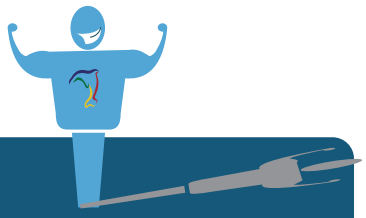
autentia

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...

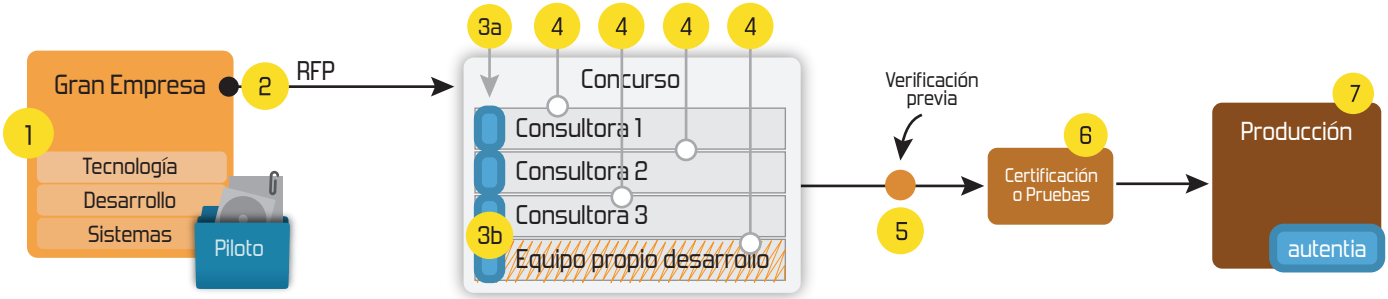
1. Desarrollo de componentes y proyectos a medida



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

AdictosAlTrabajo
Terrakas 1x04 ¡¡Ya está en la web!! ;-) terrakas.com
autentia Soporte a desarrollo informático
Hosting patrocinado por enredados
Entra en Adictos a través de
E-mail
Contraseña
Entrar
Deseo registrarme
Olvidé mi contraseña
Inicio Quiénes somos Formación Comparador de salarios Nuestro libro Más
» Estás en: Inicio Tutoriales Servicios REST documentados y probados con Swagger

Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

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

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Sourcefire is a Leader in Gartner Magic Quadrant for IPS
Access the Report
SOURCEfire

Fecha de publicación del tutorial: 2012-11-08 Tutorial visitado 5 veces Descargar en PDF

Servicios REST documentados y probados con Swagger.

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. Creando el servicio.
- 4. Acerca de Swagger.
  - 4.1 ¿Qué es Swagger?.
- 5. Documentando nuestro servicio con Swagger.
  - 5.1 pom.xml.
  - 5.2 web.xml.
  - 5.3 El filtro de Swagger.
  - 5.4 El servicio documentado.
- 6. El cliente de Swagger.
- 7. Referencias.
- 8. Conclusiones.

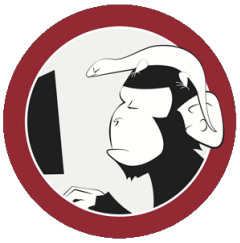
1. Introducción

¿Servicios SOAP o REST? He ahí la cuestión. Independientemente del debate, una de las ventajas de SOAP sobre REST podría considerarse que los primeros establecen un contrato más estricto gracias a la especificación WSDL. Sin embargo esta "ventaja" puede ser a su vez un inconveniente debido a que WSDL es una especificación relativamente compleja, por lo que muchos desarrolladores se decantan por REST. REST, además, cuenta con WADL (Web Application Description Language) que permite documentar este tipo de Servicios Web de una manera más sencilla que con WSDL. Nótese que REST también permite documentación con WSDL (2.0) pero, volvemos otra vez a lo mismo, suele ser considerada una especificación muy compleja.

Pues bien, en respuesta a este dilema, los chicos de Wordnik se han inventado una nueva especificación (y su correspondiente implementación para distintas tecnologías) llamada Swagger para documentar servicios REST. Y alguien podrá decir: *"¡Otra más! ¿No tenemos suficiente con dos?"*. Pues la verdad es que visto así parece que no tiene mucho sentido. De hecho creo que, como especificación, no va a tener gran acogida (tal vez me equivoque...), pero hay una característica que me parece muy interesante. Swagger proporciona un cliente web para poder acceder de forma muy cómoda y vistosa a la documentación de nuestros servicios. Además permite probar dichos servicio de forma extremadamente sencilla.

En este tutorial veremos cómo documentar y probar nuestros servicios REST con Swagger y las alternativas que ofrece a WSDL o WADL.

Catálogo de servicios Autentia



Síguenos a través de:

Últimas Noticias

» Mi retrospectiva sobre la CAS 2012

» Autentia estuvo en el Duatlón de Torrejón de Ardoz

» Participamos en la Carrera de las Empresas 2012

» ¡¡¡Terrakas 1x04 recién salido del horno!!!

» Estreno Terrakas 1x04: "Terraka por un día"

Histórico de noticias

Últimos Tutoriales

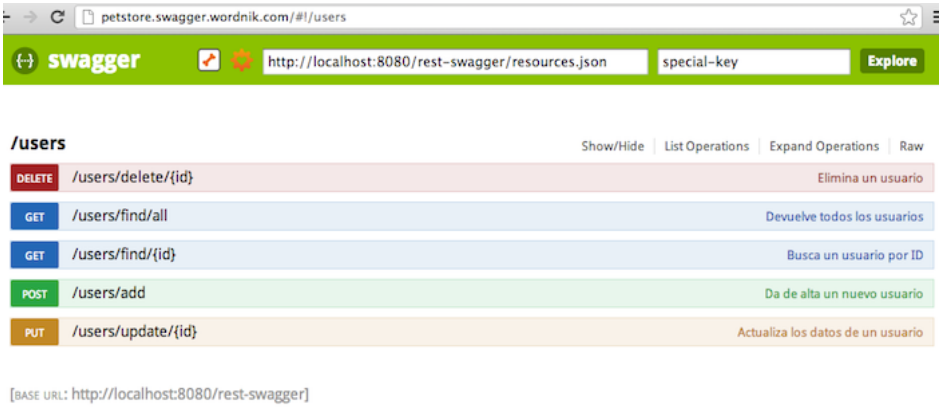
» Insertando contenidos HTML en nuestra aplicación IOS

» Integrar Barcode Scanner en nuestra aplicación Android

» Tutorial de iniciación en Framework ZK

» Creación de un Widget JavaScript usando Backbone.js y Require.js

» Robolectric: aplicando TDD en Android



2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Mountain Lion 10.8
- Entorno de desarrollo: Intellij Idea 11.1 Ultimate.
- Swagger 1.1.0
- Jersey 1.14
- Maven 3.0.3
- Apache Tomcat 6.0.32

3. Creando el servicio.

Lo primero que haremos será crear nuestro servicio. Para ello utilizaremos Jersey, la implementación de referencia de la especificación JAX-RS para la creación de Servicios REST. El que quiera saber más sobre Jersey puede consultar estos tutoriales de Germán y Álvaro.

Para gestionar la configuración haremos uso de Maven. Añadiremos las siguientes dependencias a nuestro fichero pom.xml

```
        com.sun.jersey
        jersey-server
        1.14


        com.sun.jersey
        jersey-servlet
        1.14


        com.sun.jersey
        jersey-json
        1.14
```

Nuestro servicio se compone de un API (/users/) formado por **operaciones sobre usuarios**:

- **Buscar todos** (/find/all): devuelve un listado con todos los usuarios del sistema.
- **Buscar usuario** (/find/{id}): devuelve la información relativa a un usuario (id).
- **Crear usuario** (/add): crea un usuario en el sistema.
- **Actualizar usuario** (/update/{id}): actualiza la información relativa a un usuario (id).
- **Borrar usuario** (/delete/{id}): elimina la información relativa a un usuario (id).

Nuestra clase UserService sería la siguiente (se omiten los detalles de la implementación de las operaciones):

```
package com.autentia.tutorial.rest;


import com.autentia.tutorial.rest.data.Message;
import com.autentia.tutorial.rest.data.User;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.util.HashMap;
import java.util.Map;

@Path("/users/")
@Produces(MediaType.APPLICATION_JSON)
public class UserService {

    private static final Map<Integer, User> USERS = new HashMap<Integer, User>();
    static {
        USERS.put(1, new User(1, "Juan"));
        USERS.put(2, new User(2, "Pepe"));
        USERS.put(3, new User(3, "Antonio"));
    }

    private static final Response NOT_FOUND = Response.status(Response.Status.NOT_FOUND).
            entity(new Message("El usuario no existe")).
            build();

    private static final Response USER_ALREADY_EXISTS = Response.status(Response.Status.BAD_REQUEST).
            entity(new Message("El usuario ya existe")).
```

Últimos Tutoriales del Autor

» Creación de plantillas DSL con Drools

» Introducción a Drools.

» Jugando con JSON en Java y la librería Gson

» WebSockets con Java y Tomcat 7

» Introducción a Apache ActiveMQ

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.

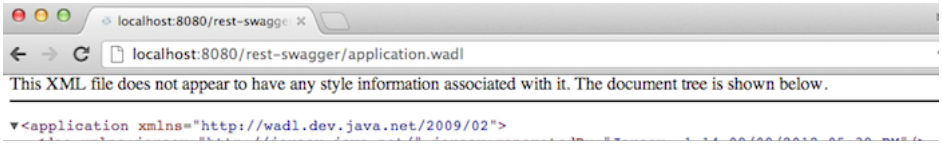
2011-07-06
Otras Sin catalogar - LUGO.

```
                build();

        private static final Response OK = Response.ok().
                entity(new Message("Operacion realizada corretamente")).
                        build();

        @GET
        @Path("/find/all")
        public Response findAll() {
            // código
        }

        @GET
        @Path("/find/{id}")
        public Response findById(@PathParam("id") int id) {
            // código
        }

        @POST
        @Path("/add")
        @Consumes({MediaType.APPLICATION_JSON})
        public Response addUser(User newUser) {
            // código
        }

        @PUT
        @Path("/update/{id}")
        @Consumes({MediaType.APPLICATION_JSON})
        public Response updateUser(@PathParam("id") int id, User userToUpdate) {
            // código
        }

        @DELETE
        @Path("/delete/{id}")
        public Response removeUser(@PathParam("id") int id) {
            // código
        }

    }
```

Por último registramos el Servlet de Jersey en nuestro web.xml

```
        JerseyTest
        com.sun.jersey.spi.container.servlet.ServletContainer

                com.sun.jersey.api.json.POJOMappingFeature
                true


                com.sun.jersey.config.property.resourceConfigClass
                com.sun.jersey.api.core.PackagesResourceConfig


                com.sun.jersey.config.property.packages
                com.autentia.tutorial.rest

        1


        JerseyTest
        /*
```

Por defecto, Jersey nos creará un WADL con la información de nuestro servicio REST.



Pues con esto ya tendríamos nuestro servicio creado. A continuación veremos cómo documentarlo y probarlo con Swagger.

4. Acerca de Swagger.

De momento no hemos visto nada nuevo que no se pudiera hacer con Jersey.

4.1 ¿Qué es Swagger?.

Como comentamos en la introducción, Swagger es una **especificación** y su correspondiente **implementación** para probar y documentar servicios REST. Uno de los objetivos de Swagger es que podamos actualizar la documentación en el mismo instante en que realizamos los cambios en el servidor.

Dicha especificación se compone de dos partes:

- Resource Listing: devuelve un listado con todas las APIs a documentar/probar.
- API Declaration: devuelve la información relativa a un API, como sus operaciones o el modelo de datos.

5. Documentando nuestro servicio con Swagger.

LLegados a este punto ya estamos en disposición de explicar cómo podemos documentar y probar nuestro servicio con Swagger. El proceso consta de una serie de pasos que iremos explicando.

5.1 pom.xml.

Lo primero que haremos será añadir la dependencia de swagger para JAX-RS en nuestro pom.xml.

```
com.wordnik
swagger-jaxrs_2.9.1
1.1.0
compile
```

5.2 web.xml.

A continuación modificamos nuestro web.xml. Debemos añadir dos parámetros al Servlet de Jersey: **swagger.api.basepath y api.version**. Además, a la propiedad **com.sun.jersey.config.property.packages** añadimos le añadimos el paquete de swagger para la especificación JAX-RS (com.wordnik.swagger.jaxrs).

```
        JerseyTest
        com.sun.jersey.spi.container.servlet.ServletContainer

                com.sun.jersey.api.json.POJOMappingFeature
                true


                com.sun.jersey.config.property.resourceConfigClass
                com.sun.jersey.api.core.PackagesResourceConfig


                com.sun.jersey.config.property.packages
                com.autentia.tutorial.rest;com.wordnik.swagger.jaxrs


                swagger.api.basepath
                http://localhost:8080/rest-swagger


                api.version
                1.0

        1
```

5.3 El filtro de Swagger.

Aunque no es necesario, puede ser recomendable utilizar un filtro que actúe sobre nuestras peticiones con el fin de configurar algún comportamiento específico de Swagger.

Por defecto, para acceder a la información de nuestras APIs con Swagger debemos añadir a nuestra URI el sufijo .{sufijo} donde sufijo es el formato en el que queramos que nos devuelva la información. Suele ser JSON o XML. Vamos a modificarlo para que no lo tenga en cuenta y la información nos la devuelva en la URI donde responde la operación de nuestra API. Lo hacemos en el método init con **JaxrsApiReader.setFormatString("")**. ¡OJO! que no se nos olvide registrar el filtro en el web.xml.

Además, queremos poder acceder a la información de nuestra API desde un navegador en una aplicación (web) que puede estar alojada en cualquier dominio. Lo haremos vía Javascript con el cliente que trae Swagger para consumir la información que devuelve sobre nuestro servicio (en el siguiente punto lo veremos con más detalle). Para ello permitimos el acceso a la información desde cualquier sitio añadiendo la información en la cabecera que se ve en el método doFilter de nuestro filtro. También indicamos los métodos (GET, POST, PUT, DELETE) que permitimos a nuestro cliente.

```
package com.autentia.tutorial.rest.filter;

import com.wordnik.swagger.jaxrs.JaxrsApiReader;

import javax.servlet.*;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class SwaggerFilter implements javax.servlet.Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        JaxrsApiReader.setFormatString("");
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException, ServletException {

        HttpServletResponse res = (HttpServletResponse) response;
        res.addHeader("Access-Control-Allow-Origin", "*");
        res.addHeader("Access-Control-Allow-Methods", "GET, POST, DELETE, PUT");
        res.addHeader("Access-Control-Allow-Headers", "Content-Type");

        chain.doFilter(request, response);

    }

    @Override
    public void destroy() {
    }

}
```

5.4 El servicio documentado.

Y lo último que nos falta es documentar nuestro servicio. Lo primero es extender de la clase **JavaHelp**:

```
package com.autentia.tutorial.rest;


import com.autentia.tutorial.rest.data.Message;
import com.autentia.tutorial.rest.data.User;
import com.wordnik.swagger.annotations.Api;
import com.wordnik.swagger.annotations.ApiOperation;
import com.wordnik.swagger.annotations.ApiParam;
import com.wordnik.swagger.jaxrs.JavaHelp;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
```

```
import java.util.HashMap;
import java.util.Map;

@Path("/users/")
@Api(value = "/users", description = "Operaciones con usuarios")
@Produces(MediaType.APPLICATION_JSON)
public class UserService extends JavaHelp {

    private static final Map<Integer, User> USERS = new HashMap<Integer, User>();
    static {
        USERS.put(1, new User(1, "Juan"));
        USERS.put(2, new User(2, "Pepe"));
        USERS.put(3, new User(3, "Antonio"));
    }

    private static final Response NOT_FOUND = Response.status(Response.Status.NOT_FOUND).
            entity(new Message("El usuario no existe")).
            build();

    private static final Response USER_ALREADY_EXISTS = Response.status(Response.Status.BAD_REQUEST).
            entity(new Message("El usuario ya existe")).
            build();

    private static final Response OK = Response.ok().
            entity(new Message("Operacion realizada corretamente")).
                    build();

    @GET
    @Path("/find/all")
    @ApiOperation(
            value = "Devuelve todos los usuarios",
            notes = "Devuelve todos los usuarios del sistema"
    )
    public Response findAll() {
        // código
    }

    @GET
    @Path("/find/{id}")
    @ApiOperation(
            value = "Busca un usuario por ID",
            notes = "Devuelve los datos relativos a un usuario"
    )
    public Response findById(
            @ApiParam(value = "ID del usuario a buscar", allowableValues = "range[1," + Integer.MAX_VALUE + "]", required = true)
            @PathParam("id") int id) {
        // código
    }

    @POST
    @Path("/add")
    @Consumes({MediaType.APPLICATION_JSON})
    @ApiOperation(
            value = "Da de alta un nuevo usuario",
            notes = "Crea un nuevo usuario a partir de un ID y un nombre. El usuario no debe existir"
    )
    public Response addUser(
            @ApiParam(value = "Datos del nuevo usuario", required = true)
            User newUser) {
                // código
    }

    @PUT
    @Path("/update/{id}")
    @Consumes({MediaType.APPLICATION_JSON})
    @ApiOperation(
            value = "Actualiza los datos de un usuario",
            notes = "Actualiza los datos del usuario que se corresponda con el id. El usuario debe existir"
    )
    public Response updateUser(@PathParam("id") int id,
                               @ApiParam(value = "Datos del usuario a actualizar", required = true)
                               User userToUpdate) {
        // código
    }

    @DELETE
    @Path("/delete/{id}")
    @ApiOperation(
            value = "Elimina un usuario",
            notes = "Elimina los datos del usuario que se corresponda con el id. El usuario debe existir"
    )
    public Response removeUser(
            @ApiParam(value = "ID del usuario a eliminar", allowableValues = "range[1," + Integer.MAX_VALUE + "]", required = true)
            @PathParam("id") int id) {
        // código
    }

}
```

Como vemos, Swagger nos ofrece una serie de anotaciones para poder documentar nuestra API, sus operaciones y parámetros. Sinceramente, no he encontrado en la documentación el listado completo de anotaciones :-S.

Una vez tenemos esto ya podemos acceder a la información que nos devolverán las operaciones Resource Listing y API Declaration de la especificación.

Obtención de la información relativa a nuestras APIs (solo tenemos una, users).


```
{"apiVersion":"1.0","swaggerVersion":"1.1","basePath":"http://localhost:8080/rest-swagger","apis":
[{"path":"/users","description":"Operaciones con usuarios"}]}
```


Obtención de la información relativa a nuestra API users.


```
{"apiVersion":"1.0","swaggerVersion":"1.1","basePath":"http://localhost:8080/rest-
swagger","resourcePath":"/users","apis":[{"path":"/users/delete/{id}","description":"Operaciones con
usuarios","operations":[{"httpMethod":"DELETE","summary":"Elimina un usuario","notes":"Elimina los datos del usuario
que se corresponda con el id. El usuario debe existir","responseClass":"void","nickname":"removeUser","parameters":
[{"name":"id","description":"ID del usuario a eliminar","paramType":"path","allowableValues":
{"valueType":"RANGE","min":1.0,"max":2.14748365E9,"valueType":"RANGE"},"required":true,"allowMultiple":false,"dataType
":"int"}]}]},{"path":"/users/find/all","description":"Operaciones con usuarios","operations":
[{"httpMethod":"GET","summary":"Devuelve todos los usuarios","notes":"Devuelve todos los usuarios del
sistema","responseClass":"void","nickname":"findAll"}]},{"path":"/users/find/{id}","description":"Operaciones con
usuarios","operations":[{"httpMethod":"GET","summary":"Busca un usuario por ID","notes":"Devuelve los datos relativos
a un usuario","responseClass":"void","nickname":"findById","parameters":[{"name":"id","description":"ID del usuario a
buscar","paramType":"path","allowableValues":
{"valueType":"RANGE","min":1.0,"max":2.14748365E9,"valueType":"RANGE"},"required":true,"allowMultiple":false,"dataType
":"int"}]}]},{"path":"/users/add","description":"Operaciones con usuarios","operations":
[{"httpMethod":"POST","summary":"Da de alta un nuevo usuario","notes":"Crea un nuevo usuario a partir de un ID y un
nombre. El usuario no debe existir","responseClass":"void","nickname":"addUser","parameters":[{"description":"Datos
del nuevo usuario","paramType":"body","required":true,"allowMultiple":false,"dataType":"User"}]}]},
{"path":"/users/update/{id}","description":"Operaciones con usuarios","operations":
[{"httpMethod":"PUT","summary":"Actualiza los datos de un usuario","notes":"Actualiza los datos del usuario que se
corresponda con el id. El usuario debe existir","responseClass":"void","nickname":"updateUser","parameters":
[{"name":"id","paramType":"path","required":true,"allowMultiple":false,"dataType":"int"},{"description":"Datos del
usuario a actualizar","paramType":"body","required":true,"allowMultiple":false,"dataType":"User"}]}]}],"models":
{"User":{"id":"User","properties":{"id":{"type":"int"},"name":{"type":"string"}}}}}
```

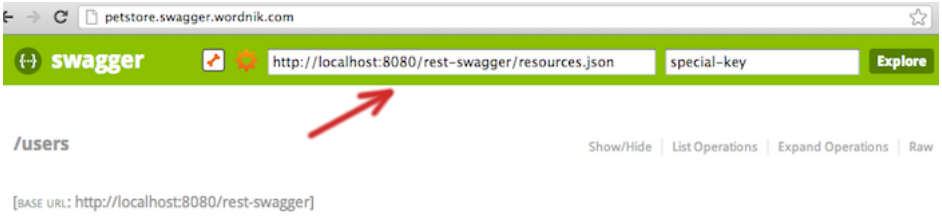

Pero no hemos montado todo esto para ver estos "chorizos" de JSON en un navegador. Vamos a ver qué nos ofrece el cliente de Swagger :-).

6. El cliente de Swagger.

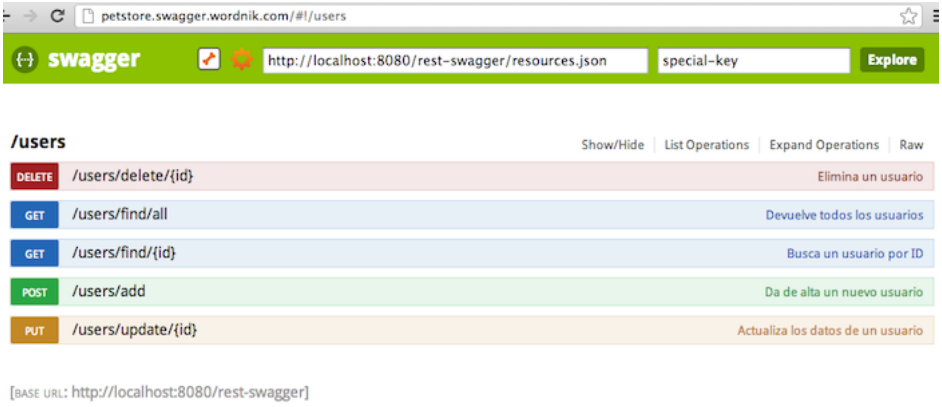
Swagger nos ofrece un cliente (web) para poder ver la información relativa a nuestras APIs y poder probarlas. Son únicamente recursos Javascript, HTML, CSS, por lo que podemos instalarla en cualquier servidor o incluso abrir directamente la página html directamente con un navegador.

Ellos la tienen colgada en esta dirección http://petstore.swagger.wordnik.com y podemos usarla tranquilamente para consumir la información de nuestra API.

Introducimos en la caja de texto de la parte superior la URL de nuestras APIs (operación Resource Listing de la especificación de Swagger) y vemos la información de todas nuestras APIs.

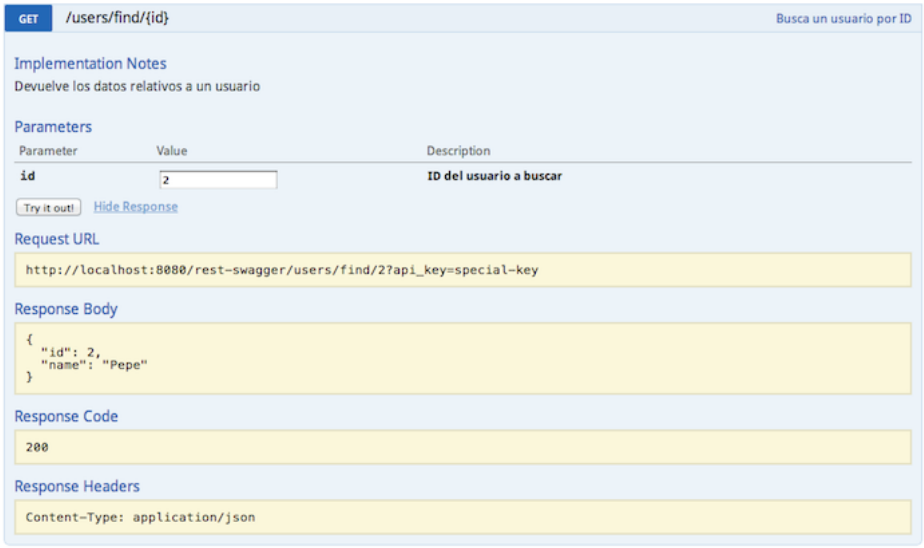


Pulsando sobre el API vemos las operaciones que la componen (operación API Declaration de la especificacion).



Si pulsamos sobre una de nuestras operaciones veremos la información relativa a la misma.

Podemos probar la operación rellenando los parámetros necesarios (si los hubiese) y pulsando sobre el botón **"Try it out!"**. Al hacerlo obtendremos los datos de respuesta.



7. Referencias.

- Swagger: integración con JAX-RS
- Swagger-ui

8. Conclusiones.

En este tutorial hemos visto cómo documentar y probar nuestros servicios REST gracias a Swagger. Me gustaría destacar algunas de las ventajas e inconvenientes:

Al ser un proyecto relativamente "joven" tiene varios inconvenientes como la falta de documentación y algún que otro bug en determinados escenarios. Por otro lado, la gran cantidad de anotaciones que requiere para la documentación puede ensuciar un poco el código.

Su gran ventaja, sin duda, el cliente web que permite acceder a la documentación del API y a sus operaciones de manera muy cómoda y comprensible, además de permitir probar las operaciones.

Si merece la pena usarlo o no dependerá de cada uno... :-)

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

SOME RIGHTS RESERVED Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

IMPULSA Impulsores Comunidad ¿Ayuda?

---- 0 personas han traído clicks a esta página

sin clicks + + + + + + + +

powered by karmacracy

Copyright 2003-2012 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto

XHTML 1.0 CSS XML RSS XML ATOM