PASOS PARA LA CREACION DE UN WEBPROYECT CON SERVLETS

# Paso 1: creación de la BD

Se creara la BD en el motor de MySql workbench. Se creara un Schema nuevo, y trabajaremos sobre el.

Crearemos las tablas necesarias para la BD. Recordad que cada tabla tendrá siempre un campo id, que será not null, primary key e auto increment.

Una vez creada la BD se rellenara con datos todas las tablas para poder tener un juego de datos de prueba

# Paso 2: creación del proyecto web dinámico

Abriendo eclipse, crearemos un nuevo proyecto web dinámico, dándole un nombre

Se mavenizara el proyecto. Para ello haciendo click derecho sobre el nombre del proyecto iremos a configure y a convert to maven. En el archivo pom de configuración añadiremos las dependencies necesarias para incluir el driver jar de conexión a BD mysql

<dependencies>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.49</version>

</dependency>

</dependencies>

Este bloque ira justo antes de la ultima etiqueta del achivo pom.xml

Dentro de este proyecto, crearemos la siguiente estructura de paquetes:

* com.ipartek.controlador
* com.ipartek.modelo
* com.ipartek.modelo.dto

En el paquete modelo crearemos (o moveremos de algún otro proyecto) una clase llamada DB\_Helper y una interfaz llamada I\_conexion. (los nombres pueden cambiar pero en el ejemplo usaremos estos)

En la clase DB\_helper, de momento deberemos contar solo con dos funciones, conectar y desconectar. Mas adelante, para cada acceso a la base de datos que hagamos, deberemos tener una función mas, que realizara la tarea correspondiente asociada.

Para la interfaz, crearemos 5 constantes en las que definiremos los datos de conexión a la BD. Recordad poner los datos correctos de usuario y conexión a la BD MySql nuestra. Las cadenas de conexión y driver que dejo en el ejemplo funcionan para el driver de conexión versión 5.1.49. otras versiones podrían modificarlos

String ***BD\_DRIVER*** = "com.mysql.jdbc.Driver";

String ***BD*** = "fruteria";

String ***CADENA\_CONEXION*** = "jdbc:mysql://localhost:3306/" + ***BD*** + "?useSSL=false";

String ***USUARIO*** = "root";

String ***PASS*** = "1234";

Una vez rellenado estos datos, en el archivo DB\_Helper haremos que implemente la Interfaz. Esta clase e interfaz en conjunto también son llamadas DAO

Ahora es el turno de los DTO. Estos serán una clase y habrá uno por cada Tabla de la BD. Cada uno de ellos lo llamaremos como la tabla a la que va a referenciar, aunque con el nombre el singular. Esta es la convención mas habitual aunque también puede darse el caso de encontrarlos nombrados en plural, o con nombres distintos como “*dto\_nombreTabla*” o similares

Cada uno de estos archivos tendremos que definir los atributos de cada clase. Estos se definirán de manera privada y luego mediante el menú contextual del botón derecho, en el apartado Source del menú que sale crearemos los constructores necesarios, los métodos get y set y de manera opcional (aunque es recomendable ponerlo) el método toString

# Paso 3: comprobación de funcionamiento

Para comprobar el buen funcionamiento de conexión a la BD, crearemos un archivo JSP y un servlet. Para realizar esta tarea, y que cada uno se coloque en su sitio, lo mejor es hacer click derecho en el nombre del proyecto, y crearlo desde allí. El servlet si nos pedirá que le indiquemos el paquete donde ira. Estos deberán colocarse en el paquete com.ipartek.controlador.

Dentro del archivo JSP que creemos, en este caso lo llamaremos index.jsp, pondremos una etiqueta de enlace <a> y en el href indicaremos el nombre del servlet. El nombre del servlet coincide normalmente con el nombre del archivo .java que lo contiene. **Respetad mayúsculas y minúsculas y recordad que al ser un servlet, no lleva extensión .java en el href**

Una vez hecha dicha tarea, modificaremos el código del servlet. Borraremos todo código de comentarios, para poder escribir bien. Ojo, no eliminar ninguna línea como esta.

@WebServlet("/ObtenerFrutas")

Ya con el código limpio, procederemos a rellenar código dentro del método doGet. Lo haremos aquí siempre, aunque la info se le mande por Post, ya que el método doPost nos reenviara a esta función. En esta sección deberemos rellenar los datos de los siguientes apartados, según proceda. Es un procedimiento estándar, y suele seguir estos pasos, aunque no siempre podremos encontrar todos, y pudiendo encontrarse con alguna variación en algún caso particular pero siendo la mayoría de casos aplicables este patrón:

## Obtención de datos

Obtendremos los datos que se nos envíen. Este apartado es solo aplicable si al servlet le llegan datos desde un formulario

Estos datos tendrán la siguiente estructura, se recogerán habitualmente en formato String. En casos avanzados, puede ser opcional realizarles un cast

String user = "";

**if** (request.getParameter("nombre") != **null**) {

user = request.getParameter("nombre");

}

## Maquetar los datos a DTO

Si se diese el caso de que recibimos datos que pudiesen meterse en un DTO, meteremos esos datos en un objeto adecuado de los DTO para así poder trabajar cómodo con esa información

## Conexión a la BD

Este paso siempre es igual. Aquí realizaremos la conexión a la BD. Es copiar y pegar estas líneas de código, que se encargaran de abrir la conexión a la BD

DB\_Helper db = **new** DB\_Helper();

Connection con = db.conectar();

## Realización de la funcionalidad

Este paso es sencillo. Simplemente llamaremos a las diferentes funciones que acceden a la BD. Estas se encontrarán en la clase DB\_Helper. Solo deberemos llamarlas y si son funciones que devuelven datos, procesarlos.

Si por ejemplo, queremos obtener todos los productos de la BD, deberemos tener en el DB\_Helper la siguiente función, devolverTodosProductos(con) y la lista que devuelva, recogerla

## Cierre de conexión

Una vez realizadas todas las tareas, cerraremos la conexión a la base de datos. Para ello usaremos la función desconectar que tenemos en el DB\_Helper.

## Empaquetado de datos

Este paso solo lo realizaremos si deseamos enviar un dato a la vista que cargaremos para salir del servlet, ya que siempre el proceso debe acabar en una vista JSP. Para ello por cada dato que queramos enviar, añadiremos un request.setAtribute para cada uno que queramos enviar. Los parámetros de esta función son un string que definiremos el nombre del atributo, como lo cogeremos en la vista a la que va destinada, y la variable que deseamos enviar

request.setAttribute("listadoAtributo",listado);

## Redireccion a una vista

Una vez acabada la tarea, no podemos quedarnos en el servlet. Al ser controlador, este no muestra interfaz de usuario, por lo que redirigiremos a una vista (normalmente un archivo jsp). Lo haremos con la sentencia request.getRequestDispatcher, pero recordad para que redirija deberemos añadirle la clausula forward.

request.getRequestDispatcher("mostrar.jsp").forward(request, response);

## recomendaciones

Realizaremos estos pasos por cada servlet. Como recordatorio, los servlets deben tener una funcionalidad minima, ser lo mas sencillos posibles ya que ello favorecerá la modularidad del proyecto

# Paso 4: gestión de las Vistas

El ultimo paso, es la gestión de las vistas. Tras realizar un servlet siempre acabaremos en una vista o archivo JSP, para lo cual puede darse la situación que el servlet le haya enviado informacion para que la vista la muestre en pantalla.

Si se diese ese caso, en la vista deberemos recoger esa informacion que nos envía el servlet que nos redirige. Esta info es la que hemos redactado en el apartado 6 indicado en el paso anterior. Para recogerla, el proceder será similar al apartado uno, con alguna ligera variación

<%

List<Registro> listadoTodosAtributo = **new** ArrayList<Registro>();

**if** (request.getAttribute("listaCompleta") != **null**) {

listadoTodosAtributo = (List<Registro>) request.getAttribute("listaCompleta");

}

%>

La diferencia seria que debemos realizar un casting implícito dentro del if ya que lo que pasamos son objetos de la clase Object y al recibirlos eso puede ser cualquier cosa, por lo que lo castearemos a los datos que realmente son.

Como medida adicional, en algún sitio del código html del documento jsp, deberemos indicar los includes de listas, arraylist y de los dto que usemos. Estos serian un ejemplo

<%@ page import=*"java.util.List"*%>

<%@ page import=*"com.ipartek.modelo.dto.Registro"*%>

<%@ page import="java.util.ArrayList"%>

Hecho esto, ya tendremos los datos preparados para recibirlos, pero recordemos que deben pasar por el servlet que es el que se encarga de enviárnoslos. Al hacerlo de esta manera, si un usuario avispado intentase saltarse este proceso no colgara la página al tratar los null que llegaran a esas variables al no seguir el proceso

Ya solo resta mostrar esa información. Para mostrarla, mediante un scriptlet realizaremos esa tarea. Si la informacion fuese una lista, la recorreremos con un bucle foreach, o si fuesen datos por ejemplo contenidos en un objetos los mostraremos directamente. Ver los ejemplos suministrados para ver su realización

## Mostrar datos de una lista

<%

**for**(Lugar lug:listaLugaresMostrar)

{

%>

<p>

id: <%=lug.getId() %>, lugar: <%=lug.getLugar()%>

</p>

<%

}

%>

## Mostrar datos de un objeto

<p>

<%=fruta.getId() %>: precio <%=fruta.getPrecio()%> Euros

</p>