DOXA EDITION

Autor:
ELVIS PACHACAMA
LENIN PACHACAMA

2025

Desarrollo Web con Java:

de la idea a la implementación I

Primera Edición

ITQ
WWW.ITQ.EDU.EC
INVESTIGACIÓN

INSTITUTO SUPERIOR TECNOLÓGICO QUITO
Excelencia en Educación Superior

DESARROLLO WEB CON JAVA: DE LA IDEA A LA IMPLEMENTACIÓN I

AUTORES:

ELVIS DAVID PACHACAMA CABEZAS

LENIN FABRICIO PACHACAMA CABEZAS

EDICIÓN: PRIMERA

AÑO: 2025

TRABAJO EN EDICIÓN:

DOXA EDITION

EQUIPO EDITORIAL

MSc. CAMILA DÍAZ DÍAZ

MSc. KEYERMAN TOAPANTA CISNEROS

ISBN:

DEDICATORIA

A mis padres, cuyo amor, sacrificio y enseñanza forjaron el carácter y la pasión para alcanzar cada meta que me propongo. Gracias por ser mi faro de luz en los momentos de oscuridad y mi apoyo incondicional en cada paso del camino.

A mi hija Keylita, cuyo espíritu vive en cada latido de mi corazón y cada sueño que persigo. Aunque no pueda tenerte en mis brazos, siempre estarás en mi alma, inspirándome a construir un legado digno de tu memoria. Esta obra es un homenaje a lo que nunca fue, pero que siempre será en mi corazón: mi amor eterno por ti.

Con profundo agradecimiento y amor, esta obra es para ustedes.

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a todas aquellas personas que hicieron posible la creación de este libro.

En primer lugar, quiero agradecer a mi familia por su amor, apoyo y paciencia durante todo el proceso. Gracias por haberme brindado el espacio y el tiempo necesario para poder concentrarme en esta tarea y por haber sido mi fuente de inspiración en cada página.

También quiero agradecer a mis amigos y colegas por su constante motivación, por haberme brindado retroalimentación y sugerencias valiosas durante la escritura de este libro. Gracias por su amistad y por haber compartido conmigo momentos de alegría y distracción.

Un agradecimiento especial a mi editor, por su profesionalismo, dedicación y paciencia en la revisión y edición de este libro. Gracias por haberme guiado en cada paso del proceso de publicación y por haber hecho posible la materialización de este proyecto.

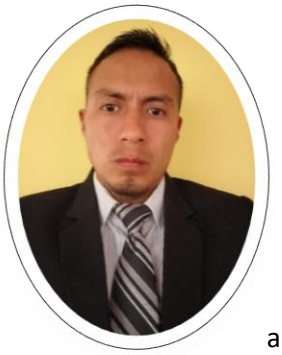
Finalmente, quiero agradecer a mis lectores por su interés en mi trabajo y por brindarme la oportunidad de compartir mis ideas, pensamientos y pasión por la educación. Espero que este libro les brinde una experiencia única y significativa.

SOBRE LOS AUTORES

Elvis Pachacama Cabezas es un educador y profesional con un título de tercer nivel en Ingeniería en Tecnologías de la Información, obtenido en la Universidad Estatal de Sumy en Ucrania (SUMDU). Se destaca por su participación en la vida estudiantil, ya que fue representante de los estudiantes ecuatorianos en el Consejo Estudiantil de la universidad, lo que sugiere un compromiso activo con los asuntos estudiantiles.

Además, Elvis se graduó con honores y recibió reconocimiento por su destacado desempeño académico en su título universitario. Esta distinción refleja su dedicación y excelencia en sus estudios. Actualmente, Elvis Pachacama Cabezas es docente en el Instituto Superior Tecnológico Quito, donde imparte cátedra en la Carrera de Desarrollo de Software. Su posición como educador indica una dedicación al campo de la enseñanza y sugiere que posee conocimientos prácticos en el desarrollo de software, que comparte con los estudiantes.

Adicionalmente, es relevante mencionar que Elvis está próximo a cursar la maestría en inteligencia artificial en la Universidad Internacional de Valencia. Este detalle indica su interés en profundizar sus conocimientos en un campo avanzado y en constante evolución, como es la inteligencia artificial.



Lenin Fabricio Pachacama Cabezas es un profesional ecuatoriano de 38 años, graduado de la Universidad Politécnica Javeriana, donde obtuvo su título en Ingeniería Mecánica Automotriz, consolidando una sólida formación técnica y científica en el área de los sistemas mecánicos, termodinámicos y automotrices. A lo largo de su trayectoria académica y profesional, ha demostrado un profundo compromiso con la innovación, la enseñanza y la aplicación práctica de los principios de la ingeniería en el ámbito educativo y técnico. Además, Lenin se graduó con Complementando su perfil técnico, el Ingeniero Pachacama obtuvo el título de Magíster en Tecnologías de la Información con mención en Educación por la Pontificia Universidad Católica del Ecuador (PUCE). Esta formación de posgrado le ha permitido integrar de manera efectiva las herramientas tecnológicas en los procesos de enseñanza-aprendizaje, impulsando una educación técnica moderna, interactiva y alineada con los desafíos de la era digital.

Actualmente, se desempeña como docente del Ministerio de Educación del Ecuador, donde imparte las asignaturas relacionadas con la Mecánica Automotriz y sus áreas complementarias: motores de combustión interna, sistemas eléctricos y electrónicos del vehículo, diagnóstico computarizado, suspensión, frenos, transmisión y mantenimiento automotriz. Su labor docente se distingue por un enfoque práctico, participativo y orientado a competencias, que busca desarrollar en sus estudiantes las habilidades técnicas y el pensamiento crítico necesarios para enfrentar los retos del sector automotor contemporáneo.

El Ingeniero Pachacama es reconocido por su vocación pedagógica, su actualización constante en nuevas tecnologías y su capacidad para vincular la ingeniería mecánica con la innovación educativa. Gracias a su compromiso con la formación técnica y su aporte al fortalecimiento del sistema educativo nacional, se ha consolidado como un referente en el ámbito de la educación automotriz, promoviendo una enseñanza integral que combina el conocimiento científico, la práctica profesional y la ética al servicio de la sociedad ecuatoriana.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN AL CONTENIDO DEL LIBRO

El desarrollo de aplicaciones de aplicaciones web se ha consolidado como una habilidad esencial en el campo de la tecnología. Con el presente libro, titulado “desarrollo web con jakarta: de la idea a la implementación”, se busca proporcionar a estudiantes de desarrollo de software y desarrolladores un enfoque claro práctico para comprender y aplicar las tecnologías que forman parte del ecosistema Jakarta EE, una evolución moderna de Java EE.

La obra está estructurada para guiarte paso a paso desde los conceptos básicos hasta la implementación de proyectos completos, integrando herramientas y tecnologías de vanguardia. A través de cada capítulo, podrás desarrollar habilidades técnicas y prácticas que te permitirán enfrentar los desafíos del desarrollo web moderno.

Capítulo 1: Fundamentos del Desarrollo Web.

El punto de partida de este libro radica en comprender los fundamentos del desarrollo web, Analizarás la arquitectura cliente-servidor y los principios básicos del protocolo HTTP, así como la interacción entre HTML, CSS y JavaScript para crear interfaces web interactivas. Este capítulo proporciona el contexto esencial para temas más avanzados.

Capítulo 2: Introducción a Jakarta para la Web

En este capítulo, aprenderás a configurar el entorno de desarrollo necesario para trabajar con Jakarta EE. Desde la instalación del JDK hasta el despliegue de aplicaciones en Apache Tomcat, también se presentarán herramientas como Intellij IDEA o Eclipse, esenciales para el desarrollo eficiente de aplicaciones web.

Capítulo 3: Servlets

Los Servlets constituyen el núcleo de las aplicaciones web en Jakarta EE. Este capítulo te llevará a entender su ciclo de vida, gestión de peticiones HTTP, manejo de sesiones y cookies, Además, se mostrará cómo estructurar aplicaciones escalables y robustas mediante estos componentes.

Capítulo 4: JavaServer Page (JSP)

Las JavaServer Pages (JSP) permiten generar contenido web dinámico mediante la combinación de código Java y HTML5. Este capítulo abarca desde las directivas básicas hasta el uso de librerías avanzadas como JSTL, integrando además JavaBeans para construir aplicaciones web bien estructuradas.

Capítulo 5: Java Beans

El concepto de JavaBean surge de la necesidad de crear componentes reutilizables, es decir, objetos que puedan ser utilizados por distintas partes de un programa sin necesidad de reescribir el mismo código una y otra vez. En una arquitectura de tipo MVC (Modelo-Vista-Controlador), los JavaBeans suelen ocupar el papel del modelo, ya que encapsulan la lógica de negocio o los datos que posteriormente serán mostrados en las vistas. Gracias a su naturaleza independiente y portable, un mismo bean puede ser integrado en diferentes aplicaciones sin modificación alguna, siempre y cuando respete las convenciones establecidas por la especificación de Java.

Capítulo 6: Expression Language con JSP

La Expression Language (EL) es un componente fundamental dentro de las tecnologías Jakarta EE (antes Java EE) que facilita la interacción entre las páginas JSP y los objetos del servidor, permitiendo acceder de manera sencilla a los datos contenidos en los JavaBeans, servlets, o en los distintos ámbitos del entorno web (request, session, application). Su principal propósito es simplificar el código al eliminar la necesidad de escribir fragmentos Java directamente en las páginas JSP, reemplazándolos por expresiones compactas y legibles. Mediante una sintaxis basada en el uso de ${} o #{}, EL evalúa dinámicamente valores y expresiones en tiempo de ejecución, lo que mejora la claridad del código y separa la lógica de negocio de la presentación. Además, soporta operaciones aritméticas, lógicas y de comparación, así como el acceso a propiedades, colecciones y métodos, haciendo posible construir vistas dinámicas con gran flexibilidad. En definitiva, la Expression Language se convierte en un puente esencial que integra la capa visual con la lógica del servidor, promoviendo el desarrollo de aplicaciones web más limpias, mantenibles y orientadas a componentes.

Capítulo 7: Manejo de JSTL (Jakarta Standard Tag Librery)

El manejo de la JSTL (Jakarta Standard Tag Library) en las páginas JSP constituye una de las herramientas más potentes y elegantes para el desarrollo de aplicaciones web dinámicas en Java. Esta biblioteca estandarizada proporciona un conjunto de etiquetas que reemplazan el uso excesivo de código Java dentro de las vistas, promoviendo así una mejor separación entre la lógica de negocio y la presentación. A través de la JSTL, es posible realizar operaciones comunes como el control de flujo (if, forEach), la manipulación de datos, el formateo de números y fechas, la internacionalización de contenido, e incluso la interacción con bases de datos mediante etiquetas especializadas. Su integración con la Expression Language (EL) permite acceder de forma directa y sencilla a los atributos almacenados en los distintos ámbitos de la aplicación, como request, session o application, sin necesidad de escribir código imperativo. En la práctica, el uso de la JSTL contribuye significativamente a mantener un código más limpio, modular y fácil de mantener, al mismo tiempo que mejora la

legibilidad de las páginas JSP, convirtiéndose en un componente esencial dentro de las arquitecturas web modernas basadas en Jakarta EE.

Capítulo 8: Modelo Vista Controlador MVC

El patrón Modelo-Vista-Controlador (MVC) es una arquitectura fundamental en el desarrollo de aplicaciones web con Jakarta EE, ya que permite organizar el código de forma estructurada, clara y mantenible. Su principal objetivo es separar la lógica de negocio, la presentación y el control del flujo de la aplicación, evitando que se mezclen responsabilidades dentro de un mismo componente. En este modelo, el Modelo (Model) representa los datos y las reglas de negocio, generalmente implementados mediante JavaBeans que encapsulan la información y la lógica interna. La Vista (View) corresponde a la capa de presentación, encargada de mostrar los datos al usuario, y suele estar construida con JSP apoyadas en Expression Language (EL) y JSTL para generar contenido dinámico. Finalmente, el Controlador (Controller) actúa como intermediario entre la vista y el modelo, recibiendo las peticiones del usuario a través de Servlets, procesando la información y determinando qué vista debe mostrarse como respuesta.

Capítulo 9. Web Services

Los Web Services con JAX-WS (Jakarta XML Web Services) en Jakarta EE constituyen una de las formas más robustas y estandarizadas para permitir la comunicación entre aplicaciones distribuidas a través de la web. Esta tecnología se basa en el intercambio de mensajes SOAP (Simple Object Access Protocol) y utiliza XML como formato de datos, garantizando la interoperabilidad entre sistemas desarrollados en diferentes lenguajes o plataformas. Con JAX-WS, los desarrolladores pueden crear servicios web de manera sencilla utilizando anotaciones Java, como @WebService o @WebMethod, que definen los puntos de acceso y las operaciones del servicio. Además, el contenedor Jakarta EE se encarga automáticamente del despliegue, la generación del WSDL y la gestión de las solicitudes, reduciendo considerablemente la complejidad técnica. Gracias a esta API, es posible implementar arquitecturas orientadas a servicios (SOA) que promueven la integración empresarial, la reutilización de componentes y la escalabilidad, convirtiendo a JAX-WS en un pilar esencial para el desarrollo de soluciones corporativas basadas en Jakarta EE.

CAPÍTULO 1

FUNDAMENTOS DEL DESARROLLO WEB

1. INTRODUCCIÓN

Durante la historia el desarrollo web ha evolucionado a pasos gigantescos desde simples páginas estáticas hasta complejas aplicaciones interactivas que conectan millones de usuarios con servicios en la nube. Este capítulo te guiará por los conceptos fundamentales que forman la base del desarrollo web, con un enfoque en el rol de Java como una tecnología clave en este ámbito.

Una página Web contiene información estática, y que debemos cambiar manualmente si deseamos ver cambios en la misma. Un sitio web es el conjunto de páginas web donde la información de la misma manera es estática. En cambio, una aplicación Web la información se puede recuperar de manera dinámica desde una base de datos.

1.1. Historia y evolución del desarrollo Web

La evolución del diseño web ha sido impresionante en los últimos años. Desde la aparición de la World Wide Web en 1989, el diseño web ha experimentado un sin número de cambios y mejoras.

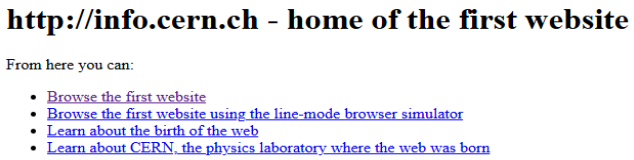
En los primeros días de la web, todas las páginas eran estáticas. En sus inicios los sitios web estaban construidas únicamente con HTML y la experiencia que tenía el usuario era limitada. Cuando se quería modificar el contenido, se lo tenía que realizar manualmente, volverlo a subir al servidor y se debía esperar que los cambias se realicen.

El desarrollo web comenzó oficialmente en 1989, cuando Tim Berners-Lee científico del CERN (Organización Europea para la Investigación Nuclear), Berners-Lee propuso desollar un sistema de gestión de información basado en hipertexto. La idea era conectar documentos a través de enlaces y de esa manera facilitar la navegación entre distintos temas.

Al inicio de la década de los 90's Berners-Lee desarrollo la primera página web del internet en el domino info.cern.ch, edonde se explicaba cómo funcionaba la web y el lenguaje HTML.

Figura 1.

Primera página web.



Nota. La imagen nos muestra la primera página web. *Fuente:* (https://info.cern.ch/, s.f.)

Como se puede observar en la figura 1. Las páginas web eran:

- **Estáticas:** Las primeras páginas web no tenían interactividad ni contenido con el usuario.
- **Código HTML:** El contenido de las páginas web solo se podían estructurar texto con encabezado, párrafos y enlaces.
- **Accesibles en navegadores primitivos:** La primera página web solo se podía ejecutar en el navegador WorldWideWeb.

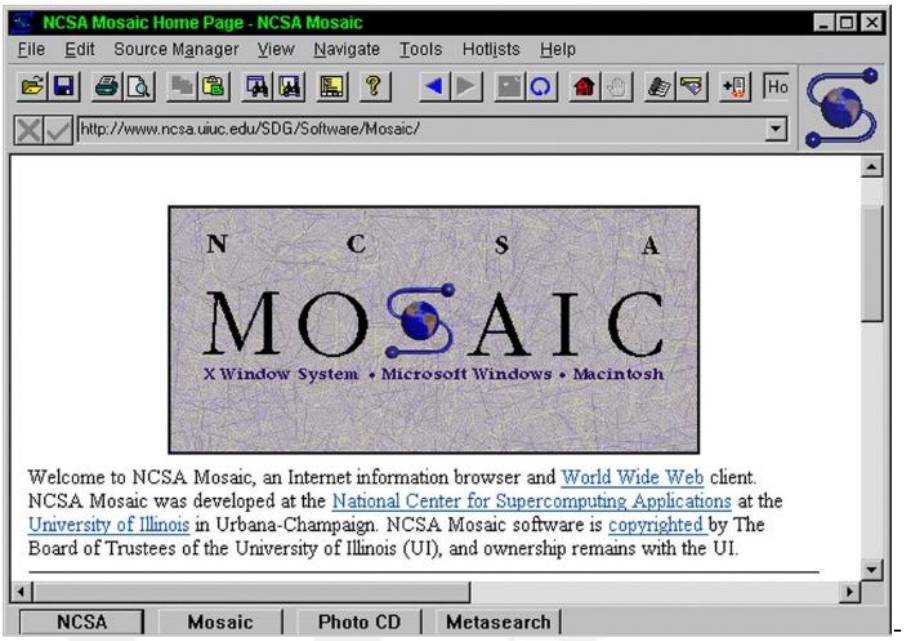
De la misma manera en esta década nace el concepto de servidor web y clientes, de esta manera estable la base del modelo cliente-servidor, en donde un cliente (navegador) realiza una petición a un servidor, el cual al responder lo hace mediante una página web.

Mientras la tecnología avanza a pasos gigantescos surgieron los primeros navegadores.

- **Mosaic 1993:** Este primer navegador permitió mostrar el texto junto a imágenes.

Figura 2.

Navegador Mosaico



Nota. La imagen nos muestra el primer navegado desarrollado para ver páginas web. *Fuente:* (i.blogs.es, s.f.)

Netscape Navigator(1994): El navegador Mosaico marca un hito desde su llegada expandiéndose por toda la web y domino el mercado hasta la llega de Internet Explorer:

Figura 3.

Navegador Netscape Navigator

Nota. La imagen nos muestra uno de los navegadores más populares de su época. *Fuente:* (i.blogs, s.f.)

Hasta la mitad de la década de los 90's, todas las páginas web eran simplemente informativas. Sin embargo, los desarrolladores se dieron cuenta que el usuario necesitaba interactuar más con las páginas web y sean más interactivas, es aquí donde surgen tecnologías claves como:

- **JavaScript (1995):** Lenguaje de programación desarrollado para el lado del cliente, esta tecnología fue desarrollada por **Netscape**, integrado interactividad como (botones dinámicos o validaciones de formularios)
- **Common Gateway Interface (CGI):** Esta tecnología permitió que los servidores ejecutaran programas el cual generan contenido dinámico, como procesar formularios o mostrar información personalizada.

1.2. La batalla de los navegadores

Ha mediado de los años 90's Microsoft entro al mercado con Internet Explore, este lo integro a Windows 95 esta decisión llevo a una feroz competencia con Netscape, con ambos navegadores introduciendo nuevas características a cada una de ellas, en la mayoría de las veces sin seguir los estándares de la W3C (World Wide Web Consortium).

En el transcurso de los años también aparecieron otras tecnologías como:

- **CSS (1996):** Hojas de Estilo en Cascada Cascading Style Sheet, tecnología que se introdujo para darle estilos a las páginas web y de esa forma mejorar su apariencia.
- **Flash (1996):** Tecnología desarrollada por Macromedia que permitía animaciones interactivas esta herramienta fue muy popular hasta 2010.
- **PHP (1995) ASP (1996):** Tecnologías desarrollados del lado del servidor para construir sitios web dinámicos integrando base de datos.

Mientras la tecnología crece a pasos gigantescos la necesidad de almacenar información y generar contenido personalizado llevó al uso de las bases de datos relacionales como MySQL, SQL y PostgreSQL

1.3. La Web 2.0 Redes Sociales, AJAX y el nacimiento de lo CMS

A inicio de los 2000, la web pasó de ser un solo medio de publicación de información a plataformas sociales e interactivas. En esta etapa nació el concepto de Web 2.0**,** esta versión se caracterizó por lo siguiente.

- La participación de los usuarios comentarios, foros, redes sociales
- Las aplicaciones se desarrollaban de una forma más dinámica y personalizada.

- El uso de AJAX (Asynchronous JavaScript and XML), esta tecnología permitió actualizar partes de una página sin necesidad de recargarla de una forma más rápida y segura.
- Nace las redes sociales como Facebook (2004), Youtube (2005) y Twitter (2006), estas plataformas dieron un giro de 160°, para la interacción de usuario en línea.
- Google lanzó Gmail y Google Maps: Estas aplicaciones utilizaron AJAX para el mejorar la experiencia de usuarios.

Para facilitar el desarrollo de aplicaciones y sitios web sin la necesidad de programar desde cero, surgieron los Sistemas de Gestión de Contenido.

- **WordPress (2003):** Se convirtió en la plataforma líder para la creación de blog y sitios web.

Para el lado del servidor se desarrolló frameworks, esta tecnología evoluciono que se simplifico la programación del lado del servidor.

- **Ruby on Rails (2005):** Este framework popularizó el enfoque de desarrollo rápido
- **Spring Framework (2002):** Esta tecnología potencio el desarrollo de aplicaciones en Java.

1.4. Aplicaciones Web Progresivas, Inteligencia Artificial y Web 3.0

Con el crecimiento gigantesco de la tecnología y de los smartphones, la web tuvo que adaptarse a esta tecnología. Google en el 2015 promueve el concepto de Progressive Web Apps (PWA), este concepto permite que las aplicaciones web funciones como aplicaciones móviles incluso si no se tiene conexión a internet.

1.5. JavaScript y los framewoks frontend

En la actualidad el desarrollo frontend se basa en frameworks avanzados como:

- **React.js (2013):** Esta tecnología fue creada por Facebook para el desarrollo de aplicaciones dinámicas.
- **Angular (2006):** Tecnología desarrollada por Google para la creación de aplicaciones complejas.
- **Vue.js (2014):** Framework progresivo para el desarrollo de interfaces de usuario.

1.6. API's, microservicios y cloud computing

El desarrollo backend ha cambiado con la integración de APIs REST y GraphQL, esto permite que los sistemas se comuniquen de manera más rápida y eficiente.

- Los microservicios de popularizaron, es una arquitectura en la cual las aplicaciones se dividen en pequeños módulos independientes.
- Los servidores tradicionales están siendo reemplazados por el cloud computing (AWS, Google Cloud, Azure).

1.7. Web 3.0 y el futuro

Hoy en día en el desarrollo web se habla de la **Web 3.0**, esta es una evolución que busca descentralizar el control de internet mediante las siguientes tecnologías.

- **Blockchain:** Esta tecnología es usado por criptomonedas y contratos inteligentes.
- **Inteligencia artificial:** La inteligencia artificial se usa en motores de recomendación y asistentes virtuales.
- **Metaverso y Realidad Aumentada:** Estas tecnologías son espacios digitales interactivos.

Desde su implementación en 1989 hasta el día de hoy, la web a cambiado radicalmente en tan poco tiempo. Hemos pasado de simples documentos estáticos a experiencias digitales interactivas y personalizadas. Con la llegada de la inteligencia artificial.

1.8. Arquitectura Web: Cómo Funcionan las Aplicaciones Web

En la actualidad las aplicaciones web forman parte de nuestra vida diaria. Cada vez que ingresamos a un sitio de e-commerce, accedemos a una red social o al utilizar una aplicación bancaria, estamos interactuando con una arquitectura web compleja.

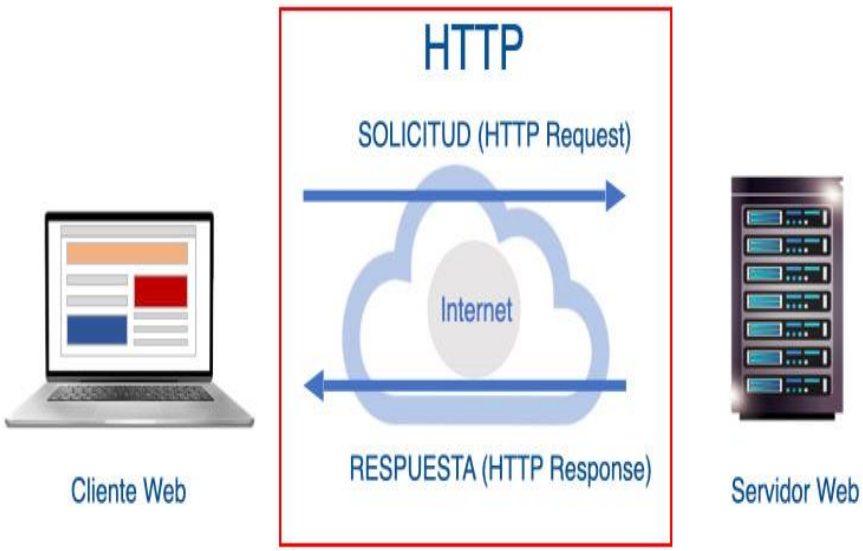
Pero sabes ¿cómo funciona todo este sistema detrás de lo que vemos? Para entender primero es necesario conocer qué es la arquitectura web con sus componentes esenciales.

1.8.1. ¿Qué es la Arquitectura Web?

Para entender cómo funciona la arquitectura web, debemos saber que la web es un diseño estructural eso quiere decir que todos los componentes de una aplicación interactúan entre sí. De esa manera garantizan que la aplicación sea eficiente, escalable y segura como se muestra en la figura 4.

Figura 4.

Comunicación cliente-servidor



Nota. La imagen nos muestra la arquitectura web al realizar una petición del cliente-servidor. *Fuente:* (ciniguez, s.f.)

1.9. Modelo Cliente-Servidor

Cada vez que queremos ingresar a un sitio web, por detrás de lo que el usuario ve, ocurren una serie de procesos invisibles en cuestión de milisegundos. Imagina que queremos ingresar al portal del Instituto Superior Tecnológico Quito para buscar una oferta académica. Este sería el flujo que sucede detrás.

- El cliente hace una solicitud al servidor
    - Cuando el usuario escribe en la url del navegador: www.itq.edu.ec y presionas la tecla “enter”, el cliente que viene a ser el navegador manda una solicitud mediante el protocolo HTTP al servidor
- El servidor recibe y procesa la solicitud.
    - El servidor recibe la petición, accede a la base de datos, ejecuta la lógica de negocios y genera la información necesaria.
- El servidor envía una respuesta la cliente:
    - Una vez encontrada la información solicitada el servidor devuelve una respuesta HTTP al cliente, esta información puede contener páginas HTML, JSP, datos en formato

JSON, imágenes, videos e incluso mensajes de error si la información no fue encontrada.

- El cliente interpreta la respuesta y se la muestra al usuario:
    - Una vez que el cliente obtiene la respuesta el usuario puede ver el contenido solicitado en el formato que se lo pidió.

Este tipo de comunicación se ha mantenido, como estándar durante décadas y hoy en día es base de todas las aplicaciones web modernas, simplemente ha tenido un cambio muy significante como se puede observar en la figura 4, la petición se realiza por el protocolo HTTP (HyperText Tranfer Protocol) y la respuesta por el mismo protocolo. Este protocolo realiza la petición de una forma insegura eso quiere decir que toda información solicitada y de respuesta viaja de una forma visible.

Para solucionar este problema se implementó un protocolo seguro, el cual encripta la información durante todo el ciclo de vida de petición de un recurso al servidor este protocolo es el siguiente HTTPS (HyperText Transfer Protocol Security), y de esta forma la información viaja oculta, y es más difícil para los ciberdelincuentes descifrar el contenido que viaja por la internet.

1.10. Componentes Claves de la Arquitectura Web.

Para desarrollar una aplicación web y que funcione de una manera eficaz, es necesario que trabaje en conjunto con distintas tecnologías. A continuación, exploraremos cada una de ellas:

1.10.1. Frontend UI o Cliente

Esta es una de las capas que interactúa directamente con los usuarios. Esta es la responsable de la experiencia visual.

Las tres tecnologías que vamos a usar en esta capa son:

- **HTML (HyperText Markup Languaje):** Esta tecnología no es un lenguaje de programación, sino un lenguaje de marcado y etiquetas. HTML es el esqueleto de nuestra aplicación web.
- **CSS (Cascading Style Sheets):** Esta tecnología es un lenguaje de estilos nos permite darle estilos y hacer visualmente atractiva a las aplicaciones web.
- **JavaScript:** Es un lenguaje de programación de alto nivel, en sus inicios JavaScript estaba solo orientado al frontend, haciendo a nuestra página web más interactiva y dinámica. Con el pasar del tiempo JavaScript fue evolucionando y hoy en día JavaScript también trabaja del lado del servidor.

Hoy en día, el desarrollo frontend, ha evolucionado con el uso de frameworks y bibliotecas como:

- **React.js:** Esta biblioteca fue desarrollada por Facebook para crear interfaces de usuarios más interactivas y dinámicas.
- **Angular:** Es un framework de diseño de aplicaciones y plataforma de desarrollo para crear aplicaciones de una sola página eficientes y sofisticadas.
- **Vue.js:** Es un marco de JavaScript progresivo de código abierto para crear interfaces de usuario UI y aplicaciones de una sola página.

Backend

En una aplicación web la lógica de programación es gestionada por el backend es toda la parte oculta de la aplicación.

El backend se encarga de las siguientes responsabilidades:

- El backen se encarga de procesar las peticiones que realiza el cliente.
- Interactuar con bases de datos para obtener y almacenar información.
- Ejecutar la lógica de negocio (ejemplo, calcula el promedio de calificaciones de un curso)

Para el backend existen distintas tecnologías que se pueden usar.

- Java (Spring, Jakarta EE, Servlets, JSP)
- Python (Django, Flask)
- Node.js (Express.js)
- PHP (Laravel, Synfony)

En este libro nos enfocares en

1.10.2. Base de Datos

Una de las estructuras de datos más utilizadas en estos días son las bases de datos relacionales y no relacionales, estas estructuras de datos almacenan una gran cantidad de información. Pueden contener datos de usuarios, estudiantes, productos, calificaciones, etc.

Como ya se mencionó existen dos tipos de bases de datos:

Base de datos relacionales (SQL): Las bases de datos relacionadas organizan la información con relaciones definidas.

Ejemplo:

- MySQL
- PostgreSQL
- SQL

Bases de datos NoSQL: Este tipo de base de datos guarda la información distintos tipos como: colecciones, documentos, grafos, clave valor entre otros.

Ejemplos:

- MongoDB.
- Firebase.
- Cassandra.

1.11. Servidores Web y Servidores de Aplicaciones

Para que una aplicación web este en línea y funcional, esta necesita ejecutarse en un servidor web ya sea local o web. Existen dos tipos de servidores.

- **Servidor web:** Este servidor maneja peticiones Http, y lo que responde son archivos estáticos como html, css y JavaSacript.
- **Servidor de aplicaciones:** Este servidor te permite ejecutar código del lado del servidor y permite gestionar la lógica de la aplicación.

  Ejemplos:

  - Apache Tomcat
  - WildFly
  - Node.js

1.12. Herramientas esenciales para el Desarrollo Web con Java.

Para desarrollar aplicaciones en Java se requiere de un sin número de herramientas que te ayuden con la escritura, pruebas, ejecución y mantenimiento del código. Empezando con entornos de desarrollo integrado (IDE), hasta servidores de aplicaciones, cada una de estas herramientas cumplen un rol clave en el desarrollo de aplicaciones robustas escalables y seguras.

1.12.1. Entornos de Desarrollo Integrados (IDE) para Java

Como se mencionó anteriormente para desarrollar aplicaciones Web en Java se necesita un IDE (Integrated Development Environment) que son aplicaciones que proporcionan un entorno unificado para escribir, depurar y ejecutar código Java. Un buen IDE nos ayuda a mejorar la productividad y facilita la gestión de proyectos modernos.

Para Java tenemos algunos IDE de desarrollo como:

- **Eclipse IDE:** Este IDE de desarrollo es uno de los más populares en el entorno Java, soporto múltiples lenguajes de programación y tiene un entorno de plugins que lo hace muy flexible.

Figura 5.

Eclipse IDE

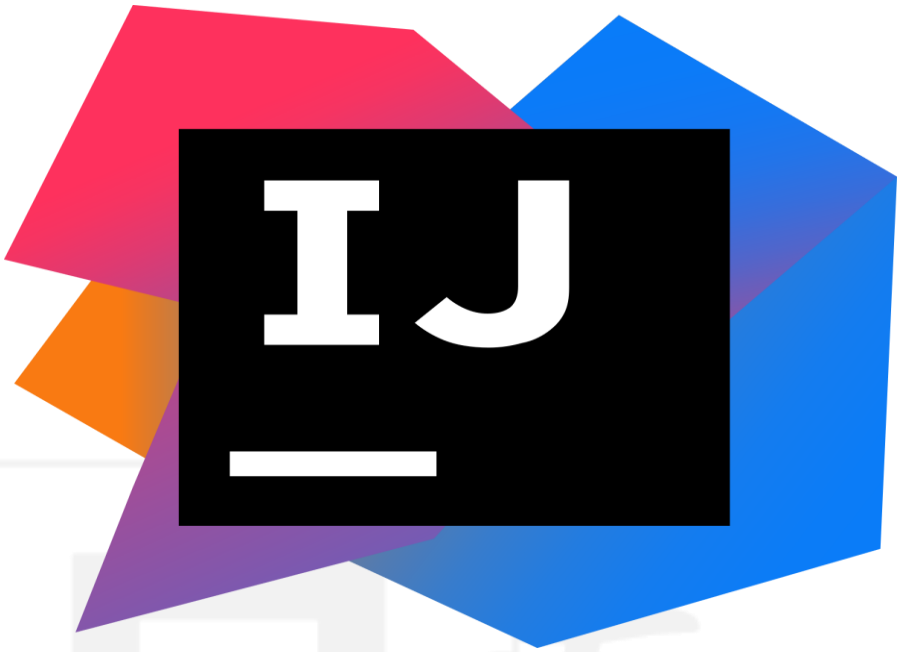
Nota. La imagen nos muestra el logo del IDE Eclipse. *Fuente:* (creativecommons, s.f.)

- **IntelliJ IDEA:** Hoy en día uno de los IDE's más avanzados para el desarrollo en Java, este tiene características de autocompletado con inteligencia artificial y un análisis de código en tipo real. Este IDE dispone dos versiones la versión gratuita (Community Edition) que tiene funcionalidades limitadas, la versión pagada (Ultimate Edition) esta versión cuenta con todas las funcionalidades necesarias para desarrollar aplicaciones en Java.

  Este IDE integra de forma nativa Spring Boot, Hibernate y Maven, soporta el desarrollo de APIs REST y de microservicios con facilidad.

Figura 6.

IntelliJ IDEA



Nota. La imagen representa el icono del IDE de desarrollo. *Fuente:* (helenjoscott, s.f.)

- **NetBeans IDE:** NetBeans es un IDE de código abierto que es mantenido por la comunidad de Apache, es fácil de usar y tiene soporte para proyectos Jakarta EE, Servlets y JSP entre otras tecnologías.

1.13. Lenguajes y Tecnologías Fundamentales.

Cuando se desarrolla aplicaciones web en Java, se necesita involucrar múltiples tecnologías que trabajen en conjunto de esta manera se puede crear aplicaciones, funcionales, interactivas y sobre todo escalables. Para construir e implementar un sitio o aplicación en Java, es necesario entender los lenguajes y herramientas que conforman todo este ecosistema, del lado del cliente como el frontend y del lado del servidor como el backend.

RESUMEN DEL CAPÍTULO 1

En el mundo de la programación el desarrollo de software es una rama fundamental en la informática hoy en día. Su propósito es el desarrollo de sitios y aplicaciones web que permiten interactuar entre usuarios y sistemas informáticos, utilizando e implementando tecnologías especializadas para el frontend y del lado del servidor o backend.

Java es uno de los lenguajes de programación más utilizado en el lado del servidor, gracias a su portabilidad, escalabilidad y seguridad. Desde sus inicios, Java a sido clave para la creación de aplicaciones empresariales y servicios de web robustos.

- **Historia y Evolución del Desarrollo Web:** La web fue creada en 1989 por Tim Berners-Lee en CERN, su primera aplicación fue un sistema de hipertexto basado en documentos HTML enlazados entre sí, el primer navegador solo permitió ver páginas estáticas. En 1995 se integra JavaScript que permite crear páginas web interactivas, Java incorpora los Servlets y los JSP, se comienza a implementar base de datos relacionales par aguardar información. Se popularizo el uso de AJAX esta tecnología permite actualizar información sin necesidad de recargar la página. Empieza el auge de las redes sociales y cambia totalmente el método de comunicación habitual. La evolución de la tecnología permite dividir las aplicaciones en pequeños servicios independientes, el desarrollo en la nube y contenedores permite desplegar aplicaciones en múltiples entornos.
- **Arquitectura Web y funcionamiento de las Aplicaciones Web:** La arquitectura web define cómo se estructura y se comunican los componentes de una aplicación web. La mayoría de las aplicaciones siguen un modelo **cliente-servidor,** donde el cliente como navegador manda una solicitud al servidor y el se encarga de buscar el recuso y devolverlo en forma de HTML, JSP, PDF entre otros.
- **Herramientas Esenciales para el Desarrollo Web con Java:** Los IDE's de desarrollo son herramientas que ayudan al desarrollador a escribir código limpio entre los cuales tenemos. Eclipse, IntelliJ IDEA, NetBeans. Además de los IDE's de desarrollo se necesita de servidores web y de aplicaciones para levantar los servicios de las aplicaciones entre los más populares tenemos. Apache Tomcat, WildFly, GlassFish. Para el backend se conecta a una base de datos la cual almacena información. Las opciones más utilizadas en Java incluyen. PostgreSQL, MySQL, MongoDB, JPA.

El desarrollo web con Java es un campo amplio y en constaten evolución. Este capítulo ha proporcionado una base sólida sobre:

- La historia y evolución del desarrollo web.

- La arquitectura web y su funcionamiento.
- Herramientas y tecnologías esenciales en el desarrollo con Java.

Aprender estos conceptos fundamentales permitirán al lector desarrollar aplicaciones modernas, escalables y segura.

CAPITULO 2

2. INTRODUCCIÓN A JAKARTA PARA LA WEB

Durante toda su historia el desarrollo web en Java a tenido varias evoluciones desde sus inicios. Jakarta EE es la versión más reciente y avanzada de Java EE, esta nueva versión ofrece un conjunto de herramientas y estándares que facilitan el desarrollo de aplicaciones escalables, seguras y eficientes.

En este segundo capítulo exploraremos su historia características principales, arquitectura y cómo comenzar a desarrollar aplicaciones con Jakarta EE.

2.1. ¿Qué es Jakarta EE?

2.1.1. Historia y evolución en Jakarta EE

Jakarta es una plataforma para el desarrollo de aplicaciones empresariales en Java, esta plataforma está diseñada para crear aplicaciones web escalables, seguras y eficientes. Esta plataforma era manejada por Oracle y se llamaba Java EE (java Enterprise Edition) hasta su última versión 8, pero en 2017, Oracle decide transferir sus derechos de desarrollo a Eclipse Foundation, en el cual su nombre es cambiado a Jakarta EE en 2018.

El cambio fue radical ya que no solo se cambió el nombre, al ser Eclipse Foundation de código abierto, esto permitió que la comunidad de desarrolladores tuviera mayor participación en la evolución de la plataforma. Desde ese momento Jakarta EE ha seguido mejorando con versiones más modernas y adaptadas a nuevas tecnologías y arquitecturas como microservicios y cloud computing.

2.1.2. Diferencias entre Jakarta EE y Java EE.

Las diferencias entre estas dos tecnologías son muy marcadas:

- Java EE era administrado por Oracle, mientras que Jakarta EE es gestionado por la empresa Eclipse Foundation.
- En Jakarta EE, carias de las especificaciones han evolucionado con mejores prácticas y sobre todo mayor soporte de arquitecturas modernas.
- Se han eliminado dependencias obsoletas y se ha mejorado la integración con tecnologías como Docker y kubernet.

2.1.3. ¿Por qué Jakarta es importante en el desarrollo web empresarial?

Java en estos días sigue siendo uno de los lenguajes más utilizados en el desarrollo de software, con su cambio de nombre Jakarta EE, no se ha quedado atrás y sigue siendo fundamental para el desarrollo de aplicaciones empresariales porque:

- Es un estándar abierto quiere decir que facilita la migración entre distintos servidores de aplicaciones.
- Hoy en día con el avance de la tecnología soporta arquitecturas modernas, se integra con microservicios, contenedores (Docker, kubernetes) y plataformas cloud.
- Incluye herramientas empresariales robustas, como JPA (persistencias), CDI (Inyecciones de dependencias) y JAX-RS (APIs RESTful).
- Hoy en día la seguridad es muy importante en nuestras aplicaciones es por eso que Jakarta EE, garantiza seguridad y escalabilidad, en todas sus aplicaciones, usado en banca, telecomunicaciones y comerci0 electrónico.

2.1.4. Componentes Principales de Jakarta EE

Como sabes para desarrollar aplicaciones empresariales se necesitar tres capas en las que se utilizan diferentes tecnologías.

- Capa de Presentación (Frontend)
  - JSP (JavaServer Pages)
  - JSF (JavaServer Faces)
  - Facelets (Plantillas avanzadas para JSF)
- Capa de Negocios (Lógica de la apliación)
  - Servlets
  - Enterprise JavaBeans (EJB)
  - CDI (Context and Dependency Injection)
- Capa de Persistencia (base de datos y comunicación con otras aplicaciones)
  - JPA (Java Persistence API)
  - JAX-RS (APIs RESTful)
  - JMS (Java Message Service)

Servlets

Un servlet es una clase en Java que se ejecuta en un servidor de aplicaciones para manejar solicitudes HTTP. A diferencia de una clase normal de java un Servlet siempre se va a extender una clase llamada **HttpServlet.**

Figura 7.

Ejemplo de un Servlet

Nota. La imagen nos muestra un ejemplo de un servlet en Java. *Fuente:* Los investigadores.

JavaServer Pages (JSP)

En el lado del cliente, los JSP permite generar páginas HTML dinámicas con código Java embebido.

Figura 8.

Ejemplo JSP

Nota. La imagen nos muestra el código de un JavaServer Pages. *Fuente:* Los investigadores

2.1.5. Arquitectura de una Aplicación Jakarta EE

Modelo Cliente-Servidor

El modelo cliente-servidor, también conocido como “principio cliente-servidor”, es un modelo de comunicación que permite la distribución de tareas dentro una red de computadoras.

Un servidor es un hardware que proporciona los recursos necesarios para otros ordenadores o programas, pero un servidor también puede ser un programa informático que se comunican con los clientes. Un servidor acepta las peticiones, el servidor las procesa y proporciona la respuesta solicitada. También existen diferentes tipos de cliente. Un computador o un programa informático se comunica con el servidor, envia solicitudes y recibe respuestas del servidor. En cuanto al modelo cliente-servidor, representa la interacción entre el servidor y el cliente.

En una aplicación de Jakarta EE, el cliente (navegador o aplicación móvil) realiza solicitudes al servidor, que procesa la lógica y devuelve una respuesta.

Flujo de una solicitud en Jakarta EE

- El usuario accede a un Servlet o una AAPI REST en el servidor.
- La aplicación porcesa la solicitud
- Si es necesario, accede a una base de datos.
- Se genera una respuesta en HTML, JSON o XML y se envía al cliente.

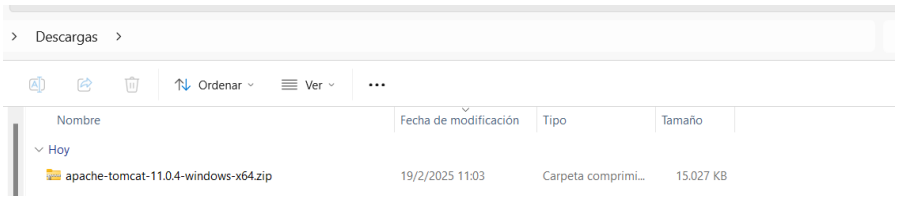
2.1.6. Instalación y Configuración del Entorno Jakarta EE

Antes de empezar a desarrollar aplicaciones en Jakarta EE se tiene que configurar un entorno de trabajo, para el libro vamos a utilizar el IDE de desarrollo IntelliJ IDEA.

- Descargamos el JDK (Java Development Kit), desde la página oficial. https://www.oracle.com/java/technologies/downloads/
- Descargamos e instalamos el IDE de desarrollo IntelliJ IDEA. http://jetbrains.com/es-es/idea/download/?section=windows
- Para el propósito de nuestro libro descargamos a la versión Ultimate pero le activamos de modo estudiante esto nos permitirá usar todos los beneficios y herramientas del IDE de desarrollo.
- Una vez descargado eh instalado el IDE, vamos a descargar e instalar un servidor de aplicaciones en este caso utilizaremos el servidor apache Tomcat la última versión. https://tomcat.apache.org/download-11.cgi.
- Una vez descargado se nos descarga un archivo zip

Figura 9.

Apache tomcat



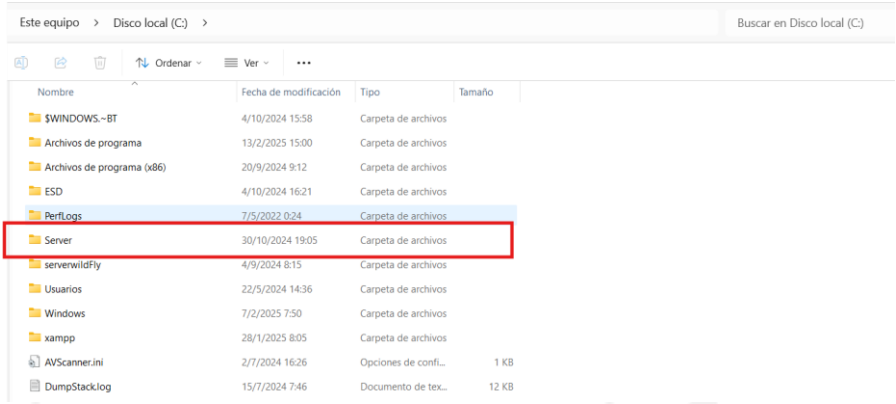
Nota. La imagen nos muestra el archivo apache-tomcat descargado en formato zip. *Fuente:* Los investigadores.

- Descomprimimos el archivo.

- Creamos una carpeta el disco C: de nombre Server. El nombre puede ser cualquier nombre, pero para fines del libro le pondremos Server.

Figura 10.

Carpeta Server

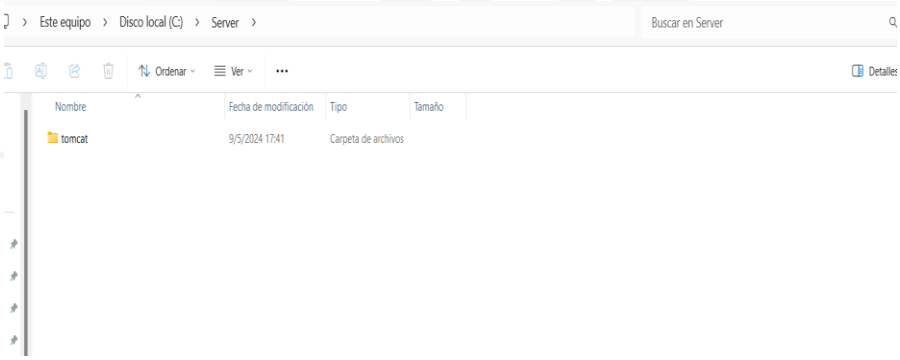


Nota. La imagen nos muestra donde esta creado el archivo Server que luego se guardara el archivo descomprimido que se descargó. *Fuente:* Los investigadores.

- Una vez creada la carpeta movemos el archivo que hemos descomprimido en el paso b.

Figura 11.

Carpeta tomcat.

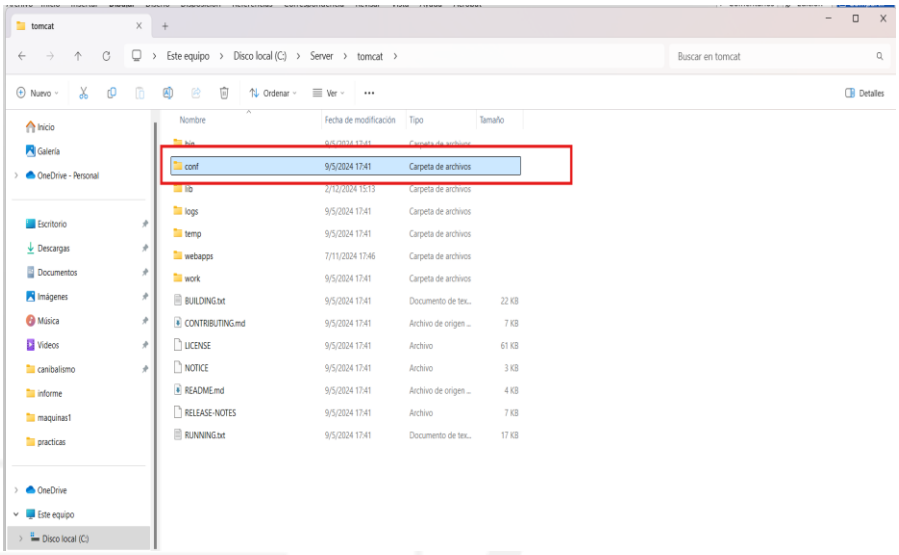


Nota. En la imagen se muestra el archivo descomprimido con el nombre de tomcat, donde se hará algunas configuraciones. *Fuente:* Los investigadores.

- Abrimos la carpeta, y dentro buscamos el archivo config.

Figura 12.

Carpeta config

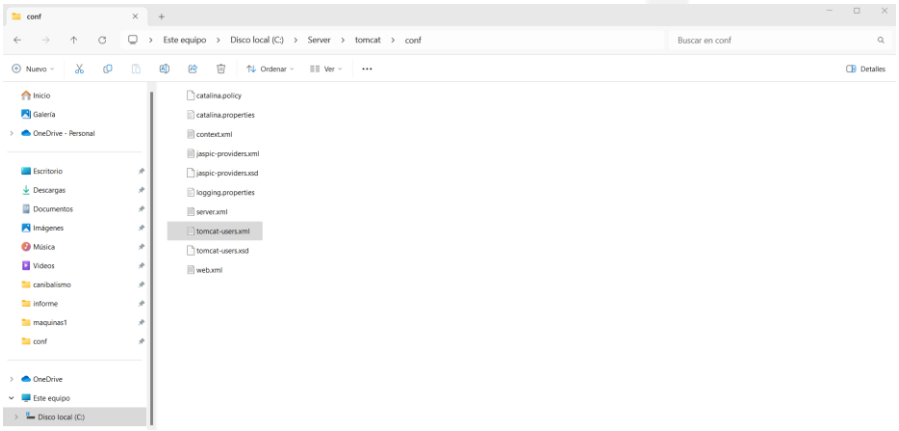


Nota. La imagen nos muestra la carpeta config donde configuraremos las variables necesarias para ejecutar las aplicaciones en el servidor. *Fuente:* Los investigadores.

- Dentro de la carpeta config, buscamos el archivo tomcat-users.xml este archivo de configuración en formato XML que define los usuarios y roles que pueden acceder a las funcionalidades administrativas de Tomcat, como el Manager APP y el Host Manager. Estos roles permiten realizar tareas como desplegar, iniciar, detener y eliminar aplicaciones web, así como gestionar hosts virtuales.

Figura 13.

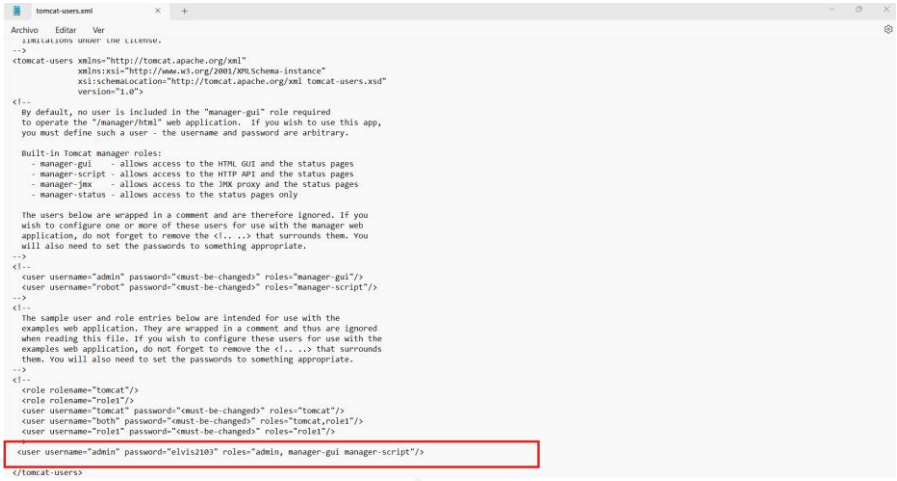
Configuración de usuarios



Nota. La imagen muestra le archivo donde se configurará el usuario y contraseña. *Fuente:* Los investigadores.

Figura 14.

Configuración Usuario



Nota. La imagen muestra la configuración de usuarios y contraseñas. *Fuente:* Los investigadores.

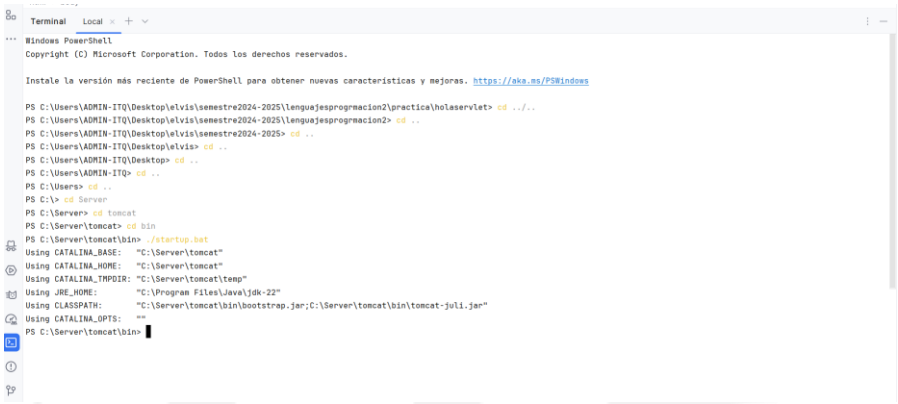
Como se muestra en la Figura 14 se está configurado usuario y contraseña, que a continuación se explicara cada uno de los argumentos que se está configurando.

- **Roles:** Los roles definen los permisos que tiene un usuario. Algunos de los roles comunes son:
    - **Manager-gui:** Este permiso nos permite acceder a la interfaz del Manager App, donde se puede gestionar aplicaciones web (desplegar, iniciar, detener, etc).
    - **Admin-gui:** Este argumento permite acceder a la interfaz gráfica del **Host Manager,** donde puedes gestionar hosts virtuales.
    - **Manager-script:** Permite acceder a las funcionalidades del Manager a través de scripts.
    - **Manager-jmx:** Permite acceder a las funcionalidades de monitoreo y gestión JMX (Java Management Extensions)
- **Usuarios <users):** Este argumento nos permite acceder a las cuentas de usuario de Tomcat. Cada usuario tiene los siguientes parámetros.
    - **Username:** Este parámetro me permite configurar el nombre de usuario del servidor.
    - **Password:** Este parámetro me permite ingresar la contraseña del usuario, este parámetro se ingresa en texto plano por lo que se debe resguardar y proteger este archivo.
    - **Roles:** Este parámetro permite asignar los distintos roles que va a tener el usuario, un usuario puede tener varios roles y esto se les va separando por una coma.

Una vez configurado los parámetros necesarios vamos a levantar el servido para eso abrimos nuestro IDE de desarrollo y desplegamos una terminal e ingresamos a la dirección donde se encuentra el servidor Tomcat en nuestro caso se encuentra en el disco C:, la carpeta Server, tomcat, bin C:\Server\tomcat\bin una vez dentro buscamos el archivo startup.bat.

Figura 15.

Ejecución de tomcat

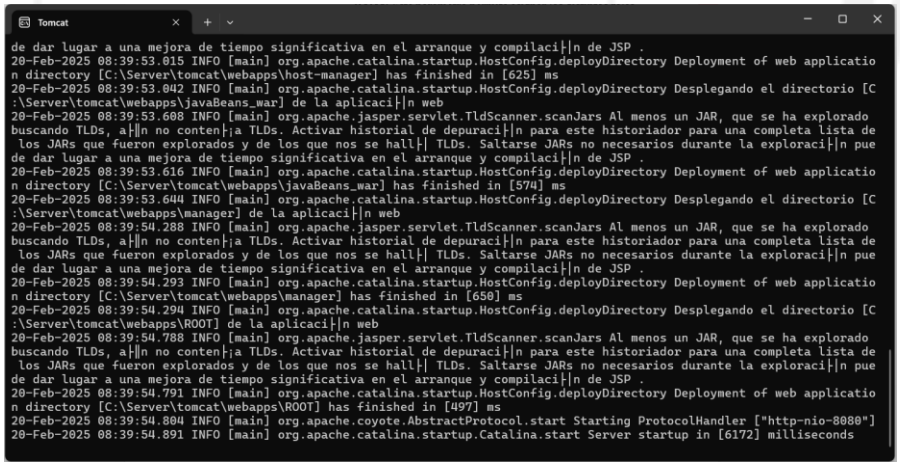


Nota. En la imagen se muestra como levantar los servicios del servidor tomcat. *Fuente:* Los investigadores.

Si todo está bien configurado se abrirá una ventana de una terminal cmd donde empieza a levantarse todos los servicios del servidor y no debería dar ningún error.

Figura 16.

Servicios levantados

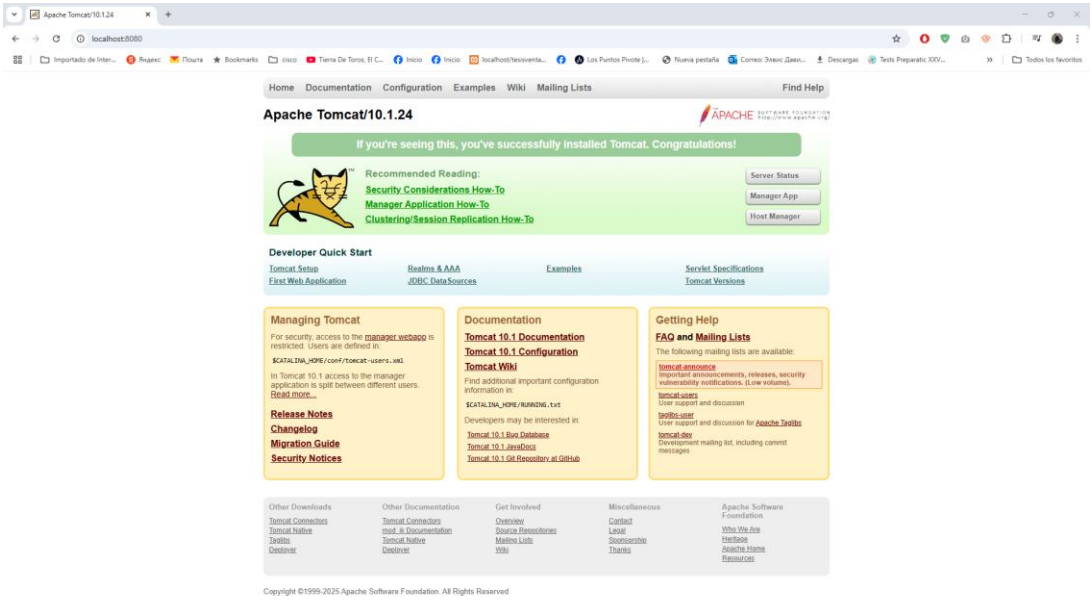


Nota. La imagen muestra cómo se van levantando los servicios del servidor tomcat. *Fuente:* Los investigadores.

Para verificar que todo este corriendo correctamente, ingresamos a un navegador y tipeamos la siguiente dirección http://localhost:8080/, y nos muestra la siguiente pantalla como se muestra en la figura 17.

Figura 17.

Servidor local



Nota. La imagen nos muestra el servidor tomcat que se está ejecutando correctamente. *Fuente:* Los investigadores.

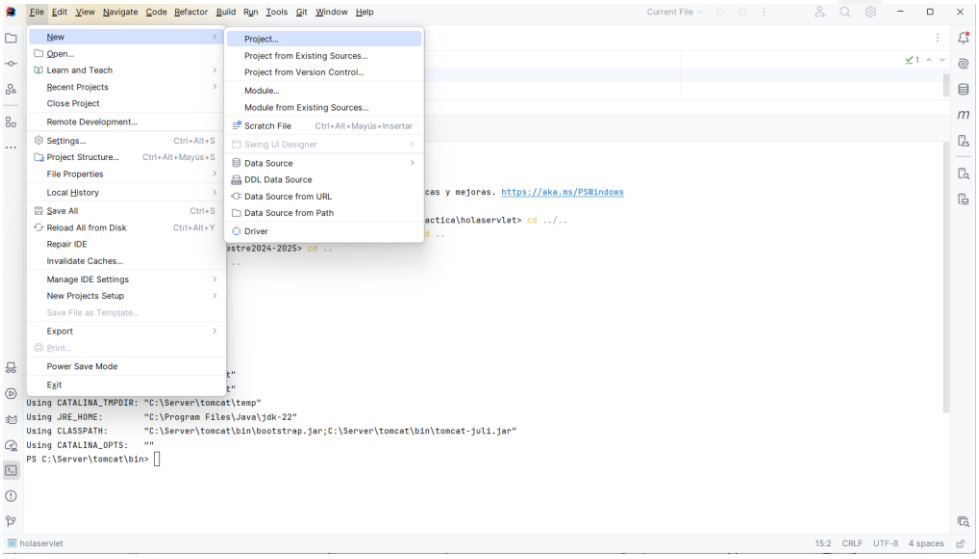
2.1.7. Estructura un proyecto básico en Jakarta EE

Para crear un proyecto Web en Jakarta EE abrimos el IDE de desarrollo IntelliJ IDEA.

- Vamos a File, New, Project

Figura 18.

Creando un Proyecto

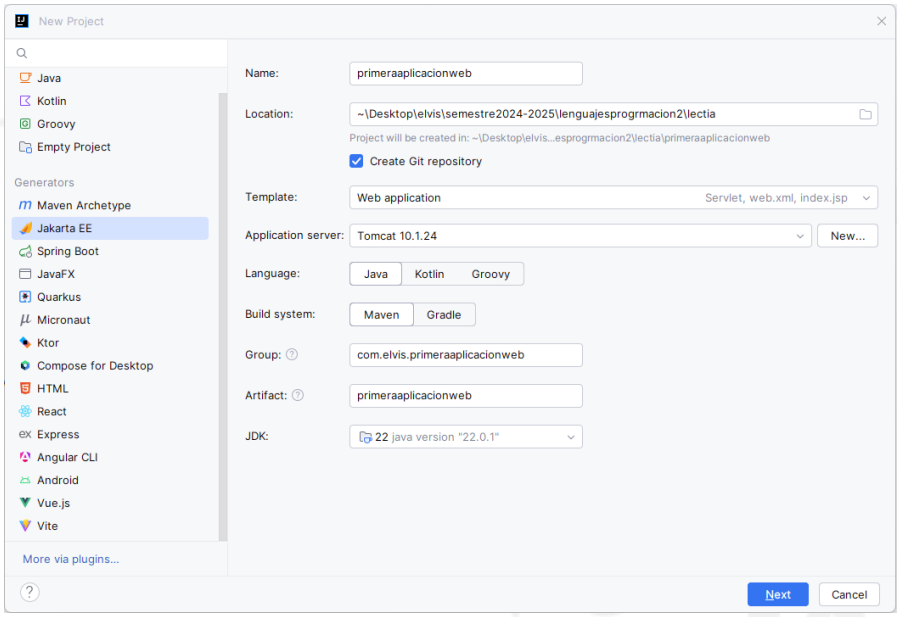


Nota. La imagen muestra como empezar a crear un proyecto web en Jakarta. *Fuente:* Los investigadores.

- Ponemos el nombre del proyecto, la ubicación donde se guarda el proyecto, tipo de proyecto en este caso elegimos web Application, escogemos el servidor de aplicaciones que utilizamos en este caso Tomcat elegimos donde se encuentra el servidor, seleccionamos el lenguaje de programación en este caso Java, seleccionamos el sistemas de construcción que será Maven, ponemos el grupo al que pertenece el proyecto, el artifact del proyecto y la versión del JDK para nuestro libro utilizaremos la última versión y presionamos el botón siguiente.

Figura 19.

Creando Primera Aplicación

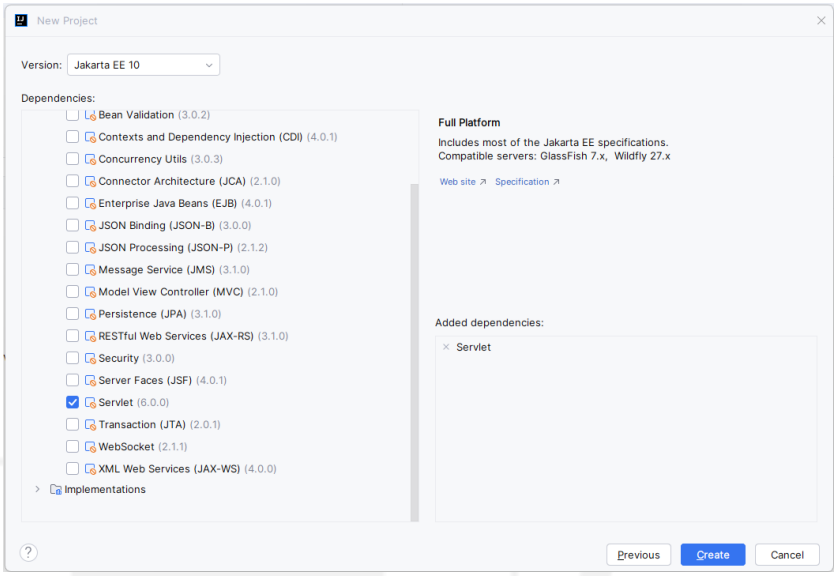


Nota. La imagen muestra cómo crear un proyecto en Jakarta EE. *Fuente:* Los investigadores.

- Elegimos la versión de Jakarta que será la versión 10 y Dependencies, marcamos la opción Servlet y le damos al botón Create.

Figura 20.

Versión del Proyecto

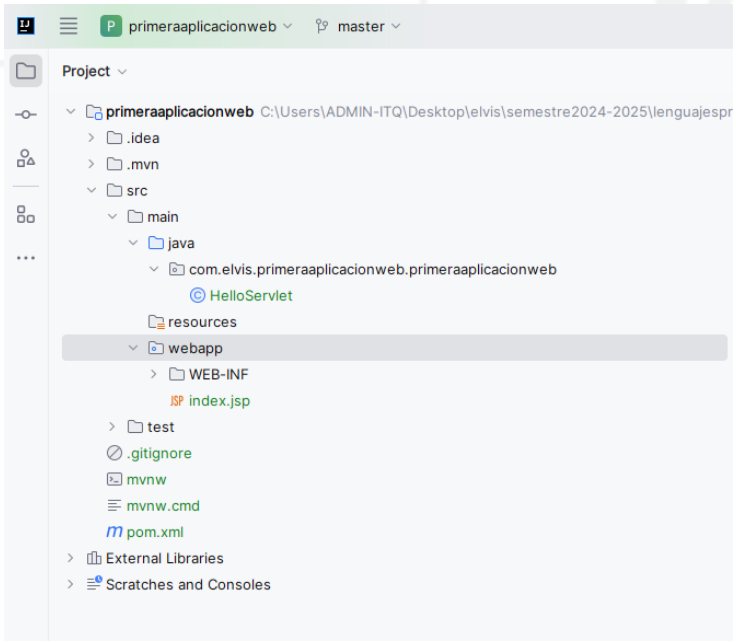


Nota. La imagen muestra como elegir la versión de Jakarta EE y las dependencias a utilizar. *Fuente:* Los investigadores.

Una vez configurado las dependencias de nuestro proyecto, este tendrá la siguiente estructura.

Figura 21.

Estructura del Proyecto



Nota. La imagen muestra cual es la estructura de un proyecto creado en Jakarta e IntelliJ IDEA, teniendo varias carpetas. *Fuente:* Los investigadores.

Como se muestra en figura 21, el proyecto tiene distintas carpetas:

- **Src:** Esta carpeta contiene coda la codificación Java
- **Webapp:** Aquí vamos a realizar toda la codificación de la parte del frontend.

RESUMEN DEL CAPÍTULO 2

Jakarta EE es la evolución de Java EE (Java Enterprise Edition) y es una plataforma estándar para el desarrollo de aplicaciones empresariales en Java. Originalmente fue desarrollado por Sun Microsystems y luego fue adquirido por Oracle, en el 2017 fue transferido a Eclipse Foundation, el cual cambia su nombre a Jakarta EE.

Jakarta EE proporciona un conjunto de especificaciones y herramientas para desarrolla aplicaciones web escalables, seguras y eficientes, que se utilizan principalmente en el ámbito empresarial.

Jakarta EE incluye diversas tecnologías organizadas en tres capas principales, estas capas son:

- Capa de presentación
- Capa de Negocio
- Capa de Persistencia y Comunicación

Las aplicaciones Jakarta EE siguen el modelo cliente-servidor, en el que los clientes (navegadores o apps móviles) envían solicitudes HTTP a un servidor, que responden con distintos tipos de archivos como: pdf, html, jsp, wav, mp3, etc.

El ciclo de vida de una aplicación en Jakarta es el siguiente:

- El usuario accede a una URL que es ejecutado por un Servlet o una API REST.
- El servidor procesa esa solicitud
- Si es necesario se consulta a una BBDD.
- Se genera una respuesta en HTML, JSON o XML u otra y se devuelve al cliente.

Ejercicios

2.1. ¿Qué es Jakarta EE y cuál es su origen?

2.2. ¿Cuál fue la razón por la que Java EE paso a llamarse Jakarta EE?

2.3. ¿Cuáles son las principales capas de una aplicación Jakarta EE?

2.4. ¿Qué es un Servlet y cuál es su función en Jakrata EE?

2.5. ¿Cómo se diferencia JSP de un Servlet en términos de funcionalidad?

2.6. Escribe un código de un Servlet básico en Jakarta EE que devuelve “Hola Mundo”.

2.7. ¿Cuál es la estructura típica de un proyecto Jakarta EE?

2.8. ¿Cómo se instala y configura un entorno de desarrollo Jakarta EE?

.

CAPITULO 3

3. SERVLETS

Un Servlet es una clase normal de Java que permite procesar peticiones Web por medio del protocolo HTTP.

Cuando un cliente envía una petición a un servidor web, el Servlet se encarga de procesarla. Esta solicitud puede incluir parámetros, ya sea provenientes de un enlace o de un formulario HTML, estableciendo así la comunicación entre el cliente y el servidor.

Tras recibir la petición y recuperar los parámetros, el Servlet realiza el procesamiento necesario, que puede incluir la consulta de una base de datos, la ejecución de cálculos en el servidor o cualquier otra lógica de negocio. Finalmente, el resultado de este procesamiento se envía de vuelta al cliente en forma de respuesta dinámica completando así el flujo habitual de interacción entre ambos.

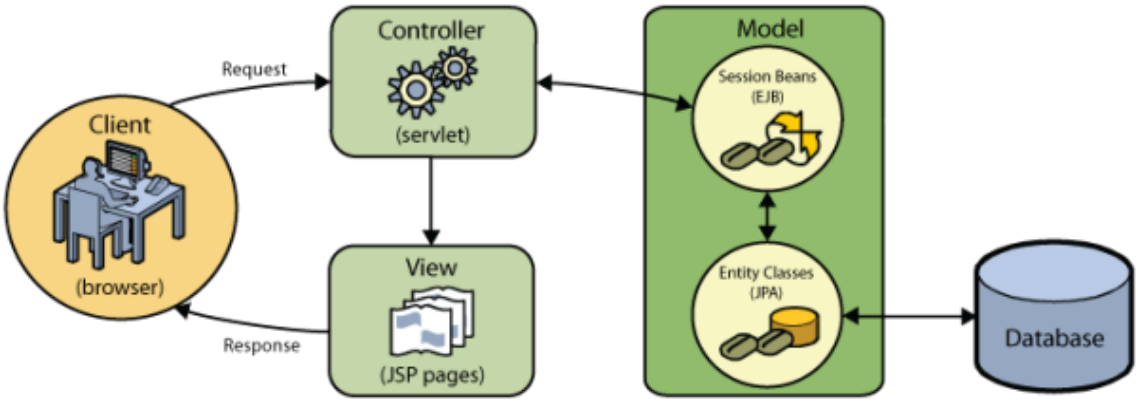
Un Servlet permite:

- Recibir y procesar información enviada por el cliente web, ya sea a través de parámetros en la solicitud HTTP, como datos de formularios o parámetros en la URL.
- Generar y enviar respuestas al cliente, devolviendo contenido dinámico en diferentes formatos, como HTML, JSON, archivos PDF, audio o video, dependiendo de las necesidades de la aplicación.

3.1 Funciones de un Servlet

Figura 22.

Función del Servlet



Nota. La imagen muestra la función de un Servlet que es un viapass entre el cliente y el servidor. *Fuente*: (elpesodeloslunes, s.f.)

Un Servlet cumple un papel fundamental en las aplicaciones web, especialmente en la gestión de flujo de control dentro de la arquitectura MVC (Modelo Vista Controlador). Su principal función es actuar como controlador, dirigiendo el procesamiento de solicitudes y la navegación dentro de la aplicación.

Un Servlet contiene código Java y puede generar código HTML de forma dinámica. Sin embargo, escribir código HTML dentro de un Servlet no es una práctica recomendada, ya que dificulta la organización y el mantenimiento del código. Por esta razón, se creó la tecnología JSP (JavaServer Pages), que permite generar HTML de manera más sencilla y estructurada.

3.2 El Rol del Servlet en el Modelo Vista Controlador (MVC)

El patrón MVC divide la aplicación web en tres componentes principales, separando la lógica de negocio de la presentación para mejorar la organización del código.

- **Vista (View):** Es la interfaz gráfica con la que interactúa el usuario. Puede ser una página JSP o HTML estático.
- **Controlador (Controller-Servlet):** Recibe las solicitudes del usuario, recupera los parámetros enviados, los procesa y determina que información se debe mostrar.
- **Modelo (Model):** Representa los datos y la lógica de negocio de la aplicación. Generalmente, se implementa utilizando JavaBeans o clases Java que gestionan la información que necesita la aplicación.

3.3 Flujo de una Aplicación MVC con Servlets

Cuando se desarrolla una aplicación web en Java como ya se mencionó el Servlet toma el papel del controlador y sigue el siguiente flujo.

- El usuario accede a una página web y visualiza información desde un JSP o una página HTML estática.
- Realiza una acción (como enviar un formulario), lo que genera una solicitud HTTP
- El Servlet como controlador recibe la solicitud, procesa los parámetros y decide la lógica a seguir.
- El Servlet se comunica con el modelo (JavaBeans o clases de negocio) para obtener información de datos o procesar datos.
- Una vez procesada la solicitud, Servlet decide que JSP debe mostrar la respuesta.
- El JSP genera la salida en formato HTML y la envía al navegador del usuario.

Este patrón de diseño permite una mejor organización del código y facilita el mantenimiento de la aplicación.

El Servlet se encarga de administrar el flujo de la aplicación, procesar peticiones y determinar que vista mostrara al usuario.

3.4 Métodos HTTP y Procesamiento con Servlets

El protocolo HTTP (HyperText Transfer Protocol) permite la comunicación entre el cliente y el servidor este protocolo, está compuesto por 8 métodos, los cuales son:

- **Options.** Este método solicita información sobre los métodos soportados por el servidor.
- **Get:** Este método permite enviar y recuperar información enviada al servidor mediante la URL eso quiere decir que los datos viajan de forma visible,
- **Head:** Este método es similar al Get este método me permite obtener información de los encabezados de la respuesta.
- **Post:** Este método envía información al servidor, pero a diferencia del método Get este lo hace ocultando la información por el cuerpo de la solicitud.
- **Put:** Este método sube un recurso al servidor o lo actualiza,
- **Delete:** Elimina un recurso en el Servidor.
- **Trace:** Este método prueba la ruta que sigue una solicitud hasta el servidor.
- **Connect:** Este método permite la conexión con un servidor proxy.

3.5 Uso de Métodos GET y POST en Servlet

Para el manejo del libro vamos a utilizar dos métodos importantes, en el desarrollo web con Jakarta EE, los métodos más utilizados son GET y POST, que permiten enviar y recibir información de los clientes.

3.5.1 Diferencia entre GET y POST

Tabla 1.

Diferencia GET y POST

Características	GET	POST
Parámetros en URL	Información Visible	Información oculta

Seguridad	Menos seguro	Más seguro
Tamaño de datos	Limitado (Depende del navegador)	Mayor capacidad de envió de datos
Cache	Puede almacenase en memoria cache	No se almacena en cache
Uso recomendado	Obtener datos	Enviar datos sensibles (formularios autenticación)

Nota. La tabla nos muestra la diferencia entre el método GET y el POST al enviar información desde el cliente al servidor. *Fuente:* Los investigadores.

Al crear una clase de tipo Servlet esta clase se debe extender de la clase jakarta.servlet.http.HttPServlet, y esta clase que se crea a su vez debe sobrescribir los métodos doGet o doPost según la petición que sea enviada por parte del cliente.

Cuando queramos procesar datos con el método GET al crear el Servlet el método a sobrescribir es el método doGet, y cuando se procese datos con el método POST el método a sobrescribir es el método doPost del Servlet.

Ejemplo de Servlet con GET y POST

Cuando se crea un Servlet en Jakarta, este como ya se explicó se debe extender de la clase HttServlet, la cual define métodos como doGet() y doPost() para procesar solicitudes HTTP.

Figura 23.

Formulario petición GET

```
<%--Método para manejar solicitudes GET--%>
<form action="miServlet" method="get">
    <div>
        <label for="nombre">Ingrese el nombre</label>
        <input type="text" id="nombre" name="nombre">

    </div>

    <div>
        <input type="submit" value="Enviar GET">
    </div>


</form>
```

Nota. El código muestra un formulario realizando una petición por el método GET al servidor. *Fuente:* Los investigadores.

Figura 24.

Petición GET

```
package com.elvis.primeraaplicacionweb.primeraaplicacionweb;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
//Creamos un path o key para enlazarnos al Servlet
@WebServlet("/miServlet")
public class Servlet extends HttpServlet {
    //Sobreescribimos el método doGet para procesar solicitudes GET
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //Obtenemns el parametro nombre y guardamos en una variable String nombre
        String nombre = request.getParameter("nombre");
        //Declaramos el contetType, que tipo de contenido de la respuesta qeu el servidor enviará al
navegador
        //En este caso, se define como "text/html", indicando que la respuesta es una página web en
HTML
        response.setContentType("text/html");
        /*
        * Definimos la codificación de caracteres como UTF-8, lo cual es importante para soportar
        * caracteres especiales (tildes, ñ, etc..*/
        response.setCharacterEncoding("UTF-8");
        //Declaramos un objeto PrintWriter, que sirve para escribir directamente en la
        //respuesta HTTP enviada al navegador
        PrintWriter out = response.getWriter();
        /*Escribimos el inicio de la página HTML que el Servlet enviara al navegador*/
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Método GET</title>");
        out.println("</head>");
        /*Iniciamos el cuerpo de la página web  donde se mostrara el contenido prioncipal */
        out.println("<body>");
        out.println("<h1>Procesando información método GET</h1>");
        /*Mostramos un saludo personalizado con el nombre recibido desde el formulario o la URL*/
        out.println("<h2>Hola," + nombre + " (Solicitud GET)</h2>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Nota. La imagen nos muestra el código de un Servlet sobrescribiendo el método doGet, que realiza. *Fuente:* Los investigadores.

Figura 25.

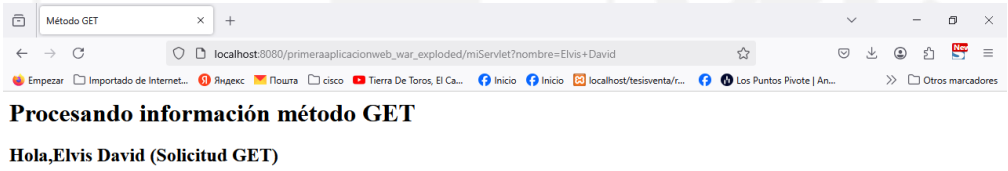
Enviando Petición GET

Nota. La imagen nos muestra la salida de pantalla donde se envía la información por el método GET al servidor.

Fuente: Los investigadores.

Figura 26.

Respuesta del Servidor al Cliente



Nota. La imagen muestra la salida de pantalla de la respuesta del servidor al cliente en archivo de formato HTML donde se captura la información del nombre que se envía en la *Figura 24*. *Fuente:* Los investigadores.

Para enviar una petición por el método Post realizamos un formulario, luego sobrescribimos el método doPost**,** en el Servlet procesamos la petición y el servidor devuelve la solicitud en forma de una página HTML.

Figura 27.

Formulario Petición POST

```
<%--Método para manejar solicitudes POST--%>
<form action="miServlet" method="POST">
    <div>
        <label for="apellido">Ingrese el apellido</label>
        <input type="text" id="apellido" name="apellido">
    </div>

    <div>
        <input type="submit" value="Enviar POST">
    </div>

</form>
```

Nota. La imagen nos muestra como realizar un formulario para realizar una petición POST al servidor. *Fuente:* Los investigadores.

Figura 28.

Código Servlet Procesando petición POST

```
//Sobreescribimos el método doPost
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        /*Declaramos una variable de tipo string
        *donde obtenemos el dato del apellido del formulario */
        String apellido = request.getParameter("apellido");
        /*
        * Declaramos el contennTyp, que tipo de contenido de la respuesta que el servidor enviará al
navegador
        * En este caso, se define como "text/html" indicando que la respuesta en una página web en
HTML*/
        response.setContentType("text/html");
        /*
        * Definimos la codificación de caracteres como UTF-8, lo cual es importante para soportar
        * caracteres especiales (tildes, ñ, etc..*/
        response.setCharacterEncoding("UTF-8");
        /*
        * Declaramos un objeto PrinterWriter, que sirve para escribir directamente en la respuesta
        * HTTP enviada al navegador*/
        PrintWriter out = response.getWriter();
        /*
        * Escribimos la sintaxis bpasica de HTML que el
        * Servlet enviara al navegador*/
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Método POST</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Procesando información método POST</h1>");
        out.println("<p>Hola, "+ apellido + " (Solicitud método POST)</p>");
        out.println("</body>");
        out.println("</html>");

    }
```

Nota. La imagen nos muestra la sobreescritura del método doPOST, y el servidor envía la información capturada del formulario de la *Figura 27*. *Fuente:* Los investigadores.

Figura 29.

Salida de Pantalla Petición POST

Nota. La imagen muestra al enviar la información al servidor por el método POST que luego será procesado y enviará a la información al cliente. *Fuente:* Los investigadores.

Figura 30.

Respuesta Servidor Petición POST

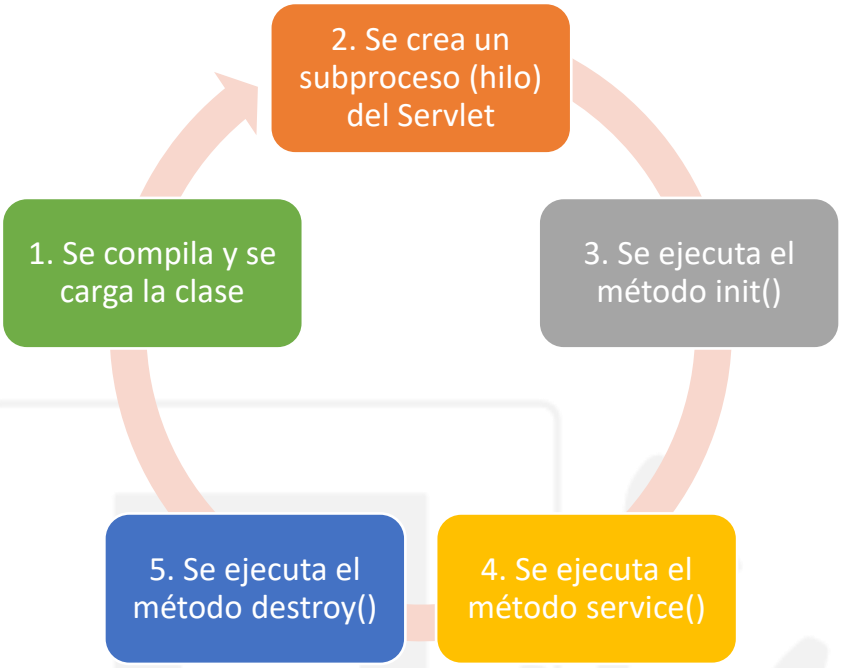
Nota. La imagen nos muestra la respuesta del servidor realizando la petición POST. *Fuente:* Los investigadores.

Los Servlets en Jakarta EE permiten procesar solicitudes HTTP GET y POST.

3.6 Ciclo de Vida del Servlet.

Figura 31.

Ciclo de Vida del Servlet



Nota. La imagen muestra el ciclo que vida que sigue un Servlet para procesar una petición del cliente. *Fuente:* Los investigadores.

En el desarrollo web con Jakarta, uno de los conceptos clave es el ciclo de vida de un Servlet. Un Servlet es una clase especial de Java que se extiende de una clase base, jakarta.servlet.http.HttpServlet**.**

Este ciclo de vida define todas las etapas por la que pasa un Servlet desde que es cargado en el servidor hasta que es destruido, abarcando desde la inicialización el manejo de solicitudes y finalmente su eliminación. Comprender este ciclo es esencial para que los lectores, estudiantes y desarrolladores puedan entender cómo se gestiona el flujo de las aplicaciones web dentro de un servidor compatible con Jakarta EE.

3.6.1 Creación y Carga en Memoria

El primer paso ocurre cuando el servidor de aplicaciones inicia o cuando el Servlet es solicitado por primera vez. En este momento, el servidor compila y carga la clase del Servlet en la memoria de la JVM (máquina virtual de Java).

Sin embargo, es importante aclarar que, a diferencia de otros objetos en Java, el servidor no crea una nueva instancia del Servlet cada vez que recibe una solicitud del cliente. En lugar de eso, crea

una única instancia de la clase Servlet y a partir de ella, genera subprocesos (hilos) que atienden cada solicitud de manera independiente.

Este enfoque basado en hilos permite que el servidor puede atender múltiples solicitudes al mismo tiempo usando un único objeto Servlet, lo que mejora el rendimiento y reduce considerablemente el uso de memoria.

3.6.2 Inicialización con el método init()

Una vez que se carga el Servlet, el servidor ejecuta el método llamado init(). Este método funciona de manera similar al constructor en una clase de Java normal. Su objetivo es preparar el entorno necesario para que el Servlet funcione correctamente, como abrir conexiones a bases de datos o cargar configuraciones iniciales.

Es importante recordar que el método init() solo se ejecuta una vez durante la vida del Servlet:

- Se ejecuta al momento de crear la primera instancia de la clase.
- Si posiblemente llegan nuevas solicitudes, el método init() ya no se vuelve a ejecutar.

En términos de eficiencia, esto evita inicializaciones repetidas y acelera el procesamiento de las siguientes peticiones.

3.6.3 Procesamiento de Peticiones con el método service()

Cuando un cliente envía una solicitud HTTP al servidor, este invoca al método service() del Servlet. Este método es el corazón del procesamiento, ya que es el encargado de identificar el tipo de solicitud (GET, POST, PUT, DELETE, etc.) y redirigirla automáticamente al método especifico correspondiente.

- doGet(), si la solicitud es GET.
- doPost(), si la solicitud es POST
- doDelete(), doPut() o cualquier otro método HTTP permitido.

Una excelente y buena práctica recomendada es no sobrescribe directamente el método service(). En su lugar, se debe sobrescribir los métodos específicos como doGet() o doPost(). Si el método service() es modificado, se interrumpe el flujo automático de identificación del método HTTP, lo que puede generar comportamientos inesperados.

3.6.4 Ejecución de la Lógica de Negocio.

Dentro del método doGet() o doPost(), el Servlet se encarga de:

- Recuperar parámetros enviados desde el cliente.

- Consultar la base de datos si es necesario.
- Procesar reglas de negocio (como validaciones o cálculos).
- Construir una respuesta que el servidor devolverá al navegador del usuario (HTML, JSON, PSF, etc).

El Servlet puede funcionar de manera independiente o bien delegar la generación de la vista a una página JSP, como parte de una arquitectura MVC.

3.6.5 Destrucción y Liberación de Recursos con destroy()

Cuando el servidor decide remover el Servlet de la memoria, ejecuta el método destroy(). Este método es similar al método finalize(), en una clase normal de Java.

Su propósito es que el desarrollador pueda cerrar conexiones, liberar memoria o realizar tareas de limpieza antes de que el Servlet sea definitivamente eliminado.

Al igual que el método init(), el método destroy() solo se ejecuta una vez, justo antes de que el Servlet sea descargado de la memoria.

En el ciclo de vida de un Servlet es una pieza fundamental dentro del entorno de Jakarta EE. Conocer cada fase permite a los desarrolladores no solo entender cómo funciona internamente el servidor, sino también para crear aplicaciones eficientes, estables y bien estructurados.

3.7 Cabeceros de Petición

En el mundo del desarrollo web moderno con Jakarta EE, las cabeceras HTTP son un componente esencial en la comunicación entre el cliente y el servidor. Estas cabeceras actúan como un conjunto de metadatos que acompañan cada solicitud enviada al servidor. Su función es proporcionar información adicional sobre la propia petición, el cliente que la origina y el entorno de ejecución.

3.7.1 Tipo de información que aportan las cabeceras HTTP

Algunos de los tipos de información más relevantes que se pueden obtener desde las cabeceras son:

- **Método HTTP utilizado:** El servidor puede identificar si la solicitud corresponde a un GET, POST, PUT, DELETE, entre otros. Esto dato es fundamental para que el Servlet determine cómo procesar la petición
- **Datos sobre el dispositivo equipo que realiza la solicitud**: Incluye la dirección IP desde donde proviene la conexión y, en algunos casos, el nombre del host o equipo del cliente. Esta

información es útil para auditoria, registro de logs o aplicar medidas de seguridad basadas en las procedencias de la solicitud.

- **Información sobre el dominio de origen:** El servidor puede saber desde que dominio especifico proviene la petición, lo que es clave cuando se trabaja con entornos distribuidos o integraciones con aplicaciones.
- **Datos sobre el navegador utilizado (User-Agent):** Cada navegador incluye una cabecera llamada User-Agent, sonde se especifica su nombre, versión, sistema operativo y otros datos técnicos. Por ejemplo, es posible diferenciar si la solicitud proviene de Google Chrome, Mozilla Firefox o un navegador móvil.
- **Otras cabeceras importantes:** Dependiendo del contexto, las cabeceras también pueden incluir información como el idioma preferido del usuario, el tipo del contenido que el cliente acepta recibir, detalles de autenticación y cookies enviadas en la sesión actual.

Cuando un desarrollador de software trabaja en aplicación web con Jakarta EE, entender cómo funcionan las cabeceras y cómo procesarlas desde un Servlet es importante por distintas razones.

- **Personalización Dinámica:** Se puede adaptar la respuesta según el navegador, el idioma preferido o el tipo de dispositivo
- **Registro y monitoreo:** Capturar las cabeceras permite construir logs de auditoria más completos, registrando quien accedió, desde dónde y con qué condiciones.
- **Seguridad:** Algunas cabeceras ayudan a detectar intentos de acceso no autorizado, bots maliciosos o peticiones sospechosas.
- **Integración y depuración:** Cuando se integran aplicaciones web con sistemas externos, las cabeceras sirven para rastrear el origen de las llamadas o depurar errores en servicios REST.

3.8 Procesamiento de Cabeceros HTTP en Jakarta EE mediante el Objeto HttpServletRequest

En el entorno de Jakarta EE, el manejo de cabeceras HTTP es responsabilidad directa del objeto HttpServletRequest, el cual es recibido automáticamente por cada Servlet cunado este procesa una solicitud HTTP, ya sea por el método doGet() o doPost(). Este objeto actúa como una representación completa de la petición HTTP, conteniendo tanto los parámetros enviados como toda la colección de cabeceras incluidas en la solicitud.

3.8.1 Principales métodos de HttpServletRequest para leer cabeceras.

El API de Jakarta Servlets ofrece un grupo de métodos específicos dentro del HttpServletRequest que permiten acceder y procesar las cabeceras recibidas.

- **getHeader(String nombre):** Este método permite recuperar el valor de una cabecera especifica si se conoce su nombre. Por ejemplo, para leer el User-Agent (información del cliente).
- **getHeaderNames():** Cuando es necesario recorres todas las cabeceras disponibles, este objeto devuelve un objeto de tipo Enumeration<String> que incluye la lista de los nombres de todas las cabeceras recibidas.
- **getHeaders(String nombre):** Es algunos casos, una misma cabecera puede tener múltiples valores. Este método recupera todos los valores asociados a una cabecera en lugar de solo uno:
- **getCookies:** Este método devuelve un arreglo de objetos Cookie**,** permitiendo al desarrollador leer las cookies almacenadas en el navegador del cliente. Estas cookies son fragmentos de información que el navegador guarda para recordar preferencias o estados entre visitas al sitio.
- **getAuthType() y getRemoteUser():** Estos métodos son útiles en aplicaciones que implementan autenticación y control de acceso:
    - **getAuthType():** Devuelve el tipo de autenticación utilizando (BASIC, DIGEST, FORM, etc.).
    - **getRemoteUser():** Devuelve el nombre del usuario autenticado, si este existe.
- **getContentLength():** Indica el tamaño en bytes de cuerpo de la solicitud, lo cual es útil al procesar formularios o cargas de archivos.
- **getContentType():** Devuelve el tipo del contenido recibido.
- **getDateHeader(String nombre):** Si alguna cabecera contiene un valor tipo fecha, este método lo devuelve como un long con el número en milisegundos desde el Epoch (1 de enero de 1970).
- **getIntHeader(String nombre):** Cuando una cabecera contiene un valor numérico entero, este método lo devuelve como un int.
- **getMethod():** Este método es fundamental, ya que indica el método HTTP que se utilizó en la petición actual del cliente.

- **getRequestURI():** Este método devuelve la ruta exacta solicitada por el cliente, sin incluir el dominio o el contexto.
- **getQueryString():** Si la solicitud incluye parámetros en la URL, este método devuelve esa cadena de consulta completa.

Ejemplo Servlet que procesa todas las cabeceras.

Figura 32.

HTML petición de cabeceros

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Manejo de Cabeceros</title>
</head>
<body>
<h1>Manejo de Cabeceros</h1>

<!-- Enlace para acceder al servlet de las cabeceras -->
<a href="/manejoCabeceros/cabeceros">Ir a servlet (Cabeceros)</a><br><br>


</body>
</html>
```

Nota. La imagen nos muestra una página web donde se realiza una apetición GET al Servlet. *Fuente:* Los investigadores.

Figura 33.

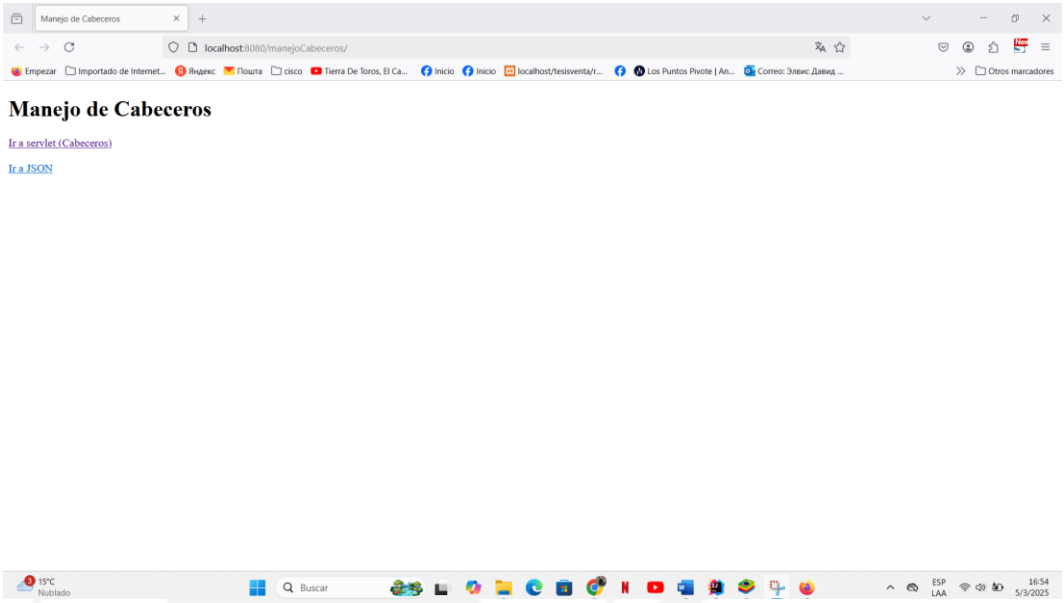
Servlet Cabeceros

```java
package controlador.Cabecero;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

@WebServlet("/cabeceros")
public class Cabecero extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        resp.setContentType("text/html;charset=UTF-8");
        String metodoHttp = req.getMethod();
        String requestURI = req.getRequestURI();
        String requestURL = req.getRequestURL().toString();
        String contextPath = req.getContextPath();
        //
        String servletPath = req.getServletPath();

        //Guardamos en una variable String lo que obtenemos, nombre del navegador
        String navegador = req.getHeader("User-Agent");

        // Obtener la IP del cliente
        String clientIP = req.getRemoteAddr();

        // Obtener el puerto por el que se está ejecutando la aplicación
        int port = req.getServerPort();

        // Obtener el esquema (http o https)
        String schema = req.getScheme();

        // Obtener el nombre del host
        String host = req.getServerName();

        // Construir las URLs requeridas
        String url1 = schema + "://" + host + contextPath + servletPath;
        String url2 = schema + "://" + clientIP + ":" + port + contextPath + servletPath;

        // Obtener los nombres de los headers
        Enumeration<String> headerNames = req.getHeaderNames();

        PrintWriter out = resp.getWriter();

        // Crear la plantilla HTML
        out.print("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset=\"utf-8\">");
        out.println("<title>Cabeceros</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Peticiones Http Request</h1>");
        out.println("<ul>");
        out.println("<li>Navegador: " + navegador + "</li>");
        out.println("<li>Método: " + metodoHttp + "</li>");
        out.println("<li>URI: " + requestURI + "</li>");
        out.println("<li>URL completa: " + requestURL + "</li>");
        out.println("<li>ContextPath: " + contextPath + "</li>");
        out.println("<li>Servlet Path: " + servletPath + "</li>");
        out.println("<li>IP del cliente: " + clientIP + "</li>");
        out.println("<li>Puerto: " + port + "</li>");
        out.println("<li>Esquema: " + schema + "</li>");
        out.println("<li>Host: " + host + "</li>");
        out.println("<li>URL construida 1 (schema+host+contextPath+servletPath): " + url1 + "</li>");
        out.println("<li>URL construida 2 (schema+ip+port+contextPath+servletPath): " + url2 + "</
li>");
        out.println("</ul>");

        out.println("<h2>Cabeceras HTTP</h2>");
        out.println("<ul>");
        while (headerNames.hasMoreElements()) {
            String headerName = headerNames.nextElement();
            String headerValue = req.getHeader(headerName);
            out.println("<li>" + headerName + ": " + headerValue + "</li>");
        }
        out.println("</ul>");

        out.println("</body>");
        out.println("</html>");
    }
}
```

Nota. La imagen nos muestra el código del Servlet donde se está procesando las peticiones de cabeceros y el servidor devolviendo en formato HTML toda información. *Fuente:* Los investigadores.

Figura 34.

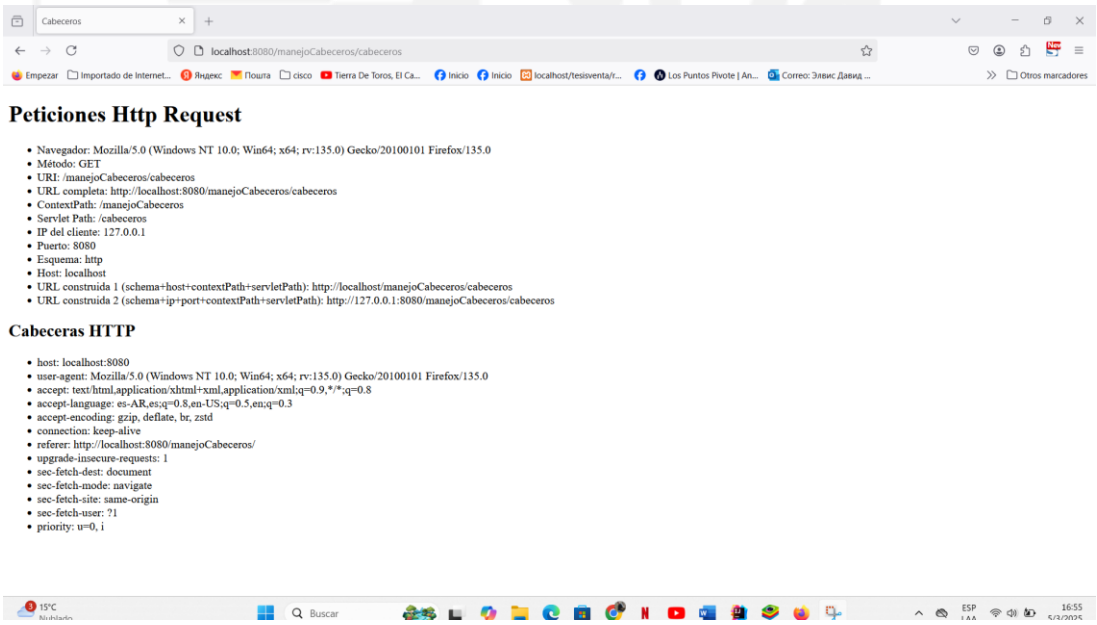
Salida de Pantalla Petición Cabeceros



Nota. La imagen nos muestra la página web donde estamos haciendo la petición GET al servidor, salida del código *Figura 32*. *Fuente:* Los investigadores.

Figura 35.

Salida Pantalla Solicitud Cabeceros



Nota. La imagen nos muestra la información de la petición realizada al servidor del código de la *Figura 33*. *Fuente:* Los investigadores.

Conocer y dominar los métodos del objeto HttpServletRequest es una habilidad esencial para todo desarrollador de aplicaciones web en Jakarta EE. Estos métodos no solo no van a permitir leer y procesar los cabeceros enviada por el cliente, sino que también facilita tareas criticas como:

- Implementar autenticación
- Manejar cookies de sesión.
- Registrar los detalles de acceso.
- Adaptar el contenido al dispositivo o navegador del cliente.

Aprender cómo funcionan estos métodos son fundamentales del protocolo HTTP desde la perspectiva de un Servlet es importante para los profesionales puedan crear aplicaciones web modernas, robustas y seguras.

3.9 Manejo de Cookies con Servlets

¿Qué son las cookies y por qué son necesarias?

En el desarrollo de aplicaciones web con Jakarte EE, las cookies juegan un papel clave para recordar información relacionada con el usuario entre una solicitud y otra. Esto es necesario debido a que el protocolo HTTP, por naturaleza, es un protocolo sin estado, lo que significa que cada solicitud HTTP es independiente y no tiene memoria sobre quien es el cliente que hizo la petición anterior.

Una cookie es un pequeño fragmento de información que el servidor web envía al navegador del cliente, y este lo almacena como un archivo de texto plano. Cada cookie contiene un nombre y un valor asociado, formando un par de clave-valor. Este mecanismo permite que, en futuras solicitudes, el navegador reenvié automáticamente esas cookies al servidor, permitiendo que el servidor recuerde ciertos datos del usuario.

3.9.1 Buenas Prácticas y Recomendaciones

Aunque las cookies son útiles para el desarrollo de aplicaciones web, no son el luga adecuado para almacenar información sensible. Esto se debe a que, las cookies se almacenan como texto plano en el navegador, esto quiere decir que cualquier persona que tenga acceso al dispositivo puede leerlas.

La información como contraseñas, número de tarjetas de créditos o datos personales críticos no deberían guardarse jamás en las cookies, para proteger la privacidad del usuario se recomienda utilizar cookies seguras y sesiones. Estas se llegarán a eliminar automáticamente al cerrar el navegador.

3.9.2 Creación Lectura y Manipulación de las Cookies desde el Servlet.

En el desarrollo de aplicaciones en Jakarta EE, los Servlets tiene la capacidad de interactuar directamente con las cookies que forman parte de una solicitud HTTP. Esta interacción incluye leer las cookies, crear nuevas Cookies, modificar su contenido y finalmente se envía las cookies al cliente como parte de la respuesta HTTP.

3.9.3 Creación de Cookies

Al API de Servlets proporciona la clase Cookie, que permite construir un nuevo objeto de tipo cookie utilizando su nombre y su valor asociado. En código, esto se hace de la siguiente manera.

Figura 36.

Creación Objeto Cookie

```
//Creación de un objeto Cookie:
      Cookie c=new Cookie("Usuario", "Elvis");
```

Nota. Aquí se muestra el código de cómo crear una Cookie instanciando el objeto Cookie y como parámetros pasamos clave-valor. *Fuente:* Los investigadores.

En este caso, la cookie creada tiene como nombre Usuario y como valor asociado el texto Elvis. Cada cookie sigue esta estructura básica de clave-valor y es posible añadir todas las cookies que sean necesarias en una misma respuesta HTTP.

3.9.4 Recuperar el Nombre y el Valor de una Cookie

Una vez que una Cookie ha sido creada, el Servlet puede acceder a sus atributos a través de los métodos.

- **getName():** Devuelve el nombre de la cookie.
- **getValue():** Recupera el valor almacenado dentro de esa cookie.

Figura 37.

Obtener los valores de la Cookie

```
//Obtenemos el nombre de la Cookie
c.getName();
//Obtenemos el valor de la cookie
c.getValue();
```

Nota. La imagen muestra el código para obtener los valores de las cookies. *Fuente:* Los investigadores..

3.9.5 Lectura de Cookies desde una solicitud HTTP

Cuando el navegador envía una nueva solicitud HTTP al servidor, todas las cookies asociadas al dominio son automáticamente enviadas junto con la petición.

Desde el Servlet, esas cookies se pueden leer utilizando el método.

Figura 38.

Lectura de Cookies

```
//Lectura de cookies desde una solicitud HTTP
        Cookie[] cookies = req.getCookies();
```

Nota. La imagen nos muestra el código para leer la Cookie desde una petición HTTP. *Fuente:* Los investigadores.

Es importante aclarar que el API de los Servlets no proporciona un método directo para obtener una cookie especifica por su nombre. En su lugar, el desarrollador debe recorrer el arreglo completo de cookies para localizar la cookie deseada.

Figura 39.

Código Petición HTTP

```
//Lectura de cookies desde una solicitud HTTP
        Cookie[] cookies = req.getCookies();
        //Creamos una variable para guardar el valor de la cookie
        String valorUsuario = null;
        //Verificamos si la cookie esta vacia
        if (cookies != null) {
            //Si la cookie no esta vacia la recorremos
            //COn un foreach de principio a fin
            for (Cookie cookie : cookies) {
                /*Verificamos si el valor obtenido es igual al nombre de
                * la cookie obtenemos el valor de la cookie*/
                if (cookie.getName().equals("Usuario")) {
                    /*En la variable valorUsuario guardamos el valor de
                    * la Cookie*/
                    valorUsuario = cookie.getValue();
                }
            }
        }
```

Nota. La imagen nos muestra el código de una petición HTTP para obtener los datos de la Cookie recorriéndola con un foreach. *Fuente:* Los investigadores.

3.9.6 Envío de Cookies en la Respuesta HTTP.

Una vez que se crea y se modifica las cookies deben ser agregadas a la respuesta HTTP, para que el navegador las almacene localmente y las reenvié en futuras solicitudes. Este proceso se lo realiza de la siguiente manera.

Figura 40.

Enviar Cookie al Cliente

```
//El servidor envia la cookie obtenida al cliente
        resp.addCookie(c);
```

Nota. La imagen nos muestra la respuesta del servidor al cliente devolviendo la cookie. *Fuente:* Los investigadores.

Con este flujo, cada petición y respuesta HTTP puede incluir cookies, permitiendo que el servidor recuerde información importante sobe el cliente.

3.10 API del Objeto Cookie

El objeto Cookie, parte clave del API de los Servlets en Jakarta EE, proporciona una serie de métodos esenciales para gestionar el ciclo de vida y el comportamiento de cada cookie. Es importante recordar que cada cookie está directamente asociada al dominio o IP del servidor que la creó, así como un tiempo de vida y, en ciertos casos, al cliente que la recibe. Esto asegura que cada navegador solo devuelva las cookies correspondientes al sitio web especifico que las emitió.

3.10.1 Métodos claves del objeto Cookie

Los métodos getDomain() y setDomain() son métodos que permiten consultar o definir el dominio al que pertenece la cookie. Em términos prácticos, esto significa que el desarrollador puede especificar de que dominio proviene la cookie y que dominios se permitirá reenviar esa cookie. Por defecto, una cookie está limitada al dominio exacto desde el cual fue enviada, pero con setDomain() es posible extender el alcance a subdominios.

Los métodos getMaxAge() y setMaxAge(), son métodos que controlan el tiempo de vida de la cookie en el navegador del cliente. El tiempo se define en segundos, y una vez transcurrido ese periodo, el navegador automáticamente elimina la cookie.

- Un valor positivo indica el tiempo exacto de la expiración de la cookie.
- Un valor de 0 fuerza la eliminación inmediata de la cookie.
- Un valor negativo indica que la cookie solo vive durante la sesión actual esta cookie se elimina al cerrar el navegador.

Figura 41.

Tiempo de vida de la Cookie

```
//Creación de un objeto Cookie:
        Cookie c=new Cookie("Usuario", "Elvis");
        //Configuramos el tiempque va a existir la cookie
        //El tiempo se le configura en segundos en esta cookie
        //durara una hora
        c.setMaxAge(3600);
```

Nota. La imagen nos muestra el código para poder configurar el tiempo de vida de la cookie, el tiempo se le configura en segundos. *Fuente:* Los investigadores.

3.11 Las Cookies y su elación con las sesiones

Si bien como hemos visto las cookies pueden parecen un simple mecanismo para almacenar preferencias, en aplicaciones web más complejas también forman parte clave en la gestión de sesiones. Por ejemplo, en Jakarta EE, el identificador único de sesión (JSESSIONID) se transporta precisamente dentro de una cookie, lo que permite al servidor identificar que cliente está realizando cada solicitud, incluso en entornos sin estado como HTTP.

Ejercicio

Crear una aplicación web en Jakarta EE donde el usuario mediante un formulario seleccione su idioma preferido, el idioma seleccionado se almacenera en una cookie y cada vez que el usuario regrese a la página, el Servlet lea la cookie y muestre un saludo en el idioma preferido

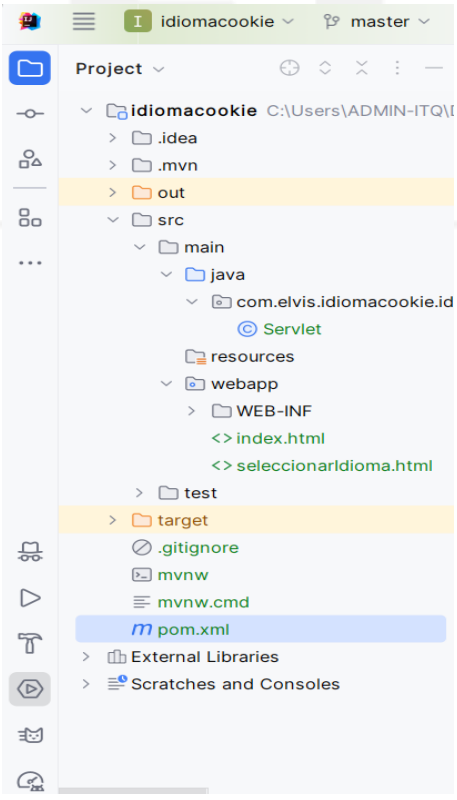
Explicación del flujo de la aplicación

La aplicación está estructurada de la siguiente manera:

- Creamos el proyecto como ya se explicó en capítulos anteriores.

Figura 42.

Estructura del Proyecto.



Nota. La imagen muestra la estructura del proyecto con cada uno de sus archivos. *Fuente:* Los investigadores.

- Se realiza la configuración del archivo pom.xml

Figura 43.

Configuración pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.elvis.idiomacookie</groupId>
    <artifactId>idiomacookie</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>idiomacookie</name>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.target>11</maven.compiler.target>
        <maven.compiler.source>11</maven.compiler.source>
        <junit.version>5.10.0</junit.version>
    </properties>

    <dependencies>
            <dependency>
                <groupId>jakarta.platform</groupId>
                <artifactId>jakarta.jakartaee-web-api</artifactId>
                <version>10.0.0</version>
                <scope>provided</scope>
            </dependency>

            <dependency>
                <groupId>org.junit.jupiter</groupId>
                <artifactId>junit-jupiter-api</artifactId>
                <version>${junit.version}</version>
                <scope>test</scope>
            </dependency>
            <dependency>
                <groupId>org.junit.jupiter</groupId>
                <artifactId>junit-jupiter-engine</artifactId>
                <version>${junit.version}</version>
                <scope>test</scope>
            </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-site-plugin</artifactId>
                    <version>3.12.1</version>
                </plugin>

                <plugin>
                    <groupId>org.apache.tomcat.maven</groupId>
                    <artifactId>tomcat7-maven-plugin</artifactId>
                    <version>2.2</version>
                    <configuration>
                        <url>http://localhost:8080/manager/text/url</url>
                        <username>elvis</username>
                        <password>elvis2103</password>
                    </configuration>
                </plugin>

                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>3.4.0</version>
                </plugin>
            </plugins>
        </build>
</project>
```

Nota. La imagen nos muestra el código, para la configuración del archivo pom.xml donde se encuentran todas las dependencias que se usaran en el proyecto. *Fuente:* Los investigadores.

- Se crea un archivo index.html donde se va a enviar peticiones get al servidor.

Figura 44.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
    <title>Selección de idiomas</title>
</head>
<body>
<header>
    <h1>BIENVENIDO A MI PAGINA WEB</h1>
</header>

    <p class="parrafo"><a href="preferenciaIdiomaServlet">Ver mensaje en mi idioma preferido</a></p>
    <p class="parrafo"><a href="seleccionarIdioma.html" id="link">Cambiar Idioma Preferido</a></p>
</body>
</html>
```

Nota. La imagen nos muestra el código html, donde se realizan peticiones get al servidor, un enlace hace referencia al servlet y el otro enlace hace referencia al archivo seleccionarIdioma.html. *Fuente:* Los investigadores.

- Se crean un archivo seleccionarIdioma.html donde se realiza una petición POST al servidor para cambiar de idioma, este archivo tiene un formulario con una etiqueta select, en donde se tiene tres tipos de idiomas: español, inglés y ruso.

Figura 45.

Código Formulario

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>SELECCIONAR IDIOMA</title>
</head>
<body>
    <header>
        <h1>Bienvenido al Sistema de preferencia de Idioma</h1>
    </header>
    <div>
        <form action="preferenciaIdiomaServlet" method="post">
            <div>
                <label for="idioma">SELECCIONE EL IDIOMA</label>
                <select name="idioma" id="idioma">
                    <option value="">---SELECCIONE EL IDIOMA---</option>
                    <option value="es">ESPAÑOL</option>
                    <option value="en">ENGLISH</option>
                    <option value="ru">RUSO</option>
                </select>
            </div>
            <div>
                <input type="submit" value="GUARDAR">
            </div>
        </form>
    </div>
        <div>
            <a href="index.html">Volver al inicio</a>
        </div>
</body>
</html>
```

Nota. La imagen muestra el código del formulario donde se va a elegir el idioma preferido para guardar en la cookie. *Fuente:* Los investigadores.

- Implementamos el Servlet.java el que se va a encargar de realizar toda la lógica de negocio. En este archivo Java sobrescribimos el método doGet y doPost para procesar las peticiones del cliente.

  En el método doPost, vamos a obtener la preferencia del idioma desde el formulario, luego se va a crear la Cookie mediante el objeto Cookie con su respectivo nombre-valor (“idiomaPreferido”, idioma), le daremos un tiempo de validez a la cookie, posteriormente enviaremos la cookie la cliente y por último redirigiremos mediante un enlace al index, y de esta forma queda creada la cookie.

Figura 46.

Servlet método doPost

```
ackage com.elvis.idiomacookie.idiomacookie;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

//Crear elpath para tener acceso al Servlet
@WebServlet("/preferenciaIdiomaServlet")
public class Servlet extends HttpServlet {

    /*Sobrescribimos el método doPost*/

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        //Leer el idioma enviado desde el formulario HTML
        String idioma = req.getParameter("idioma");

        //Crear la Cookie y la configuramos
        Cookie cookieIdioma = new Cookie("idiomaPreferido", idioma);
        //Le damos un tiempo de validez
        cookieIdioma.setMaxAge(86400);

        //enviamosla cookie al cliente o añadimos la cookie
        resp.addCookie(cookieIdioma);

        //redirigimos a la página principal index.html
        resp.sendRedirect("index.html");
    }
```

Nota. La imagen muestra el código de la petición doPost donde se crea la cookie y se envía al cliente. *Fuente:* Los investigadores.

- En este paso vamos a sobrescribir el método doGet, este método se encarga de mostrar la preferencia del idioma leyendo la cookie que se creó en el método doPost, creamos una variable predeterminada para el idioma en nuestro caso en español, verificamos si la información traída del servidor no es nula quiere decir que en la cookie hay algo, recorremos todos los parámetros de la cookie con un foreach, verificamos si lo que trajo es igual al nombre de la cookie, si es igual obtenemos el valor de la cookie.

  Después mostramos el mensaje con el idioma seleccionado, para eso implementamos la estructura html comprando la cookie y mostrando el mensaje en el idioma seleccionado y por último un enlace para redirigirse al inicio.

Figura 47.

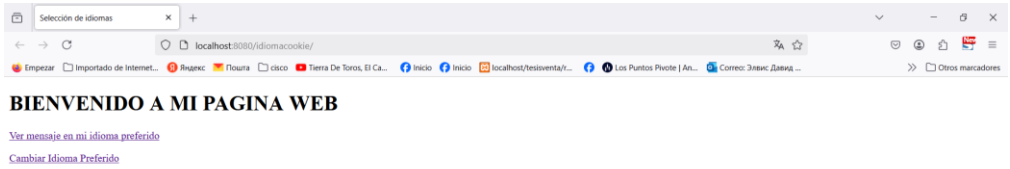
Código método doGet

```java
//Sobreescribimos el método doGet

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
         //Leer las cookies existentes
        Cookie[] cookies = req.getCookies();
        //Creamos una variable de tipo estring con un valor por defecto
        String idioma ="es";

        //Verificamos si la Cookie no esta vacio o no es nula
        if (cookies != null) {
            //Recorremos los valores de la cookie
            for (Cookie cookie : cookies) {
                //Verificamos si la cookie que estamos envia es igual a la que esta guardada
                if ("idiomaPreferido".equals(cookie.getName())) {
                    //Guardamos en la variable idioma el valor de la cookie
                    idioma = cookie.getValue();
                    //hacemos un salto de linea
                    break;
                }
            }
        }
        //Mostramos el mensaje de salido con el idioma pseleccionado
        resp.setContentType("text/html;charset=UTF-8");
        //Creamos un objeto de tipo PrintWriter para mostrar en pantalla el contenido
        PrintWriter out = resp.getWriter();
        //Creamos la estructura que va a devolver el servidor
        //al cliente
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Mensaje Personalizado</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Mensaje Personalizado</h1>");
        if("es".equals(idioma)){
            out.println("<h1>Bienvenido a nuestro sitio!</h1>");
        } else if ("en".equals(idioma)) {
            out.println("Welcome to our site!");
        } else if ("ru".equals(idioma)) {
            out.println("Добро пожаловать на наш сайт!");
        }else {
            out.println("<h1>Bienvenido a nuestro sitio!</h1>");
        }
        out.println("<br>");

        out.println("<a href='seleccionarIdioma.html'>Cambiar de idioma</a>");
        out.println("</body>");
        out.println("</html>");


    }
```

Nota. La imagen nos muestra el código del método doGet donde se procesa la cookie para mostrar el mensaje con el idioma seleccionado. *Fuente:* Los investigadores.

Figura 48.

Salida de pantalla index.html



Nota. La imagen nos muestra la salida de pantalla de la aplicación donde nos da a escoger dos opciones ver mensaje en mi idioma preferido o cambiar idioma preferido. El archivo es el index.html. *Fuente:* autores.

Figura 49.

Ventana seleccionarIdioma.html

Nota. La imagen muestra la salida de pantalla donde vamos a seleccionar el idioma preferido que posterior se guardara en la cookie la opción elegida. *Fuente:* Los investigadores.

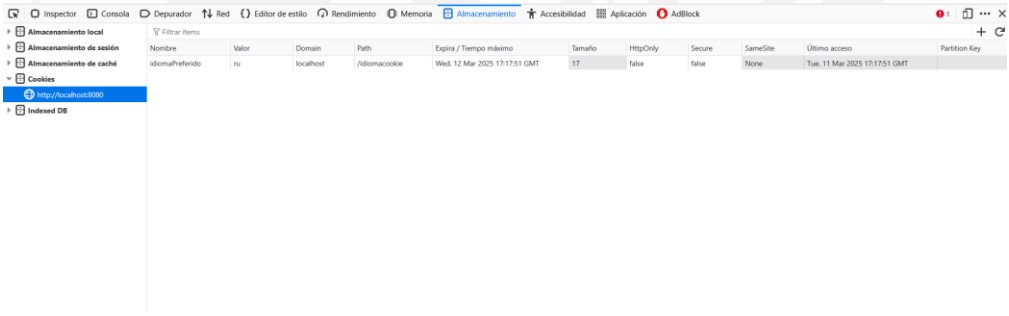
Figura 50.

Salida de Pantalla del Servlet

Nota. La imagen nos muestra la salida de pantalla del servidor al cliente después de haber obtenido la cookie, y nos muestra el mensaje con el idioma seleccionado. *Fuente:* Los investigadores.

Figura 51.

Salida de Pantalla de la Cookie



Nota. La imagen nos muestra la cookie que se está generando con el nombre y el valor en este caso idiomaPreferido y el valor ru. *Fuente:* Los investigadores.

Se desarrollado un ejercicio aplicando todos los conceptos vistos en esta sección de cookies.

Con este enfoque centrado en el Servlet se respeta el principio de que la lógica de negocio y procesamiento se mantenga dentro del backend, mientras que el HTML solo presenta la interfaz de usuario.

3.12 Manejo de Sesiones en Jakarta EE con HttpSession

En el desarrollo de aplicaciones web con Jakarta EE, es fundamental gestionar la información de los usuarios entre múltiples solicitudes HTTP. Para lograr esto, el API de los Servlets proporciona el objeto HttpSession, el cual permite mantener datos del usuario durante toda su sesión activa en el servidor.

Como ya hemos mencionado el protocolo HTTP no tiene memoria y el uso de sesiones es esencial para manejar y asociar múltiples peticiones HTTP que están asociadas a un mismo usuario, asegurando que su información permanezca disponible mientras dure la sesión.

3.12.1 ¿Cómo funcionan las sesiones en Jakarta EE?

Cuando un usuario accede a un sitio web, el servidor crea automáticamente una sesión para él si no existe una previamente. Esta sesión es única para cada cliente y se mantiene a lo largo de todas sus solicitudes HTTP hasta que expira o se invalida manualmente.

- Si un usuario accede al sitio web desde dos navegadores diferentes en la misma computadora: Firefox y Chrome, cada navegador será tratado como un cliente distinto esto hace que se generen dos sesiones distintas.
- SI el usuario abre una nueva ventana de cualquier navegador, la sesión seguirá siendo la misma, ya que los navegadores comparten la misma información de sesión.

3.12.2 Manejo de Sesiones en Jakarta Métodos Clave del Objeto HttpSession

Para gestionar una sesión en una aplicación web con Jakarta EE, utilizamos el objeto HttpSession, el cual permite almacenar y recuperar información del usuario, a lo largo de múltiples solicitudes HTTP. El primer paso para trabajar con sesiones en un Servlet es obtener una instancia de HttpSession. Esto se logra a través del objeto HttpServletRequest, utilizando el siguiente método.

Figura 52.

Objeto Session

```
HttpSession sesion=req.getSession();
```

Nota. La imagen muestra el código para obtener la sesión mediante el objeto HttpSession. *Fuente:* Los investigadores.

Cuando se llama al método:

- Si el usuario ya tiene una sesión activa, se devuelve la misma.
- Si la sesión no existe, se crea una nueva sesión para el usuario.

Cada sesión está asociada al cliente que realizo la solicitud, lo que significa que cada usuario tiene su propia sesión única en el servidor.

3.12.3 Principales métodos de HttpSession

Para poder obtener un atributo de la sesión se utiliza el método getAttribute(String nombre). Este método permite recuperar información previamente almacenada en la sesión del usuario. Es útil cuando queremos acceder a datos como el nombre del usuario, su rol, preferencias, etc.

Figura 53.

Obtener Atributos de la sesión

```
//Declaremos una variable donde vamos a guardar el parametro de la sesión
        String nombreUsuario=(String) sesion.getAttribute("usuario");
        /*Creamos un condicional if para verificar que
        * nombreUsuario que sea distinto a nulo*/
        if(nombreUsuario != null){
            //Imprimimos el nombre del Usuario
            System.out.println("Usuario en sesion "+nombreUsuario);
        }else{
            //Si no se trae nada en la variableUsuario  mostramos un mensaje
            //que diga no hay usuario en la sesión
            System.out.println("no hay usuario en sesion");
        }
```

Nota. La imagen muestra el código donde se obtiene el nombre de usuario de la sesión. *Fuente:* Los investigadores.

Para almacenar información en la sesión se utiliza el método setAttribute(String nombre, Object valor). Este método se utiliza para guardar datos en la sesión, permitiendo que estos valores estén disponibles para solicitudes futuras mientras la sesión siga activa.

Figura 54.

Código para almacenar información de la sesión

```
//Seteamos los atributos de la sesión de usuario con su
        //respectivo valor
        sesion.setAttribute("usuario","Norma Jami");
        //Seteamos el rol del usuario como administrador
        sesion.setAttribute("rol", "admin");
```

Nota. La imagen muestra el código para poder almacenar la información de la sesión en este ejemplo guardamos el nombre del usuario que tiene como valor Norma Jami y el rol del usuario que tiene un rol admin. *Fuente:* Los investigadores.

En este caso, se está guardando dos atributos en la sesión:

- "usuario" con el valor "Norma Jami"
- "rol" con el valor de "admin"

Esta información estará disponible en todas las solicitudes posteriores que realice este usuario.

Para remover un atributo de la sesión utilizamos el método removeAttribute(String nombre), si ya no necesitamos un atributo dentro de la sesión, podemos eliminarlo con el siguiente método.

Figura 55.

Método que remueve la información de la sesión

```
//Eliminamos el atributo de usuario que se encuentra en la sesión
        sesion.removeAttribute("usuario");
```

Nota. La imagen muestra el código del método para poder remover algún atributo de la sesión. *Fuente:* Los investigadores.

Este método no destruye la sesión completa, solo elimina el atributo especifico en este caso estamos eliminando el nombre del usuario de la sesión.

Para poder eliminar o finalizar una sesión se utiliza el método invalidate(), este método cierra la sesión completamente y elimina todos los atributos almacenados en la sesión.

Figura 56.

Método para invalidar la sesión

```
//Cierra la sesión del usuario y elimina todos los datos almacenados
       //en la sesión
       sesion.invalidate( );
```

Nota. La imagen muestra el código del método para destruir la sesión y eliminar toda la información de esta.

Fuente: Los investigadores.

4 Ejercicio con Sesiones

Desarrollar una aplicación web en Jakarta EE que permita administrar sesiones de usuario mediante HttpSession. Los estudiantes deberán seguir los siguientes pasos.

- Verificar si el usuario ya tiene una sesión activa con request.getSession().
- Si no tiene una sesión, crear una nueva sesión y almacena un atributo “usuario”.
- Si la sesión ya existe, muestre un mensaje con el nombre del usuario
- Agregar un enlace a Cerrar Sesión.
- Crear un Servlet para cerrar la sesión.
- Recupera la sesión existe con request.getSession(false).
- Si la sesión existe, elimínala con el método invalidate().
- Redirige al usuario a index.html para indicar que ha cerrado sesión.
- Crear una página principal index.html.
- Contiene enlaces al archivo ManejoSesion para iniciar sesión.
- Agregar un enlace a CerrarSesion para que el usuario pueda salir.

Figura 57.

Código index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>MANEJO DE SESIONES EN JAKARTA</title>
</head>
<body>
    <header>
        <h1>Manejo de Sesiones</h1>
    </header>
    <div>
        <a href="manejoSesion">Manejo de Sesiones</a>

    </div>
    <div>
        <a href="cerrarSesion">Cerrar Sesiones</a>
    </div>
</body>
</html>
```

Nota. La imagen muestra el código HTML donde se va a implementar dos enlaces donde se crea la sesión y otro donde se termina la sesión. *Fuente:* Los investigadores.

Figura 58.

Salida de Pantalla index.html

Nota. La imagen nos muestra la salida de pantalla del código *Figura 57*. *Fuente:* Los investigadores.

Figura 59.

Código Servlet manejoSesion

```java
package com.elvis.manejosesiones.manejosesiones;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/manejoSesion")
public class ManejoSesion extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();

        //Obtenemos o creamos una nueva sesion
        HttpSession sesion=req.getSession();

        //Obtener el atributo "usuario" si existe
        String usuario= (String) sesion.getAttribute("usuario");

        //Mandamos la información al cliente
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset=\"UTF-8\">");
        out.println("<title>Manejo Sesion en Jakarta</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Manejo Sesion</h1>");
        if(usuario==null){
            //Si no hay un usuario en lasesion la creamos
            usuario="Norma Jami";
            sesion.setAttribute("usuario",usuario);
            out.println("<h2>¡Bienvenido, ! "+ usuario + " </h2>");
            out.println("<p>Tu sesión a sido creada</p>");

        }else{
            //Si ya hay una sesión activa, mostramos el usuario
            out.println("<h2>Bienvenido de nuevo, " + usuario + "!</h2>");
            out.println("<p>Tu sesión sigue activa.</p>");
        }

        //Opcion para cerrar sesión
        out.println("<p><a href='cerrarSesion'>Cerrar Sesión</a></p>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Nota. La imagen muestra el código del servlet donde se crea una sesión se obtiene y muestra el saludo al cliente.

Fuente: Los investigadores.

Figura 60.

Salida del Pantalla manejoSesion.

Nota. La imagen muestra la salida de pantalla del código de la *Figura 60*, donde muestra un mensaje de bienvenida al usuario que inicia sesión. *Fuente:* Los investigadores.

Figura 61.

Código cancelar Sesión

```java
package com.elvis.manejosesiones.manejosesiones;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

import java.io.IOException;

@WebServlet("/cerrarSesion")
public class CerrarSesion extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        //Obtenemos la sesión existente, sin crear una nueva
        HttpSession sesion = req.getSession(false);

        //creamos una condicional para preguntar si la sesión es distinta de nula
        if (sesion != null) {
            //Cerramos la sesión
            sesion.invalidate();
        }
        resp.sendRedirect("index.html");
    }
}
```

Nota. La imagen muestra el código del Servlet que destruye la sesión creada en el código de la Figura 59. *Fuente:* Los investigadores.

El objeto HttpSession en Jakarta EE es una herramienta esencial para gestionar la iteración de los usuarios en aplicaciones web. Este objeto nos permite:

- Mantener solicitudes persistentes entre solicitudes HTTP.
- Distinguir sesiones únicas para cada usuario.
- Cerrar sesiones de manera controlada para evitar problemas de seguridad o consumo excesivo de memoria en el servidor.

Dominar estos conceptos es importante para cualquier desarrollador web, ya que las sesiones son la base de la autenticación de usuarios, carritos de compra y muchas otras funcionalidades esenciales en aplicaciones web modernas.

RESUMEN DEL CAPÍTULO 3

Un Servlet es un componente del lado del servidor que extiende la funcionalidad de un servidor web procesando peticiones HTTP y generando respuestas dinámicas. Se define dentro de la especificación de Jakarta EE y forma parte del paquete jakarta.servlet.http.

Cuando un usuario interactúa con una aplicación web (por ejemplo, enviando un formulario o accediendo a una URL), el servidor recibe la solicitud HTTP y la direcciona a un Servlet que se encarga de procesarla y generar una respuesta.

El ciclo de vida de un Servlet estás compuesto por cinco fases principales:

- Carga y Creación: Cuando el servidor recibe la primera solicitud para un Servlet, lo carga en memoria y crea una única instancia.
- Inicialización (init()): Se ejecuta una sola vez cuando el Servlet es creado. Aquí se pueden definir configuraciones iniciales.
- Procesamiento de Peticiones (service()): Cada vez que el Servlet recibe una solicitud, ejecuta el método service(), que a su vez llama a doGet(), doPost(), doPut(), etc.
- Generación de Respuesta: El Servlet procesa los datos, interactúa con la lógica de negocio y devuelve una respuesta al cliente.
- Destrucción (destroy()): Ocurre cuando el servidor elimina el Servlet de la memoria, liberando los recursos utilizados.

Las cookies permiten almacenar información en el navegador del usuario y enviarla al servidor en cada solicitud. Son útiles para recordar información como preferencia de idioma, autenticación o configuración personalizada.

Cuando se genera una cookie nunca almacenar información en las cookies.

EL protocolo HTTP no mantiene información entre solicitudes, por lo que es necesario utilizar sesiones para recordar datos del usuario mientras navega en la aplicación.

Los Servlets con la base del desarrollo web en Jakarta EE, permitiendo a los desarrolladores gestionar solicitudes HTTP, manejar cookies y administrar sesiones de usuario de forma eficiente.

Ejercicios Propuestos

1. ¿Qué es un Servlet y cuál es su función principal en Jakarta EE?
2. ¿Cuál es el ciclo de vida de un Servlet y qué métodos intervienen en cada fase?
3. ¿Cuál es la diferencia entre los métodos doGet() y doPost() en un Servlet?

4. ¿Cómo se obtiene el objeto HttpServletRequest en un Servlet y para qué sirve?
5. ¿Qué método de HttpServletResponse se utiliza para enviar contenido HTML como respuesta?
6. ¿Cómo se almacena y se recupera información de una sesión en Jakarta EE?
7. ¿Qué es una cookie y cuál es su función en una aplicación web?
8. ¿Cómo se crea una cookie en un Servlet y cómo se envía al cliente?
9. ¿Cuál es la diferencia entre una sesión y una cookie en términos de almacenamiento de información del usuario?
10. ¿Qué método se utiliza para cerrar una sesión en un Servlet y eliminar todos sus atributos?
11. ¿Qué ocurre cuando se llama al método invalidate() en un objeto HttpSession?
12. ¿Cómo se obtiene una sesión en un Servlet y qué sucede si no existe una sesión previa?
13. ¿Qué es el método setMaxInactiveInterval() y cómo afecta a una sesión en Jakarta EE?
14. ¿Cómo se pueden recuperar todas las cookies enviadas por el cliente en un Servlet?
15. ¿Cuáles son las buenas prácticas de seguridad al utilizar cookies y sesiones en aplicaciones Jakarta EE?
16. Crear un Servlet que permita a los usuarios elegir su color de fondo favorito y almacenar esta preferencia en una cookie. Cada vez que el usuario regrese a la página, el fondo debe cambiar automáticamente al color almacenado en la cookie.
17. Implementar un sistema básico de autenticación utilizando sesiones en Jakarta EE.

CAPITULO 4

JAVASERVER PAGES

En el desarrollo web con Jakarta EE, los JavaServer Pages (JSP) juegan un papel crucial en la capa de presentación de las aplicaciones. Son componentes del lado del servidor diseñados para facilitar la generación de contenido dinámico en aplicaciones web, permitiendo la integración de HTML con código Java dentro de una misma página.

Los JSP son especialmente útiles para mostrar información que ha sido procesada por otros componentes, como los Servlets, y permiten construir interfaces dinámicas sin necesidad de generar manualmente cada línea de HTML dentro del código Java.

4.1 Funciones de los JSP

Un JSP contiene código HTML y a través de etiquetas vamos a agregar el código Java para poder manejar la cuestión dinámica dentro de las aplicaciones web.

En el desarrollo de aplicaciones web bajo la arquitectura MVC (Modelo-Vista-Controlador), los JavaServer Pages desempeñan un papel fundamental en la capa de presentación. Su función principal es mostrar la información procesada por el Servlet, permitiendo generar contenido dinámico basado en la interacción con el usuario.

Los JSPs no solo se limitan a mostrar datos en la interfaz del usuario, sino que también pueden enviar información de vuelta al Servlet a través de formularios HTML, facilitando así la comunicación ente la vista (JSP) y el controlado (Servlet).

4.2 Integración entre un JSP y Servlet en una Aplicación Web.

El flujo típico de una aplicación web con JSP y Servlet siguen el siguiente esquema:

- El usuario realiza una solicitud HTTP, por ejemplo, ingresa datos en un formulario o accede a una URL especifica.
- El Servlet recibe la solicitud, procesa información, interactúa con la lógica de negocio y la base datos.
- El Servlet envía los datos al JSP, una vez procesados, los datos son enviados al JSP mediante el objeto request.
- El JSP genera la respuesta HTML, muestra la información de manera dinámica basada en los datos proporcionados por el Servlet.

4.3 Ciclo de vida de un JSP

El ciclo de vida de un JSP es muy similar al ciclo de vida de un Servlet, la única diferencia que tenemos en el ciclo de vida comparado con un Servlet, es que un JSP se debe traducir a un Servlet, este es el primer paso que sucede una vez que se ha solicitado un recurso JSP.

Una vez que se traduce este JSOP a Servlet, este Servlet va a tener el mismo ciclo de vida que ya hemos estudiado en capítulos anteriores, es decir se va a compilar, se va a cargar la clase del Servlet en el web Server y posteriormente se crea un subproceso del servlet conocido como un hilo, se ejecuta el método init() del Servlet, se ejecuta el mpetodo service() del Servlet y por último se ejecuta el método destroy().

<entonces se observa que el único paso extra dentro del proceso de los JSPs, es que un JSP se va a traducir a un Servlet.

Figura 62.

Ciclo de vida del JSP

Nota. La imagen nos muestra el ciclo de vida de un JSP. *Fuente:* Los investigadores.

4.4 Elementos de un JSP

Para el desarrollo de aplicaciones web con Jakarta EE, los JavaServer Page (JSP) permiten generar contenido dinámico mediante la combinación de HTML y código Java. Dentro de un JSP existen distintos elementos que permiten incrustar código Java de manera estructura y eficiente. Estos elementos son:

- Expresiones JSP <%=%>
- Scriplets JSP <% %>
- Declaraciones JSP <%! %>
- Sintaxis XML en JSP

Cada uno de estos componentes cumple una función específica dentro de un JSP y es importante conocerlos para escribir código limpio, eficiente y mantenible.

4.4.1 Expresiones JSP

Las expresiones en JSP se utilizan para mostrar valores directamente en la página HTML. La sintaxis de una expresión es como se muestra en la Figura 63

Figura 63.

Expresiones en JSP

```
<%--
  Created by IntelliJ IDEA.
  User: ADMIN-ITQ
  Date: 16/3/2025
  Time: 22:20
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Expresiones en JSP</title>
</head>
<body>

    <%--Expresiones en JSP--%>
    <%= expresionJava%>

</body>
</html>
```

Nota. La imagen muestra un archivo JPS con la sintaxis de una expresión. *Fuente:* Los investigadores.

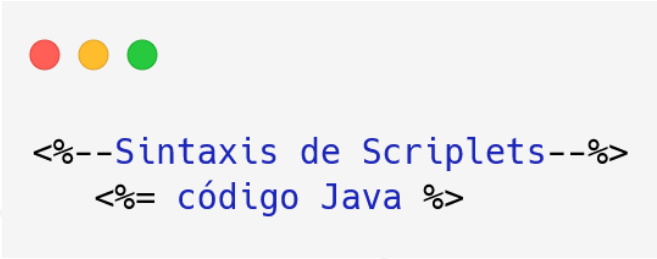
Una expresión de utiliza para imprimir valores en la respuesta HTML sin necesidad de escribir out.printnl(). Las expresiones pueden contener valores como cadenas de texto, cálculos matemáticos o el retorno de una función, las expresiones no pueden contener llamadas a métodos void, ya que siempre deben devolver un valor.

4.4.2 Scriplets JSPT

Los scriplets permiten escribir código Java dentro de un JSP. A diferencia de las expresiones JSP, un scriplets no imprime automáticamente el resultado en la respuesta HTML, sino que se usa para ejecutar lógica dentro del JSP.

Figura 64.

Sintaxis Scriplets



Nota. La imagen muestra la sintaxis para implementar un scriplets en JSP. *Fuente:* Los investigadores.

Para escribir scriplets en JSO existen varias buenas prácticas para implementarlo.

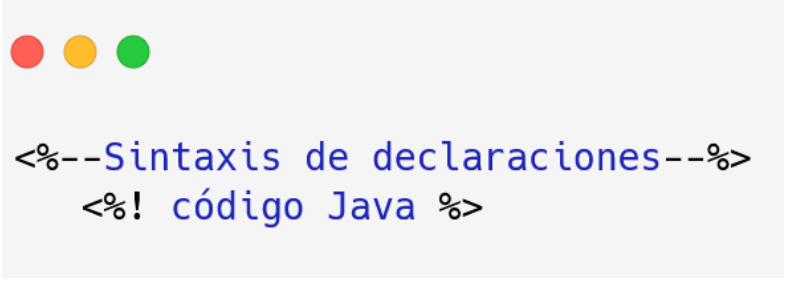
- Evitar escribir demasiada lógica de negocio a un Servlet o un Bean y solo usar JSP para mostrar la información.
- No abusar de los scriplets, ya que dificultan la separación entre la lógica de negocio y la presentación.

4.4.3 Declaraciones en JSP

Las declaraciones JSP permiten definir variables y métodos que serán parte de la clase del Servlet generado a partir del JSP. A diferencia de los scriplets, las variables definidas dentro de una declaración no son locales, sino que pertenecen a la clase del Servlet generado por el JSP.

Figura 65.

Sintaxis declaraciones JSP



Nota. La imagen nos muestra el código de una declaración en JSP la primera línea hace referencia a los comentarios de un JSP. *Fuente:* Los investigadores.

Los JavaServer Pages ofrecen múltiples formas de incrustar código Java dentro de una página Web, permitiendo generar contenido dinámico. Es importante conocer las diferencias entre Expresiones, Scriplets y Declaraciones.

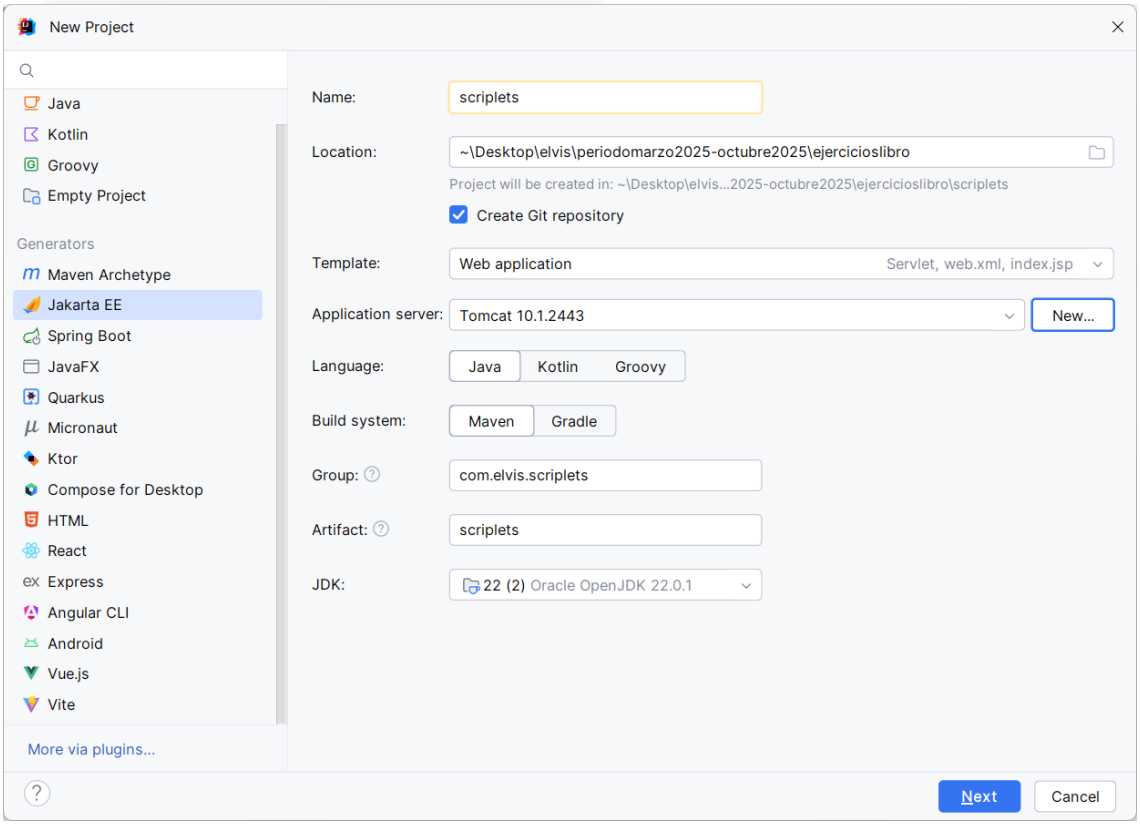
Ejercicio con Scriplets

Paso 1. Crear el Proyecto en IntellijIdea

- Abre IntellijIdea y selecciona File > New > Project
- Crea el proyecto.

Figura 66.

Creando el proyecto de Scriplets



Nota. La imagen nos muestra donde y como crear el proyecto. *Fuente:* Los investigadores.

- Configura el archivo pom.xml

Figura 67.

Configuración del archivo pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.elvis.scriplets</groupId>
    <artifactId>scriplets</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>scriplets</name>
    <packaging>war</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.target>11</maven.compiler.target>
        <maven.compiler.source>11</maven.compiler.source>
        <junit.version>5.10.0</junit.version>
    </properties>

   <dependencies>

            <dependency>
                <groupId>jakarta.platform</groupId>
                <artifactId>jakarta.jakartaee-api</artifactId>
                <version>9.1.0</version>
                <scope>provided</scope>
            </dependency>


        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <groupId>org.apache.maven.plugins</groupId>
                    <version>3.11.0</version>
                </plugin>

                <plugin>
                    <groupId>org.apache.tomcat.maven</groupId>
                    <artifactId>tomcat7-maven-plugin</artifactId>
                    <version>2.2</version>
                    <configuration>
                        <url>http://localhost:8080/manager/text</url>
                        <username>admin</username>
                        <password>elvis2103</password>
                    </configuration>
                </plugin>

                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>3.4.0</version>
                </plugin>
            </plugins>
        </build>
</project>
```

Nota. La imagen nos muestra el código donde se configura el archivo pom.xml, para poder levantar los servicios del servidor Tomcat y poder desplegar la aplicación en la web de forma local. *Fuente:* Los investigadores.

- Index.html

Figura 68.

Creación de Scriplets

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo de Scriplets JSP</title>
</head>
<body>
    <h1>Ejemplo de Scriplets</h1>
    <br>
    <a href="scriplets.jsp">Ejemplos de Scriplets</a>
    <br>
    <br>
    <form action="fondoColor.jsp">
        <label>Proporciona el color de fondo (ejemplo reed, blue, yellow, white, etc.)</label>
        <input type="text" name="colorFondo">
        <br>
        <input type="submit" value="Enviar">

    </form>

</body>
</html>
```

Nota. La imagen nos muestra el código de la creación de scripets en JSP. *Fuente:* Los investigadores.

- Crea el archivo JSP (scriplets.jsp)

Figura 69.

Manejo de Scriplets

```
<%--
  Created by IntelliJ IDEA.
  User: ADMIN-ITQ
  Date: 3/9/2025
  Time: 10:23
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>JSP CON SCRIPLETS</title>
</head>
<body>
    <h1>JSP CON SCRIPLETS</h1>
    <br>
    <%-- Scriplets para enviar información al navegador--%>
    <%
        out.print("Saludos desde un scriplets");
    %>
    <%--Scriplet para manipular los objetos implicitos--%>
    <%
        String nombreAplicacion= request.getContextPath();
        out.print("Nombre de la aplicación: " + nombreAplicacion);
    %>
    <br>
    <%--Scriplet con código condicional--%>
    <%
        if (session != null && session.isNew()){
    %>
    la sesión SI es nueva
    <%
        } else if (session != null) {


    %>
    la sesión NO es nueva
    <%}%>
    <br>
    <a href="index.html">Regresar al inicio</a>

</body>
</html>
```

Nota. La imagen muestra el código de cómo manejar scriplets en Java. *Fuente:* Los investigadores.

- Crear el archivo (fondoColor.jsp).

Figura 70.

Manejo de Scriplets

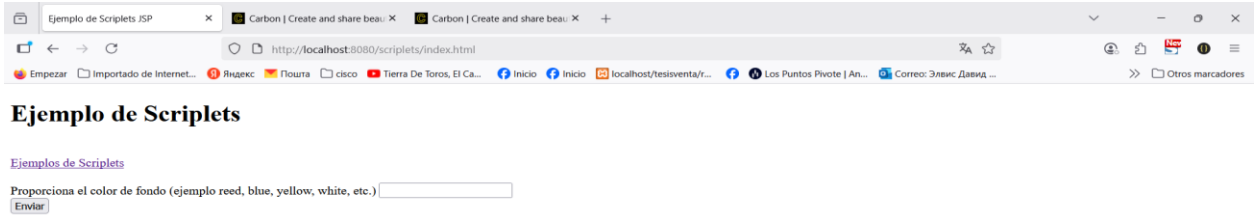
```jsp
<%--
  Created by IntelliJ IDEA.
  User: ADMIN-ITQ
  Date: 3/9/2025
  Time: 10:36
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    String fondo = request.getParameter("colorFondo");
    if (fondo == null || fondo.trim().equals("")){
        fondo = "white";
    }
%>
<html>
<head>
    <title>JSP cambio de color</title>
</head>
<body bgcolor="<%=fondo%>">
    <h1>JSP cambio de color</h1>
    <br>
    Color de fondo aplicado: <%=fondo%>
    <br>
    <a href="index.html">Regresar al inicio</a>
</body>
</html>
```

Nota. La imagen nos muestra el código de como implementar condicionales con scriplets. *Fuente:* Los investigadores.

- Salida de pantalla.

Figura 71.

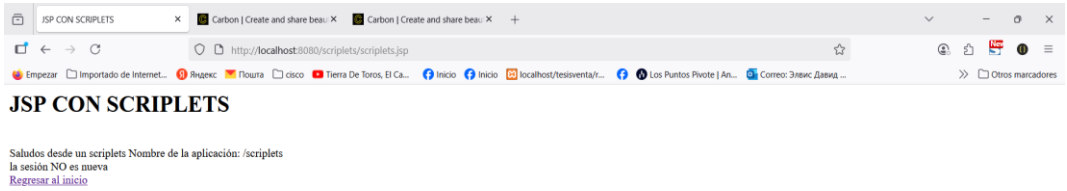
Salida de pantalla index.html



Nota. La imagen muestra la salida de pantalla del index.html donde haremos las peticiones a los JSPs. *Fuente:* Los investigadores.

Figura 72.

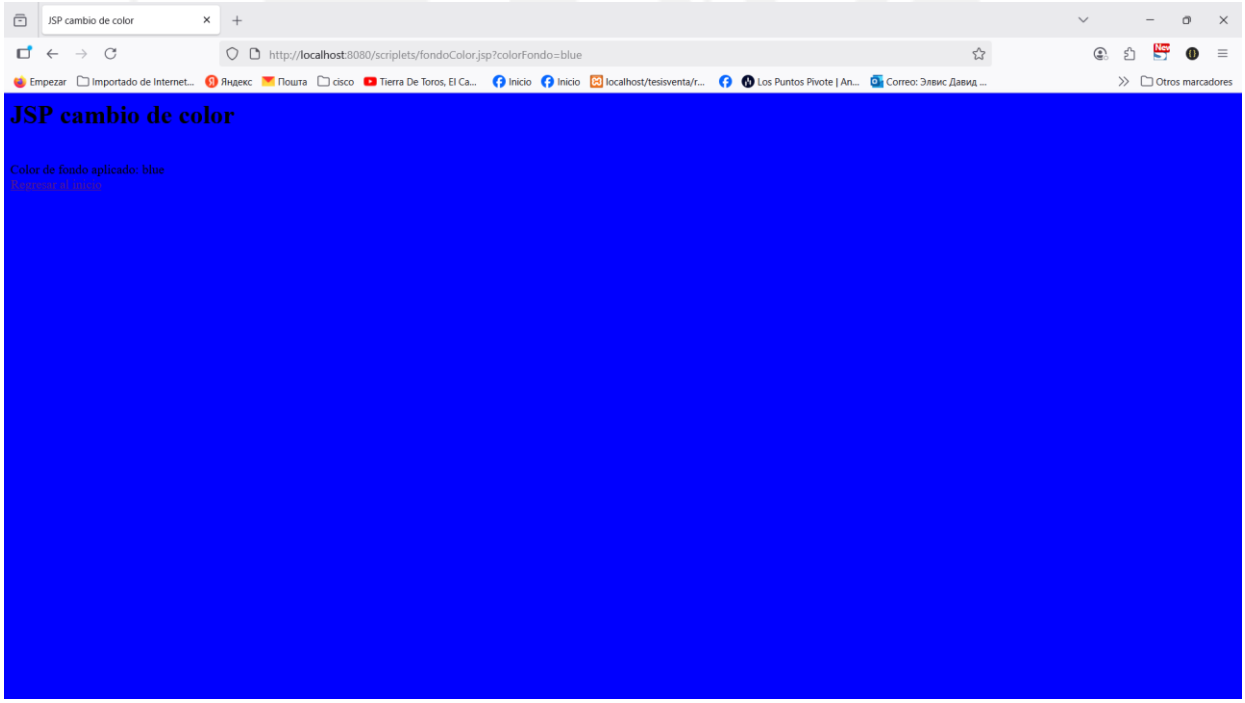
Salida de pantalla Manejo de Sesiones con JSP



Nota. La imagen muestra manejo de sesiones con jsp y scriplets. *Fuente:* Los investigadores.

Figura 73.

Manejo de colores Scriplets



Nota. La imagen nos muestra el fondo de color de pantalla al hacer la petición al servidor cambiando de color al momento en que enviamos la solicitud. *Fuente:* Los investigadores.

Ejercicios con declaraciones

Para el ejercicio vamos a realizar un contador de visitas usando declaraciones en JSP

4.5 Variables Implícitas en los JSP's

En JavaServer Pages existen una serie de objetos implícitos que están pre instanciados y listos para ser utilizados dentro del código, sin necesidad de declararlos o de inicializarlos manualmente. Estos objetos permiten interactuar con el ciclo de vida de la solicitud HTTP, la sesión del usuario, la respuesta del servidor y otros elementos fundamentales en el desarrollo web con Jakarta EE.

Entre los objetos implícitos más utilizados en un JSP, se encuentran:

- Request, representa la solicitud HTTP y permite accede a parámetros y cabeceras.
- Response, contiene la respuesta HTTP enviada al cliente
- Out. Se usa para enviar datos al cliente, similar a PrintWriter en los Servlet.
- Sesión, permite acceder a la sesión activa del usuario HttpSession.
- Application, representa el contexto de la aplicación web

Estos objetos no necesitan se instanciados en el JSP, ya que el contenedor de servlets los proporciona automáticamente, en cada ejecución de la página.

4.6 Directivas de un JSP

Las directivas de un JSP permiten controlar el comportamiento de un JSP, por ejemplo, las clases que se utilizan dentro de un JSP y al momento de hacer el import de la clase Java, se debe especificar el tipo de MIME (Multipurpose Internet Mail Extensions) con el que vamos a responder a nuestro cliente.

También con estas directivas podemos especificar si un JSP va a participar en el concepto de sesiones, o se especifica el tamaño del buffer de salida, especificar el JSP de error en caso de que tengamos alguna excepción dentro de nuestro JSP, o indicar si nuestro JSP va a manejar el concepto de multihilos, entre varias características más.

4.7 Atributos de las Directivas de un JSP7

Algunos de los atributos que podemos indicar en la directiva page son:

Import: El atributo de import dentro de la directiva page va a permitir especificar cuáles son las clases que se debe importar dentro del JSP e indirectamente al servlet generado a partir de la solicitud al JSP. Aunque el import se lo puede declarar en cualquier parte del JSP, lo más recomendables es ponerlo al principio como una buena práctica.

Para especificar las clases que se desea importar en el JSP se puede utilizar la sintaxis como se puede observar en la *Figura 74*.

Figura 74.

Directiva JSP.

```
3    <%@ page import="paquete.Clase1, paquete.ClaseN" %>
```

Nota. La imagen muestra cómo se debe utilizar la directiva import en un archivo JSP para llamar a distintas clases del proyecto. *Fuente:* Los investigadores.

Se puede detectar que es una directiva debido a que estamos utilizando ahora el símbolo @ y posteriormente el nombre page, seguido del atributo import y se indica el nombre completo calificado de las clases, es decir, incluimos el paquete y el nombre de la clase, si se quiere especificar más de una clase a utilizar vamos a separarlas por coma.

Posteriormente se tiene el atributo contetType <%@page contentType=”MIME-TYPE” %>. Este atributo nos permite especificar el tipo de respuesta a nuestro cliente web.

Session: <% @page sesión=” true” %> Tambien se puede indicar por medio de esta directiva page si el JSP va a poder acceder al objeto sesión que se haya creado anteriormente por ejemplo desde un JSP o desde otro Servlet. Por defecto un JSP está configurado para que podamos acceder al objeto session, si se quiere indicar lo contrario se tendrá que especificar false en esta directiva lo que sucede que dentro de las clases implícitas de un JSP ya no se va a tener acceso a la variable.

isELIgnored:<%@ page isELIgnored=”false” %> También se tiene otro atributo llamado isELIgnored esto lo que significa es que si se queremos deshabilitar el manejo de EL (expresión Languages), tendríamos que indicarlo por medio de esta directiva page indicando el atributo isELIgnored el valor de true, por defecto un JSP a partir de la versión 2.4 puede utilizar de manera automática el concepto de EL pero anterior a la versión 2.3 de los servlets los JSPs por defecto no pueden acceder al lenguaje conocido como EL.

RESUMEN DEL CAPÍTULO 4

Los JavaSever Pages son una tecnología clave en Jakarta EE utilizada para la generación dinámica de contenido web. A diferencia de los Servlets, los JSP's permiten escribir código HTML con código Java incrustado, facilitando la construcción de interfaces web interactivas si necesidad de generar HTML manualmente en un Servlet.

En este capítulo hemos explorado los conceptos fundamentales de los JSP, su ciclo de vida, los elementos principales que los componen y la forma en que interactúan con otros componentes como Servlets, sesiones y objetos implícitos.

Un JSP es una página web que puede contener tanto HTML estático como código Java dinámico. Se utilizan principalmente en la capa de presentación, de una aplicación web, siguiendo el modelo MVC, donde los JSP cumplen el rol de la vista y los Servlet el rol de controlador.

En Java, tanto la clase **"**vector", como la clase "ArrayList" forman parte del marco de colecciones y se utilizan para almacenar grupos de objetos. Aunque ambos ofrecen funcionalidades similares al manejar listas dinámicas, hay ciertas diferencias claves en cómo se comportan y se utilizan en diferentes contextos.

Estas clases almacena secuencias de objetos de cualquier tipo: las colecciones este tipo de estructura de datos se diferencian en la forma de organizar los objetos y, por consecuencia en la manera en las que se puede recuperar.

Ejercicios propuestos

1. ¿Qué es un JSP y cuál es su función en Jakarta EE?
2. ¿Cuál es la principal diferencia entre un JSP y un Servlet en una aplicación web?
3. ¿Cómo funciona el proceso de conversión de un JSP a un Servlet?
4. ¿Cuál es la extensión de los archivos JSP y en qué carpeta deben ubicarse dentro de una aplicación web?

Referencias

Ahmad Dar, M. (2020). *JAVA Programming Simplified.* Walter de Gruyter GmbH.

ciniguez. (s.f.). Obtenido de ciniguez: https://ciniguez.github.io/balava/appwebcom/protocolo_http.html

creativecommons. (s.f.). Obtenido de creativecommons: https://creativecommons.org/licenses/by-sa/3.0/

Deitel, H., & Feitek, P. (2019). *Java: Cómo Programar (11° edición),.* Person Eduacation.

Deitel, P. (2014). *Java SE8 for Programmers.* Pearson Education.

elpesodeloslunes. (s.f.). Obtenido de elpesodeloslunes: https://elpesodeloslunes.wordpress.com/wp-content/uploads/2014/09/mvc-diagrama.png

Ernest, M. (2012). *Java SE 7 Programming Essentials.* Wiley.

Garnica, C. C. (s.f.). *Principios Básicos de la POO | POO en JAva*.

Garrido Abenza, P. (2015). *Comenzando a programar con JAVA.* Universidad Miguel Hernández.

Goling, J., Joy, B., & Steele, G. (2018). *The Java Programming Languaje.*

helenjoscott. (s.f.). Obtenido de helenjoscott: https://www.helenjoscott.com/wp-content/uploads/2020/09/1200px-IntelliJ_IDEA_Logo.png.

Hennessy, J. L., & Paterson, D. A. (2019). *Arquitectura de Computadoras: Un enfoque Cuantitativo.* Pearson Education.

https://info.cern.ch/. (s.f.). Obtenido de https://info.cern.ch/: https://info.cern.ch/

i.blogs. (s.f.). Obtenido de i.blogs: https://i.blogs.es/2173cf/welcome_to_netscape/1024_2000.jpg

i.blogs.es. (s.f.). Obtenido de i.blogs.es: https://i.blogs.es/5bfa64/mosaic3/1024_2000.jpg.[

Joyanes Aguilar, L. (2011). *Programación en Java 6 algoritmos, programación orientada a objetos.* Mexico: McGraw-Hill Interamericana de España S.L.; 1er edición (16 Septiembre 2011).

Joyanes Aguilar, L., & Zahonero Martinez, I. (2011). *Programación en Java 6.* Mexico: McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V.

Mughal, K. (2016). *A Programmer's Guide to Java SE 8 Oracle Certified Associate (OCA).* Pearson Education.

Rodríguez Rancel, M. L. (2012). *Aprender a programar en Java desde cero.* aprenderaprogramar.com.

Schildt, H. (2014). *Java 8. .* Anaya Multimedia.

Toro, J. (09 de 10 de 2018). *Tipos de Progrmas que se pueden hacer en Java.* Obtenido de Applets: http://joseltoro.blogspot.com/2018/10/que-tipos-de-programas-se-pueden-hacer.html

Грин, А. и. (2019). *Java Swing. Создание графического интерфейса. .* Питер.

Шилдт, Г. (2019). *Java. Полное руководство (11-е издание).* Вильямс.

Cadenhead, R. y Lemay, L. (2007), Teach Yourself Java 6 in 21 Days, Indianápolis: Sams

Caules, C. Á. (2023). Las versiones de Java y su historia. Arquitectura Java. https://www.arquitecturajava.com/las-versiones-de-java/