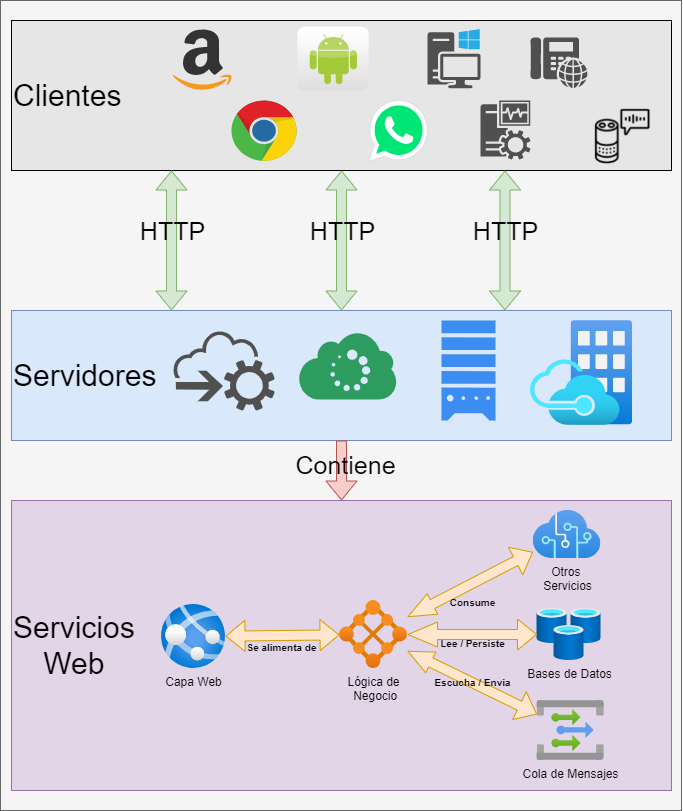
Web Services

# ¿Qué es un servicio web?

Un servicio web se puede definir como una colección estándar o protocolos de intercambio de información contenida en una aplicación o componente de aplicación para comunicaciones cliente-servidor.

Esta aplicación o componente puede ser invocada de forma remota utilizando el protocolo HTTP mediante peticiones estándar, por lo que permite exponer a la red o internet funcionalidades de un software. Una vez expuesto, otras aplicaciones, independientemente la tecnología en la que están basadas, aprovechan estas funcionalidades para interconectarse.



# Tipos de servicios web

Los dos tipos de servicios web mas utilizados hoy en día son:

1. SOAP: SOAP (Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. En java, se puede implementar mediante la librería JAX-WS. (<https://mkyong.com/tutorials/jax-ws-tutorials/>)
2. REST: REST (**RE**presentational **S**tate **T**ransfer) es un estilo de arquitectura de servicio web que se basa en HTTP para comunicación y paquetes de mensajes bajo texto plano, HTML, XML o JSON y es utilizado comúnmente para interconectar. En java, se puede implementar mediante la librería JAX-RS.   
   (<https://mkyong.com/tutorials/jax-rs-tutorials/>)

Ver comparación:

<https://www.upwork.com/resources/soap-vs-rest-a-look-at-two-different-api-styles?utm_source=google&utm_campaign=SEM_GGL_INTL_NonBrand_Marketplace_DSA&utm_medium=cpc&utm_content=113089129402&utm_term=&campaignid=11384804789&matchtype=b&device=c&gclid=Cj0KCQjw78yFBhCZARIsAOxgSx3_JCKFAebL6FrhrSEgSYkMi8x3nD-rBBl7pMnnsv-w1jWSYiwJzR8aAot8EALw_wcB>

# Características de los servicios web

* Acoplamiento débil

Un cliente de un servicio web no está fijado al servicio web directamente. La interfaz del servicio web puede soportar la innovación a lo largo del tiempo sin negociar la capacidad del cliente para comunicarse con el servicio. Un sistema estrechamente acoplado significa que la lógica del cliente y del servidor están estrechamente ligadas entre sí, lo que indica que si una interfaz cambia, otra debe ser actualizada. Aceptar una arquitectura poco acoplada tiende a hacer que los sistemas de software sean más manejables y permite una integración más sencilla entre varios sistemas.

* De grano grueso

Las tecnologías orientadas a los objetos, como Java, exponen sus funciones a través de métodos individuales. Un proceso específico es una operación demasiado fina para proporcionar una capacidad adecuada a nivel corporativo. Construir un programa Java desde cero requiere la creación de varias funciones de grano fino que luego se recogen en una función de grano grueso que es consumida por un cliente o por otro servicio.

Las empresas y las interfaces que prueban deben ser de grano grueso. La tecnología de servicios web implementa un método natural para definir servicios de grano grueso que se acercan a la cantidad adecuada de lógica empresarial.

* Capacidad de ser sincrónico o asincrónico

La sincronización especifica la vinculación del cliente a la ejecución de la función. En las invocaciones sincrónicas, el cliente se bloquea y tarda en completar su servicio antes de continuar. Las operaciones asíncronas permiten al cliente invocar una tarea y luego ejecutar otras funciones.

Los clientes asíncronos obtienen su resultado en un momento posterior, mientras que los clientes síncronos reciben su efecto cuando el servicio ha finalizado. La capacidad asíncrona es un método esencial para habilitar sistemas poco acoplados.

* Admite llamadas a procedimientos remotos (Remote Procedure Calls)

Los servicios web permiten a los consumidores invocar procedimientos, funciones y métodos en objetos remotos mediante un protocolo especifico. Los sistemas remotos exponen un marco de entrada y salida que un servicio web debe soportar.

Una función web soporta RPC proporcionando servicios propios, equivalentes a los de una función tradicional, o traduciendo las invocaciones entrantes en una invocación de un EJB o un componente .NET.

* Admite el intercambio de documentos

Una de las ventajas esenciales de esquemas como XML, JSON o HTML es su forma genérica de representar no sólo datos, sino también documentos complejos. Estos documentos pueden ser tan sencillos como la descripción de una dirección actual, o pueden ser tan complejos como la definición de un libro entero o una solicitud de oferta (RFQ). Los servicios web permiten la transferencia transparente de documentos para facilitar la integración empresarial.

# REST

## ¿Qué es una REST?

Una REST es un estilo arquitectónico para una interfaz de programa de aplicación (API) que utiliza solicitudes HTTP para acceder y utilizar datos. Esos datos pueden utilizarse por los métodos GET, PUT, POST y DELETE, que se refieren a la lectura, la actualización, la creación y la eliminación de operaciones relativas a los recursos.

Por lo general, se prefiere la tecnología REST a otras tecnologías similares. Esto suele ser así porque REST utiliza menos ancho de banda, lo que la hace más adecuada para un uso eficiente de Internet. Las APIs REST también pueden construirse con lenguajes de programación como Java, JavaScript o Python.

El REST utilizado por los navegadores puede considerarse el lenguaje de Internet. Con el aumento del uso de la nube, los consumidores de esta utilizan las API para exponer y organizar el acceso a los servicios web. REST es una opción lógica para construir APIs que permitan a los usuarios conectarse, gestionar e interactuar con los servicios en la nube de forma flexible en un entorno distribuido. Las APIs RESTful son utilizadas por sitios como Amazon, Google, LinkedIn y Twitter.

Como las llamadas no tienen estado, REST es útil en las aplicaciones en la nube. Los componentes sin estado pueden volver a desplegarse libremente si algo falla, y pueden escalar para adaptarse a los cambios de carga. Esto se debe a que cualquier petición puede dirigirse a cualquier instancia de un componente; no puede haber nada guardado que tenga que ser recordado por la siguiente transacción. Esto hace que REST sea preferible para su uso en la web. El modelo RESTful también es útil en los servicios en la nube porque la vinculación a un servicio a través de una API es una cuestión de controlar cómo se decodifica la URL. Es casi seguro que la computación en nube y los microservicios harán que el diseño de APIs RESTful sea la norma en el futuro.

## ¿Como funciona un API REST?

Una API RESTful utiliza comandos para obtener recursos. El estado de un recurso en una fecha determinada se denomina representación del recurso. Una API RESTful utiliza las metodologías HTTP existentes definidas por el protocolo RFC 2616, como:

* GET para recuperar un recurso;
* PUT para cambiar el estado o actualizar un recurso, que puede ser un objeto, archivo o bloque;
* POST para crear ese recurso; y
* DELETE para eliminarlo.

Con REST, los componentes en red son un recurso al que el usuario solicita acceso, como una caja negra cuyos detalles de implementación no están claros. Todas las llamadas son sin estado; el servicio RESTful no puede retener nada entre ejecuciones.

Los formatos de datos que admite la API REST son

* application/json
* application/xml
* application/x-wbe+xml
* application/x-www-form-urlencoded
* multipart/form-data

## REST vs SOAP

REST y SOAP ofrecen diferentes métodos para invocar un servicio web. REST es un estilo arquitectónico, mientras que SOAP define una especificación de protocolo de comunicación estándar para el intercambio de mensajes basados en XML. Las aplicaciones REST pueden utilizar SOAP. RESTful web services son stateless (sin estado).

Una implementación basada en REST es sencilla en comparación con SOAP, pero los usuarios deben entender el contexto y el contenido que se transmite, ya que no existe un conjunto de reglas estándar para describir la interfaz de los servicios web REST. Los servicios REST son útiles para dispositivos de perfil restringido, como los móviles, y son fáciles de integrar en los sitios web existentes.

SOAP requiere menos código de fontanería -es decir, código de bajo nivel y de infraestructura que conecta los principales módulos de código- que el diseño de servicios REST. El lenguaje de descripción de servicios web (WSDL) describe un conjunto común de reglas para definir los mensajes, los enlaces, las operaciones y la ubicación del servicio. Los servicios web SOAP son útiles para el procesamiento asíncrono y la invocación.

# Desarrollo de API REST en Java

JAX-RS es la API de Java utilizada para crear servicios web RESTful. Los servicios web REST están diseñados para ser lo más sencillos posible para mejorar la usabilidad tanto para los desarrolladores de los servicios como para los clientes que los consumen. Para mantener un nivel de simplicidad, los servicios web deben adherirse a varios estándares para ser considerados RESTful.

Java EE 7 es compatible con JAX-RS 2.0, lo que hace que el desarrollo de servicios web RESTful y la adhesión a sus estándares de forma muy sencilla. JAX-RS utiliza varias anotaciones para definir el comportamiento de los servicios web. Estas anotaciones se colocan directamente en la clase de servicio para crear diferentes tipos de endpoints y para definir parámetros.

La siguiente tabla resume las anotaciones disponibles para el resto de la raíz RESTful:

|  |  |
| --- | --- |
| **Anotación** | **Descripción** |
| @ApplicationPath | La anotación @ApplicationPath se aplica a la subclase de la clase javax.ws.rs.core.Application y define el URI base para el servicio web. |
| @Path | La anotación @Path define el URI base para toda la clase raíz o para un método individual. La ruta puede contener una ruta estática explícita, como "fpDual", o puede contener una variable que se pasará en la petición. Este valor es referenciado usando la anotación @PathParam. |
| @Consumes | La anotación @Consumes define el tipo de contenido que la solicitud aceptada. Si se envía un tipo incompatible al servicio, el servidor devuelve el error HTTP 415, Unsupported Media Type. Los parámetros aceptables son application/json, application/xml, text/html o cualquier otro tipo MIME. |
| @Produces | La anotación @Produces define el tipo de contenido de la respuesta que devuelve la clase o el método del servicio. Los parámetros aceptables son application/json, application/xml, text/html o cualquier otro tipo MIME. |
| @GET | La anotación @GET se aplica a un método para crear un punto final para el tipo de solicitud HTTP GET, comúnmente utilizado para recuperar datos. |
| @POST | La anotación @POST se aplica a un método para crear un punto final para el tipo de solicitud HTTP POST, comúnmente utilizado para guardar o crear datos. |
| @DELETE | La anotación @DELETE se aplica a un método para crear un punto final para el tipo de solicitud HTTP DELETE, comúnmente utilizado para eliminar datos. |
| @PUT | La anotación @PUT se aplica a un método para crear un punto final para el tipo de solicitud HTTP PUT, comúnmente utilizado para actualizar los datos existentes. |
| @PathParam | La anotación @PathParam se utiliza para recuperar un parámetro pasado a través del URI, como http://localhost:8080/fpDual/api/mensaje/1. |
| @QueryParam | The @QueryParam annotation is used to retrieve a parameter passed in through the URI as a query parameter, such as http://localhost:8080/hello-web/api/hello?id=1. |

## Creación de servicio web

A continuación, se presentan una serie de pasos para crear una aplicación web con Maven y esta implementación en eclipse.

1. Creación de aplicación web Maven indicando los siguientes puntos:
   1. Arquetipo con id “maven-archetype-webapp” del group id “org.apache.maven.archetypes”.
   2. Group id de la aplicación: edu.fpdual.web
   3. Artifact id de la aplicación: fpDualWeb
2. Crea y asocia al build path las carpetas estándar de un proyecto Maven como: src/main/java, src/main/resources, src/test/java y src/test/resources.
3. Indicamos al proyecto que implemente las siguientes dependencias para implementar lombok y una API REST.

*…*

*<lombokVersion>1.18.20</lombokVersion>*

*…*

*<dependency>*

*<groupId>javax</groupId>*

*<artifactId>javaee-api</artifactId>*

*<version>7.0</version>*

*<scope>provided</scope>*

*</dependency>*

*<dependency>*

*<groupId>org.projectlombok</groupId>*

*<artifactId>lombok</artifactId>*

*<version>${lombokVersion}</version>*

*<scope>provided</scope>*

*</dependency>*

1. Dentro de “src/main/java” creamos los siguientes paquetes:
   1. edu.fpdual.web.fpdualweb.application
   2. edu.fpdual.web.fpdualweb.controller
   3. edu.fpdual.web.fpdualweb.api.dto
2. Dentro del paquete “application” creamos la siguiente clase:

*package edu.fpdual.web.fpdualweb.application;*

*import javax.ws.rs.ApplicationPath;*

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

*@ApplicationPath("/api")*

*public class RestApplication extends Application {*

*}*

Esta clase sirve para iniciar a la aplicación web el punto de partida de nuestro servicio, además de ser quien se encargará de despachar las diferentes llamadas al servicio.

1. Dentro del paquete “dto” creamos la siguiente clase:

*package edu.fpdual.web.fpdualweb.api.dto;*

*import lombok.AllArgsConstructor;*

*import lombok.Getter;*

*import lombok.NoArgsConstructor;*

*import lombok.Setter;*

*@Getter*

*@Setter*

*@AllArgsConstructor*

*@NoArgsConstructor*

*public class Notification {*

*private int id;*

*private String title;*

*private String body;*

*}*

Nota: para que acepte las anotaciones, deben activar el “Annotation Processor” en su proyecto. En caso de no querer usar Lombok, ignorar las anotaciones e indicar los setter y getters así como el constructor con todos sus parámetros de forma manual.

1. Dentro del paquete “controller” crearemos nuestro controlador que contendrá cada una de las posibles llamadas que podemos hacer a nuestro servicio.

*package edu.fpdual.web.fpdualweb.controller;*

*import javax.ws.rs.Consumes;*

*import javax.ws.rs.GET;*

*import javax.ws.rs.POST;*

*import javax.ws.rs.PUT;*

*import javax.ws.rs.Path;*

*import javax.ws.rs.PathParam;*

*import javax.ws.rs.Produces;*

*import javax.ws.rs.QueryParam;*

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

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

*import edu.fpdual.web.fpdualweb.api.dto.Notification;*

*@Path("/notifications")*

*public class NotificationController {*

*@GET*

*@Path("/ping")*

*public Response ping() {*

*return Response.ok().entity("Service online").build();*

*}*

*@GET*

*@Path("/get/{id}")*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Response getNotification(@PathParam("id") int id) {*

*return Response.ok().entity(new Notification(id, "john", "test notification")).build();*

*}*

*@PUT*

*@Path("/get/{id}/{name}")*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Response getNotification(@PathParam("id") int id, @PathParam("name") String name) {*

*return Response.ok().entity(new Notification(id, name, "test notification")).build();*

*}*

*@GET*

*@Path("/get/{id}/name")*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Response getNotificationWithParameters(@PathParam("id") int id, @QueryParam("name") String name) {*

*if (name == null || name.trim().isEmpty()) {*

*return Response.status(400).entity("Name no present in the request").build();*

*} else {*

*return Response.ok().entity(new Notification(id, name, "test notification")).build();*

*}*

*}*

*@GET*

*@Path("/getXML/{id}")*

*@Produces(MediaType.APPLICATION\_XML)*

*public Response getNotificationXML(@PathParam("id") int id) {*

*return Response.ok().entity(new Notification(id, "john", "test notification")).build();*

*}*

*@POST*

*@Path("/post/")*

*@Consumes(MediaType.APPLICATION\_JSON)*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Response postNotification(Notification notification) {*

*return Response.status(201).entity(notification).build();*

*}*

*}*

De esta clase debemos destacar que:

* 1. El inicio de la clase define la anotación “@Path("/notifications")” mediante la cual de define el path o ruta que controla esta clase. Todos los métodos contenidos en este podrán ser accedidos a partir de esta ruta.
  2. El cuerpo de la clase contiene 3 metodos los cuales definen comportamientos que son accedidos de forma diferente:
     1. El primero se accede mediante la ruta “/ping” indicado por la anotación @Path, a través por el método HTTP GET indicado por la anotación @GET.
     2. El segundo se accede mediante la ruta “/get/{id}” indicado por la anotación @Path especificando que recibe un parámetro en la ruta del servicio. El método HTTP GET es indicado por la anotación @GET y como respuesta retorna un mensaje en formato JSON según indica en la anotación “@Produces”.

Dado que la ruta (@Path) indica que se recibe un parámetro en ella, este debe ser recibido como parámetro del método. Para asociar el parámetro del método a un valor recibido en la ruta, se debe utilizar la anotación “@PathParam” a la cual se le indica el nombre del parámetro del cual obtendrá el valor. Ej: @PathParam("id") int id.

* + 1. El tercero se accede mediante la ruta “/post” indicado por la anotación @Path. A diferencia de los anteriores, se accede por método HTTP POST es indicado por la anotación @POST, por lo que recibe sus datos mediante un objeto en formato JSON indicado por la anotación “@Consumes” y como respuesta retorna un mensaje en formato JSON según indica en la anotación “@Produces”.

Dado que es accedido mediante POST, el cuerpo del mensaje viene en una parte del paquete HTTP llamado “payload” la cual al indicarse que viene en formato JSON automáticamente es transformada a un objeto Java conocido, por lo que la firma del método puede contener explícitamente el objeto que esperamos recibir.

1. Para esta guía, se debe desplegar la aplicación en un servidor local y probar el servicio en un GlassFish (Paraya) o Wildfly de Redhat. Para instalar el segundo seguir los siguientes pasos: <https://www.baeldung.com/eclipse-wildfly-configuration>
2. Una vez arrancado el servidor con la aplicación asociada, se puede consumir el servicio. Para ello puede llamarse la url <http://localhost:8080/fpDualWeb/api/notifications/> desde cualquier navegador.

Si se accede directamente al servicio dará error 404, por lo que se debe indicar de forma adicional el método u operación que se desea ejecutar, por ejemplo si se desea llamar al método “ping”, solo indicamos este nombre en la ruta y automáticamente será llamado el método que tenga esta ruta en su anotación “@Path”. Ejemplo: <http://localhost:8080/fpDualWeb/api/notifications/ping>.

## Respuestas en formato XML

Una vez montado el servicio, podemos indicar diversos formatos de respuesta, entre ellos el formato XML. Si se desea retornar un mensaje en este formato de forma automática, debemos seguir los siguientes pasos:

1. Añadir al objeto que se desea convertir a formato XML la anotación @XmlRootElement.
2. Crear un método dentro del controlador “NotificationController” de la siguiente forma:

*@GET*

*@Path("/getXML/{id}")*

*@Produces(MediaType.APPLICATION\_XML)*

*public Response getNotificationXML(@PathParam("id") int id) {*

*return Response.ok()*

*.entity(new Notification(id, "john", "test notification"))*

*.build();*

*}*

Como se puede ver, el método es casi idéntico a los demás con la salvedad de que produce un elemento del tipo “XML” indicado en la anotación “@Produces”.

# Referencias:

* REST API

<https://searchapparchitecture.techtarget.com/definition/RESTful-API#:~:text=A%20RESTful%20API%20is%20an,deleting%20of%20operations%20concerning%20resources>.

* Web Service

<https://www.javatpoint.com/what-is-web-service>

<https://www.tutorialspoint.com/webservices/what_are_web_services.htm>

* Web Service en Java

<https://mkyong.com/tutorials/jax-rs-tutorials/>

<https://mkyong.com/tutorials/jax-ws-tutorials/>

<https://www.baeldung.com/jax-rs-spec-and-implementations>