|  |  |  |  |
| --- | --- | --- | --- |
| **ARQUITECTURA WEB** | **10/01/2023** | | |
|  | | |
|  | | |
| Cualquier tipo de aplicación Web de Java tiene arquitectura de programa Cliente y programa Servidor, que se comunican entre sí.  El Servidor Web suele ser pasivo y están disponibles 24h los 7 días de la semana. A menos que tengan una incidencia, mantenimiento o estén caído. Usaremos, y que es más usado, es “TomCat”. | |  | |
| ¿Qué es un contenedor en programación orientada a objetos?  Los contenedores son una forma de virtualización del sistema operativo. Un solo contenedor se puede usar para ejecutar cualquier cosa, desde un microservicio o un proceso de software a una aplicación de mayor tamaño.  El CLIENTE puede ser un navegador (implementado por html, css, JavaScript, JFS para móviles,…).  El PROTOCOLO es el flujo que conecta cliente y servidor. Puede ser HTTP o HTTPS  El protocolo http funciona mandando solo texto, en un tipo de archivo. Características:   * Es de petición y respuesta. Es un proceso síncrono. * Normalmente no es visible al usuario (cliente).   Con Java se utiliza el Web Socket para realizar petición al servidor.  Con JavaScript se utiliza AJAX, el cual fue creado para simular el tiempo de recarga o espera.  PROTOCOLO HTTP es SIN ESTADO 🡪 este no guarda ninguna petición anterior o presente. Esto suele dar problemas en lado servidor, cuando se trabaja ahí, porque para muchas veces es necesario tener un recuerdo de estas peticiones, para poder realizar las operaciones que se necesite. Habrá que plantear unas buenas “**Estrategias de aseguramiento de peticiones**”.  Características del protocolo HTTP:   |  |  | | --- | --- | | * Es un TEXTO. * Es PETICIÓN/RESPUESTA * Es SIN ESTADO | Estas nos obligan a hacer en el lado del Servidor una serie de planteamientos que nos ayuden a saltar o utilizar estos aspectos del http. | | | | |
|  |  | | |
| Apache **TOMCAT** (Jakarta Tomcat) es un contenedor de SERVLETS que se puede usar para compilar y ejecutar aplicaciones web realizadas en Java. Implementa y da soporte tanto a SERVLETS como a páginas JSP (Java Server Pages) o Java Sockets.  Lo utilizaremos como intermediario al servidor ya que es simple y fácil de usar, que recibirá las peticiones que hagamos (como un servidor). Dentro podremos tener uno o muchos servidores y aplicaciones, como por ejemplo puede darse en el caso de una granja de servidores. Con el podríamos mandar a todos esos servidores peticiones de acción y estos servidores pueden tener cada uno miles de aplicaciones ejecutándose a la vez.  CLIENTE 🡪 HTTP 🡪 TOMCAT 🡪 APLICACIÓN (en lado del servidor)  TomCat sabe a cuál servidor debe de pasarle la información por la URL que viene en la información del fichero.  Nosotros programaremos COMPONENTES 🡪 que harán modificar, pintar gráficos, estadísticas, etc.  Características a tener en cuenta para los componentes:   |  |  | | --- | --- | | * El objeto lo crea el Servidor: ni lo instanciamos y ni creamos. * Servidor es el que llama a los métodos, y ni nosotros llamamos a componentes. * Son destruidos por el Servidor. | GESTIONADAS  POR SERVIDOR | | | | |
|  |  | | |
| Lo que si hacemos es que escribimos la clase correspondiente para utilizar ese componente.  Los componentes que podemos hacer son:   |  |  | | --- | --- | | * Servlet 🡪 más habitual * Filtros | Se diferencian en el TIME, cuando son llamados por el servidor hay que aprenderse el ciclo de vida de los componentes para ver cómo funcionan. | | Los “escuchadores de eventos del servidor” 🡪 se definen con implementación de interfaces. | | | | | |
|  | | | |
| ¿Para hacer una aplicación en JEE que más se puede necesitar?  CONTEXTO  En Java son objetos que tienen un ciclo de vida.   * Sirven para llamar a los métodos que necesitamos, estos los tienen ya predefinidos. * Tienen en común que gestionar atributos (es un “map” donde se guardan los objetos, con una clave única y valor. Los map sirven para tracking de sesión.   Tipos de “contexto”, que vienen de clases o interfaces distintos:   * APLICACIÓN   + Es el mismo para todas las peticiones y aplicaciones que lleguen.   + Se crean al arrancar la aplicación (por esto se dice que es única para este evento).   + Se destruyen al parar la aplicación (por esto se dice que es única para este evento).   *Ejemplo: aplicaciones web privadas que te piden identificarte una vez y ya guardan esa información.*   * SESIÓN   + Habrá tantos como usuarios conectados. Lo garantiza el servidor que es quien lo mantiene. Mueren al salir 🡪 Servidor crea y destruye.   + Haremos seguimiento de la sesión.   *Ejemplo: el espacio que nos facilita Mega para subir archivos.*   * PETICIÓN   + Tantos como lleguen de parte del usuario.   + Dura lo que dure esta, aunque el usuario tenga otras peticiones.   + Sirve para pasar información que tenga validez en tratamiento que dure la petición. | | | |
| ***SERVLET***  Los servlets son programas Java™ que utilizan la interfaz de programación de aplicaciones (API) de servlet Java. Debe empaquetar los servlets en un archivo WAR (archivador de aplicación web) o un módulo web para su despliegue en el servidor de aplicaciones.  Funcionamiento: El Servlet procesa los argumentos de la petición, es decir, el contenedor de Servlets delega la petición a un Servlet en particular elegido de entre los Servlets que contiene. El Servlet, que es un objeto java, se encarga de generar el texto de la página web que se entrega al contenedor. | | | |
| **CICLO DE VIDA DEL SERVLET**  Primero llama al método INIT (que tiene código vacío en la clase padre), el cual debemos sobrescribir para que haga algo. Cuando llegue la petición a este componente, servidor llamará al método.  SERVICE si tiene código y analiza que tipo de protocolo llega de http y según el cual, llama a uno u otro método. Hay 7 tipos de métodos o peticiones que funcionan en distintos protocolos es http.  Lo más utilizados son el Get (DoGet) y Post (DoPost), ambos vacíos. Dependerá del tipo de petición que haya llamado el navegador.  El método Service no se toca. Para modificar DoGet y DoPost hay que sobrescribirlo.  DESTROY se llama una única vez.  El proceso sería así: INIT se ejecuta una vez cuando arranca, cuando llega la petición se crea múltiples SERVICE (Get y Post), y se destruye cuando acaba. | | |  |
| Se crean CLONACIONES para que así se puedan responder a un número múltiple de peticiones, se consigue responder más rápido y se evitan instancias de objetos y destrucciones. | | | |
| **ECLIPSE** | | | |
| Primera vez que abrimos, nos pide Workspace, indicamos ruta de nuestra carpeta de trabajo, aconsejable que este fuera de la carpeta de eclipse y que tenga una ruta en la que no tenga ni espacios y ni caracteres extraños, por ejemplo: D:\00Curso\CursosHarzeta\CURSO-Hazerta-1-2023\WORKSPACE  Damos a “Launch”   |  |  | | --- | --- | | Tras abrirse eclipse le damos, lo de a continuación se puede hacer de diferente formas:   1. Boton derecho New/Project/Web/Dynamic Web Project 2. File/new/ Dynamic Web Project   En la ventana que aparece ponemos:   * Project name: (nombre sin espacios) * Target runtime: (elegiremos un servidor si tuviésemos alguno, aunque se puede agregar más tarde) * Dynamic Web Module Version: (versión Java EE con la que se realiza nuestro proyecto) 3.1 = JEEv.7   (next)  Pantalla “Configure project for building a Java application” 🡪 dejamos igual  (next)  Pantalla “Configure web module settings” 🡪 señalamos “Generate web.xml deployment descriptor  (Finish) | Así quedaría la estructura: | |  |   web.xml🡪 es muy importante porque en cualquier proyecto de Arquitectura Web señalar esta pestaña porque nos crea una serie de carpetas necesarias más adelante para nuestro proyecto.  Para crear nuestra primera aplicación web crearemos primero un paquete donde estará todo nuestro proyecto. Encima de “src/main/java” le damos botón derecho y new/package. Le ponemos de nombre por ejemplo “com.servlet” y no tenemos señalada ninguna pestaña. (Finish)  A continuación botón derecho encima del paquete ya creado anteriormente y le damos a “Servlet”:   * Class Name: prueba\_Servlet   (next)  Pantalla 2- Enter servlet deployment descriptor specific information 🡪 opciones configuración de este componente especifico   * Name: (es opcional) se puede dejar mismo nombre clase o cambiar, es nombre componente y lo que no puede es repetir al nombre de otro componente que ya exista * Description: (es opcional, pero aconsejable) se describe brevemente lo que hará el componente * Url Mappings: (se deja lo que nos aparezca, pero se puede cambiar) / prueba\_Servlet 🡪 le indicamos al servidor a quien debe distribuir la peticion   (next)  Pantalla 3-Specify modifiers, interfaces to implement, and method stubs to generate 🡪 método que contendrá nuestro servlet, clases que se crearan por herencia y heredadas de la clase padre, normalmente están vacias.   * Señalamos: doPost y DoGet   (finish)  Ya estaría todo creado, y genera la class, aparecen errores porque no se han añadido las librerías (de los import y métodos). Eso lo añadiremos adjuntándole, además, un servidor TomCat (en este caso). En la siguiente clase. (imagen código que nos aparece) | | | |
|  |  | | |