Unidad 3

Contenidos

Introducción a JAVA Web

Construir una primera aplicación web con acceso a base de datos, requiere de un gran esfuerzo, ya que debe poner en práctica varios conocimientos, y también adquirir otros más.

En este material hemos seleccionado el lenguaje Java, tomando en cuenta que las aplicaciones web se pueden construir con diversos lenguajes y tecnologías, aquí destacaremos la relevancia de Java en el mundo del desarrollo web.

El desarrollo web con Java se ha convertido en una de las tecnologías más populares y versátiles para crear aplicaciones web dinámicas y escalables. Java, es un lenguaje de programación de propósito general, ha demostrado su robustez y capacidad para manejar sistemas complejos, y esta misma fortaleza se extiende al desarrollo de aplicaciones web.

Las aplicaciones web son muy útiles en la vida moderna. Se utilizan constantemente en las computadoras y en los dispositivos móviles para llevar a cabo diversos tipos de negocios y para acceder a una gran variedad de servicios. Es por esto que la industria del software necesita desarrolladores web y ofrece empleos relativamente bien remunerados. Por lo tanto, hay un gran interés entre los que eligieron profesiones relacionadas con la informática, por aprender a elaborar y mantener aplicaciones web.

Sin embargo, el aprendizaje de este tema es un proceso complejo, porque se debe integrar y poner en práctica los conocimientos adquiridos recientemente en su carrera y además adquirir otros más para poder explorar y adentrarse en el emocionante mundo del desarrollo web con Java.

¿Qué es una aplicación Web?

El término Web proviene del inglés, y significa red o malla, este término ha sido adoptado para referirse al internet. Una aplicación Web es un conjunto de páginas que funcionan en internet, estas páginas son las que el usuario ve a través de un navegador de internet (Internet Explorer de Microsoft, Chrome, Mozilla Firefox, etc.) y están codificadas en un lenguaje especial. Existen varios tipos de páginas Web: HTML, JSPs, XML, etc. En este curso trabajaremos con las JavaServer Pages (JSPs). Las páginas JSP se ejecutan en una máquina virtual de Java, el resultado de la ejecución es código HTML listo para correr en el navegador. Las JSP constituyen la interfaz de la aplicación con el usuario.

Las aplicaciones Web se almacenan en un servidor, el cual es una computadora que se encarga de que éstas sean accesibles a través de internet. Como se ilustra en la Figura I -1, una aplicación Web corre en un servidor bajo el control de un software especial, al cual se le llama también servidor. Para evitar confusiones es importante aclarar que el software servidor corre en una computadora servidor.

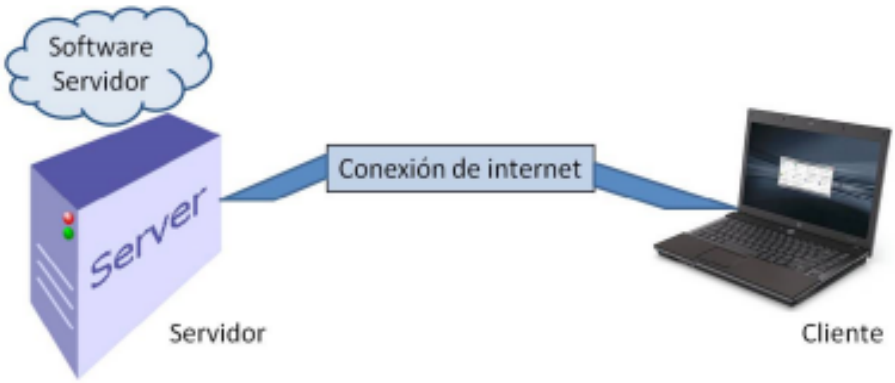

Figura I-1. Una aplicación web funciona con una computadora-servidor y una o varias computadoras-cliente conectadas a través de internet

Uno del software servidores más ampliamente utilizados es el Apache Tomcat, es de distribución libre. GlassFish es otro software servidor que también es muy utilizado.

Es muy común que las aplicaciones Web hagan uso de una base de datos ubicada en la computadora-servidor, como MySQL. El manejador de base de datos permite que varios clientes compartan la información, éste es uno de los aspectos más útiles de las aplicaciones web, ya que permite el comercio en línea (tiendas virtuales, reservaciones de hoteles, vuelos, etc.) y facilita la organización en las instituciones (solicitudes en línea, sistemas de inscripción, control de préstamos bibliotecarios, etc.), como se ilustra en la Figura I-2.

Figura I-2. Gracias al administrador de base de datos que corre en el servidor, las computadoras-cliente comparten una misma información.

Arquitectura REST

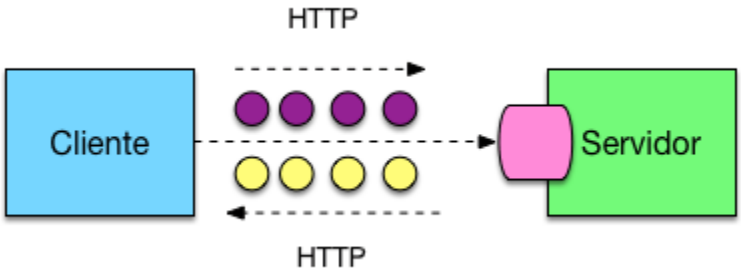

Figura I–3 Arquitectura REST

La arquitectura REST (Representational State Transfer) es un estilo arquitectónico utilizado en el diseño de aplicaciones web que se basa en principios y restricciones específicas para facilitar la comunicación entre sistemas distribuidos.

REST es una arquitectura de desarrollo web que puede ser utilizada en cualquier cliente HTTP. Además, es mucho más simple que otras arquitecturas ya existentes, como pueden ser XML-RPC o SOAP. Esta simplicidad se consigue porque emplea una interfaz web que usa hipermedios para la representación y transición de la información.

Fue propuesta por Roy Fielding en su tesis doctoral en 2000 y se ha convertido en una de las bases fundamentales de los servicios web modernos.

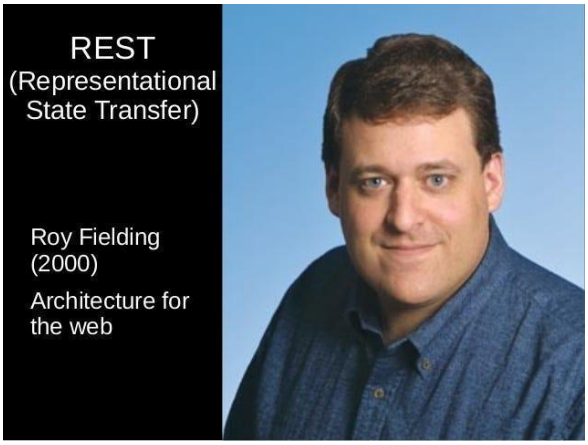

Figura I–4 Roy Fielding

A continuación, se presentan los conceptos clave de la arquitectura REST:

Recursos: En la arquitectura REST, todo es considerado como un recurso, que puede ser un objeto, un conjunto de datos o cualquier entidad. Cada recurso tiene una

identificación única llamada URI (Uniform Resource Identifier), que se utiliza para acceder y manipular el recurso.

- Las URI recibirán nombres que no deben implicar una acción, es decir, se debe evitar colocar verbos en ellas. Esto se debe a que se pone énfasis a los sustantivos.
- Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
- Deben ser independiente de formato, es decir, no debe representar ninguna extensión.
- Deben mantener una jerarquía lógica. La jerarquía es el criterio por el que se ordena los elementos.

Métodos HTTP: Los métodos HTTP (GET, POST, PUT, DELETE, etc.) se utilizan para definir las operaciones que se pueden realizar sobre los recursos. Por ejemplo, el verbo GET se utiliza para recuperar información de un recurso, mientras que POST se utiliza para crear un nuevo recurso.

Representaciones: Los recursos pueden tener diferentes representaciones, como JSON, XML, HTML, etc. La representación de un recurso es el formato en el que se devuelve la información al cliente que realiza la solicitud.

Estado de la Aplicación: La comunicación entre el cliente y el servidor en REST es stateless, lo que significa que cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender y procesar esa solicitud. El estado de la aplicación se mantiene en el cliente o en el servidor, pero no en el protocolo.

Interfaz Uniforme: REST sigue una interfaz uniforme que simplifica el diseño y la interacción entre los componentes de la aplicación. Esta interfaz se basa en el uso consistente de los verbos HTTP, URI y las representaciones de los recursos.

Sin Estado (Stateless): Las interacciones entre el cliente y el servidor en REST son independientes y no almacenan información de estado en el servidor entre solicitudes. Cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender y procesar esa solicitud.

Sistemas Distribuidos: REST está diseñado para ser utilizado en sistemas distribuidos, lo que significa que puede escalarse fácilmente y puede ser utilizado en aplicaciones que se ejecutan en diferentes servidores y plataformas.

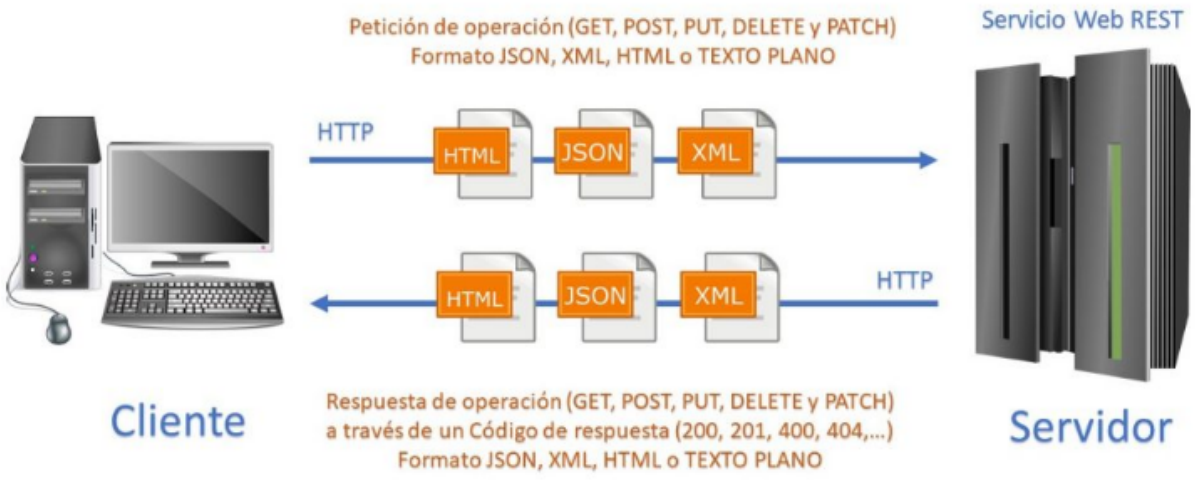

Figura I-5 Esquema Web Service REST

La arquitectura REST es un enfoque popular para el diseño de servicios web debido a su simplicidad, escalabilidad y flexibilidad. Permite una comunicación eficiente entre los componentes de una aplicación y facilita la interoperabilidad entre diferentes sistemas.

REST se ha convertido en el estándar predominante para el desarrollo de API (Interfaces de Programación de Aplicaciones) en la actualidad y es ampliamente utilizado en el desarrollo de aplicaciones web y móviles.

Métodos HTTP. GET. POST

Cuando hablamos de la arquitectura REST sabemos que se basa en recursos, donde cada uno de ellos está identificado de forma única.

Para poder interactuar con estos recursos utilizaremos, principalmente, los siguientes métodos HTML:

- GET: se utiliza para acceder a los distintos recursos.
- POST: se utiliza para realizar acciones de creación de nuevos recursos.
- PUT: se utiliza para modificar los recursos existentes.
- DELETE: se utiliza para eliminar los recursos.

Video explicativo: https://youtu.be/EHHDimfLyHg

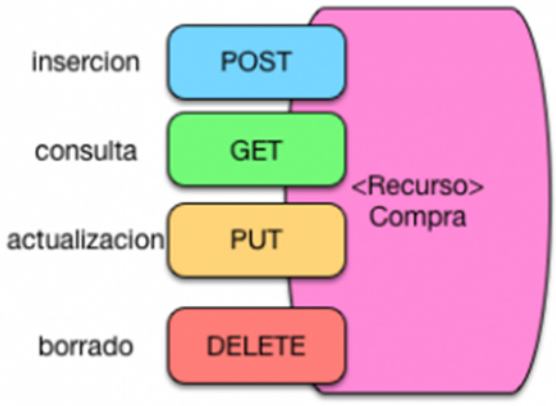

Figura I–4 Roy Fielding

- Método HTTP: GET

Es uno de los métodos HTTP, este realiza una petición a un recurso específico. No permite él envió de datos a excepción si dichos datos se envían como parámetro en la Url que realiza la petición. Esta petición retorna tanto la cabecera como el contenido. Ahora, este método GET puede retornar una respuesta en formato HTML, JSON, XML, Imágenes o JavaScript. Semánticamente se utiliza para consultar información como una SELECT a la base de datos, se puede filtrar datos empleando los datos enviados por la Url. Por otro lado, violando la semántica; los datos enviados por la Url se pueden usar para otros fines como una inserción, actualización y eliminación de un registro. Si se compara con las sentencias SQL es similar a un SELECT.

Formato de URL sin datos

- www.misitioweb/index.html

Formato de URL con datos

- www.misitioweb/index.html?nombre=williams&apellido=morales

En este caso se envía los datos de nombre y apellido donde las claves son nombre y apellido y valores williams y morales respectivamente.

Formato envió de datos por Formulario

- También se usa el método GET en un formulario como se muestra a continuación:

```
<form action="www.misitioweb/index.html" method="get">
    Nombre: <input type="text" name="nombre"><br>
  Apellido: <input type="text" name="apellido"><br>
        <input type="submit" value="Enviar">
```

```
</form>
```

Respuesta HTTP

200	Ok (Respuesta Positiva)
400	Bad Request
404	No Found

- Método HTTP: POST

Es otro de los métodos HTTP, este puede enviar datos al servidor por medio del cuerpo (body) y nada por la Url como se emplea en el método GET. El tipo de cuerpo de solicitud se define en la cabecera Content-Type. Semánticamente se utiliza para registrar información, similar al INSERT de datos a nivel de base de datos. A pesar de eso se puede forzar este método de petición HTTP para otras acciones como actualización, eliminación de registro, como carga de archivos, etc. También, se emplea para acciones que no tienen relación con el registro de información se debe considerar que el método POST no es Idempotente, es decir cada petición realiza un cambio diferente en el recurso del servidor web. El POST usa enviando los datos por formularios, formato JSON entre otros. Si se compara con las sentencias SQL es similar a un INSERT.

Formato envió datos por Formulario

Por otro lado, se puede enviar usar el método POST en un formulario como se muestra a continuación:

```
<form action="www.misitioweb/index.html" method="post">
    Nombre: <input type="text" name="nombre"><br>
    Apellido: <input type="text" name="apellido"><br>
    <input type="submit" value="Enviar">
</form>
```

Respuesta HTTP

200	Ok (Respuesta Positiva)
201	Creado
400	Bad Request
404	No Found

- Método HTTP: PUT

Es similar al método de petición POST, solo que el método PUT es idempotente; es decir puede ser ejecutado varias veces y tiene el mismo efecto, caso contrario a un POST que cada vez que se ejecuta realiza la agregación de un nuevo objeto, ya que semánticamente es como una inserción de un nuevo registro. Semánticamente el método HTTP PUT se utiliza para la actualización de información existente, es semejante a un UPDATE de datos a nivel de base de datos. Los requests de un PUT

usualmente se envían los datos por formularios, formato JSON entre otros. Si se compara con las sentencias SQL es similar a un UPDATE.

Respuesta HTTP

200	Ok (Respuesta Positiva)
201	Creado
204	No Responde, Si el servidor no devuelve ningún contenido
400	Bad Request
404	No Found

- Método HTTP: DELETE

Este método de petición permite eliminar un recurso específico. También es idempotente; es decir puede ser ejecutado varias veces y tiene el mismo efecto similar al PUT y GET. Semánticamente se utiliza para eliminar información existente, es semejante a un DELETE de datos a nivel de base de datos.

Respuesta HTTP

200	Elimina correctamente con body
202	La acción fue realizado correctamente, pero no se ha promulgado
204	Responde sin body

Por ejemplo, en el caso de una API sobre películas.

GET: Consulta y lee información sobre películas.

“/movies” nos permite acceder a todas ellas

“/movies/419” accedemos a la película con ID 419

```
GET https://www.domain.com/movies/419
Respuesta:
Status Code - 200 (OK)
{
 "id": 419,
 "title": "Eraserhead",
 "year": "1977"
}
```

POST: Crea una nueva película

“/movies”

```
POST https://www.domain.com/movies
Respuesta:
Status Code - 201 (CREATED)
{
 "movie": {
    "id": 419,
    "title": "Eraserhead",
    "year": "1977"
 }
}
```

PUT: Actualiza los datos de una película

“/movies/419” Actualizamos los datos de la película con id: 419

```
PUT https://www.domain.com/movies/419
Respuesta:
Status Code - 200 (OK)
```

DELETE: Elimina una película

“/movies/419” Elimina la película con id: 419

```
DELETE https://www.domain.com/movies/419
Respuesta:
Status Code - 204 (NO CONTENT)
```

Con esto deberíamos ofrecer las 4 funciones básicas de un modelo de datos. También conocidas con las siglas CRUD. Crear (Create), Leer (Read), Actualizar (Update) y Eliminar (Delete).

Las respuestas del servidor deben contener códigos de estado para avisar al cliente del éxito o no de cada operación. Para cada verbo HTTP tenemos una serie de códigos esperados que nos indicarán la ejecución correcta de cada operación:

GET: Código 200 (OK)

POST: Cödigo 201 (CREATED)

PUT: Código 200 (OK)

DELETE: Código 204 (NO CONTENT)