JSP (Java Server Pages)

Presentado por:

Oscar Alejandro González Soto

Presentado a:

Noé Arcos Muñoz

Universidad Distrital Francisco José de Caldas

14 de septiembre del 2024

Contenido

Definición de conceptos

JSP (JavaServer Pages):

Definición:

La tecnología JavaServer Pages (JSP) es una herramienta avanzada dentro de la plataforma Java Enterprise Edition (Java EE), diseñada para el desarrollo de aplicaciones web dinámicas. JSP facilita la creación de aplicaciones web al permitir la integración de contenido estático con lógica dinámica del servidor, simplificando así el proceso de desarrollo web en Java (Hunt & Thomas, 2002).

Características:

1. Inserción de código Java en documentos estáticos:

    - Scriptlets**:** Según Cervantes Ojeda (2017), "un scriptlet es un fragmento de código en Java que se incrusta en una página JSP". En las páginas JSP, se puede incluir código Java utilizando los delimitadores <% %>. Este método permite a los desarrolladores insertar lógica Java directamente en el archivo de la página web.
    - Expresiones**:** Las expresiones Java se insertan en una página JSP usando la sintaxis <%= %>. Esta técnica permite la evaluación y despliegue de valores directamente dentro del HTML (Jain & Arun, 2008).
    - Declaraciones**:** Las declaraciones en JSP se utilizan para definir variables y métodos que estarán disponibles en el resto de la página, usando los delimitadores <%! %>. Estas declaraciones permiten una mayor flexibilidad en la manipulación de datos y lógica dentro de las páginas JSP (Schildt, 2014).
2. Separación de lógica y presentación:

    - Modelo-Vista-Controlador (MVC): JSP se usa frecuentemente en el contexto del patrón de diseño Modelo-Vista-Controlador (MVC), donde JSP actúa como la capa de vista (Sharma, 2013). En este patrón, la lógica de negocio y el control del flujo se gestionan mediante servlets o frameworks adicionales como Spring, mientras que JSP se encarga de la presentación de la información al usuario.
3. Ciclo de Vida:

    - Compilación**:** La primera vez que se accede a una página JSP, el servidor de aplicaciones compila el archivo .jspen un servlet Java. Este servlet se convierte en una clase Java que se ejecuta para generar el contenido HTML dinámico (Geary & Bates, 2009).

- Ejecución**:** En cada solicitud posterior, el servlet generado maneja la solicitud y produce la respuesta HTML que se envía al navegador del cliente (Hall, 2010).
- Integración con Java: JSP se integra profundamente con Java, lo que permite el uso de objetos Java, clases y bibliotecas directamente en las páginas JSP. Esto incluye la capacidad de interactuar con bases de datos, manejar sesiones de usuario y realizar otras tareas del lado del servidor (Schildt, 2014).

Ejemplos:

```jsp
<%--
    Document   : index
    Created on : 12/09/2024, 7:59:49 a. m.
    Author     : Estudiante
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<!--
Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h1>Tabla de multiplicar</h1>
        <section>
            <h2>Tabla del 9 - 1000</h2>
            <table>
                <thead>
                    <th>Numero</th>
                    <th>Operador</th>
                    <th>Iterador</th>
                    <th>Resultado</th>
                </thead>
                <tbody>
                    <% for(int i = 1; i < 101; i++){%>
                    <tr>
                        <td>9</td>
                        <td>X</td>
                        <td><%= i %></td>
                        <td>=</td>
                        <td><%= i * 9%></td>
                    </tr>
                    <% } %>
                </tbody>
            </table>
        </section>
    </body>
</html>
```

En ejemplo anterior, realizado el 12 de septiembre junto con el profesor Noe Arcos (2024), se ilustra la creación de una página JSP que tiene como finalidad usar Java dentro de un documento HTML para presentar la tabla de multiplicar del 9 con 100 iteraciones, a partir de indicadores como los scriptlets.

- En las lineas 1 a 5, se hace un comentario mediante el indicador <%--, --%>. El cuál contiene la información general del documento.
- En las linea 7, a través de la etiqueta “page”, la tecnología JSP, permite definir las propiedades de la página web, como el lenguaje, scripting y tipo de contenido.
- En la linea 35, se usan scriptlets para incrustar un bucle for que va a tomar 100 iteraciones, y de manera continua se implementa en formato HTML, la estructura básica de la tabla de multiplicar, y en las lineas 34, 36 y 38 se hace uso de estos indicadores para presentar el múltiplo, expresar el resultado como operación aritmética y cerrar el bucle for.

Servlets:

Definición:

Los *servlets* son componentes clave en la plataforma Java Enterprise Edition (Java EE) utilizados para crear aplicaciones web dinámicas. Un servlet es una clase en Java que se ejecuta en el servidor y gestiona las solicitudes y respuestas en aplicaciones web. Su propósito principal es procesar las solicitudes del cliente, generar respuestas y, en general, manejar la lógica del servidor en aplicaciones web basadas en Java (Hunter & Crawford, 2004).

Características:

1. Ciclo de Vida:

- Inicialización**:** Cuando un servlet es solicitado por primera vez, el contenedor de servlets lo carga y lo inicializa. Este proceso invoca el método “init()” del servlet, donde se puede configurar el servlet y realizar inicializaciones necesarias (Hall, 2010).
- Procesamiento de Solicitudes: Después de la inicialización, el servlet puede manejar solicitudes a través del método “service()”. Este método recibe solicitudes HTTP y genera respuestas basadas en la lógica del servlet. El método "doGet()” maneja solicitudes GET, mientras que “doPost()” maneja solicitudes POST (Geary & Bates, 2009).
- Destrucción**:** Cuando el servlet ya no es necesario o el contenedor se apaga, se invoca el método “destroy()”, permitiendo que el servlet libere recursos y realice tareas de limpieza antes de ser

destruido (Fowler, 2003).

2. Interacción con Otros Componentes:

    - Acceso a Objetos Implicítos: Los servlets tienen acceso a objetos implicítos como HttpServletRequest y HttpServletResponse, lo que permite manejar datos de la solicitud del cliente y construir respuestas adecuadas. Por ejemplo, el objeto HttpServletRequest proporciona información sobre la solicitud del cliente, como parámetros de formulario y cabeceras HTTP (Hunter & Crawford, 2004).
    - Integración con JSP: Los servlets se integran frecuentemente con JavaServer Pages (JSP) en aplicaciones web. En el patrón de diseño Modelo-Vista-Controlador (MVC), los servlets suelen gestionar la lógica de negocio y la navegación, mientras que las páginas JSP se encargan de la presentación (Sharma, 2013).

3. Ventajas y Buenas Prácticas:

    - Desempeño**:** Los servlets son eficientes porque permanecen cargados en la memoria del servidor, lo que minimiza el tiempo de respuesta para solicitudes repetidas (Geary & Bates, 2009).
    - Modularidad**:** La estructura de servlets permite una clara separación de la lógica de negocio de la capa de presentación, facilitando el mantenimiento y la escalabilidad de las aplicaciones web (Hall, 2010).
    - Uso de Filtros y Listeners: Los servlets pueden ser complementados con filtros y listeners para manejar tareas transversales como la autenticación, la autorización, y la gestión de sesiones (Fowler, 2003).

Ejemplo:

```java
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;

@WebServlet("/HolaServlet")
public class HolaServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>Hola, Mundo desde un Servlet!</h1>");
        out.println("</body></html>");
    }
}
```

El servlet anterior es un sencillo ejemplo que hace uso del metodo doGet para enviar un saludo desde la lógica del servidor. Y su código se compone la siguiente estructura:

1. Desde la línea 1 hasta la 7, se accede al JDK para hacer uso de las librerías necesarias para:

    - Manejo de excepciones
    - La clase HttpServlet, que de forma analógica funciona como el contenedor de Java EE para componentes de Java como lo son los servlet, y es el encargado de gestionar su ejecución correspondiente y en este orden manejar la configuración determinada para una clase que contenga esta funcionalidad, como el ciclo de vida del servlet, el envío de parámetros a los métodos doGet() y doPost() y entre otros…
    - La interfaz de las solicitudes HTTP que provienen del usuario, las cuales son las encargados de la transferencia de datos desde la vista o archivo .jsp hacia el controlador, o servlet
    - La interfaz de respuesta HTTP, que proporciona funcionalidades esenciales como el Writter que va a permitir generar una respuesta en formato HTML a la solicitud del cliente

2. La linea 9 define el patrón URL para acceder al servlet, en otras palabras, el nombre del servlet al que tendrá acceso la vista del usuario

3. La linea 13 define el método doGet(), el cual tiene los parametros de solicitud y respuesta, el cuál será ejecutado cada vez que se realice una petición del tipo HTTP GET.

4. La linea 14 del ejemplo hace referencia al formato con que se va a generar la respuesta, en este caso mediante lenguaje de etiquetas html

Método GET:

Definición:

El método GET es uno de los métodos más utilizados en HTTP (Protocolo de Transferencia de Hipertexto). Se emplea para solicitar datos de un servidor mediante una URL, donde los datos se envían como parte de la cadena de consulta de la propia URL (Leffler, 2020).

Características:

- Exposición de datos: Los datos se envían como parámetros en la URL, lo que significa que pueden ser visibles en la barra de direcciones del navegador.
- Uso de caché: Las peticiones GET se pueden almacenar en caché por los navegadores, lo que puede mejorar el rendimiento cuando se solicitan los mismos datos repetidamente.
- Limitación en el tamaño de los datos: La cantidad de datos que se pueden enviar con GET está limitada, ya que los navegadores y servidores suelen restringir la longitud de la URL.
- Idempotencia: Las peticiones GET son idempotentes, lo que significa que realizar la misma solicitud repetidamente no cambiará el estado del servidor.
- No seguro para datos sensibles: Dado que los parámetros se incluyen en la URL, GET no es adecuado para enviar datos sensibles como contraseñas o información personal (Tan, 2019).

Método POST:

Definición*:*

El método POST se utiliza para enviar datos al servidor, normalmente para el propósito de crear o actualizar un recurso. A diferencia de GET, los datos se envían en el cuerpo de la solicitud HTTP en lugar de ser parte de la URL (Fielding et al., 1999).

Características*:*

- Los datos no están en la URL: A diferencia de GET, los datos se envían en el cuerpo de la solicitud, lo que significa que no son visibles en la barra de direcciones del navegador.
- Mayor seguridad: Aunque no es completamente seguro por sí solo, POST es más apropiado para enviar datos sensibles, ya que los datos no son visibles en la URL. Sin embargo, para asegurar la transmisión de datos, es recomendable usarlo con HTTPS.
- Tamaño ilimitado: No hay una limitación estricta sobre la cantidad de datos que se pueden enviar con POST, lo que lo hace adecuado para el envío de grandes cantidades de información.
- No es idempotente: A diferencia de GET, una solicitud POST no es idempotente, lo que significa que realizar la misma solicitud varias veces puede producir efectos diferentes (Davies, 2018).
- Uso en formularios y APIs: POST es comúnmente utilizado para enviar formularios y hacer peticiones a APIs donde se deben crear o modificar recursos en el servidor.

Diferencias entre el método GET y POST		
Diferencia:	GET	POST
Ubicación de los datos	URL (Son visibles por el usuario)	Protocolo HTTP, (Están ocultos para el usuario)
Tamaño de los datos	Restringido por algunos navegadores o servidores, generalmente 2048 caracteres	El tamaño de los datos no tiene limitaciones mediante el envío POST por más complejo que sea el formulario de envío
Caché	Generalmente estas solicitudes se almacenan en cache y permiten se reutilizados en futuras solicitudes.	Generalmente no se almacenan en cache porque modifican el estado del servidor
Idempotencia (Se realizan muchas solicitudes al servidor y no alteran su estado)	Idempotente	No idempotente, (manipulación de datos)

Patrones de diseño

DTO (Data Transfer Object)

Definición:

El patrón DTO, es un patrón de diseño estructural, que consiste en un objeto que es usado habitualmente para transferir datos de manera encapsulada y coherente desde una capa a otra, promoviendo así la comunicación eficaz entre subsistemas, y la abstracción en general de los requerimientos del negocio.

¿Qué es un patrón estructural?

Los patrones estructurales son un tipo de solución que busca organizar las clases y objetos, de manera que promueven la cohesión entre los distintos componentes de un sistema y asimismo mejoran la cohesión a partir del ordenamiento.

Características:

- Estructura Simple: Los DTOs suelen tener una estructura simple y estar compuestos únicamente por propiedades o campos de datos. No contienen lógica de negocio ni comportamientos complejos.
- Sin Lógica de Negocio: A diferencia de los objetos de dominio, los DTOs no tienen métodos que realicen operaciones sobre los datos. Su propósito es solo almacenar y transferir datos.
- Serialización: Los DTOs suelen ser fáciles de serializar y deserializar, lo cual es útil para el intercambio de datos entre sistemas distintos o para la persistencia en almacenamiento.
- Inmutabilidad: Aunque no es un requisito estricto, es común que los DTOs sean inmutables, es decir, sus propiedades no cambian una vez que se han establecido. Esto ayuda a evitar problemas de sincronización y facilita el trabajo con datos en diferentes contextos.

Ejemplo:

```java
import java.util.Date;

public class TaskDTO {
    private Long id;
    private String title;
    private String description;
    private Date dueDate;

    public TaskDTO() {
    }

    public TaskDTO(Long id, String title, String description, Date dueDate) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.dueDate = dueDate;
    }
    public Long getId() {return id;}
    public String getTitle() {return title;}
    public String getDescription() {return description;}
    public Date getDueDate() {return dueDate;}

    public void setId(Long id) {this.id = id;}
    public void setTitle(String title) {this.title = title;}
    public void setDescription(String description) {this.description = description;}
    public void setDueDate(Date dueDate) {this.dueDate = dueDate;}

    @Override
    public String toString() {
        return "TaskDTO{" + "id=" + id + ", title='" + title + '\'' + ", description='" + description +
        '\'' +", dueDate=" + dueDate +'}';
    }
}
```

- Línea 1: Importa la clase `Date` de la biblioteca estándar de Java. La clase `Date` se utiliza para representar fechas y horas. En este caso, se usará para manejar la fecha de vencimiento (dueDate) de la tarea.

- Línea 3: Esta línea define una nueva clase pública llamado TaskDTO. La palabra clave public indica que la clase puede ser accesible desde cualquier otra clase.

- Líneas 4 a 7: Se declaran las variables (id, title, description, dueDate) que va a usar la clase, también conocidas como atributos, el termino private indica que están encapsulados y hace que ningún componente pueda acceder a ellos a excepción de la misma clase.

- Línea 9: Este es el constructor sin parámetros de la clase TaskDTO. Es necesario para crear una instancia de la clase sin inicializar los atributos en el momento de la creación.

- Líneas 12 a 17: Este es el constructor con parámetros de la clase TaskDTO. Permite crear una instancia de TaskDTO e inicializar todos los atributos de la clase en el momento de la creación.

- Líneas 18 a 26: Son los métodos o comportamientos que tendrá la clase para acceder al valor de los atributos o modificarlos desde otra parte esencial del sistema, estos métodos se ocupan mayormente de la transferencia de datos.

- Líneas 28 a 32: El método toString(), permite la obtención de un texto mediante el cual se puedan evidenciar los valores de las propiedades del objeto, el @override significa que está sobrescribiendo un método de la clase abstracta o padre “Object”

Definición:

El patrón DAO, es una capa de diseño, que me permite separar la lógica del negocio con la persistencia de los datos.

¿Qué es una capa de diseño?

Una *capa de diseño* se refiere a una estructura en la arquitectura de software que organiza los componentes del sistema en niveles jerárquicos, cada uno de los cuales tiene responsabilidades específicas. Esta organización facilita la separación de preocupaciones y mejora la modularidad, permitiendo una mejor comprensión, mantenimiento y evolución del sistema (Sommerville, 2015). Según Pressman (2014), las capas de diseño permiten que cada nivel de la arquitectura cumpla con un rol definido, facilitando la gestión de la complejidad y la reutilización del software. Además, Gamma et al. (1994) destacan que las capas proporcionan una abstracción que ayuda a encapsular detalles

específicos de implementación y a manejar la interacción entre diferentes partes del sistema de manera ordenada.

Componentes del patrón DAO:

1. DAO (Data Access Object):
    - Es una interfaz o clase abstracta que define los métodos para realizar operaciones sobre la fuente de datos, como crear, leer, actualizar y eliminar (CRUD).
2. Implementación de DAO:
    - Es una clase concreta que implementa la interfaz DAO y proporciona la lógica específica para interactuar con la fuente de datos. Por ejemplo, puede usar JDBC para acceder a una base de datos SQL o utilizar una API para acceder a un servicio web.
3. Modelo o Entidad:
    - Representa los datos que se están manipulando. Es una clase que modela la estructura de los datos de acuerdo con la fuente de datos.
4. Cliente de DAO:
    - Es el componente de la lógica de negocio que utiliza el DAO para realizar operaciones de acceso a datos. El cliente de DAO no necesita conocer los detalles de cómo se accede a la fuente de datos; solo interactúa con la interfaz DAO.

EJEMPLO:

```java
import java.sql.*;
import java.util.Date;

public class TaskDAOImpl implements ITaskDAO {
    private static final String URL = "jdbc:mysql://localhost:3306/basedatos";
    private static final String USER = "localhost";
    private static final String PASSWORD = "123456";
    private static final String INSERT_TASK_SQL = "INSERT INTO tasks (id, title, description, due_date)
    VALUES (?, ?, ?, ?)";
    private static final String SELECT_TASK_BY_ID_SQL = "SELECT * FROM tasks WHERE id = ?";
    private static final String UPDATE_TASK_SQL = "UPDATE tasks SET title = ?, description = ?,
    due_date = ? WHERE id = ?";
    private static final String DELETE_TASK_SQL = "DELETE FROM tasks WHERE id = ?";

    @Override
    public void createTask(TaskDTO task) {
        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement preparedStatement = connection.prepareStatement(INSERT_TASK_SQL)) {
            preparedStatement.setLong(1, task.getId());
            preparedStatement.setString(2, task.getTitle());
            preparedStatement.setString(3, task.getDescription());
            preparedStatement.setDate(4, new java.sql.Date(task.getDueDate().getTime()));
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
            throw new RuntimeException("Error creating task", e);
        }
    }

```

Recuperado de "TutorialesPoint. (2023). Interfaz IUserDAO, Clase UserDTO, Clase UserDAOImpl. TutorialesPoint. https://www.tutorialspoint.com/java/dao_example.htm"

- Línea 1: import java.sql.*: Importa todas las clases del paquete java.sql, que es necesario para trabajar con bases de datos en Java.
- Línea 2: import java.util.Date; : Importa la clase Date de java.util, que se usa para manejar fechas.
- Línea 4: public class TaskDAOImpl implements ITaskDAO { : Declara la clase TaskDAOImpl, que implementa la interfaz ITaskDAO.
- Línea 5: private static final String URL = "jdbc:mysql://localhost:3306/basedatos"; : Define una constante URL que especifica la URL de la base de datos MySQL. Aquí localhost es el host y 3306 es el puerto por defecto de MySQL. basedatos es el nombre de la base de datos.
- Línea 6: private static final String USER = "localhost"; : Define una constante USER que contiene el nombre de usuario para conectarse a la base de datos. Nota: Aquí parece haber un error; típicamente debería ser un nombre de usuario como "root".
- Línea 7: private static final String PASSWORD = "123456"; : Define una constante PASSWORD que contiene la contraseña para la conexión a la base de datos.
- Línea 8: private static final String INSERT_TASK_SQL = "INSERT INTO tasks (id, title, description, due_date) VALUES (?, ?, ?, ?)"; : Define una constante INSERT_TASK_SQL que contiene la sentencia SQL para insertar una nueva tarea en la tabla tasks. Los signos de interrogación (?) son marcadores de posición para los valores que se insertarán.
- Línea 9: private static final String SELECT_TASK_BY_ID_SQL = "SELECT * FROM tasks WHERE id = ?"; : Define una constante SELECT_TASK_BY_ID_SQL que contiene la sentencia SQL para seleccionar una tarea de la tabla tasks según su id.
- Línea 10: private static final String UPDATE_TASK_SQL = "UPDATE tasks SET title = ?, description = ?, due_date = ? WHERE id = ?"; : Define una constante UPDATE_TASK_SQL que contiene la sentencia SQL para actualizar una tarea en la tabla tasks. Se actualizan los campos

title, description, y due_date para una tarea específica identificada por su id.

- Línea 11: private static final String DELETE_TASK_SQL = "DELETE FROM tasks WHERE id = ?"; : Define una constante DELETE_TASK_SQL que contiene la sentencia SQL para eliminar una tarea de la tabla tasks según su id.

Interfaces:

Definición:

Una interfaz es una referencia que define un conjunto de métodos que una clase debe implementar. En lugar de proporcionar una implementación concreta para estos métodos, una interfaz solo especifica la firma de los métodos (es decir, el nombre del método, los parámetros y el tipo de retorno). Las clases que implementan una interfaz deben proporcionar la implementación de estos métodos.

Características de las Interfaces:

1. *Contrato sin Implementación*: Una interfaz define un contrato que las clases deben cumplir, pero no proporciona implementación de los métodos. Esto permite que diferentes clases puedan implementar la interfaz de diferentes maneras.
2. *Métodos Abstractos*: En las versiones anteriores a Java 8, todos los métodos en una interfaz son abstractos por defecto (es decir, no tienen implementación). Desde Java 8, las interfaces pueden tener métodos con implementación (métodos por defecto y estáticos), pero aún se espera que los métodos no implementados sean implementados por las clases que las usan.
3. *Herencia Múltiple*: A diferencia de las clases, una interfaz puede extender múltiples interfaces. Esto permite una forma de herencia múltiple, que no es posible con clases en Java.
4. *Implementación por Clases*: Las clases implementan interfaces y proporcionan la implementación concreta de los métodos definidos en la interfaz.
5. *Interfaz y Modificadores de Acceso*: Los métodos en una interfaz son, por defecto, públicos y abstractos. En las interfaces, no se necesita especificar el modificador de acceso public para los métodos, pero es una buena práctica hacerlo explícito.

6. *Constantes*: Las interfaces pueden contener constantes, que son implícitamente public, static y final.

7. *Compatibilidad hacia Adelante*: Las interfaces permiten que el sistema sea más flexible y compatible con versiones futuras, ya que se puede añadir nuevos métodos a una interfaz (con un enfoque cuidadoso para mantener la compatibilidad).

Javax y jakarta

Javax:

En el ecosistema de Java, el prefijo "javax" se utiliza para designar paquetes y clases que forman parte de las extensiones de la API estándar de Java, proporcionando funcionalidades adicionales más allá del núcleo esencial del Java Development Kit (JDK). Aunque estas bibliotecas son importantes y a menudo utilizadas, no están incluidas en el núcleo esencial del JDK, permitiendo así que el lenguaje y sus herramientas básicas se mantengan más simples y coherentes. El uso de "javax" ayuda a garantizar que las nuevas funcionalidades no interfieran con el código existente y facilita la evolución y expansión de la plataforma Java.

Jakarta:

Jakarta EE representa la evolución de Java EE, un conjunto de especificaciones que definen la plataforma estándar para el desarrollo de aplicaciones empresariales en Java. Con la transición de Java EE a Jakarta EE, se busca fomentar una mayor innovación y colaboración dentro de la comunidad de desarrolladores, al tiempo que se mantiene la compatibilidad con versiones anteriores.

¿Por qué el cambio de nombre?

La principal razón detrás del cambio de nombre de Java EE a Jakarta EE se debe a cuestiones relacionadas con marcas registradas. Oracle, propietario de la marca Java, decidió centrar sus esfuerzos en el lenguaje Java en sí mismo, y permitió a la comunidad de desarrolladores tomar las riendas de las especificaciones Java EE bajo el auspicio de la Fundación Eclipse.

¿Qué tecnologías incluye Jakarta EE?

Jakarta EE abarca una amplia gama de tecnologías, incluyendo:

- Servlets: Para crear aplicaciones web dinámicas.

- JSP: Para generar contenido HTML de forma dinámica.

- EJB: Para desarrollar componentes empresariales reutilizables.
- JPA: Para interactuar con bases de datos relacionales.
- JMS: Para enviar y recibir mensajes de forma asíncrona.
- JTA: Para gestionar transacciones distribuidas.
- WebSockets: Para establecer comunicaciones bidireccionales en tiempo real.
- JSON-P: Para procesar datos en formato JSON.
- RESTful Web Services: Para crear servicios web basados en REST.

Diferencias entre JAVAX y JAKARTA

Característica	javax (Java EE)	jakarta (Jakarta EE)
Espacio de nombres	javax.*	jakarta.*
Organización	Oracle (JCP)	Fundación Eclipse
Proceso de desarrollo	JCP (Java Community Process)	Eclipse Development Process
Enfoque	Más centrado en empresas grandes	Más abierto y colaborativo, orientado a la comunidad
Ciclo de lanzamiento	Más lento	Más rápido y ágil
Innovación	Más conservador	Mayor énfasis en nuevas características y tecnologías
Licencia	Oracle Binary Code License	Licencias de código abierto (por ejemplo, Eclipse Public License)
Neutralidad del proveedor	Menos neutral, influenciado por Oracle	Totalmente neutral, sin un proveedor dominante

Referencias

- Bergsten, H. (2003). *JavaServer Pages*. O'Reilly Media.
- Hall, J. (2010). *Core Servlets and JavaServer Pages*. Prentice Hall.
- Sharma, R. (2013). *JavaServer Pages: The Complete Reference*. McGraw-Hill.

- Cervantes Ojeda, J. (2017). *Introducción a JavaServer Pages (JSP).*
- Davies, E. (2018). *Understanding REST: Methods and Verbs*. O'Reilly Media.
- Fielding, R. T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext Transfer Protocol -- HTTP/1.1*. RFC 2616. https://www.rfc-editor.org/rfc/rfc2616
- Leffler, J. (2020). *Web Forms and HTTP: GET vs POST*. Web Development Journal.
- Tan, W. (2019). *HTTP Protocol: The GET and POST Methods Explained*. Tech Insights.