Clientes JAXRS - FHIR RESTFULL

Dirección General de Sistemas de Información y Comunicaciones

Áreas de Gobierno Tecnológico de SSII y del Dato

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Versión

Guía de implementación Vigente

Versión**: N/A**

Estado: ACTIVO

Entrada en vigor desde: N/A

Obligado cumplimiento desde: N/A

Guía de implementación PRE-RELEASE

Versión: 1.0.0

Estado: PRE-RELEASE

Entrada en vigor desde:N/A

Obligado cumplimiento desde: N/A

Tabla de Contenido

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

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	Pre-release	Adopción	Activa	Retiro	Alcance
v01r00	8 oct 2023				• Versión inicial • Recomendaciones para la implementación de FHIR con JAX-RS

1. Introducción

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

1.2. Ámbito de aplicación

Esta arquitectura de referencia será de aplicación obligatoria(*) en los siguientes casos:

- Desarrollo de nuevos sistemas de información corporativos.
- Refactorización de sistemas de información existentes.
- Evolución de sistemas de información que requieran la implementación de nuevos procesos o funcionalidades y que requiera de desarrollo en funciones "core" del sistema.

Se recomienda su aplicación:

- Sistemas comerciales que permitan una arquitectura tecnológica flexible y alineada con este modelo de referencia.
- Desarrollos locales con previsión de escalar a sistemas corporativos.
- Cualquier desarrollo para implementar nuevos procesos sobre los sistemas de información existentes.

Se recomienda analizar en detalle con la unidad de Arquitectura de la STIC y particularizarla para los siguientes casos:

- Sistemas analíticos o que gestionen grandes volúmenes de datos (**).

(*) Cualquier propuesta que difiera de esta arquitectura deberá ser aprobada por el Área de Gobernanza de la STIC, previa solicitud y justificación en su caso.

(**) En proceso de elaboración de una arquitectura de referencia para sistemas analíticos y big data.

1.3. Ciclo de vida de la arquitectura de referencia

Los cambios normativos dentro de la arquitectura de referencia seguirán el siguiente ciclo de vida:

- **Pre-release:** Periodo durante el cual la normativa se hace pública aunque no es necesario adherirse inmediatamente. En este periodo aún puede sufrir pequeñas correcciones.
- **Adopción:** Periodo durante el cual la normativa inicia un periodo de tiempo flexible para su aplicación de forma obligatoria al ámbito en el cual está destinado.
- **Activa:** Periodo durante el cual es obligatorio su cumplimiento para el ámbito en el cual está destinado.
- **Retiro:** Periodo durante el cual dejará de aplicarse en el ámbito al cual está destinado en sustitución de nuevas versiones o normas.

2. Uso de JAX-RS

Acorde con la arquitectura de referencia seguida por la STIC, en la cual se especifica el uso de Weblogic 12c como servidor de aplicaciones, se debe usar la especificación JAX-RS 2.0 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 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.RequestScopedpublic 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": "juang.arriaza.
exts@juntadeandalucia.es",  "phone": "955124356",  "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": "juang.arriaza.exts@juntadeandalucia.
es",    "phone": "955124356",    "address": "Americo
Vespucio",    "cityRegion": "SV",    "ccNumber": "10"  },  "service": "000646",  "admissionDate": 1368698
400000}
```

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 ha 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, mientras que el servicio "EpisodesREST" está securizado siguiendo la normativa de comunicaciones entre aplicativos y requiere que se envíe 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"

2.2 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

```
@Testpublic 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

```
@Testpublic 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 respueta siempre que exista respueta 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.

3. Uso de Jax-rs con Hapi-Fhir

Acorde con la arquitectura de referencia seguida por la STIC, y al estándar FHIR que sigue la OTI en la definición de los contratos Rest, se proporcionan las siguientes librerías que facilitan los desarrollos:

- arquitectura-framework-kernel > 1.6.0
- sas-hapi-fhir-util-r4 > 1.1.0
- sas-hapi-fhir-util-r4b > 1.1.0
- sas-hapi-fhir-util-r5 > 1.1.0
- sas-hapi-fhir-jax-rs > 1.1.0
- sas-hapi-fhir-jax-rs-r4 > 1.1.0
- sas-hapi-fhir-jax-rs-r4b > 1.1.0
- sas-hapi-fhir-jax-rs-r5 > 1.1.0

3.1 Cómo comenzar

Esta sección identificará y mostrará un ejemplo de implementación de todos los elementos que intervienen en el desarrollo de un controlador basado en Jax-rs con Fhir. Para facilitar la incorporación del estándar FHIR dentro de los desarrollos se ha hecho uso de la librería de HAPI-FHIR.

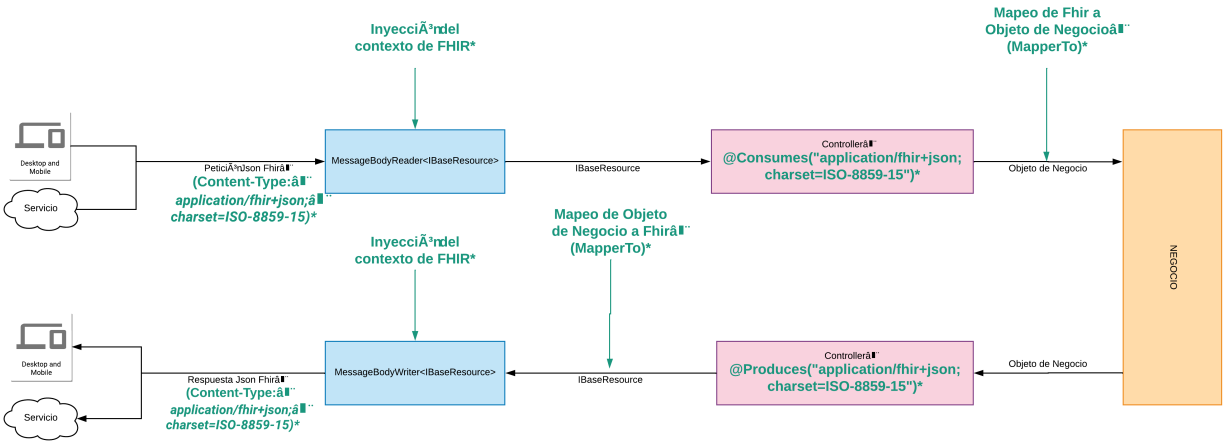
3.1.1 Caso de Éxito

A continuación se puede observar un gráfico correspondiente a una petición REST realizada de forma satisfactoria.

La integración de esta libreria con su solución sólo requiere incidir en:

- Definir los content types correctos que debe escuchar.
- Proveer una instancia del contexto de hapi FHIR de tipo Application Scoped.
- Implementar los mappers necesarios para realizar la conversión de sus objetos de dominio a los objetos fhir y viceversa.

Caso de Ã▮xito



* Todo lo marcado hay que tenerlo en cuenta para configurar la aplicaciÃ³nprevio a la ejecuiciÃ³n

draw.io

Diagram attachment access error: cannot display diagram

Dentro de este gráfico podemos distinguir los siguientes elementos que intervienen en la implementación de un servicio REST que cumple el estándar JAX-RS y utiliza HAPI-FHIR como librería de tratamiento del estándar FHIR:

- Cliente
- MessageBodyReader<IBaseResource>
- Controlador
- Mapeador de FHIR Negocio
- Mapeador de Negocio FHIR
- MesasageBodyWriter<IBaseResource>

Ámbito: Cliente	
Cualquier sistema consumidor de la api (p. ejem UI, Servicio, etc..). El cliente debe tener en cuenta las siguientes consideraciones al crear y recibir peticiones	
Pauta	**Descripción**
P1	El Content-Type de la petición enviada debe ser compatible con ***application/fhir+json; charset=ISO-8859-15***
P2	El Content-Type de la respuesta a la petición debe ser compatible con ***application/fhir+json; charset=ISO-8859-15***

Nota	*El charset ISO-8859-15 es el correspondiente a los carácteres latinos con la inclusión principal del carácter del €*

Ejemplo Petición Curl

```
curl --location --request POST 'http://localhost:9080/ScedueSchedule-fhir' \ --header 'Content-Type: application/fhir+json; charset=ISO-8859-15' \ --data-raw '{ "resourceType": "Schedule", "serviceCategory": [ { "id": 12, "coding": [ { "code": 1, "display": "test" } ] } ], "actor": [ { "id": 1, "reference": "test" } ], "planningHorizon": { "start": "2021-07-20T11:11:11+12:12", "end": "2021-07-20T11:11:11+12:12" } }'
```

Ejemplo Respuesta

```
POST /ScScheduedulee-fhir HTTP/1.1 Content-Type: application/fhir+json; charset=ISO-8859-15
```

Ámbito: MessageBodyReader<IBaseResource>	
Interceptor proporcionado por ***sas-hapi-fhir-jax-rs*** que permite transformar el body de las peticiones a objetos *Hapi-Fhir*. El tipo ***IBaseResource*** es un tipo proporcionado por Hapi-Fhir para englobar cualquier objeto FHIR de cualquier versión.	
Pauta	**Descripción**
P3	Se le debe proveer, a través de CDI, un contexto de Hapi-FHIR con la versión concreta con la que esté definido el contrato del servicio (R4,R4B, R5 ...)
P4	En caso de necesitar funcionalidad complementaria se puede extender del MessageBodyReader
Nota ***Rendimiento***	*La creación del contexto de Hapi-FHIR es costoso por lo que se recomienda crearlo una sóla vez.* *"**This class is expensive** to create, as it scans every resource class it needs to parse or encode to build up an internal model of those classes. For that reason, you should try to create one FhirContext instance which remains for the life of your application and reuse that instance. Note that it will not cause problems to create multiple instances (ie. resources originating from one FhirContext may be passed to parsers originating from another) but you will incur a performance penalty if a new FhirContext is created for every message you parse/encode."*
Nota ***Concurrencia***	*" **This class is thread safe** and may be shared between multiple processing threads, except for the `registerCustomType(java.lang.Class<? extends org.hl7.fhir.instance.model.api.IBase>)` and `registerCustomTypes(java.util.Collection<java.lang.Class<? extends org.hl7.fhir.instance.model.api.IBase>>)` methods."*

Ejemplo Contexto Hapi-FHIR R4

```
@ApplicationScoped public class HapiFhirR4Context {         //Para asignar por cdi una versión concreta de HAPI FHIR a través de su contexto         @Produces         @ApplicationScoped         public FhirContext getHapiFhirContext(){                return FhirContext.forR4();         } }
```

Ámbito: Controlador	
Elemento proporcionado por Jax-rs para recepcionar las peticiones HTTP	
Pauta	**Descripción**
P5	En caso de recibir objetos FHIR se debe configurar el controlador o el end-point para que consuma bodies compatibles con " ***application/fhir+json; charset=ISO-8859-15"***
P6	En caso de responder objetos FHIR se debe configurar el controlador o el end-point para que produzca bodies compatibles con " ***application/fhir+json; charset=ISO-8859-15"***
Nota	*En caso de usar OpenApi, hay un bug para la generación automática con los objetos Hapi-FHIR. En caso de querer disponer de esta documentación se debe proporcionar de forma manual.*

Ejemplo Configuración Jax-rs

```
//Ejemplo de configuración del controlador@Consumes("application/fhir+json; charset=ISO-8859-15")
@Produces("application/fhir+json; charset=ISO-8859-15")public class SchedueScheduleFhirRestController
{        //Ejemplo de configuración del end-point    @GET    @Path("/")        @Consumes("application
/fhir+json; charset=ISO-8859-15")       @Produces("application/fhir+json; charset=ISO-8859-15")
public Response crSchedlecreateSchedule(@RequestBody Schedule schedule) {        ...    }}
```

Ámbito: Mapeador(FHIR Negocio)	
Elemento encargado de transformar la paramétrica en formato FHIR en objetos de negocio para usar en las demás capas	
Pauta	**Descripción**
P7	El mapper debe implementar la Interfaz Mapper<S,D>
P8	Se provee una clase base que implementa el mapeo unidireccional de colecciones, BaseMapperTO<S,D>
Nota	*Revisar el apartado técnico de mapeo para más detalle*

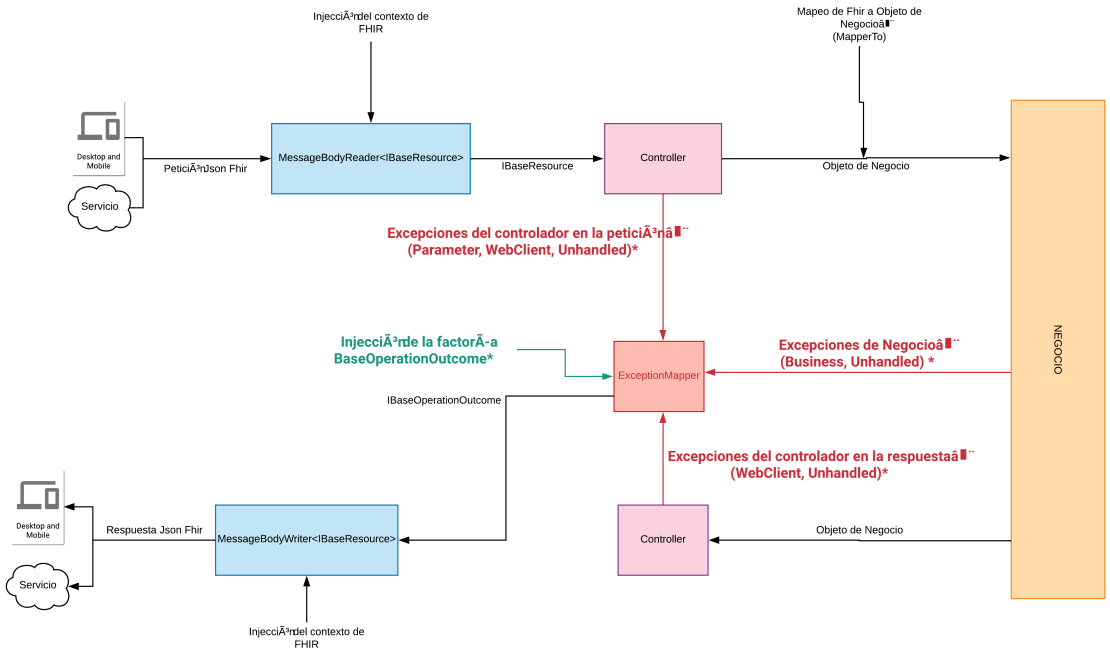
Ámbito: Mapeador(Negocio FHIR)	
Elemento encargado de transformar la paramétrica en formato FHIR en objetos de negocio para usar en las demás capas	
Pauta	**Descripción**
P9	El mapper debe implementar la Interfaz Mapper<S,D>
P10	Se provee una clase base que implementa el mapeo unidireccional de colecciones, BaseMapperTO<S,D>
Nota	*Revisar el apartado técnico de mapeo para más detalle*
Nota	*Revisar el apartado técnico de mapeo con paginación fhir para más detalle en la creación de mapeadores para resultados paginados*

Ámbito: MessageBodyWriter<IBaseResource>	
Interceptor proporcionado por ***sas-hapi-fhir-jax-rs*** que permite transformar la respuesta del end-point de las peticiones a objetos *Hapi-Fhir*. El tipo ***IBaseResource*** es un tipo proporcionado por Hapi-Fhir para englobar cualquier objeto FHIR de cualquier versión.	
Pauta	**Descripción**
P11	Se le debe proveer, a través de CDI, un contexto de Hapi-FHIR con la versión concreta con la que esté definido el contrato del servicio (R4,R4B, R5 ...)
P12	En caso de necesitar funcionalidad complementaria se puede extender del MessageBodyWriter
Nota ***Rendimiento***	*La creación del contexto de Hapi-FHIR es costoso, por lo que se recomienda crearlo una sóla vez.* *"**This class is expensive** to create, as it scans every resource class it needs to parse or encode to build up an internal model of those classes. For that reason, you should try to create one FhirContext instance which remains for the life of your application and reuse that instance. Note that it will not cause problems to create multiple instances (ie. resources originating from one FhirContext may be passed to parsers originating from another) but you will incur a performance penalty if a new FhirContext is created for every message you parse/encode."*
Nota ***Concurrencia***	*" **This class is thread safe** and may be shared between multiple processing threads, except for the `registerCustomType(java.lang.Class<? extends org.hl7.fhir.instance.model.api.IBase>)` and `registerCustomTypes(java.util.Collection<java.lang.Class<? extends org.hl7.fhir.instance.model.api.IBase>>)` methods."*

3.1.2 Caso de Error

A continuación se puede observar un gráfico correspondiente a una petición REST que contiene errores durante su ejecución.

Caso de Error



* Todas las excepciones son interceptadas por su mapeador correspondiente

* Todo lo marcado hay que tenerlo en cuenta para configurar el uso de excepciones en la aplicaciÃ³n

draw.io

Diagram attachment access error: cannot display diagram

Dentro de este gráfico podemos distinguir los siguientes elementos que intervienen en la gestión de excepciones dentro de la implementación de un servicio REST que cumple el estándar JAX-RS y utiliza HAPI-FHIR como librería de tratamiento del estándar FHIR:

- ExceptionMapper
- UnhandledException
- ParameterException
- WebClientException
- BusinessException

Ámbito: ExceptionMapper<T>	
Interfaz a implementar por parte de JAX-RS para la interceptación de las excepciones de un tipo concreto	
Pauta	**Descripción**

P13	Todas las excepciones deben tener un ExceptionMapper definido que lo capture, ya sea por su tipo o por un tipo derivado
P14	Se le debe proveer, a través de CDI, una factoria de Operation Outcome Hapi-FHIR con la versión concreta con la que esté definido el contrato del servicio (R4,R4B, R5 ...)
Nota	*Revisar el apartado técnico de manejo de excepciones y manejo de excepciones FHIR para más detalle*

Ámbito: UnhandledException	
Son todas aquellas excepciones no controladas dentro del desarrollo (HTTP STATUS 500)	
Pauta	**Descripción**
P15	No deberían aparecer ninguna.
P16	En caso de detectar alguna se debe controlar y tipificar como algún tipo de excepción de negocio
Nota	*Revisar el apartado técnico de manejo de excepciones y manejo de excepciones FHIR para más detalle*

Ámbito: ParameterException	
Son todas aquellas excepciones provenientes de la validación de los parámetros de entrada del controlador	
Pauta	**Descripción**
P17	Todas las excepciones originadas por una validación incorrecta de un parámetro de entrada del controlador deben heredar de ParameterException
P18	Acorde a la normativa de la OTI, el httpstatus de estas excepciones debe ser 422
Nota	*Revisar el apartado técnico de manejo de excepciones y manejo de excepciones FHIR para más detalle*

Ámbito: WebClientException	
Son todas aquellas excepciones provenientes de la interactuación con todo lo relativo al cliente web (HTTP STATUS 4XX)	
Pauta	**Descripción**
P19	Se le debe proveer, a través de CDI, un contexto de Hapi-FHIR con la versión concreta con la que esté definido el contrato del servicio (R4,R4B, R5 ...)
P20	En caso de necesitar funcionalidad complementaria se puede extender del MessageBodyWriter
Nota	*Revisar el apartado técnico de manejo de excepciones y manejo de excepciones FHIR para más detalle*

Ámbito: BusinessException	
Son todas aquellas excepciones provenientes de una validación funcional del proceso	
Pauta	**Descripción**
P21	Todas las excepciones originadas por una validación incorrecta de un parámetro de entrada del controlador deben heredar de BusinessException

Nota	*Revisar el apartado técnico de manejo de excepciones y manejo de excepciones FHIR para más detalle*

3.2 Detalles técnicos

3.2.1 Requisitos Tecnológicos

JDK	11. Nota: Si se va a hacer uso en un proyecto que se despliegue en openliberty se recomienda AdoptJdk OpenJ9
arquitectura-framework-kernel	> 1.6.0
sas-hapi-fhir-util-r4	> 1.1.0
sas-hapi-fhir-util-r4b	> 1.1.0
sas-hapi-fhir-util-r45	> 1.1.0
sas-hapi-fhir-jax-rs	> 1.1.0
sas-hapi-fhir-jax-rs-r4	> 1.1.0
sas-hapi-fhir-jax-rs-r4b	> 1.1.0
sas-hapi-fhir-jax-rs-r	> 1.1.0
jax-rs	2.1
cdi-api	2.0

3.2.2 Inclusión de librerías

Para incluir la librería en un proyecto Maven es necesario realizar los siguientes pasos:

1. Agregar el repositorio de la Junta de Andalucía. Para ello, seguir las indicaciones de la página Repositorio de artefactos.
2. Especificar las dependencias en el POM del proyecto que requiere la funcionalidad

Repositorio versión final Versión R5

```
<properties>        <arquitectura.framework.version>1.6.0</arquitectura.framework.version>        <maven.
compiler.target>11</maven.compiler.target>        <maven.compiler.source>11</maven.compiler.
source>        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>      ....
<version.sas.hapi-fhir-base>1.1.0</version.sas.hapi-fhir-jax-rs>      ....    </properties>
<dependencyManagement>        <dependencies>           .....            <dependency>
<groupId>es.ja.csalud.sas.componentescomunes.fhir</groupId>               <artifactId>sas-hapi-fhir-jax-
rs</artifactId>               <version>${version.sas.hapi-fhir-jax-rs}</version>            <
/dependency>            <dependency>               <groupId>es.ja.csalud.sas.componentescomunes.fhir<
/groupId>               <artifactId>sas-hapi-fhir-util-r5</artifactId>
<version>${version.sas.hapi-fhir-base}</version>            </dependency>
<dependency>               <groupId>es.ja.csalud.sas.arquitectura-framework</groupId>
<artifactId>arquitectura-framework</artifactId>               <version>${arquitectura.framework.version}
</version>               <type>pom</type>               <scope>import</scope>            <
/dependency>          .....        </dependencies>
```

3.2.3 Arquitectura Framework Kernel

A continuación se describe los aportes de cada librería :

- El kernel base proporciona mecanismos simplificados para la gestión de cuestiones transversales, que a continuación se detallan:
- Manejo de Excepciones

La extensión del modelo excepciones base que se proporciona permitirá a las utilerías de jax-rs fhir implementar de forma generalizada la captura y conversión al formato FHIR.

  - ***BusinessException*** : Extienda este modelo de excepción, que se lanza por algún incumplimiento de regla de negocio.
  - ***ParameterException*** : Extienda este modelo de excepción, que se lanza cuando algún parámetro recibido no cumple alguno de los requisitos establecidos.

Ejemplo de Business Exception

```
//Ejemplo de personalización BussinesExceptionpublic PersonalizadoBussinesException extends
BussinesException{    public     PersonalizadoBussinesException (String message) {super(message);  }
public      PersonalizadoBussinesException (String message, Throwable cause) { super(message, cause);
}    public      PersonalizadoBussinesException (Throwable cause) { super(cause);   }}
```

Ejemplo de Parameter Exception

```
//Ejemplo de personalización ParameterExceptionpublic PersonalizadoParameterException extends
ParameterException{    public   PersonalizadoParameterException (String message) {super(message);  }
public   PersonalizadoParameterException (String message, Throwable cause) { super(message, cause);
}    public   PersonalizadoParameterException  (Throwable cause) { super(cause);   }}
```

- Mapeadores

Se proporcionan unas interfaces y clases abstractas que reduzcan el boilerplate y agilicen la implementación de mapeadores:

- ***MapperTo*** : Interfaz que permite definir los mapeos de objetos negocio a objetos FHIR
- ***BaseMapperTo*** : Clase Abstracta que aporta una funcionalidad por defecto para el mapeo de colecciones. Se aplica una estrategia de merge de las dos colecciones siendo la referencia la colección de source. Los casos de uso son:
    - *Existe en source y no en destination* se añade aplicándole la función mapTo a la colección de destination.
    - *No existe en source y si en destination* se borra de la colección de destination.
    - *Existe en source y en destination* se añade sobreescribe el elemento aplicándole la función mapTo a la colección de destination.

Ejemplo de Mapeador

```
public class EjemploMapper extends BaseMapperTO<Ejemplo, EjemploFhir> implements MapperTo<Ejemplo,
EjemploFhir> {        @Override        public EjemploFhir mapTo(Ejemplo source) {                 return
mapTo(source, new EjemploFhir());         }                 @Override        public EjemploFhir mapTo
(Ejemplo source, EjemploFhir destination) {                 if(Objects.isNull
(source))                         return null;                if(Objects.isNull
(destination))                         return mapTo(source);                ...                 //lógica
del mapeo                ...                 return destination;         } }
```

3.2.4 Hapi Fhir Util {Version=(R4,R4b,R5)}

En esta sección se describe la información técnica para poder utilizar la librería de utilerías de HAPI-FHIR versión R4:

Factorías

Se proporcionan unas serie de interfaces e implementaciones para R4 de algunos objetos FHIR que se consideran globales a cualquier proyecto:

- **BaseMetaFactory**: Interfaz de construcción del objeto FHIR META
- **BaseOperationOutcomeFactory**: Interfaz de construcción del objeto FHIR OperationOutcome empleado en el la transmisión de excepciones y errores de la aplicación.
- **Meta{Version}Factory**: Implementación por defecto de la factoría
- **OperationOutcome{Version}Factory**: Implementación por defecto de la factoría

Ejemplo de OperationOutcomeFactory CDI

```
@ApplicationScoped public class HapiFhir{Version}OperationOutcomeFactory {    //Para asignar por cdi la
factoría de la versión HAPI FHIR para el Operation Outcome (modelo de errores)         @Produces
@ApplicationScoped         public BaseOperationOutcomeFactory getHapiFhirOperationOutcomeFactoryContext()
{                 return new OperationOutcome{Version}Factory();         }}
```

Ejemplo de OperationOutcomeFactory CDI Versión R5

```
@ApplicationScoped public class HapiFhirR5OperationOutcomeFactory {    //Para asignar por cdi la
factoría de la versión HAPI FHIR para el Operation Outcome (modelo de errores)        @Produces
@ApplicationScoped         public BaseOperationOutcomeFactory getHapiFhirOperationOutcomeFactoryContext()
{                 return new OperationOutcomeR5Factory();         }}
```

Utilería

Herramientas genéricas para el tratamiento con objetos FHIR

- **HapiFhirUtil**: clase de utilería para la lectura/escritura de objetos FHIR con HAPI que sea independiente de la versión de FHIR

3.2.5 Hapi Fhir JAX-RS

En esta sección se describe la documentación técnica que permita el uso de la librería HAPI-FHIR con utilerías para su integración con JAX-RS

ExceptionMapper

Se proporcionan una serie de mapeadores genéricos que generan un OperationOutcome que, de forma general, cumplen con el contrato de la OTI. Esos mapeadores vienen configuradas para inyectarse directamente a la implementación del servicio REST con Jax-rs y CDI

- ***BusinessHapiFhirExceptionMapper*** : Mappeador Jax-rs para las excepciones de tipo BusinessException.
- ***ParameterHapiFhirExceptionMapper*** : Mappeador Jax-rs para las excepciones de tipo ParameterException.
- ***ServerErrorHapiFhirExceptionMapper*** : Mappeador Jax-rs para las excepciones no controladas de tipo RuntimeException.
- ***WebApplicationHapiFhirExceptionMapper*** : Mappeador Jax-rs para las excepciones de tipo WebApplicationException.
- ***HapiFhirJaxRsExceptionUtil:*** Utilería para la conversión de las excepciones a Response de Jax-rs

MessageBody

Interceptores Writer y Reader para el parseo de objetos FHIR usando la librería HAPI-FHIR. Esos mapeadores vienen configurados para inyectarse directamente a la implementación del servicio REST con Jax-rs y CDI.

- ***HapiFhirMessageBodyReader*** : Interceptor de entrada para objetos de tipo IBaseResource (HAPI-FHIR). Estos interceptores son independientes de la versión de FHIR. Esta versión se define con la inyección del contexto por CDI (Ejemplo documentado en el apartado anterior)
- ***HapiFhirMessageBodyWriter*** : Interceptor de salida para objetos de tipo IBaseResource (HAPI-FHIR). Estos interceptores son independientes de la versión de FHIR. Esta versión se define con la inyección del contexto por CDI (Ejemplo documentado en el apartado anterior)

3.2.6 Hapi Fhir JAX-RS {Version=(R4,R4b,R5)}

En esta sección se describe la documentación técnica que permita el uso de la librería HAPI-FHIR con utilerías para su integración con JAX-RS para la versión R4

Mapper

Se proporcionan mapeadores genéricos para objetos de la versión R4 de FHIR

- ***BasePaginationFhir{Version}Mapper:*** clase abstracta que implementa el mapeo de la paginación a expensas de que el desarrollador implemente el método de construcción de cada elemento concreto (*createEntry)*

Ejemplo de mapeador para la paginación con FHIR

```
public class EjemploPaginationMapper extends BasePaginationFhir{Version}Mapper<Ejemplo> {        private
final EjemploMapper ejemploMapper;         public SchedulePaginationMapper() {                  this.
ejemploMapper = new EjemploMapper();         }         @Override protected Bundle.BundleEntryComponent
createEntry(final Ejemplo ejemplo) {                  BundleEntryComponent bundelEntry = new
BundleEntryComponent();                  EjemploFHIR ejemploFhir = ejemploMapper.mapTo
(ejemplo);                 bundelEntry.setResource(schedule);                return
bundelEntry;         }}
```

Ejemplo de mapeador para la paginación con FHIR Versión R5

```
public class EjemploPaginationMapper extends BasePaginationFhirR5Mapper<Ejemplo> {        private final
EjemploMapper ejemploMapper;         public SchedulePaginationMapper() {                  this.
ejemploMapper = new EjemploMapper();         }         @Override protected Bundle.BundleEntryComponent
createEntry(final Ejemplo ejemplo) {                  BundleEntryComponent bundelEntry = new
BundleEntryComponent();                  EjemploFHIR ejemploFhir = ejemploMapper.mapTo
(ejemplo);                 bundelEntry.setResource(schedule);                return
bundelEntry;         }}
```

Normativa

Normativa	Descripción	Versión Activa	Fecha de activa	Versión Pre-Release	Fecha de Pre-release
Arquitectura tecnológica de referencia	Diagrama conceptual e introducción a la arquitectura de referencia independientemente de la tecnología usada.	N/A	N/A	1.0	pte de especificar
Arquitecturas de referencia para el desarrollo	Diagrama conceptual e introducción a la arquitectura de referencia independientemente de la tecnología usada.	1..1	1 jun 2017	N/A	N/A
Arquitectura de referencia Java EE	Arquitectura de referencia a seguir para proyectos Java.	3.1	30 sept 2019	N/A	N/A
Arquitectura de referencia .Net	Arquitectura de referencia a seguir para proyectos .Net.	1.0	1 jun 2017	N/A	N/A
Normativa de clientes web	Normativa sobre compatibilidad de aplicaciones web con navegadores de escritorio	1.1	21 nov 2016	N/A	N/A
Securización en comunicaciones	Normativa sobre securización y comunicación entre aplicativos	1.7	13 dic 2016	N/A	N/A
Integración continua y calidad en código	Normativa sobre procesos de entrega de código, integración continua y calidad en código	2.1	10 mar 2021	N/A	N/A
Normativa Git	Normativa sobre Git	1.57	16 jun 2021	N/A	N/A
Normativa de auditoría	Normativa sobre auditoría de sistemas de información	1.0	12 jun 2017	N/A	N/A
Normativa Pruebas de Carga	Normativa sobre la ejecución de las pruebas de carga de sistemas	2.10	10 mar 2021	N/A	N/A
Generación de WebServices SOAP	Normas y recomendaciones sobre como crear servicios y clientes SOAP.	1.1	18 nov 2016	N/A	N/A
Generación de WebServices REST	Normas y recomendaciones sobre como crear servicios y clientes REST.	1.3	26 abr 2021	N/A	N/A
Clientes JAXRS - FHIR RESTFULL [WIP]	Normas y recomendaciones sobre como crear servicios y clientes REST.	N/A	N/A	1.0	pte de especificar

Guías de diseño

Guía Implementación	Descripción	Versión Activa	Fecha de activa	Versión Pre-Release	Fecha de Pre-release

Guías de desarrollo

Normativa	Descripción	Versión Activa	Fecha de activa	Versión Pre-Release	Fecha de Pre-release
Android		1.0	11 oct 2019		
IOS		1.0	11 oct 2019		