TABLA DE CONTENIDO

[TABLA DE CONTENIDO 1](#_Toc494662445)

[Crear Servicio REST– DUMMY 2](#_Toc494662446)

[Objetivo: 2](#_Toc494662447)

[Crear Servicio REST. 2](#_Toc494662448)

[Notas: 3](#_Toc494662449)

[Validar disponibilidad del servicio REST 4](#_Toc494662450)

[Probar el servicio REST 4](#_Toc494662451)

[Revisar Módulo Generado 4](#_Toc494662452)

[Leyenda 5](#_Toc494662453)

[Servicio REST 5](#_Toc494662454)

[La clase con el servicio web 6](#_Toc494662455)

[@Component 7](#_Toc494662456)

[@Reference 7](#_Toc494662457)

[Configurar CXF Endpoints 8](#_Toc494662458)

[Conectarse como administrador (test) y agregar */api* al contexto de la variable de sistema CXF Endpoints 8](#_Toc494662459)

[Configurar REST Extender 8](#_Toc494662460)

Crear Servicio REST– DUMMY

### Objetivo:

Crear un servicio REST DUMMY, que devuelva una lista de países en formato Json.

### Crear Servicio REST.

1. Crear paquete: com.pais.controller
2. Crear clase PaisController.java.

**package** com.pais.controller;

**import** javax.ws.rs.ApplicationPath;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.Application;

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

**import** org.osgi.service.component.annotations.Component;

**import** jodd.json.JsonSerializer;

@Component(immediate = **true**, service = Application.**class**)

@ApplicationPath("/pais")

**public** **class** PaisController **extends** Application {

**private** String paises = "{\"paises\":["

+ "{\"nombre\":\"Argentina\",\"capital\":\"Buenos Aires\",\"moneda\":\"Peso\",\"idioma\":\"Espa\u00f1ol\"},"

+ "{\"nombre\":\"Brasil\",\"capital\":\"Brasilia\",\"moneda\":\"Cruzeiro Real\",\"idioma\":\"Portugues\"},"

+ "{\"nombre\":\"Colombia\",\"capital\":\"Bogota\",\"moneda\":\"Peso\",\"idioma\":\"Espa\u00f1ol\"},"

+ "{\"nombre\":\"Panam\u00e1\",\"capital\":\"Panam\u00e1\",\"moneda\":\"Balb\u00f3a\",\"idioma\":\"Espa\u00f1ol\"},"

+ "{\"nombre\":\"Canad\u00e1\",\"capital\":\"Otawa\",\"moneda\":\"D\u00f3lar\",\"idioma\":\"Ingles\"},"

+ "{\"nombre\":\"Italia\",\"capital\":\"Roma\",\"moneda\":\"Lira\",\"idioma\":\"Italiano\"},"

+ "{\"nombre\":\"Jap\u00f3n\",\"capital\":\"Tokio\",\"moneda\":\"Yen\",\"idioma\":\"Japon\u00e9s\"},"

+ "{\"nombre\":\"Venezuela\",\"capital\":\"Caracas\",\"moneda\":\"Bol\u00edvar\",\"idioma\":\"Espa\u00f1ol\"}"

+ "]}";

@GET

@Path("getPaises/")

@Produces(MediaType.***APPLICATION\_JSON***)

**public** String getPaises() {

return new JsonSerializer().serialize(paises);

}

// ------------------------

// Rest Application Config

// ------------------------

@Override

public Set<Object> getSingletons() {

return Collections.singleton(this);

}

}

getSingletons : devuelven los recursos y proveedores JAX-RS pertinentes

### Notas:

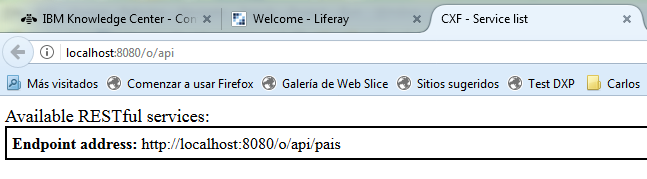
* Liferay soporta JAX-WS y JAX-RS a través de la implementación de Apache CXF.
* Extensores SOAP: Requiere publicar servicios web JAX-WS. Cada extensor SOAP puede implementar los servicios a uno o más puntos finales CXF y puede utilizar un conjunto de controladores JAX-WS para aumentar los servicios.
* Extensores REST: Requiere publicar servicios web JAX-RS. Los extensores REST para servicios JAX-RS son análogos a los extensores SOAP para servicios JAX-WS. Para crear servicios JAX-RS que puedan funcionar en diferentes implementaciones JAX-RS, debe proporcionar una implementación de javax.ws.rs.core.Application al marco OSGi. Puede hacerlo registrando una instancia de esta implementación como un servicio OSGi a través de BundleContext o la anotación Declarative Services @Component. La aplicación JAX-RS abarca los servicios que representan los puntos finales de JAX-RS y los servicios que representan a los proveedores de JAX-RS.

Los extensores SOAP y los extensores REST son subsistemas que rastrean los servicios registrados por el desarrollador de aplicaciones en OSGi (aquellos que coinciden con los filtros OSGi proporcionados) y los despliegan bajo los extremos CXF especificados

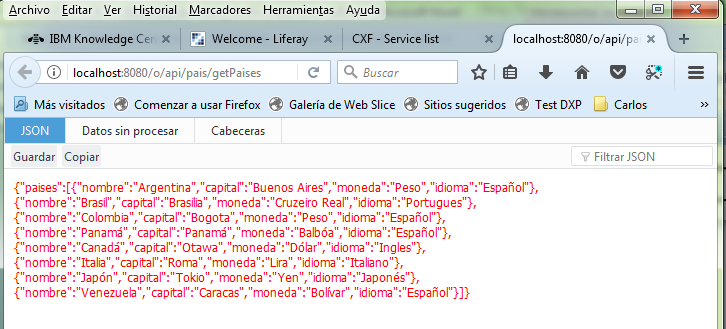
* Los valores en la anotación @Component indican a Liferay DXP que este módulo proporcionará servicios JAX-RS y que la aplicación será configurable usando la API de configuración de OSGi. La anotación @ApplicationPath indica cuál será el segmento de URL de nivel superior para todos los servicios expuestos por esta aplicación. El método getSingletons sobreescrito entonces enumera al tiempo de ejecución de JAX-RS qué proveedores y recursos esta aplicación proporcionará.

JAX-RS tiene el concepto de "proveedores", que son clases que se conectan al framework JAX-RS y proporcionan características útiles como filtros / interceptores, clasificación de contenido, etc. En este caso estamos usando librerías JSON para convertir automáticamente objetos Java a / desde JSON.

### Validar disponibilidad del servicio REST

Con el servidor levantado. En el navegador invocar: <http://localhost:8080/o/api>

### Probar el servicio REST

Con el servidor levantado. En el navegador invocar: <http://localhost:8080/o/api/pais/getpaises>

### Revisar Módulo Generado

Gogo Shell - Comando b + ID\_modulo (**b 613**). Despliega los datos del servicio REST.

### 

Leyenda

### Servicio REST

**RE**presentational **S**tate **T**ransfer (REST) es un tipo de servicio con un estilo de arquitectura para sistemas distribuidos hipermedia como por ejemplo WWW. En definitiva es un protocolo que define una serie de reglas de comunicación entre componentes.

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

* Utiliza los métodos HTTP de manera explícita
* No mantiene estado
* Expone URIs con forma de directorios
* Transfiere XML, JavaScript Object Notation (JSON), o ambos

Una de las características claves de los servicios web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP.

* se usa POST para crear un recurso en el servidor
* se usa GET para obtener un recurso
* se usa PUT para cambiar el estado de un recurso o actualizarlo
* se usa DELETE para eliminar un recurso

### La clase con el servicio web

La clase que va a ser nuestro servicio Web no es más que una clase normal de java, con métodos. Únicamente hay que poner las anotaciones específicas de *JAX-RS* para que el contenedor de aplicaciones lo convierta en servicio web cuando arranque. Las anotaciones son las siguientes:

* **@Path** para indicar el path en la url donde estará nuestro servicio web.
* **@Produces** para indicar el formato en el que el servicio web nos devuelve el resultado. Son habituales "application/xml" y "application/json".
* **@Consumes** para indicar el formato en el que nuestro servicio web admite los datos. Igual que @Produces, son habituales "application/xml" y "application/json".
* El protocolo http define [varios métodos](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) para las peticiones http. En los web services son habituales las cuatro siguientes:
  + **@GET** Cuando queremos pedir datos al servicio web. Por ejemplo, una petición @GET a http://servidor/webservice/empleados podría devolver la lista de empleados y una llamada a http://servidor/webservice/empleados/23 podría devolver los datos del empleado cuyo identificador es 23.
  + **@POST** Cuando enviamos datos al servicio web para que el servicio web haga con ellos lo que considere oportuno. Por ejemplo, una llamada @POST a http://servidor/webservice/crea\_empleado pasando los datos de un empleado, puede crear el empleado en base de datos.
  + **@PUT** Cuando queremos guardar datos en una url específica. Si esos datos no existen en esa URL, se crean. Si ya existen, deben modificarse para que sean lo que nosotros hemos pasado. Por ejemplo, una llamada @PUT a http://servidor/webservice/empleados/23 pasándole datos de un empleado, modificaría los datos del empleado cuyo identificador es el 23 para que sean los que nosotros pasamos. Si no hay ningún empleado con identificador 23, debería crearse un nuevo empleado y asignarle ese identificador concreto.
  + **@DELETE** Para borrar los datos de una url específica. Por ejemplo, una llamada @DELETE a http://servidor/webservice/empleados/23 debería eliminar el empleado de identificador 23.

### @Component

Anotación de "Servicios Declarativos" que define una implementación de servicios. En OSGi permite declarar un servicio de forma dinámica y permite que otros componentes puedan conectarse al componente.

Hay tres atributos principales en esta anotación:

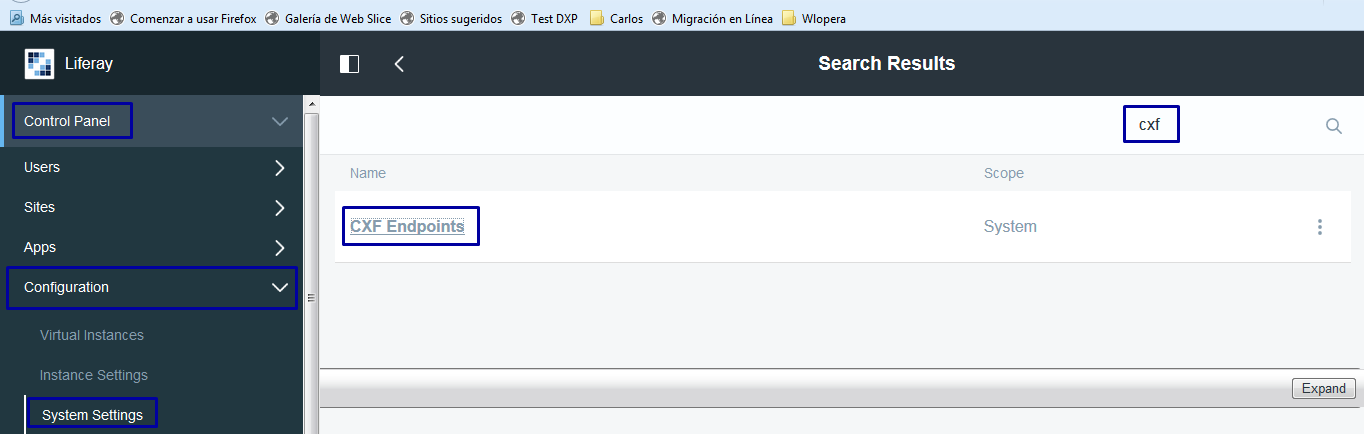
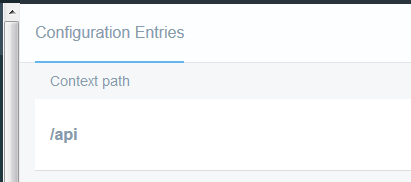
* **immediate**: a menudo establecido en true, esto asegurará que el componente se inicie de inmediato y no esperar a un cableado de referencia o de inicio lento.
* **property**: Se utiliza para pasar en un conjunto de propiedades OSGi para enlazar con el componente. El componente puede ver las propiedades, pero lo más importante es que otros componentes también podrán ver las propiedades. Estas propiedades ayudan a configurar el componente pero también se utilizan para admitir el filtrado de componentes.
* **service**: Define el servicio que el componente implementa. A veces esto es opcional, pero a menudo es obligatorio para evitar la ambigüedad en el servicio que el componente quiere anunciar. El servicio listado es a menudo una interface, pero también puede usar una clase concreta para el servicio.

### @Reference

Esta es la contrapartida de la anotación @Component. @Reference se utiliza en OSGi para inyectar una referencia de componente en su componente. Esto es clave aquí, ya que OSGi está haciendo la inyección, sólo va a trabajar en una clase OSGi @Component. Las anotaciones @Reference se van a ignorar en los no componentes, y de hecho también se ignoran en las subclases también. Cualquier referencia inyectada que necesite, debe hacerse en la propia clase @Component.

### Configurar CXF Endpoints

### Conectarse como administrador (test) y agregar */api* al contexto de la variable de sistema CXF Endpoints



### Configurar REST Extender

Conectarse como administrador (test) y agregar ***/api*** al contexto de la variable de sistema CXF Endpoints

