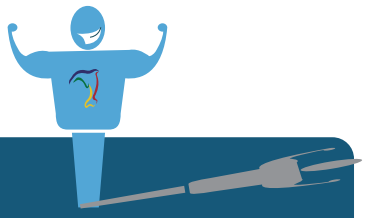
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

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

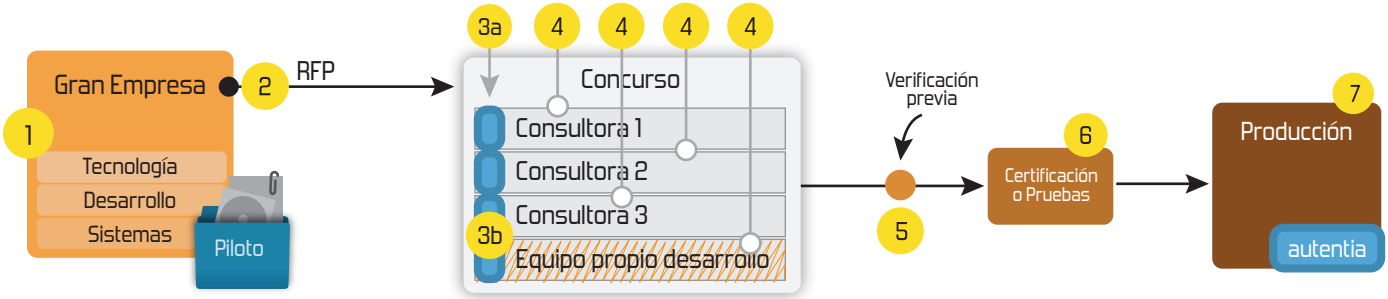
1. Desarrollo de componentes y proyectos a medida



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

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

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

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

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Carlos García Pérez

Técnico especialista en informática de empresa (CEU).

Ingeniero Técnico en Informática de Sistemas (UPM)

Creador de MobileTest, Haaala!, Girillo, toi18n.

Charla sobre desarrollo de aplicaciones en Android.

Seguir a @cgpcosmad 92 seguidores

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2009-02-26

Tutorial visitado 2 veces Descargar en PDF

Creación de un API REST protegido por OAuth2

Índice de contenidos

1. Introducción.
   - Sobre los APIs REST.
   - Sobre OAuth2.
2. El ejemplo, la aplicación SmallNotes:
   - Tecnologías usadas.
   - Descripción de los endpoints.
   - Estructura de la base de datos a los que accederá el API REST.
   - Datos de prueba a los que consultará el API REST.
   - Estructura del proyecto.
   - Las clases más importantes
   - Archivos de configuración
   - Descargar código fuente.
3. Referencias y enlaces interesantes.
4. Conclusiones.

Introducción

En este tutorial vamos a construir una aplicación web que **expone un API Rest** en donde los usuarios registrados podrán usar dicho API para gestionar sus notas (alta, baja, modificaciones y consultas).

El API estará **protegido por el protocolo estándar OAuth2** para que aplicaciones externas puedan consultar el API en nombre del usuario.

Sobre los API REST:

Rest es un estilo de arquitectura elegante y legible en donde se hace uso de los verbos HTTP para realizar las operaciones de alta (POST), baja (DELETE), modificación (PUT) y consulta (GET) de información. (Endpoints de la aplicacion del tutorial:)

Una finalidad que se persigue es la de **facilitar el trabajo a los programadores** que hacen uso de ella, de manera que se incremente su productividad.

Una buena API REST debe intentar cumplir los siguientes requisitos (obviamente además debe ser rápida y escalable):

1. Estar bien documentada.
2. En caso de errores notificar al programador de una forma clara y suficiente para que el pueda averiguar que ha pasado y si es culpa suya corregir el problema.
3. Proporcionar un mecanismo de filtro o consulta de selección de resultados.
4. Proporcionar un mecanismo de respuesta parcial, en donde el programador pueda indicar que campos de información desea obtener, normalmente para ahorrar ancho de banda e incrementar velocidad en consultas desde dispositivos con recursos limitados.
5. Estar versionado, para no afectar en futuras versiones a aplicaciones que usen versiones más antiguas del API.
6. Ofecer un mecanismo de selección del formato de los datos de salida (normalmente XML o JSON).

En este tutorial cubriremos los puntos: 2, 3, 4 y 5.

El punto 1 está documentado pero en los comentarios del código fuente, faltaría una url o documento que podrían consultar los programadores).

Para conseguir el punto 6, puedes consultar el siguiente artículo Spring MVC. Servicios REST respondiendo en JSON o XML.

Sobre OAuth2:

El **protocolo estándar OAuth2** permite que las aplicaciones puedan acceder **de manera delimitada** a los datos (u operaciones) ubicados en otro servicio o aplicación (proveedor de servicio) en nombre del usuario sin tener que tener que para ello darle las credenciales (usuario/password, etc) a esta aplicación tercera.

En la sección referencias y enlaces interesantes tenéis una **excelente charla en castellano sobre OAuth2**.

Este protocolo ofrece grandes posibilidades de integración entre aplicaciones, sin tener que compartir contraseñas, permitiendo además que el usuario pueda **revocar privilegios** a las aplicaciones cuando desee.

Anteriormente a OAuth, existian diversas soluciones propietarias para que las aplicaciones externas pudieran acceder a sus datos y/o servicios, por ejemplo, Google ya ofrecía ClientLogin y AuthSub.

OAuth2 se basa en el concepto de token de accesso, de manera que hay diversas formas de conseguir el token de acceso necesario para poder acceder al servicio sin tener que en ningún momento introducir las credenciales de acceso en la aplicación cliente que desea acceder a los datos. A este proceso de obtención del token de acceso se le conoce como **baile OAuth2**

En este tutorial se cubrirá el baile **authorization_code**, pues la aplicación cliente es una aplicación ubicada en un servidor (tomcat, jetty, apache, etc).

Es un tema extenso, si quereis profundizar más en el tema os recomiendo que leais el libro Getting Started With OAuth2 de O'Relly o que veais la charla en castellano que comenté previamente.

El ejemplo, la aplicación SmallNotes:

SmallNotes es una aplicación web que expone un API Rest en donde los usuarios registrados podrán usar dicho API para gestionar sus notas (alta, baja, modificaciones y consultas).

Las notas se componen de un titulo, un contenido y una serie de enlaces.

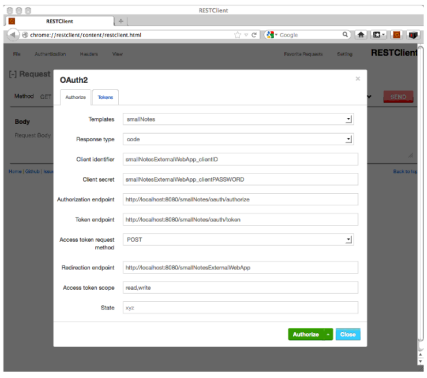
Los endpoints (urls) que expone dicho API junto con su documentación está detallado en la sección descripción de los endpoints.

Para probar el API he creado una aplicación web cliente que de consume el API que expone SmallNotes, poniéndose previamente en funcionamiento el baile OAuth2 para obtener el accessToken necesario para invocar dichos endPoints del API.

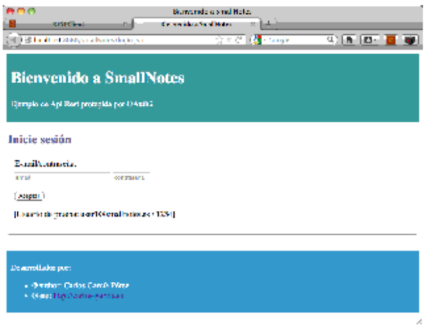
También usar el **plugin RestClient para Firefox** el cual además de permitirte hacer peticiones REST, hace de cliente OAuth2 sabiendo que tiene que hacer cuando recibe del servidor el error de "necesitas un accessToken" para acceder al recurso.
A continuación vemos unas capturas de pantalla de todo el ciclo:

Haz click en las imágenes para agrandarlas:

a) Configuración de OAuth2:

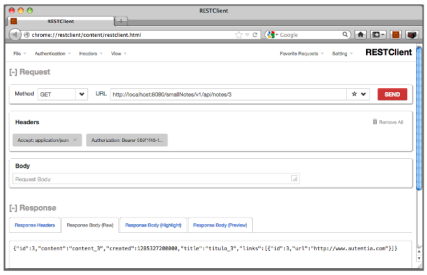


b) Solicita al usuario que se autenticación en la aplicación SmallNotes:



c) Solitica al usuario que autorización de los permisos que solicita acceder la aplicación externa:

d) Respuesta recibida:



Últimos Tutoriales del Autor

» JQuery Mobile: Desarrollo de aplicaciones y portales web para dispositivos móviles

» Geoposicionamiento Web con HTML5 y Google Maps

» Validación de formularios con el plugin JQuery Validator

» Instalación y uso del plugin de comentarios de Facebook en nuestra Web

» Construcción personalizada de objetos JSON en cliente (JavaScript)

Últimas ofertas de empleo

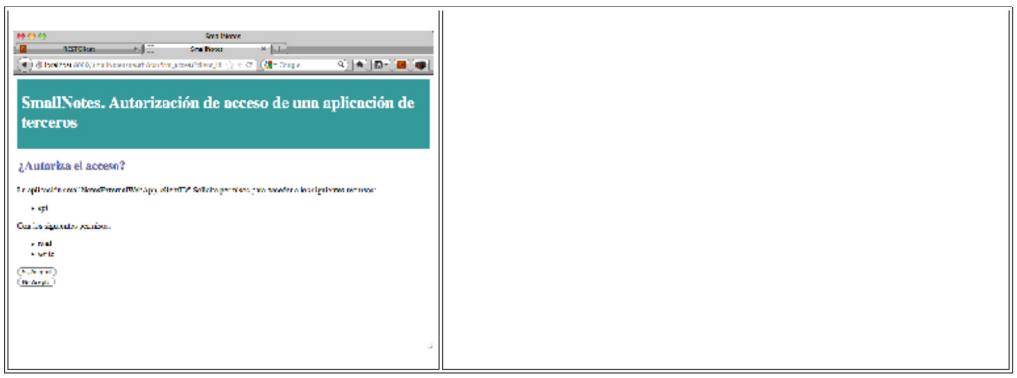
2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.



También puedes ejecutar la aplicación SmallNotesExternalWebApp:

Tecnologías usadas:

Para construir la aplicación he elegido las siguientes tecnologías:

- ***Git:***
  Como sistema de control de versiones que además permite un trabajo más ágil y cómodo.
- ***Maven:***
  Para la gestión de la construcción proyecto.
- ***JPA2:***
  Como framework de persistencia.
- ***Liquibase:***
  Como gestor del ciclo de vida de la base de datos, así como para introducir datos de prueba.
- ***Mockito, JUnit:***
  Para la creación de tests unitarios y/o integración.
- ***MySQL y H2:***
  Como base de datos para entorno de desarrollo y/o pruebas. (si no fuera un tutorial usaría **Redis** u otra solución)
- ***Spring MVC:***
  Cómo framework de desarrollo Web y su soporte REST.
- ***Spring Security:***
  Cómo framework de seguridad.
- ***Spring Security OAuth2:***
  Implementación de OAuth2 basada de Spring MVC y Spring Security.

Descripción de los endpoints:

URL:	Verbo HTTP:	Descripción:	Resultado (JSON):
/v1/api/notes/	GET	Todas las notas del usuario logado.	[{"id":1,"content":"content_1","created":1285322400000,"title":"titulo_1","links":[{"id":1,"url":"http://www.carlos-garcia.es"},{"id":2,"url":"http://www.adictosaltrabajo.com"}]},{"id":2,"content":"content_2","created":1285326000000,"title":"titulo_2"},{"id":3,"content":"content_3","created":1285327200000,"title":"titulo_3","links":[{"id":3,"url":"http://www.autentia.com"}]}]
/v1/api/notes?fields=id,title	GET	Los campos id y title de todas las notas del usuario logado **(respuesta parcial)**.	[{"id":1,"title":"titulo_1"},{"id":2,"title":"titulo_2"},{"id":3,"title":"titulo_3"}]
/v1/api/notes?q=title:titulo_1	GET	Todas las notas del usuario logado cuyo valor para el campo "title" sea "titulo_1" **(consulta o filtro)**.	[{"id":1,"content":"content_1","created":1285322400000,"title":"titulo_1","links":[{"id":1,"url":"http://www.carlos-garcia.es"},{"id":2,"url":"http://www.adictosaltrabajo.com"}]}]
/v1/api/notes/1	GET	Recupera la nota 1	{"id":1,"content":"content_1","created":1285322400000,"title":"titulo_1","links":[{"id":1,"url":"http://www.carlos-garcia.es"},{"id":2,"url":"http://www.adictosaltrabajo.com"}]}
/v1/api/notes/1/links	GET	Recupera los links de la nota con id 1 **(asociativa)**	[{"id":1,"url":"http://www.carlos-garcia.es"},{"id":2,"url":"http://www.adictosaltrabajo.com"}]
/v1/api/notes/	POST	**Crea** una nota. En la petición debemos enviar la cabecera HTTP **Content-Type:**	HTTP 200 OK

URL:	Verbo HTTP:	Descripción:	Resultado (JSON):
		application/json y en el playload de la petición: { "title": "mobiletest", "content": "Un complemento educativo para profesores y alumnos", "links": [{ "url": "http://www.mobiletest.es" }, { "url": "http://www.carlos-garcia.es" }, { "url": "http://www.autentia.com" }] }	
/v1/api/notes/5	DELETE	**Elimina** la nota con identificador 5	HTTP 200 OK
/v1/api/notes/1	PUT	**Actualiza** el contenido de la nota con identificador 1 En la petición debemos enviar la cabecera HTTP **Content-Type: application/json** y en el playload de la petición: { "title": "title_1 actualizado", "content": "content_1 actualizado", }	HTTP 200 OK

Estructura de la base de datos a los que accederá el API REST:

Al ejecutar el proyecto, **Liquibase creará la base de datos** si fuese necesario con el siguiente esquema (definidos en el archivo `/src/main/resources/liquibase/changeLog_1_0.xml`):

Aunque no esté familiarizado con Liquibase, creo que observando el archivo es suficiente como para ver las tablas y sus relaciones.

/src/main/resources/liquibase/changeLog_1_0.xml

view plain print ?

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
03.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04.     xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns
    /dbchangelog/dbchangelog-2.0.xsd">
05.
06.     <changeSet id="create_tables" author="cgarcia" context="test, developer, pre-produccion">
07.         <comment>table creations</comment>
08.
09.         <createTable tableName="smallNotesUsers">
10.             <column name="id" type="INT" autoIncrement="true">
11.                 <constraints nullable="false" primaryKey="true" />
12.             </column>
13.             <column name="email" type="VARCHAR(100)">
14.                 <constraints nullable="false" unique="true" />
15.             </column>
16.             <column name="secret" type="VARCHAR(50)">
17.                 <constraints nullable="false" />
18.             </column>
19.             <column name="name" type="VARCHAR(100)">
20.                 <constraints nullable="false" />
21.             </column>
22.             <column name="role" type="VARCHAR(10)">
23.                 <constraints nullable="false" />
24.             </column>
25.         </createTable>
26.
27.         <createTable tableName="notes">
28.             <column name="id" type="INT" autoIncrement="true">
29.                 <constraints nullable="false" primaryKey="true" />
30.             </column>
31.             <column name="title" type="VARCHAR(100)">
32.                 <constraints nullable="false" />
33.             </column>
34.             <column name="content" type="VARCHAR(500)">
35.                 <constraints nullable="false" />
36.             </column>
37.             <column name="created" type="DATETIME">
38.                 <constraints nullable="false"/>
39.             </column>
40.
41.             <column name="owner" type="INT">
42.                 <constraints nullable="false"/>
43.             </column>
44.         </createTable>
45.         <addForeignKeyConstraint constraintName="fk_user_notes" baseTableName="notes" baseColumnNames="owi
46.
47.         <createTable tableName="links">
48.             <column name="id" type="INT" autoIncrement="true">
49.                 <constraints nullable="false" primaryKey="true" />
50.             </column>
51.             <column name="url" type="VARCHAR(255)">
52.                 <constraints nullable="false" />
53.             </column>
54.             <column name="noteId" type="INT">
55.                 <constraints nullable="false" />
56.             </column>
57.         </createTable>
```

```
        <addForeignKeyConstraint constraintName="fk_links_notes" baseTableName="links" baseColumnNames="n
    </changeSet>
</databaseChangeLog>
```

Datos de prueba a los que consultará el API REST

Liquibase introducirá los siguientes datos de prueba definidos en el siguiente archivo `/src/main/resources/liquibase/changeLog_1_1-test.xml`):

/src/main/resources/liquibase/changeLog_1_1-test.xml

```
view plain print ?
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-2.0.xsd">

    <changeSet id="small_notes_inserts" author="cgarcia" context="test, developer">
        <comment>Inserts for small_notes_users</comment>
        <insert tableName="smallNotesUsers">
            <column name="id"        value="1" />
            <column name="email"     value="user1@smallnotes.es" />
            <column name="secret"    value="81dc9bdb52d04dc20036dbd8313ed055" />
            <column name="name"  value="Carlos García Pérez" />
            <column name="role"  value="ROLE_USER" />
        </insert>
        <insert tableName="smallNotesUsers">
            <column name="id"        value="2" />
            <column name="email"     value="user2@smallnotes.es" />
            <column name="secret"    value="81dc9bdb52d04dc20036dbd8313ed055" />
            <column name="name"  value="Cristóbal García" />
            <column name="role"  value="ROLE_USER" />
        </insert>
        <insert tableName="notes">
            <column name="id"        value="1" />
            <column name="title"     value="titulo_1" />
            <column name="content"   value="content_1" />
            <column name="created"   value="2010-09-24 12:00:00" />
            <column name="owner"     value="1" />
        </insert>
        <insert tableName="notes">
            <column name="id"        value="2" />
            <column name="title"     value="titulo_2" />
            <column name="content"   value="content_2" />
            <column name="created"   value="2010-09-24 13:00:00" />
            <column name="owner"     value="1" />
        </insert>
        <insert tableName="notes">
            <column name="id"        value="3" />
            <column name="title"     value="titulo_3" />
            <column name="content"   value="content_3" />
            <column name="created"   value="2010-09-24 13:20:00" />
            <column name="owner"     value="1" />
        </insert>
        <insert tableName="notes">
            <column name="id"        value="4" />
            <column name="title"     value="titulo note 4" />
            <column name="content"   value="content note 4" />
            <column name="created"   value="2010-09-24 13:00:00" />
            <column name="owner"     value="2" />
        </insert>
        <insert tableName="links">
            <column name="id"    value="1" />
            <column name="url"   value="http://www.carlos-garcia.es" />
            <column name="noteId"    value="1" />
        </insert>
        <insert tableName="links">
            <column name="id"    value="2" />
            <column name="url"   value="http://www.adictosaltrabajo.com" />
            <column name="noteId"    value="1" />
        </insert>
        <insert tableName="links">
            <column name="id"    value="3" />
            <column name="url"   value="http://www.autentia.com" />
            <column name="noteId"    value="3" />
        </insert>
    </changeSet>
</databaseChangeLog>
```

Estructura del proyecto:

Cómo se puede ver en la siguiente captura de pantalla, el proyecto tiene bastante contenido. Para hacer más legible el tutorial nos centraremos en los que directamente afectan a la temática (REST y OAuth2), dejándo a un lado detalles como Liquibase, JPA, Mockito, etc.

Creo que las clases y la configuración del proyecto están bien estructuradas semáticamente por paquetes (modelo, persistencia, servicios, oauth2, tests, etc.).

Captura de pantalla de la estructura del proyecto SmallNotes (haz clic para agrandar la imagen)

Las clases más importantes

es.carlosgarcia.smallnotes.service.SmallNotesServiceImpl

Esta clase proporciona la funcionalidad necesaria para cumplir la funcionalidad que expone el API REST.

view plain print ?

```
01. package es.carlosgarcia.smallnotes.service;
02.
03. import java.util.Date;
04. import java.util.List;
05. import java.util.Map;
06.
07. import javax.inject.Inject;
08.
09. import org.apache.commons.collections.CollectionUtils;
10. import org.slf4j.Logger;
11. import org.slf4j.LoggerFactory;
12. import org.springframework.stereotype.Service;
13. import org.springframework.transaction.annotation.Transactional;
14.
15. import es.carlosgarcia.smallnotes.model.Link;
16. import es.carlosgarcia.smallnotes.model.Note;
17. import es.carlosgarcia.smallnotes.model.User;
18. import es.carlosgarcia.smallnotes.repository.LinkRepository;
19. import es.carlosgarcia.smallnotes.repository.NoteRepository;
20. import es.carlosgarcia.smallnotes.repository.UserRepository;
21.
22. /**
23.  * Main implementation of service to work with notes.
24.  * All methods are transactional.
25.  *
26.  * @author Carlos García.
27.  * @see http://www.carlos-garcia.es
28.  * @see http://www.autentia.com
29.  */
30. @Service
31. @Transactional(readOnly=false)
32. public class SmallNotesServiceImpl implements SmallNotesService {
33.     private final Logger logger = LoggerFactory.getLogger(SmallNotesServiceImpl.class);
34.
35.     private NoteRepository noteRepository;
36.     private UserRepository userRepository;
37.     private LinkRepository linkRepository;
38.
39.     public SmallNotesServiceImpl(){
40.         super();
41.     }
42.
43.     @Inject
44.     public SmallNotesServiceImpl(NoteRepository noteRepository, UserRepository userRepository, LinkReposi
45.         super();
46.
47.         this.noteRepository = noteRepository;
48.         this.userRepository = userRepository;
49.         this.linkRepository = linkRepository;
50.     }
51.
52.     @Override
53.     public void create(Note note) {
54.         logger.debug("creating note title {}, owner: {}", note.getTitle(), note.getOwner().getId());
55.
56.         note.setCreated(new Date());
57.
58.         // Ensure that links has reference to its note
59.         List<Link> links = note.getLinks();
60.         if (links != null){
61.             for (Link link : links){
62.                 link.setNote(note);
63.             }
64.         }
65.
66.         this.noteRepository.saveOrUpdate(note);
67.
68.         if (CollectionUtils.isNotEmpty(links)){
69.             for (Link link : links) {
70.                 logger.debug("creating note link url: {}", link.getUrl());
71.                 link.setNote(note);
72.                 this.linkRepository.saveOrUpdate(link);
73.             }
74.         }
75.     }
76.
77.     @Override
78.     public void update(Note note) {
79.         Note persistedNote = this.getNote(note.getId());
80.
81.         if (persistedNote == null){
82.             throw new IllegalArgumentException("Note not exists, id: " + note.getId());
83.         }
84.
85.         persistedNote.setContent(note.getContent());
86.         persistedNote.setTitle(note.getTitle());
87.
88.         this.noteRepository.saveOrUpdate(persistedNote);
89.     }
90.
91.     @Override
92.     public void delete(Note note) {
93.         logger.debug("deleting note id {}, owner: {}", note.getId(), note.getOwner().getId());
94.         this.noteRepository.delete(note);
95.     }
96.
97.     @Override
98.     @Transactional(readOnly=true)
99.     public List<Note> getAll(User user, Map<String, Object> params) {
100.         logger.debug("getting notes by user id {}", user.getId());
```

```
101.
102.         List<Note> notes = noteRepository.getAllByUserEmail(user.getEmail(), params);
103.
104.         logger.debug("num notes readed {}", CollectionUtils.size(notes));
105.         return notes;
106.     }
107.
108.     @Override
109.     @Transactional(readOnly=true)
110.     public Note getNote(Integer noteId) {
111.         Note note = noteRepository.findByPK(Note.class, noteId);
112.         return note;
113.     }
114.
115.     @Override
116.     public User getUserByEmail(String email) {
117.         return this.userRepository.getUserByEmail(email);
118.     }
119.
120.
121.     @Override
122.     public int deleteNotes(User user) {
123.         List<Note> notes = this.getAll(user, null);
124.         int numDeleted = 0;
125.
126.         if (notes != null) {
127.             for (Note note : notes) {
128.                 this.noteRepository.delete(note);
129.                 numDeleted++;
130.             }
131.         }
132.
133.         return numDeleted;
134.     }
135. }
```

es.carlosgarcia.smallnotes.web.oauth2.api.SmallNotesAPI

Controller de Spring MVC que **implementa el API REST securizado por OAuth2**

Observe que:

- Los endpoints están protegidos con `@PreAuthorize` haciendo uso de las funciones definidas en el expresion handler `oauthWebExpressionHandler`
  (observe el archivo /src/main/resources/security/smallNotes-security-rules.xml)
- La expresión de `@PreAuthorize` indica que debe ser un accessToken perteneciente a un usuario authenticado y que además tenga un escope y un role especifico.
  Con "usuario authenticado" me refiero a que hay un baile OAuth2 de nombre Client Credentials Flow para acceder a recursos protegidos por OAuth2 desde aplicaciones sin intervención del usuario.
- Observer la sección **Exception Handlers** en donde mapeamos excepciones de negocio a instancias de RestApiError para notificar al usuario del problema.
  En la sección referencias dispone de un enlace a otra forma de realizar esta tarea así como documentación útil.

```
view plain print ?
01. package es.carlosgarcia.smallnotes.web.oauth2.api;
02.
03. import java.util.List;
04. import java.util.Map;
05.
06. import javax.servlet.http.HttpServletRequest;
07. import javax.servlet.http.HttpServletResponse;
08.
09. import org.apache.commons.lang.StringUtils;
10. import org.apache.commons.lang.Validate;
11. import org.slf4j.Logger;
12. import org.slf4j.LoggerFactory;
13. import org.springframework.beans.factory.InitializingBean;
14. import org.springframework.beans.factory.annotation.Autowired;
15. import org.springframework.http.HttpStatus;
16. import org.springframework.http.ResponseEntity;
17. import org.springframework.security.access.prepost.PreAuthorize;
18. import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
19. import org.springframework.security.core.Authentication;
20. import org.springframework.security.core.context.SecurityContextHolder;
21. import org.springframework.security.core.userdetails.UserDetails;
22. import org.springframework.security.oauth2.provider.OAuth2Authentication;
23. import org.springframework.stereotype.Controller;
24. import org.springframework.web.bind.annotation.ExceptionHandler;
25. import org.springframework.web.bind.annotation.PathVariable;
26. import org.springframework.web.bind.annotation.RequestBody;
27. import org.springframework.web.bind.annotation.RequestMapping;
28. import org.springframework.web.bind.annotation.RequestMethod;
29. import org.springframework.web.bind.annotation.RequestParam;
30. import org.springframework.web.bind.annotation.ResponseBody;
31. import org.springframework.web.bind.annotation.ResponseStatus;
32.
33. import es.carlosgarcia.smallnotes.exception.AccessDeniedException;
34. import es.carlosgarcia.smallnotes.exception.NoteNotFoundException;
35. import es.carlosgarcia.smallnotes.model.Link;
36. import es.carlosgarcia.smallnotes.model.Note;
37. import es.carlosgarcia.smallnotes.model.User;
38. import es.carlosgarcia.smallnotes.service.SmallNotesService;
39.
40. /**
41.  * API protected by OAuth2
42.  * @author Carlos Garcia
43.  * @see http://www.carlos-garcia.es
44.  * @see http://www.autentia.com
45.  */
46. @Controller
47. @RequestMapping(value = "/v1/")
48. public class SmallNotesAPI implements InitializingBean {
```

```
49.     private static final Logger logger  = LoggerFactory.getLogger(SmallNotesAPI.class);
50.
51.     private SmallNotesService smallNotesService;
52.
53.     public SmallNotesAPI(){
54.         super();
55.     }
56.
57.     @Autowired
58.     public SmallNotesAPI(SmallNotesService smallNotesService){
59.         logger.debug("Creating SmallNotesAPIImpl");
60.         this.smallNotesService = smallNotesService;
61.     }
62.
63.     @Override
64.     public void afterPropertiesSet() throws Exception {
65.         Validate.notNull(this.smallNotesService);
66.     }
67.
68.     /**
69.       * @param filter If not null, filter to be applied. Example: q?title:titulo1,content:Conteni
70.       * @param fields If not null. Partial response. Example: ?fields=id,title
71.       * @return Note list of the logged user.
72.       */
73.     @RequestMapping(value = "/api/notes", method = RequestMethod.GET)
74.     @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('read')")
75.     @ResponseBody
76.     public List<Note> getNotes(@RequestParam(required=false, value="q") String filter, @RequestParam(requ
77.         logger.debug("SmallNotesAPIImpl.getNotes. filter {}, fields {}", filter, fields);
78.
79.         User user = this.getLoggedUser();
80.
81.         Map<String, Object> params = null;
82.
83.         if (StringUtils.isNotBlank(filter)){
84.             params = SmallNotesApiHelper.parseFilter(filter);
85.         }
86.
87.         List<Note> notes = this.smallNotesService.getAll(user, params);
88.
89.         if (logger.isDebugEnabled()){
90.             int numNotes = 0;
91.             if (notes != null){
92.                 numNotes = notes.size();
93.             }
94.             logger.debug("num notes that match filter {}", numNotes);
95.         }
96.
97.         // ¿Is partial response requested?
98.         if (StringUtils.isNotBlank(fields) && (notes != null)){
99.             return SmallNotesApiHelper.applyPartialResponse(notes, fields);
100.        } else {
101.            return notes;
102.        }
103.    }
104.
105.
106.    /**
107.      * Create a note.
108.      * @param note Note to be created
109.      */
110.    @RequestMapping(value = "/api/notes", method = RequestMethod.POST)
111.    @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('write')")
112.    @ResponseStatus(value=HttpStatus.OK)
113.    public void createNote(@RequestBody Note note) {
114.        User user = this.getLoggedUser();
115.        note.setOwner(user);
116.        smallNotesService.create(note);
117.    }
118.
119.    /**
120.      * Delete all notes of user logged in.
121.      * @return The number of notes deleted.
122.      */
123.    @RequestMapping(value = "/api/notes", method = RequestMethod.DELETE)
124.    @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('write')")
125.    @ResponseStatus(value=HttpStatus.OK)
126.    @ResponseBody
127.    public int deleteNotes() {
128.        User user = this.getLoggedUser();
129.        int  numDeleted = this.smallNotesService.deleteNotes(user);
130.        return numDeleted;
131.    }
132.
133.
134.    /**
135.      * Get a note by id.
136.      * @param noteId Note identify.
137.      * @param fields If not null. Partial response. Example: ?fields=id,title
138.      * @return A note.
139.      */
140.    @RequestMapping(value = "/api/notes/{noteId}", method = RequestMethod.GET)
141.    @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('read')")
142.    @ResponseBody
143.    public Note getNote(@PathVariable Integer noteId, @RequestParam(required=false, value="fields") Strin
144.        Note note = this.getUserNote(noteId);
145.
146.        logger.debug("SmallNotesAPIImpl.getNote Id: {}", noteId);
147.
148.        if (StringUtils.isBlank(fields)){
149.            return note;
150.        } else {
151.            return SmallNotesApiHelper.applyPartialResponse(note, fields);
152.        }
153.    }
154.
```

```
155.     /**
156.      * Delete a note.
157.      * @param noteId Code of the note to be deleted
158.      */
159.     @RequestMapping(value = "/api/notes/{noteId}", method = RequestMethod.DELETE)
160.     @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('write')")
161.     @ResponseStatus(value=HttpStatus.OK)
162.     public void deleteNote(@PathVariable Integer noteId) {
163.         Note note = this.getUserNote(noteId);
164.         if (note != null){
165.             this.smallNotesService.delete(note);
166.         }
167.     }
168. 
169.     /**
170.      * Update a note.
171.      * @param noteId Note identify.
172.      * @param note Note content to be created
173.      */
174.     @RequestMapping(value = "/api/notes/{noteId}", method = RequestMethod.PUT)
175.     @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('write')")
176.     @ResponseStatus(value=HttpStatus.OK)
177.     public void updateNote(@PathVariable Integer noteId, @RequestBody Note note) {
178.         if ((note == null) || (noteId != note.getId())){
179.             throw new IllegalArgumentException("Invalid note");
180.         }
181. 
182.         // this method check that logger user is the owner of note
183.         Note notePersisted = this.getUserNote(noteId);
184. 
185.         note.setOwner(notePersisted.getOwner());
186. 
187.         this.smallNotesService.update(note);
188.     }
189. 
190.     /**
191.      * Get links of a note
192.      * @param noteId Note code
193.      * @return The links of the note.
194.      */
195.     @RequestMapping(value = "/api/notes/{noteId}/links", method = RequestMethod.GET)
196.     @PreAuthorize("#oauth2.isUser() and #oauth2.clientHasRole('ROLE_USER') and #oauth2.hasScope('read')")
197.     @ResponseBody
198.     public List<Link> getNoteLinks(@PathVariable Integer noteId) {
199.         Note note = this.getUserNote(noteId);
200. 
201.         logger.debug("SmallNotesAPIImpl.getNoteLinks Id: {}, user {}", noteId);
202. 
203.         return note.getLinks();
204.     }
205. 
206.     /**
207.      * Error if user try to access to any other url.
208.      */
209.     @RequestMapping(value = "/api/**", method = RequestMethod.GET)
210.     public void notExistsUrlHandler(HttpServletRequest request) {
211.         throw new IllegalArgumentException("Requested url not exists: " + request.getRequestURI());
212.     }
213. 
214.     /**
215.      * Get a note validate that user is the owner.
216.      * @param noteId Note code to get.
217.      * @return Return the note
218.      */
219.     private Note getUserNote(Integer noteId) {
220.         Note note = this.smallNotesService.getNote(noteId);
221. 
222.         if (note == null){
223.             throw new NoteNotFoundException(noteId);
224.         }
225. 
226.         User user = this.getLoggedUser();
227.         if (! user.equals(note.getOwner())){
228.             throw new AccessDeniedException(noteId);
229.         }
230. 
231.         return note;
232.     }
233. 
234.     /**
235.      * @return The logged user
236.      */
237.     private User getLoggedUser(){
238.         String email =  null;
239.         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
240.         Object     principal  = authentication.getPrincipal();
241. 
242. 
243.         if (principal instanceof UserDetails) {
244.             email =  ((UserDetails) principal).getUsername();
245.         } else if (principal instanceof UsernamePasswordAuthenticationToken){
246.             email = ((UsernamePasswordAuthenticationToken) principal).getName();
247.         } else if (principal instanceof OAuth2Authentication){
248.             email = ((OAuth2Authentication) principal).getUserAuthentication().getName();
249.         }
250. 
251.         logger.debug("Logged user email {}", email);
252. 
253.         Validate.notNull(email);
254. 
255.         User user  = this.smallNotesService.getUserByEmail(email);
256. 
257.         logger.debug("Logged user {}", user);
258. 
259.         return user;
260.     }
```

```
261.
262.
263.     /**
264.      * Exception Handlers
265.      */
266.
267.     @ExceptionHandler(NoteNotFoundException.class)
268.     protected @ResponseBody ResponseEntity<RestApiError> handleNoteNotFoundException(NoteNotFoundExceptio
269.         RestApiError restApiError = new RestApiError(HttpStatus.NOT_FOUND, ApiErrorCode.NOTE_NOT_FOUND, n
270.         return SmallNotesApiHelper.createAndSendResponse(restApiError);
271.     }
272.
273.     @ExceptionHandler(AccessDeniedException.class)
274.     protected @ResponseBody ResponseEntity<RestApiError> handleAccessDeniedException(AccessDeniedExceptio
275.         RestApiError restApiError = new RestApiError(HttpStatus.UNAUTHORIZED, ApiErrorCode.ACCESS_DENIED,
276.         return SmallNotesApiHelper.createAndSendResponse(restApiError);
277.     }
278.
279.     @ExceptionHandler(SecurityException.class)
280.     protected @ResponseBody ResponseEntity<RestApiError> handleSecurityException(SecurityException except
281.         RestApiError restApiError = new RestApiError(HttpStatus.UNAUTHORIZED, ApiErrorCode.SECURITY, excep
282.         return SmallNotesApiHelper.createAndSendResponse(restApiError);
283.     }
284.
285.     @ExceptionHandler(Exception.class)
286.     protected  @ResponseBody  ResponseEntity<RestApiError> handleException(Exception exception, HttpServl
287.         RestApiError restApiError = new RestApiError(HttpStatus.BAD_REQUEST, ApiErrorCode.GENERIC, except
288.         return SmallNotesApiHelper.createAndSendResponse(restApiError);
289.     }
290.
291.     private String getInfoUrl(ApiErrorCode code){
292.         return "http://yourAppUrlToDocumentedApiCodes.com/api/support/" + code.ordinal();
293.     }
294. }
```

es.carlosgarcia.smallnotes.web.oauth2.api.SmallNotesApiHelper

Esta clase tiene métodos de utilidad relacionados con el procesado de parámetros de filtro y respuesta parcial.

```
view plain  print  ?
01. package es.carlosgarcia.smallnotes.web.oauth2.api;
02.
03. import java.beans.PropertyDescriptor;
04. import java.lang.reflect.InvocationTargetException;
05. import java.util.ArrayList;
06. import java.util.HashMap;
07. import java.util.List;
08. import java.util.Map;
09. import java.util.Set;
10. import java.util.StringTokenizer;
11. import java.util.TreeSet;
12.
13. import org.apache.commons.beanutils.PropertyUtils;
14. import org.apache.commons.lang.ArrayUtils;
15. import org.springframework.beans.BeanUtils;
16. import org.springframework.http.HttpHeaders;
17. import org.springframework.http.MediaType;
18. import org.springframework.http.ResponseEntity;
19.
20. import es.carlosgarcia.smallnotes.model.Note;
21.
22. /**
23.  * Helper method for SmallNotesApi
24.  * @author Carlos García
25.  * @see http://www.carlos-garcia.es
26.  * @see http://www.autentia.com
27.  */
28. public class SmallNotesApiHelper {
29.
30.     public static ResponseEntity<RestApiError> createAndSendResponse(RestApiError restApiError) {
31.         HttpHeaders headers = new HttpHeaders();
32.         headers.set("Cache-Control", "no-store");
33.         headers.set("Pragma", "no-cache");
34.         headers.setContentType(MediaType.APPLICATION_JSON);
35.         ResponseEntity<RestApiError> responseEntity = new ResponseEntity<RestApiError>
(restApiError, headers, restApiError.getHttpStatusCode());
36.         return responseEntity;
37.     }
38.
39.     /**
40.      * Parse filter string to build a map with format <paramFieldKey,paramFieldValue>.
41.      * @param filter Query String, Example: title:titulo1,content:Contenido1
42.      * @return A map with format <paramFieldKey,paramFieldValue>
43.      */
44.     public static Map<String, Object> parseFilter(String filter) {
45.         HashMap<String, Object> map = new HashMap<String, Object>();
46.         StringTokenizer fields  = new StringTokenizer(filter, ",");
47.         Note note = new Note();
48.         String fieldName  = null;
49.
50.         try {
51.             while (fields.hasMoreTokens()){
52.                 StringTokenizer field = new StringTokenizer(fields.nextToken(), ":");
53.                 if (field.countTokens() != 2){
54.                     throw new IllegalArgumentException(filter);
55.                 }
56.
57.                 fieldName  = field.nextToken();
58.                 String fieldValue = field.nextToken();
59.
60.                 // Verify that fieldName exists as field on Note class
61.                 PropertyUtils.getSimpleProperty(note, fieldName);
62.
63.                 map.put(fieldName, fieldValue);
```

```
            }
        } catch (NoSuchMethodException e) {
            throw new IllegalArgumentException(String.format("Property %s does not exists", fieldName));
        } catch (IllegalAccessException ex){
            throw new IllegalArgumentException(filter);
        } catch (InvocationTargetException ex){
            throw new IllegalArgumentException(filter);
        }

        return map;
    }

    /**
     * Apply partial response to Note
     * @param notes Full property note
     * @param fields Fields to return.
     * @return A Note with only "fields" with values.
     */
    public static Note applyPartialResponse(Note note, String fields){
        String[] fieldsToIgnore =  SmallNotesApiHelper.constructPartialResponseFieldsToIgnore(fields);
        Note partialResponseNote = new Note();
        BeanUtils.copyProperties(note, partialResponseNote, fieldsToIgnore);
        return partialResponseNote;
    }


    /**
     * Apply partial response.
     * @param notes Full property note list
     * @param fields Fields to return.
     * @return A Note list with only "fields" with values.
     */
    public static List<Note> applyPartialResponse(List<Note> notes, String fields){
        String[] fieldsToIgnore =  SmallNotesApiHelper.constructPartialResponseFieldsToIgnore(fields);

        List<Note> partialResponseNotes = new ArrayList<Note>(notes.size());
        for (Note note : notes){
            Note partialResponseNote = new Note();
            BeanUtils.copyProperties(note, partialResponseNote, fieldsToIgnore);
            partialResponseNotes.add(partialResponseNote);
        }

        // Important: Note class is has @JsonSerialize(include = Inclusion.NON_DEFAULT) setting c
        return partialResponseNotes;
    }


    /**
     * @param fields Fields that user want to get response only.
     * @return And String[] with the others fields, (fields to be ignored)
     */
    private static String[] constructPartialResponseFieldsToIgnore(String fields){
         String[] partialResponseFields = fields.split(",");
         String   currentFieldName = null;

         Note note = new Note();

         // First, validate that all fields exists on class
         try {
             for (int i = 0, num = partialResponseFields.length; i < num; i++){
                 currentFieldName = partialResponseFields[i];
                 PropertyUtils.getSimpleProperty(note, currentFieldName);
             }
         } catch (NoSuchMethodException e) {
             throw new IllegalArgumentException(String.format("Property %s does not exists", currentFieldN
         } catch (IllegalAccessException ex){
             throw new IllegalArgumentException(fields);
         } catch (InvocationTargetException ex){
             throw new IllegalArgumentException(fields);
         } catch (Exception ex){
             throw new IllegalArgumentException(fields);
         }


         Set<String> fieldsToIgnoreList = new TreeSet<String>();

        // Constuct Field to Ignore array
        PropertyDescriptor[] properties = BeanUtils.getPropertyDescriptors(note.getClass());
        for (int i = 0, numProperties = properties.length; i < numProperties; i++){
            if (! ArrayUtils.contains(partialResponseFields, properties[i].getName())){
                fieldsToIgnoreList.add(properties[i].getName());
            }
        }
        String[] fieldsToIgnore =  fieldsToIgnoreList.toArray(new String[0]);

        return fieldsToIgnore;
    }
}
```

es.carlosgarcia.smallnotes.web.oauth2.api.RestApiError

Representa el error de negocio que será enviado (en JSON) al cliente cuando sea necesario.

```
view plain  print  ?
package es.carlosgarcia.smallnotes.web.oauth2.api;

import org.springframework.http.HttpStatus;

/**
 * Rest api error.
 * @author Carlos García
 * @see http://www.carlos-garcia.es
 * @see http://www.autentia.com
 */
```

```
11. public class RestApiError {
12. 
13.     private final HttpStatus httpStatusCode;
14.     private final ApiErrorCode apiCode;
15.     private final String message;
16.     private final String devMessage;
17.     private final String infoUrl;
18. 
19.     public RestApiError(HttpStatus httpStatusCode, ApiErrorCode apiCode, String message, String devMessag
20.         this.httpStatusCode = httpStatusCode;
21.         this.apiCode = apiCode;
22.         this.message = message;
23.         this.devMessage = devMessage;
24.         this.infoUrl = infoUrl;
25.     }
26. 
27.     public HttpStatus getHttpStatusCode() {
28.         return this.httpStatusCode;
29.     }
30. 
31.     public ApiErrorCode getApiCode() {
32.         return this.apiCode;
33.     }
34. 
35.     public String getMessage() {
36.         return this.message;
37.     }
38. 
39.     public String getDeveloperMessage() {
40.         return this.devMessage;
41.     }
42. 
43.     public String getInfoUrl() {
44.         return this.infoUrl;
45.     }
46. }
```

es.carlosgarcia.smallnotes.web.oauth2.api.SmallNotesApiTest

Tests unitarios de la clase `es.carlosgarcia.smallnotes.web.oauth2.api.SmallNotesApi`.

```
view plain print ?
01. package es.carlosgarcia.smallnotes.web.oauth2.api;
02. 
03. import org.junit.Before;
04. import org.junit.Test;
05. import org.junit.runner.RunWith;
06. import org.mockito.Mock;
07. import org.mockito.Mockito;
08. import org.mockito.runners.MockitoJUnitRunner;
09. import org.springframework.security.core.Authentication;
10. import org.springframework.security.core.context.SecurityContext;
11. import org.springframework.security.core.context.SecurityContextHolder;
12. import org.springframework.security.oauth2.provider.OAuth2Authentication;
13. 
14. import es.carlosgarcia.smallnotes.model.Note;
15. import es.carlosgarcia.smallnotes.model.User;
16. import es.carlosgarcia.smallnotes.service.SmallNotesService;
17. 
18. /**
19.  * Unit tests for class SmallNotesApiImpl
20.  * @author Carlos García
21.  * @see http://www.carlos-garcia.es
22.  * @see http://www.autentia.com
23.  */
24. @RunWith(MockitoJUnitRunner.class)
25. public class SmallNotesApiTest {
26. 
27.     @Mock
28.     private SecurityContext securityContext;
29. 
30.     @Mock
31.     private OAuth2Authentication principal;
32. 
33.     @Mock
34.     private SmallNotesService smallNotesService;
35. 
36.     @Mock
37.     private Authentication authentication;
38. 
39.     @Mock
40.     private User loggedUser;
41. 
42. 
43.     @Before
44.     public void initAuthentication(){
45.         final String USER_EMAIL_1 = "cgpcosmad@gmail.com";
46. 
47.         Mockito.when(securityContext.getAuthentication()).thenReturn(authentication);
48.         Mockito.when(authentication.getPrincipal()).thenReturn(principal);
49. 
50.         Mockito.when(principal.getUserAuthentication()).thenReturn(authentication);
51.         Mockito.when(authentication.getName()).thenReturn(USER_EMAIL_1);
52.         Mockito.when(smallNotesService.getUserByEmail(USER_EMAIL_1)).thenReturn(loggedUser);
53. 
54.         SecurityContextHolder.setContext(securityContext);
55.     }
56. 
57.     @Test
58.     public void shouldGetAllUserNotesWhenNoFilter(){
59.         Mockito.when(smallNotesService.getAll(loggedUser, null)).thenReturn(null);
60. 
61.         SmallNotesAPI smallNotesApi = new SmallNotesAPI(smallNotesService);
62. 
```

```
        smallNotesApi.getNotes(null, null);

        // Verify that getAll method has been called
        Mockito.verify(smallNotesService).getAll(loggedUser, null);
    }


    @Test
    public void shouldCreateNote(){
        Note note = Mockito.mock(Note.class);
        Mockito.when(note.getId()).thenReturn(null);

        SmallNotesAPI smallNotesApi = new SmallNotesAPI(smallNotesService);
        smallNotesApi.createNote(note);

        // Verify that create method has been called
        Mockito.verify(smallNotesService).create(note);
    }


    @Test
    public void shouldDeleteNote(){
        final int NOTE_ID_TO_BE_DELETED = 1;

        Note note = Mockito.mock(Note.class);
        Mockito.when(note.getId()).thenReturn(NOTE_ID_TO_BE_DELETED);
        Mockito.when(note.getOwner()).thenReturn(this.loggedUser);
        Mockito.when(smallNotesService.getNote(NOTE_ID_TO_BE_DELETED)).thenReturn(note);

        SmallNotesAPI smallNotesApi = new SmallNotesAPI(smallNotesService);
        smallNotesApi.deleteNote(NOTE_ID_TO_BE_DELETED);

        // Verify that delete method has been called.
        Mockito.verify(smallNotesService).delete(note);
    }
}
```

Archivos de configuración

/src/main/resources/security/security.xml

Configuración normal de los beans necesarios para una seguridad clásica por formulario de usuario/contraseña:

```
view plain print ?
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:sec="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org
/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/security http://www.springframework.org/schema
/security/spring-security-3.1.xsd
                http://www.springframework.org/schema/mvc http://www.springframework.org/schema
/mvc/spring-mvc-3.1.xsd
                http://www.springframework.org/schema/security/oauth2 http://www.springframework.org
/schema/security/spring-security-oauth2-1.0.xsd">

    <sec:jdbc-user-service id="userService" data-source-ref="dataSource"
    users-by-username-
query="select email as username, secret as password, true as enabled from smallNotesUsers where email=?"
    authorities-by-username-
query="select email as username, role as authorities from smallNotesUsers where email=?" />

    <sec:authentication-manager alias="authenticationManager">
        <sec:authentication-provider user-service-ref="userService">
            <sec:password-encoder hash="md5" />
        </sec:authentication-provider>
    </sec:authentication-manager>
</beans>
```

/src/main/resources/security/smallNotes-security-rules.xml

Definición de la protección de endpoints, en donde podemos observar:

- En la url `/oauth/token` es donde en el baile oauth2, un usuario ya authenticado (en la url /oauth/authorize) obtiene un accessToken para el token de autorización que obtuvo en la primera parte del baile de oauth2.

  Los parámetros que debe recibir son los especificados en el estándar de OAuth2 (code, el hash de client_id+client_secret, state y redirect_uri), estos serán procesados en el filtro `clientCredentialsTokenEndpointFilter` y validados en `clientAuthenticationManager`

- La url `/v1/api/notes/**` define la protección del Api REST por OAuth2. Será el filtro `resourceServerFilter` el que extraiga el accessToken de BD (haciendo uso de TokenStore) e introducciendo el OAuth2Authentication en el contexto de seguridad de SpringSecurity
  `SecurityContextHolder.getContext().setAuthentication(authentication);`

```
view plain print ?
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:sec="http://www.springframework.org/schema/security" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance"  xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org
/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/security http://www.springframework.org/schema
/security/spring-security-3.1.xsd
                http://www.springframework.org/schema/mvc http://www.springframework.org/schema
/mvc/spring-mvc-3.1.xsd
                http://www.springframework.org/schema/security/oauth2 http://www.springframework.org
/schema/security/spring-security-oauth2-1.0.xsd">
```

```
09.
10.     <sec:http pattern="/oauth/token" create-session="stateless" authentication-manager-
    ref="clientAuthenticationManager" entry-point-ref="oauthAuthenticationEntryPoint" xmlns="http:
    //www.springframework.org/schema/security">
11.         <sec:intercept-url pattern="/oauth/token" access="IS_AUTHENTICATED_FULLY" />
12.         <sec:anonymous enabled="false" />
13.         <sec:http-basic entry-point-ref="oauthAuthenticationEntryPoint" />
14.         <sec:custom-filter ref="clientCredentialsTokenEndpointFilter" before="BASIC_AUTH_FILTER" />
15.         <sec:access-denied-handler ref="oauthAccessDeniedHandler" />
16.     </sec:http>
17.
18.     <sec:http pattern="/v1/api/notes/**" create-session="never" use-expressions="true"
19.               entry-point-ref="oauthAuthenticationEntryPoint" access-decision-manager-
    ref="accessDecisionManager" xmlns="http://www.springframework.org/schema/security">
20.         <sec:anonymous      enabled="false" />
21.         <sec:custom-filter          ref="resourceServerFilter" before="PRE_AUTH_FILTER" />
22.         <sec:access-denied-handler  ref="oauthAccessDeniedHandler" />
23.         <sec:expression-handler     ref="oauthWebExpressionHandler" />
24.     </sec:http>
25.
26.
27.     <sec:http  access-denied-page="/WEB-INF/views/messages/notPrivileged.jsp" disable-
    url-rewriting="true">
28.         <sec:intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
29.         <sec:form-login authentication-failure-url="/login.jsp" login-page="/login.jsp"  default-target-
    url="/home.jsp"  login-processing-url="/j_spring_security_check"/>
30.         <sec:logout  logout-success-url="/login.jsp" logout-url="/logout" />
31.         <sec:anonymous />
32.     </sec:http>
33. </beans>
```

/src/main/resources/security/smallNotes-security-oauth2.xml

En el siguiente archivo destacaremos:

- **tokenStore**: Es la clase que se encargará de acceder a las tablas de base de datos relacionadas con los accessTokens.
- **clientDetailsService**: Es la clase que se encargará de acceder a las tablas de base de datos relaconadas con las aplicaciones clientes que solicitan autorización, es decir, esta clase se usa principalmente en el baile OAuth2 para conseguir el accessToken).
- **userApprovalHandler**: Esta clase se encarga de ver si una determinada petición ya estaba aprobada por el usuario.
- **accessConfirmationController**: Esta clase se encarga de mostrar el gui de solicitud de autorización así como de recibir la respuesta del usuario en ese gui.
- **oauth:authorization-server**: Este tag se engarga de crear y configurar los Beans y Filtros necesarios tanto para el proceso de authorización como para el endpoint de entrega de accessToken una vez que el usuario ha autorizado a la aplicación. Más concretamente este tag crea dos Beans principales:
  - **AuthorizationEndPoint:** Se encarga del proceso de autorización OAuth2 generando el token de autorización y redirigiendo al usuario al endPoint dónde escucha TokenEndPoint.
  - **TokenEndPoint:** Se encarga de generar un accessToken (y persistirlo) para un código de autorización que recibe como parámetro.
- **resourceServerFilter**: Esta clase se encarga de registrar un filtro que extraerá el accessToken de la cabecera HTTP **Authorization** (o el parametro accessToken) consultará en BD ese accessToken y si existe construirá el **OAut2Authentication** y lo meterá en el contexto de SpringSecurity.

```
view plain  print  ?
01. <?xml version="1.0" encoding="UTF-8"?>
02. <beans xmlns="http://www.springframework.org/schema/beans"
03.         xmlns:sec="http://www.springframework.org/schema/security" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance"  xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
04.             xmlns:mvc="http://www.springframework.org/schema/mvc"
05.         xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org
    /schema/beans/spring-beans.xsd
06.                 http://www.springframework.org/schema/security http://www.springframework.org/schema
    /security/spring-security-3.1.xsd
07.                 http://www.springframework.org/schema/mvc http://www.springframework.org/schema
    /mvc/spring-mvc-3.1.xsd
08.                 http://www.springframework.org/schema/security/oauth2 http://www.springframework.org
    /schema/security/spring-security-oauth2-1.0.xsd">
09.
10.     <oauth:expression-handler id="oauthExpressionHandler" />
11.     <oauth:web-expression-handler id="oauthWebExpressionHandler" />
12.
13.     <sec:global-method-security pre-post-annotations="enabled" proxy-target-class="true">
14.         <sec:expression-handler ref="oauthExpressionHandler" />
15.     </sec:global-method-security>
16.
17.
18.     <bean id="tokenStore" class="org.springframework.security.oauth2.provider.token.JdbcTokenStore">
19.         <constructor-arg index="0" ref="dataSource"/>
20.     </bean>
21.
22.     <bean id="clientDetailsUserService" class="org.springframework.security.oauth2.provider.client.ClientI
23.         <constructor-arg  ref="clientDetailsService"/>
24.     </bean>
25.
26.     <bean id="clientDetailsService" class="org.springframework.security.oauth2.provider.JdbcClientDetails
27.             <constructor-arg index="0" ref="dataSource"/>
28.     </bean>
29.
30.
31.     <sec:authentication-manager id="clientAuthenticationManager">
32.         <sec:authentication-provider user-service-ref="clientDetailsUserService" />
33.     </sec:authentication-manager>
34.
35.
36.     <bean id="tokenServices" class="org.springframework.security.oauth2.provider.token.DefaultTokenServic
37.         <property name="tokenStore"       ref="tokenStore" />
38.
39.         <!-- Default is 12 hours but i increase to 5 days -->
40.         <property name="accessTokenValiditySeconds" value="432000" />
41.
42.         <property name="supportRefreshToken" value="true" />
43.     </bean>
44.
```

```
    <bean id="userApprovalHandler" class="es.carlosgarcia.smallnotes.web.oauth2.UserApprovalHandler">
        <property name="tokenServices" ref="tokenServices" />
    </bean>

    <bean id="accessConfirmationController" class="es.carlosgarcia.smallnotes.web.oauth2.AccessConfirmatic
        <property name="clientDetailsService" ref="clientDetailsService" />
    </bean>

    <bean id="oauthAuthenticationEntryPoint" class="org.springframework.security.oauth2.provider.error.OAu

            <bean id="oauthAccessDeniedHandler" class="org.springframework.security.oauth2.provider.error
    <bean id="clientCredentialsTokenEndpointFilter" class="org.springframework.security.oauth2.provider.c
        <property name="authenticationManager" ref="clientAuthenticationManager"/>
    </bean>
    <bean id="accessDecisionManager" class="org.springframework.security.access.vote.UnanimousBased">
        <constructor-arg>
            <list>
                <bean class="org.springframework.security.oauth2.provider.vote.ScopeVoter" />
                <bean class="org.springframework.security.access.vote.RoleVoter" />
                <bean class="org.springframework.security.access.vote.AuthenticatedVoter" />
            </list>
        </constructor-arg>
    </bean>

    <oauth:authorization-server client-details-service-ref="clientDetailsService" token-services-
ref="tokenServices"
                                user-approval-handler-ref="userApprovalHandler"
                                token-endpoint-url="/oauth/token" authorization-endpoint-url="/oauth
/authorize"
                                user-approval-page="redirect:/oauth/confirm_access">
        <oauth:authorization-code />
    </oauth:authorization-server>

    <oauth:resource-server id="resourceServerFilter" resource-id="api" token-services-
ref="tokenServices" />
</beans>
```

/src/main/resources/liquibase/changeLog_OAuth2.xml

Es un archivo de creación de las tablas relacionadas con OAuth2 en formato Liquibase, el cuál se he creado extrayendo dicha definición del código fuente de Spring Security OAuth2.

Un par de observaciones a destacar:

1. El campo authentication de la tabla oauth_access_token es una serialización de la instancia **OAuth2Authentication asociada a ese accessToken, es decir a los permisos exactos que concedió el usuario en el momento de crear el token.**
   Posteriormente, un filtro de Spring Security OAuth2 se encargará de consultar esta tabla para ver si el token recibido en la Cabecera HTTP Authorization (o parámetro accessToken) existe e asociarlo al contexto de Spring Security.
2. La tabla oauth_client_details contiene las **aplicaciones autorizadas** para obtener accessTokens. (mediante el baile OAuth).
3. El campo authorized_grant_types de la tabla oauth_client_details indica que clases de bailes permitimos para ese cliente, implicit, authorization_code, etc. (es un tema extenso, ver la documentación o preguntarme).
4. El campo scope de la tabla oauth_client_details nos sirve para securizar los recursos de forma lógica.

   Por ejemplo scopes de Facebook y los scopes de Google

```
view plain  print  ?
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns
/dbchangelog/dbchangelog-2.0.xsd">

    <changeSet id="oauth2_db_model_create" author="cgarcia" context="test, developer, pre-produccion">
        <createTable tableName="oauth_access_token">
            <column name="token_id" type="VARCHAR(256)"/>
            <column name="token" type="BLOB"/>
            <column name="authentication_id" type="VARCHAR(256)"/>
            <column name="user_name" type="VARCHAR(256)"/>
            <column name="client_id" type="VARCHAR(256)"/>
            <column name="authentication" type="BLOB"/>
            <column name="refresh_token" type="VARCHAR(256)"/>
        </createTable>

        <createTable tableName="oauth_client_details">
            <column name="client_id" type="VARCHAR(256)">
                <constraints nullable="false" primaryKey="true"/>
            </column>
            <column name="resource_ids" type="VARCHAR(256)"/>
            <column name="client_secret" type="VARCHAR(256)"/>
            <column name="scope" type="VARCHAR(256)"/>
            <column name="authorized_grant_types" type="VARCHAR(256)"/>
            <column name="web_server_redirect_uri" type="VARCHAR(256)"/>
            <column name="authorities" type="VARCHAR(256)"/>
            <column name="access_token_validity" type="INT"/>
            <column name="refresh_token_validity" type="INT"/>
            <column name="additional_information" type="VARCHAR(4096)"/>
        </createTable>

        <createTable tableName="oauth_code">
            <column name="code" type="VARCHAR(256)"/>
            <column name="authentication" type="BLOB"/>
        </createTable>

        <createTable tableName="oauth_refresh_token">
            <column name="token_id" type="VARCHAR(256)"/>
            <column name="token" type="BLOB"/>
            <column name="authentication" type="BLOB"/>
        </createTable>
    </changeSet>

    <changeSet id="oauth_client_details_Inserts" author="cgarcia" context="test">
```

```
45.         <comment>Insert one OAuth2Client</comment>
46.         <insert tableName="oauth_client_details">
47.             <column name="client_id"                value="smallNotesExternalWebApp_clientID" />
48.             <column name="client_secret"            value="smallNotesExternalWebApp_clientPASSWORD" />
49.             <column name="authorized_grant_types"   value="authorization_code" />
50.             <column name="authorities"              value="ROLE_USER, ROLE_CLIENT" />
51.             <column name="scope"                value="read,write" />
52.             <column name="resource_ids"             value="api" />
53.
54.             <column name="access_token_validity" value="500" />
55.             <column name="refresh_token_validity"   value="50000" />
56.         </insert>
57.     </changeSet>
58. </databaseChangeLog>
```

Descargar código fuente

A continuación puede descargarse los proyectos.

- Descargar proyecto SmallNotes
- Descargar proyecto SmallNotesExternalWebApp.

Ambos proyectos son projectos maven, por lo que puede por ejemplo iniciarlos mediante el comando:

- `mvn tomcat:run -Dmaven.tomcat.port=8080` (smallNotes)
- `mvn tomcat:run -Dmaven.tomcat.port=9080` (smallNotesExternalWebApp)

Para smallNotes, debes establecer las propiedades de conexión a la bd (que será creada automáticamente) en el archivo `appcore.properties`

Referencias y enlaces interesantes:

- Charla de SpringIO 2012: Securing your REST API with OAuth.
- Especificación del protocolo OAuth 2.0.
- Github de Spring Security OAuth2
- Spring MVC REST Exception Handling Best Practices.
- Los scopes de Facebook y los scopes de Google

Conclusiones

A fecha de hoy, 25-Julio-2012, el proyecto Spring Security OAuth2, está a punto de ser una release (continua en milestone). Desde mi punto de vista, aunque el código fuente es de gran calidad y viene con un buen ejemplo, he echado mucho en falta más documentación para poder comprender como encajan las piezas, así que en muchos casos he tenido que ir analizando el código fuente.

Supongo que será cuestión de tiempo que la comunidad vaya nutriendo este estupendo proyecto con más documentación, tutoriales y ejemplos.

Es una implementación madura de la especificación OAuth2 (y OAuth1), además es muy completa pues se cubren **todos los flujos (grantTypes)** para la obtención de accessToken para cada tipo de aplicación (JavaScript, móviles, servidor, dispositivo tipo televisión, ...)

Espero que os haya sido de utilidad.

Un saludo.
Carlos García.


W3C XHTML 1.0
W3C CSS
XML RSS
XML ATOM