WebServices REST

Dirección General de Tecnologías de la Información y Comunicaciones

Áreas de Gobierno Tecnológico de SSII y del Dato

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Contenido

Resumen

- **Versión:** v01r03
- **Fecha publicación:** 11 oct 2016
- **Entrada en vigor desde:** 11 oct 2016

Cumplimiento normativo

Las normas expuestas son de obligado cumplimiento. La STIC podrá estudiar los casos excepcionales los cuales serán gestionados a través de los responsables del proyecto correspondiente y autorizados por el Área de Gobernanza de la STIC. Asimismo cualquier aspecto no recogido en estas normas deberá regirse en primera instancia por las guías técnicas correspondientes al esquema nacional de seguridad y esquema nacional de interoperabilidad según correspondencia y en su defecto a los marcos normativos y de desarrollo software establecidos por la Junta de Andalucía, debiendo ser puesto de manifiesto ante la STIC.

La STIC se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata según la planificación de cada proyecto.

En el caso de que algún actor considere conveniente y/o necesario el incumplimiento de alguna de las normas y /o recomendaciones, deberá aportar previamente la correspondiente justificación fehaciente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida por la STIC para proceder a su validación técnica.

Contacto Arquitectura: l-arquitectura.stic@juntadeandalucia.es

Obsolescencia

Los nuevos desarrollos enmarcados dentro de la STIC hacen uso de algunas soluciones tecnológicas más actuales. El documento actual, pretende servirle de ayuda para aquellos desarrollos basados en tecnología JEE, puede ser que su solución provea de una pila tecnológica más actual que la que aquí está descrita.

Histórico de cambios

Los cambios en la normativa vendrán acompañados de un registro de las modificaciones. De este modo se podrá realizar un seguimiento y consultar su evolución. Ordenándose de mas recientes a menos recientes, prestando especial cuidado a las cabezeras de la tablas dónde se indican las fechas de entrada en vigor y versión.

Versión	v01r03	**Fecha publicación**	26 abr 2021	**Fecha entrada en vigor**	9 jun 2020
Alcance					
• Pautas para la implementación de OPTIONS y otras consideraciones para clientes web.					

Versión	v01r02	**Fecha publicación**	7 abr 2020	**Fecha entrada en vigor**	11 oct 2016
Alcance					

- Modificaciones de las rutas de los repositorios.
- Advertencias de obsolescencia tecnológica para los nuevos desarrollos.

Versión	v01r01	**Fecha publicación**	11 oct 2016	**Fecha entrada en vigor**	11 oct 2016
Alcance					

- Versión inicial sobre las normas y recomendaciones sobre como crear servicios y clientes REST.

1. Introducción

En el siguiente texto tiene como objetivo mostrar cómo debe generarse y usarse un servicio REST siguiendo la normativa JEE de la STIC. Para comprender mejor las indicaciones dadas, se ha creado un ejemplo ilustrativo partiendo del Arquetipo base "javaee7-sample" de la STIC. Vea la documentación asociada al arquetipo base para comprender cómo configurar el proyecto.

El ejemplo modificado puede descargarse de la siguiente ruta: http://git.sas.junta-andalucia.es/gobernanza/javaee7-sample/-/tree/RestIntegration/

El ejemplo realizado cuenta con las siguientes funcionalidades:

- Recurso REST usando JAX-RS
- Cliente REST usando JAX-RS
- Autenticación de usuarios en llamadas REST

Otras funcionalidades relacionadas presentes en el ejemplo:

- Autentificación y autorización de usuarios con MACO
- Securización de la información usando Javascript Object Signing and Encryption (JOSE)
- Uso de Apache Shiro como sistema de gestión de autenticación y autorización de usuarios.

De cara a priorizar futuras funcionalidades, si algún proveedor tiene la necesidad de alguna funcionalidad adicional se deberá de comunicar a l-arquitectura.stic.sspa@juntadeandalucia.es

2. Uso de JAX-RS

Acorde con la arquitectura de referencia seguida por la STIC, se debe usar la especificación JAX-RS 2.0 o superior para operar con servicios web RESTful.

- En la siguiente web podrá consultar toda la documentación de especificación: https://jcp.org/en/jsr/detail?id=339
- En el siguiente enlace puede encontrarse toda la documentación sobre cómo crear servicios web REST en JEE7: https://docs.oracle.com/javaee/7/tutorial/jaxrs.htm

2.1 Implementación de verbos:

Además de los verbos previstos dentro del api rest, GET,UPDATE,DELETE,PATCH que serán determinados por las necesidades de negocio identificadas, es de obligada implementación el verbo OPTIONS para todos los recursos rest que se expongan.

La implementación de los verbos se realizará en base al standard definido en https://tools.ietf.org/html/rfc7231 y con las apreciaciones que la Oficina Técnica de Interoperabilidad pudiera realizar al respecto.

2.1.1 OPTIONS:

El metodo HTTP OPTIONS se usa para definir las opciones de comunicación para un determinado recurso, es decir, el cliente puede mandar una petición OPTIONS para averiguar que métodos y otras opciones son admitidas por dicho recurso. Estas peticiones solo devuelven datos, es decir, el servidor no debe cambiar su estado.

Estas solicitudes se utilizan en el navegador como una "solicitud de verificación previa", un mecanismo en CORS para comprobar si el recurso destino aceptará la petición final o no. Dicho mecanismo consiste en enviar el metodo HTTP OPTIONS con `Access-Control-Allow-Origin`, `Access-Control-Request-Method` y `Access-Control-Request-Headers` en la cabecera para notificar al servidor sobre el tipo de petición que se quiere enviar. La respuesta deberá ser 200 (OK) para que que la petición real sea enviada.

Para que lar respuesta dada por el servidor a una petición OPTIONS se considere correcta, su solución debe enviar todos los campos en el header que puedan indicar características adicionales implementadas y aplicables al recurso contextualizado por la URI (pej CORS) . Es decir puede incluir información especial y adicional que no está en el standard indicado RFC7231 por esta via.

Debe tener en consideración las siguientes cuestiones:

- Especifique un Content-lenght con valor 0 si no hay payload en el body de la respuesta.
- Si por parte del cliente espera algo en el payload de la request, es necesario definir el tipo mime del documento que espera recibir en el verbo options, ya que el cliente se verá obligado a informarlo según el estandard.

2.1.1.1 CORS Prefligh Request:

La actual especificación para clientes web, determina hacer uso del estandard de fetch que está implementado en los navegadores web. Tenga en cuenta las apreciaciones de dicho estandard al respecto de como tratar CORS haciendo uso del verbo OPTIONS en su implementación, ya que será de obligado cumplimiento para que sus servicios puedan consumirse correctamente desde los clientes web.

Concretamente tenga en cuenta que su servicio debe implementar soporte para las solicitudes de preflight y estas deben responder en las cabeceras con:

- `Access-Control-Request-Method` Indicando que verbos se esperan en futuras peticiones para el recurso contextualizado.
- `Access-Control-Request-Headers`Indicando que headers se esperan en futuras peticiones para el recurso contextualizado.

2.2 Headers:

Las especificaciones de clientes web basadas en fetch, presentan características que hasta ahora no eran habituales en la cabezeras, concretamene la capitalización de las mismas. Tenga en cuenta las siguientes consideraciones que son de obligado cumplimiento para que sus servicios operen correctamente con clientes web:

- Todas las propiedades de las cabeceras a incluir han de ser byte-case-insensitive, aunque se recomienda el uso de: PascalCase (capitalizando todas las palabras) + kebab-case (separando las palabras con guiones) p.e. "Content-Type"

2.3 Servicios existentes en el proyecto de ejemplo

Centrándonos en el proyecto de ejemplo que se ha creado para mostrar las capacidades de esta especificación, nos encontramos con las siguientes clases de mayor interés:

RestConfig.java

Clase encargarda de crear el contexto JAX-RS, permite también realizar algunas acciones de configuración.

```
import  javax.ws.rs.ApplicationPath;
import  javax.ws.rs.core.Application;

@ApplicationPath ( "/rest" )
@javax .enterprise.context.RequestScoped
public class RestConfig  extends Application {

    public RestConfig() { }
}
```

La anotación `@ApplicationPath("/rest")` especifica la raíz de los servicios REST publicados, en nuestro ejemplo sería `http://localhost:7001/sample/rest/`

PatientsREST.java

Clase que ofrece un recurso REST para acceder a historias clínicas:

```
import javax.inject.Inject;
import  javax.ws.rs.GET;
import  javax.ws.rs.Path;
import  javax.ws.rs.PathParam;
import  javax.ws.rs.Produces;
import  javax.ws.rs.core.MediaType;

@Path ( "/patients" )
public class PatientsREST {

    private Patients patients;

    @Inject
    public PatientsREST(Patients patients) {
        this .patients = patients;
    }

    @GET
    @Produces (MediaType.APPLICATION_JSON)
    @Path ( "/{nuhsa}" )
    public Patient get( @PathParam ( "nuhsa" ) String nuhsa) {

        return patients.findByNuhsa(nuhsa);
    }
}
```

En este caso, existe un único servicio para acceder a una historia clínica a partir de su nuhsa. Por ejemplo, con la siguiente petición GET solicitaríamos la historia con nuhsa "AN000000002": `http://localhost:7001/sample/rest/patients/AN000000002`

Cuya respuesta sería:

```
{
   "id" :  3 ,
   "nuhsa" :  "AN000000002" ,
   "name" :  "Juan German" ,
   "email" :  "un.usuario.cualquiera@juntadeandalucia.es" ,

   "phone" :  "955000000" ,
   "address" :  "Americo Vespucio" ,
   "cityRegion" :  "SV" ,
   "ccNumber" :  "10"
}
```

EpisodesREST.java

Importante: Este servicio está securizado, por favor lea el apartado "Notas sobre securización" Clase que ofrece un recurso REST para acceder a episodios clínicos:

```
import java.util.logging.Logger;

import javax.inject.Inject;
import  javax.ws.rs.GET;
import  javax.ws.rs.Path;
import  javax.ws.rs.PathParam;
import  javax.ws.rs.Produces;
import  javax.ws.rs.container.ContainerRequestContext;
import  javax.ws.rs.core.Context;
import  javax.ws.rs.core.MediaType;

@Path ( "/episodes" )
public class EpisodesREST {
    private Logger log =
Logger.getLogger(EpisodesREST. class .getName());

    private Episodes episodes;

    @Inject
    public EpisodesREST(Episodes episodes) {
        this .episodes = episodes;
    }

    @GET
    @Produces (MediaType.APPLICATION_JSON)
    @Path ( "/{episodeId}" )
    public Episode get( @Context ContainerRequestContext request,

@PathParam ( "episodeId" )  int episodeId) {
        return episodes.find(episodeId);
    }
}
```

De forma similar al caso anterior, se ofrece un servicio de ejemplo que permite acceder a un episodio a partir de su identificador. Un ejemplo de llamada GET a este servicio sería: `http://localhost:7001/sample/rest/episodes/10`

Cuya respuesta debe ser:

```
{
  "id" : 10,
  "patient" : {
    "id" : 3,
    "nuhsa" : "AN000000002",
    "name" : "Juan German",
    "email" : "un.usuario.cualquiera@juntadeandalucia.es",

    "phone" : "955000000",
    "address" : "Americo Vespucio",
    "cityRegion" : "SV",
    "ccNumber" : "10"
  },
  "service" : "000646",
  "admissionDate" : 1368698400000
}
```

Nota sobre securización

Este mismo proyecto ha sido creado con el propósito de ser un ejemplo para la creación de servicios REST así como para ser un ejemplo de cómo securizar las comunicaciones entre módulos. Con el objetivo de mostrar las diferentes posibilidades se parametrizado la aplicación para que el servicio "PatientsREST" NO esté securizado, es decir, se puede realizar peticiones al servicio sin enviar ningún tipo de credencial, y el servicio "EpisodesREST" está securizado siguiendo la normativa de comunicaciones entre aplicativos, es decir, para poder realizar una llamada al servicio "EpisodesREST" se debe enviar las credenciales del usuario que está realizando la petición. Vea la normativa de comunicación y securización entre aplicativos para obtener más información.

Cómo probar el ejemplo

Para probar los servicios REST podemos hacer uso de aplicaciones de terceros destinadas a este propósito, como por ejemplo puede ser Postman. A la hora de probar el ejemplo nos encontramos dos escenarios diferentes:

- PatientsREST: Como se ha mencionado anteriormente, este servicio NO está securizado, por lo que no se necesita aportar ninguna credencial, siendo suficiente realizar una petición GET para probar el servicio.
    - Ejemplo: http://localhost:7001/sample/rest/patients/AN000000002
- EpisodesREST: Este servicio, al estar securizado, requiere que su invocación sea acompañada de un parámetro en el header de la petición llamado credentials que debe llevar la información del usuario que está realizando la petición. Puesto que los tickets JWE tienen fecha de expiración, es necesario generar un nuevo ticket para probar este servicio. De la misma forma, los tests proporcionados con el ejemplo (JweTest.java) hacen uso de tickets JWE de ejemplo e igualmente hay que regenerarlos para poder ejecutar los tests. Acceda a la ruta http://localhost:7001/sample/faces/pages/login/permissions.xhtml, lógese con un usuario de MACO Preproducción y le generará el ticket firmado en JWE. Finalmente, para probar el servicio bastaría con invocar mediante GET la ruta http://localhost:7001/sample/rest/episodes/10 añadiendo un parámetro en el header de la petición llamado "credentials", cuyo valor será el texto que hemos copiado previamente llamado "JWS encriptado con JWE"

2.4 Clientes REST

Hasta ahora hemos hablado de servicios REST que ofrece nuestro proyecto de ejemplo, pero igualmente podemos crear clientes REST para conectarnos a servicios externos. En el proyecto de ejemplo se han creado las clases PatientsClientTest.java y EpisodesClientTest.java, ambas dentro del paquete de Test que incorpora el proyecto donde se realiza una serie de tests de comunicación con servicios REST con y sin securización de llamadas. Es importante destacar que para el caso concreto del test EpisodesClientTest.java, es necesario modificar el usuario y contraseña de MACO Pre antes de lanzar el test para poder generar el ticket de autenticación correctamente.

En general nos encontramos con dos formas de crear clientes REST:

Opción 1

```
@Test
public void simpleGetPatientCorrect() {
     String nuhsa =  "AN000000002";

     try{
         Patient patient = ClientBuilder.newClient()
                 .target( "http://localhost:7001/sample/rest" )
                 .path( "patients/{nuhsa}" )
                 .resolveTemplate( "nuhsa", nuhsa)
                 .request(MediaType.APPLICATION_JSON)
                 .get(Patient.class );

         Assert.assertNotNull(patient);
         Assert.assertEquals(nuhsa, patient.getNuhsa());
     } catch (NotFoundException e) {
         Assert.fail( "Patient " + nuhsa +  " not found in server" );

     }
}
```

Esta opción tiene como positivo el hecho de que la petición get devuelve directamente un objeto del tipo de la entidad que estamos trabajando, en este caso Patient, liberando al programador de tener que procesar la respuesta. Por contra, requiere añadir tantos capturadores de excepciones como sean necesarios para capturar todos los posibles errores que se deseen capturar, como el caso de NotFoundException en nuestro ejemplo.

Opción 2

```
@Test
public void simpleGetPatientCorrect() {
    String nuhsa =  "AN000000002";

    Response response = ClientBuilder.newClient()
            .target( "http://localhost:7001/sample/rest" )
            .path( "patients/{nuhsa}" )
            .resolveTemplate( "nuhsa" , nuhsa)
            .request(MediaType.APPLICATION_JSON)
            .get();

    if (response.getStatus() != Response.Status.OK.getStatusCode())

{
        Assert.fail( "Response error: " +
Response.Status.OK.getStatusCode() +  " - " +
Response.Status.OK.getReasonPhrase());
    }  else {
        Patient patient = response.readEntity(Patient. class );

        Assert.assertNotNull(patient);
        Assert.assertEquals(nuhsa, patient.getNuhsa());
    }
}
```

Esta opción tiene como positivo el hecho de que la petición genera una respuesta siempre que exista respuesta del servidor (independientemente del código de estado devuelto), pudiendo consultar posteriormente el estado de la respuesta que hemos obtenido y, en caso de respuesta correcta, recuperar la entidad solicitada. Por contra, requiere que cada petición sea seguida de una pequeña lógica de negocio encargada de interpretar la respuesta.