

Ministerio de Educación Superior, Ciencia y Tecnología (Mescyt)

Centro de Tecnología y Educación Permanente TEP

Pontificia Universidad Católica Madre y Maestra (PUCMM)

## **Diplomado en Programación en Lenguaje JAVA**

### **Laboratorio 01**

Módulo III

### **Programación Web Java**

Facilitador: Ing. Eudris Cabrera

Fecha : 23 Julio del 2016

Santiago de los Caballeros, República Dominicana

## Introducción

### ¿Qué es una aplicación web?

'Por definición, se trata de algo más que un 'sitio web'.

Se trata de una aplicación cliente / servidor que utiliza un navegador Web como su programa cliente, y por consiguiente constituye un servicio interactivo mediante la conexión con los servidores a través de Internet (o Intranet).

Una aplicación web presenta contenido adaptado dinámicamente en función de parámetros de la petición, los comportamientos de los usuarios seguidos, y consideraciones de seguridad.

Una aplicación Web Java puede ser representada como una jerarquía de directorios y archivos, que a su vez contiene:

- Componentes Web (Servlets, JavaServer Pages, entre otros)
- Recursos estáticos (páginas html e imágenes).
- Clases Java.
- Librerías (Archivos Jars).
- Un archivo descriptor de despliegue (web.xml).

Una aplicación web de Java se puede implementar como un archivo **".war"**.

El archivo **".war"** es un archivo zip que contiene todo el contenido de la aplicación web correspondiente.

Las aplicaciones java web normalmente no se ejecutan directamente en el servidor, sino que se ejecutan dentro de un **contenedor** en el servidor. El contenedor proporciona un entorno de ejecución para aplicaciones web en Java.

El contenedor es para aplicaciones web en Java lo que la JVM (Java Virtual Machine) es para las aplicaciones Java que se ejecutan locales. El contenedor en sí se ejecuta en la JVM.

En general, Java distingue dos contenedores: El **contenedor web** y el **contenedor Java EE**.

Un contenedor Web apoya la ejecución de Servlets Java y Java Server Pages.

Un contenedor compatible con Java EE provee funcionalidades adicionales, tales como, gestor de ejecución de los Enterprise JavaBeans, interfaz de conexión entre el servidor Java EE y aplicaciones clientes.

Contenedores web típicos en el mundo Java son **Tomcat** o **Jetty**.

La especificación **Servlet** es el fundamento de las aplicaciones web Java. Muchos frameworks de alto nivel incluyendo **JAX-RS**, **JavaServer Faces (JSF)**, y **Spring MVC** están basados en **servlets** y **JavaServer Pages**.

Los frameworks web populares en Java son: Spring, Java Server Faces y Vaadin. Estos frameworks web por lo general se ejecutan en un contenedor web.

## Guía del laboratorio

Este laboratorio está enfocado en las tecnologías Servlet/JSP, elementos básicos en el desarrollo de aplicaciones Web con Java.

## Requerimientos

Java 1.7+ (Recomendado Java 8)

Netbeans 8.0.1+

Apache Maven 3.2.1+

Tomcat 7.0.54 ó superior (Recomendado Apache Tomcat 8.0.27.0 ó superior)

Mysql Server 5.0+

## Escenario

Se desea crear un proyecto en forma de **task list** para llevar un registro de nuestras tareas.

Nuestro objetivo principal es aplicar los conceptos de **jsp** and **servlets** desarrollando una pequeña aplicacion web java.

## Creación y configuración de la base de datos.

Crear una base de datos en mysql llamada **taskapp\_diplomado**.

```
create database taskapp_diplomado;
```

Ejecutar el archivo **taskapp\_diplomado.sql** situado en el directorio **programacion-web-java/labs/lab01/sql** dentro del repositorio del diplomado.

### Paso I

Crear una aplicación **Maven** del tipo **Web Application**, ir a la opción: **New Project -> Maven -> Web Application**.

### Información del proyecto

Nombre : **taskapp-web**

artifactId: valor por defecto

groupId: **org.diplomado.pucmm.mescyt**

version: **valor por defecto "1.0-SNAPSHOT"**

package: **org.diplomado.pucmm.mescyt.taskapp.web**

### Modificar archivo pom.xml

Remover la dependencia **javaee-web-api**, agregada por defecto y agregar las siguientes dependencias.

### Servlet

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

## Mysql

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.36</version>
</dependency>
```

## Paso II

Crear la siguiente estructura de paquetes

<b>org.diplomado.pucmm.mescyt.taskapp.web.controller</b>	Para las clases que tienen que ver con la navegación.
<b>org.diplomado.pucmm.mescyt.taskapp.web.utilidades</b>	Paquete para agrupar cualquier clase que sea un utilitario.
<b>org.diplomado.pucmm.mescyt.taskapp.web.entidades</b>	Clase que representan el dominio o negocio (POJO).
<b>org.diplomado.pucmm.mescyt.taskapp.web.servicios</b>	Para contener las clases que tienen que ver con acceso a datos.
<b>org.diplomado.pucmm.mescyt.taskapp.web.filtros</b>	Filtros para el manejo de acceso y autenticación
<b>org.diplomado.pucmm.mescyt.taskapp.web.sesiones</b>	Contener las clases para iniciar o concluir una sesión.

## Paso III:

Crear clase que representen las tablas que tenemos en la base de datos.

Crear clase **Task** en el paquete **entidades** y crear métodos setter y getter.

## Paso IV:

Configurar datasource en el archivo **META-INF/context.xml**

**Datasource** (Fuente de datos) es un nombre dado a la conexión establecida para una base de datos desde un servidor. Se utiliza comúnmente cuando se crea una consulta a la base de datos.

Según **Sun Microsystems** un objeto **DataSource** representa una fábrica (factory) para las conexiones a la fuente de datos. Es una alternativa a la instalación de DriverManager, un objeto de origen de datos (DataSource) es el medio preferido para conseguir una conexión. Un objeto que implementa la interfaz **DataSource** típicamente se ha registrado en un servicio de nombres basado en la API **Java Naming and Directory Interface (JNDI)**.

La interfaz de origen de datos (Datasource) se lleva a cabo por un proveedor del controlador.

Hay tres tipos de implementaciones:

**Implementación básica** - produce un objeto de conexión estándar.

**Implementación agrupación de conexiones (Connection pooling).**

Produce un objeto Connection que participará automáticamente en la agrupación de conexiones. Esta aplicación funciona con un administrador de conexión a nivel de un la capa media ( middle-tier).

**Implementación transacción distribuida**

Produce un objeto de conexión que puede utilizarse para transacciones distribuidas y casi siempre participa en la agrupación de conexiones.

Esta aplicación funciona con un gestor de transacciones de nivel medio y casi siempre con un administrador de conexión común.

```
<Resource auth="Container" driverClassName="com.mysql.jdbc.Driver" global="jdbc/TaskApp"
maxActive="100" maxIdle="20" maxWait="10000" minIdle="5" name="jdbc/TaskApp"
password="rootweb" type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/taskapp_diplomado" username="root"/>
```

**jdbc/TaskApp** es el nombre que le hemos dado a un nuestro datasource

url es el string de conexión a la base de datos

**username** usuario de la base de datos

**password** clave de la base de datos

**minIdle, maxIdle, maxActive** está relacionado con los tiempos que puede durar activa una conexión antes de pasar a inactivo.

**driverClassName** nombre del driver

## Paso V:

1- Crear excepción personalizada para manejar las excepciones de nuestra aplicación.

Crear una nueva clase y llamarle **TaskAppException**, en el paquete de utilidades, luego debemos extender de Exception.

Ahora necesitamos sobrecargar los **constructores** disponible en la clase **Exception**.

Click derecho en el cuerpo de la clase y seleccionar la opción **Constructor**.

Marcar todos los constructores disponibles.

2- Crear servicios para tener acceso a la data de nuestras entidades

a) Crear clase **ServicioPersistenciaBase** que va a contener una serie de métodos que podemos reutilizar en los demás servicios.

Crear método **getConeccion()** para manejar las conexiones con la base de datos.

Cuando estamos trabajando desde un contenedor web o un servidor de aplicaciones debemos usar un datasource en lugar de llamar la clase **DriverManager**.

En pasos anteriores creamos el datasource **jdbc/TaskApp** y este es el momento de usarlo.

Para acceder a dicho recurso debemos crear una instancia de la interfaz **Context** y usar el método **lookup**.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:/comp/env/jdbc/TaskApp");
```

java:comp/env es el nodo en el árbol JNDI, donde puedes encontrar propiedades para una aplicación web, o un EJB.

En el código anterior la interfaz DataSource contiene un método que nos permite obtener la conexión.

```
Connection con = ds.getConnection();
```

Debemos validar si la conexión devuelve valores nulos y lanzar una excepción en caso que sea así.

```
if (con == null) {
    throw new TaskListException("No pudo establecer conexión con la base de datos");
}
```

Poniendo todo junto nuestro método **getConeccion()** quedará de la siguiente manera:

```
protected Connection getConeccion() throws TaskListException {
    try {
        Context ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup("java:/comp/env/jdbc/TaskApp");

        Connection con = ds.getConnection();

        if (con == null) {
            throw new TaskListException("No pudo establecer conexion con la base de
datos");
        }
        return con;
    } catch (NamingException | SQLException ex) {
```



```

        ogger.getLogger(ServicioPersistenciaBase.class.getName()).log(Level.SEVERE,
null, ex);
        throw new TaskListException(ex);
    }
}

```

Crear los siguientes métodos:

- ResultSet consultarTodas(String sql)
- ResultSet consultarPorId(String sql, Integer idEntidad)
- boolean borrarRegistro(String sql, Integer idEntidad)
- boolean actualizarRegistro(PreparedStatement stmt)

Estos métodos son comunes para la mayoría de servicios que podemos necesitar y deben ser construido lo más genérico posible, de tal forma que se puedan utilizar en clases que hereden de **ServicioPersistenciaBase**.

b) Crear clase **ServicioTask** y extender de **ServicioPersistenciaBase**

Crear los métodos siguientes :

- List<Task> consultarTodas()
- Optional<Task> consultarTaskPorId(int id)
- boolean borrarTask(int id)
- guardarTask(Task task)
- boolean modificarTask(Task task)

Estos métodos deben utilizar los métodos definidos en la clase padre.

**Paso VI:**