Unidad 8

Componentes para el acceso a datos

Acceso a datos

Índice

Introducción

Uno de los sistemas de programación es el modular, que consiste en dividir la aplicación en pequeños componentes para realizar una funcionalidad especifica y mostrar una interfaz para su acceso, en su conjunto resolver un problema general.

Los componentes pueden ser gráficos en modo visual o no visuales, directamente funcionales por código.

En java se dispone de dos versiones Java SE y Java EE, ambas versiones disponen de sistemas modulares, basado en componentes, mediante JavaBeans y con EjB.

En esta unidad se verá en profundidad el sistema modular basado en JavaBeans y EjB, y algunos ejemplos de uso.

Al finalizar esta unidad

- Aprenderemos los componentes del software.
- Conoceremos los JavaBeans y EjB de Java.
- Comprenderemos la forma de desarrollo de aplicaciones web.
- Conoceremos los elementos: servlets y JSP de java para web.
- Estudiaremos el modelo MVC.
- Aprenderemos a desarrollar aplicaciones web.

8.1. Componentes de software

El modelo modular permite dividir en partes más pequeñas funcionalidades que unidas en conjunto componen un sistema complejo. Por ejemplo, un equipo informático está compuesto por una placa base, y diferentes módulos, la memoria, microprocesador, dispositivos de almacenamiento, etc.

En la ingeniería del software este modelo cobra una especial relevancia, debido a que permite descomponer la aplicación en módulos o componentes, permitiendo su reutilización en diferentes partes, la facilidad de mantenimiento y la calidad del software.

Un componente es un bloque o parte de código que define perfectamente la funcionalidad que tiene que hacer y la forma de interactuar con él, a partir de su interfaz. El uso de componentes es muy utilizado en sistemas electrónicos y software.

8.2. Modelos de componentes

El modelo de componentes se basa principalmente en disponer de una infraestructura de software con unos servicios básicos y que permita desplegar componentes según unos requisitos y unos mecanismos de interacción.

Los primeros modelos de componentes fueron visuales, como OLE y COM, posteriormente surgieron los componentes no visuales, como los DAO, objetos de acceso a datos.

Como modelo de componentes nos encontramos:

- ActiveX
- COM+
- DCOM
- JavaBeans
- EjB (Enterprise JavaBeans)
- CORBA

8.3. La plataforma Java: Java SE y Java EE

Java es un lenguaje de programación orientado a objetos que dispone de dos versiones para su entorno de ejecución (JRE), Java SE y Java EE.

Java SE

Es la especificación que incluye bibliotecas de clases de uso general. Se puede aplicar el modelo de componente con esta versión basada en **JavaBeans**.

JavaEE

Es la especificación más avanzada, está destinada para aplicaciones empresariales. Esta versión ha ido evolucionando y pasando por diferentes comunicades, Sun, Oracle y Eclipse que han ido cambiando su nombre a EE4J o Jakarta EE.

Esta versión incluye soporte a servidores de aplicaciones para desplegar diversos tipos de elementos, por ejemplo, servlets y contenedores de componentes **EjB**.

8.4. JavaBeans

Los JavaBeans son clases base igual que los POJOs (clases que representan objetos básicos en java) que deben de cumplir una serie de requisitos y mecanismos:

- Tener un constructor vacío.
- Puede haber constructores parametrizados.
- Hacer uso de la interfaz Serializable.
- Propiedades con ámbito privado.
- Métodos get y set por cada propiedad.
- Cada método get y set tiene que esta nombrado correctamente, especificar si hay parámetros y valor de retorno.

Se puede agrupar los JavaBean para empaquetarlos en un fichero tipo .jar para distribuirlos.

Mecanismos para gestionar los JavaBeans

Persistencia

Un JavaBean debe ser posible almacenarlo y restaurarlo en cualquier medio de persistencia. El uso de la interfaz Serializable ya especifica un mecanismo básico de persistencia.

Reflexión e introspección

Mecanismo para poder conocer los detalles de cualquier clase, atributos y métodos. Como la posibilidad de obtener cualquier información referente al JavaBean. Se puede ampliar esta funcionalidad usando la interfaz BeanInfo.

Personalización

En concreto para componentes visuales, se debe de poder modificar la apariencia y conducta, en el modo diseño

8.5. El modelo MVC para desarrollo de aplicaciones web con Java

La gran mayoría de aplicaciones, en concreto las aplicaciones web se basan en el modelo MVC, una formar de diseñar una aplicación, basada en separar en tres capas, el modelo de datos, las vistas y los controladores.

- **Capa modelo.** Contiene las clases que aplican la persistencia del modelo de datos. Las clases que aparecen en el modelo pueden ser los POJOS, **JavaBeans**, o **EjB**.
- **Capa vista.** Contiene todas las interfaces que muestran de forma gráfica los datos obtenido de la capa del modelo. Esta es la capa que puede ver el usuario. Las interfaces que se emplean en java para desarrollos web son del tipo **JSP.**
- **Capa controlador.** Contiene clases encargadas de delegar acciones sobre la capa vista y la capa modelo. Estas clases controlan el flujo de llamadas dentro del código al resto de clases, funciones o métodos. En java se implementa mediante **servlets**.

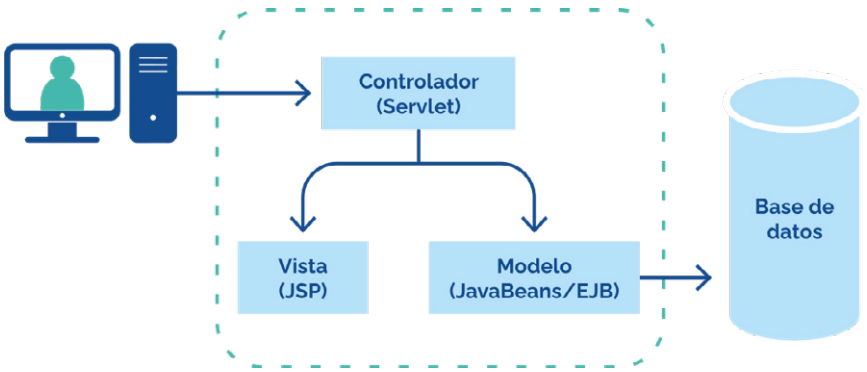


Imagen 1. Modelo MVC en java

Aplicar el modelo MVC es recomendable debido a los beneficios que aporta a la hora del desarrollo de cualquier aplicación, destacando la fiabilidad, robustez, escalabilidad y facilidad de mantenimiento.

Existen frameworks que por defecto ya aplican y trabajan sobre el modelo MVC, dotando a la aplicación toda la infraestructura necesaria. Nos podemos encontrar frameworks como JSF, Struts y Spring.

8.6. JSP (JavaServer Pages)

JavaServer Pages (JSP) son ficheros de java para poder desarrollar páginas web dinámicas. Actúan en la capa de vista para diseñar interfaces donde interacciona el usuario y su funcionamiento es aplicar código java directamente sobre HTML y XML, un concepto similar a PHP y ASP.

El contenido que puede tener un fichero JSP es:

- Etiquetas <html> o <xml>
- Directivas de JSP
- Código del lenguaje java

8.6.1. Directivas de JSP

Las directivas son especificaciones propias de JSP que permiten indicar como se configurará y ejecutará la interfaz o página. La sintaxis es, `<% atributo="valor" %>`.

Atributo	Descripción
`import`	Importa clases y paquetes para usar dentro del fichero JSP.
`session`	Permite indicar si se usan datos de la sesión.
`contentType`	Especifica el tipo de contenido del fichero, por defecto es "`text/html`".
`errorPage`	Indica la página de error a la que dirigirse en cado de una excepción.
`isErrorPage`	Determina si la página gestiona excepciones.

8.6.2. Scriptlets

Son partes que contienen código java. Se usa en cualquier parte del documento con la sintaxis, `<% código java %>`.

8.6.3. Variables implícitas

Son variable definidas en el contexto de la página web.

Variable implicita	Clase
pageContext	javax.servlet.jsp.PageContext
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
config	javax.servlet.ServletConfig
application	javax.servlet.ServletContext
page	java.lang.Object
exception	java.lang.Exception

Los cuatro ámbitos que existen en JSP de menor visibilidad a mayor son:

- Página (page)
- Petición (request)
- Sesión (sesión)
- Aplicación (application)

8.6.4. Referencias a variables de Java

Las variables definidas en cualquier scriptlet con código java se pueden usar en otras partes, mediante la sintaxis `<%= nombre variable %>`.

Existe otra forma de poder acceder al contenido de la variable, mediante el método out.print, su sintaxis es `<%out.print(variable)%>`.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<%@ page import="java.util.Date, java.text.SimpleDateFormat" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>Saludo</title>
    </head>
    <body>
        <h2>Hola <%= request.getParameter("nombre") %>!</h2>
        <%
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            Date fecha = new Date();
        %>
        <h3> La fecha de la petición es: <% out.println(sdf.format(fecha)); %> </h3>
    </body>
</html>
```

Imagen 2. Ejemplo de fichero JSP

8.7. Servlets

Los servlets son programas contenidos en un módulo para ejecutarlo en el lado del servidor web que tiene contenedor de servlets.

Los clientes pueden invocar los servlets usando el protocolo HTTP. Gestiona las peticiones, las procesa con la información de la invocación y genera una respuesta, normalmente HTML o JSP.

Imagen 3. Diagrama de iteracción cliente servlet a partir de HTTP

Las características que pueden aportar los servlets son:

- **Rendimiento.** Por cada servlet que crea un único proceso. Cada petición en el servlet creará un hilo de ejecución.
- **Portabilidad.** Heredado de Java, se puede llevar de un servidor a otro, sin necesidad de adaptarlo.
- **Rápido desarrollo.** Es una estructura modular y se usa todas las librerías Java
- **Robustez.**
- **Amplio soporte.** Existe una comunidad muy amplia y compañías que usan esta tecnología.

Estructura de un servlet

Un servlet es una instancia de alguna de las clases del API Java Servlets, *GenericServlet* o *HttpServlet*. Dependiendo de la versión del contenedor de servlets estas dos clases pueden estar en los paquetes *javax.servlet* o *jakarta.servlet*

```
public class Servlet extends HttpServlet {

    public void init() throws ServletException {
        // Inicialización de recursos
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Acciones
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Acciones
    }

    public void destroy() {
        // Liberar recursos
    }

}
```

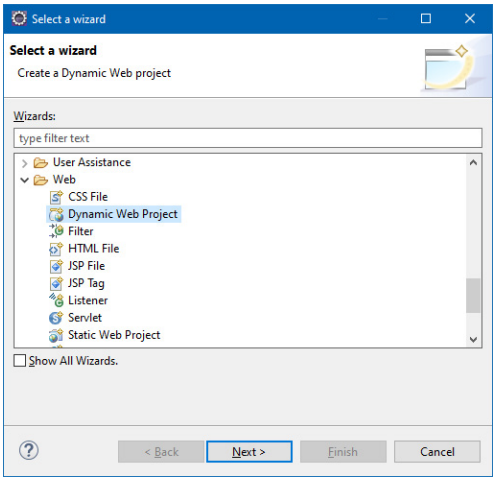
Imagen 4. Estructura en código de un servlet

8.8. Desarrollo de una aplicación web MVC basada en JavaBeans

A continuación, se va a ver paso a paso la creación de una aplicación web MVC con JavaBeans. Como acceso a datos se usará la persistencia mediante ORM con Hibernate.

8.8.1. Creación de la aplicación web

Se genera un proyecto del tipo Dynamic Web Project, desde "File", "New", "Project". Este tipo de proyecto genera una estructura base de aplicación web, con las carpetas META-INF, WEB-INF.



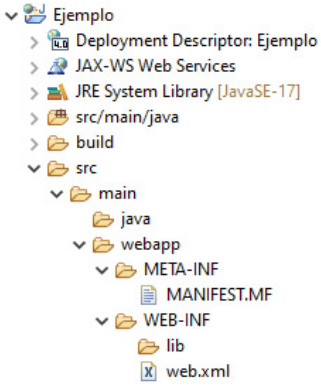


Imagen 5. Creación de un proyecto Dynamic Web Project

8.8.2. Persistencia de objectos con Hibernate

Como base de datos se va a emplear un diseño de modelo de datos de ventas, en el que intervendrán, clientes, ordenes de productos, la orden general y los productos. En la siguiente imagen aparece las sentencias para construir la base de datos.

```
CREATE DATABASE ventas;
USE ventas;

CREATE TABLE IF NOT EXISTS clientes (
  id int(11) NOT NULL AUTO_INCREMENT,
  nombre varchar(100) NOT NULL,
  email varchar(100) NOT NULL,
  telefono varchar(15) NOT NULL,
  direccion text NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS productos (
  id int(11) NOT NULL AUTO_INCREMENT,
  nombre varchar(200) NOT NULL,
  descripcion text NOT NULL,
  precio float(10,2) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS orden (
  id int(11) NOT NULL AUTO_INCREMENT,
  cliente_id int(11) NOT NULL,
  precio_total float(10,2) NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY fk_orden_cliente(cliente_id)
    REFERENCES clientes (id) ON DELETE CASCADE ON UPDATE NO ACTION
);

CREATE TABLE IF NOT EXISTS orden_productos (
  id int(11) NOT NULL AUTO_INCREMENT,
  orden_id int(11) NOT NULL,
  producto_id int(11) NOT NULL,
  cantidad int(5) NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY fk_orden_productos_orden_id(orden_id)
    REFERENCES orden (id) ON DELETE CASCADE ON UPDATE NO ACTION,
  FOREIGN KEY fk_orden_productos_producto_id(producto_id)
    REFERENCES productos (id) ON DELETE CASCADE ON UPDATE NO ACTION
);

CREATE USER 'usuarioVentas'@'localhost' IDENTIFIED BY 'usuarioVentas'
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,EXECUTE
    ON ventas.* TO 'usuarioVentas'@'localhost';
```

Imagen 6. Sentencias DDL para la base de datos de ventas

Tal y como se aprendió en el tema 4, se realizarán las mismas tareas para configurar y arrancar hibernate.

1. Crear la conexión con la base de datos.
2. Crear los ficheros hibernate.cfg.xml e hibernate.reveng.xml
3. Crear el fichero HibernateUtil.java
4. Crear los POJOs de cada tabla de base de datos.

Las librerías de hibernate y el driver de conexión a la base de datos se tendrá que ubicar en *src/main/webapp/WEB-INF/lib*

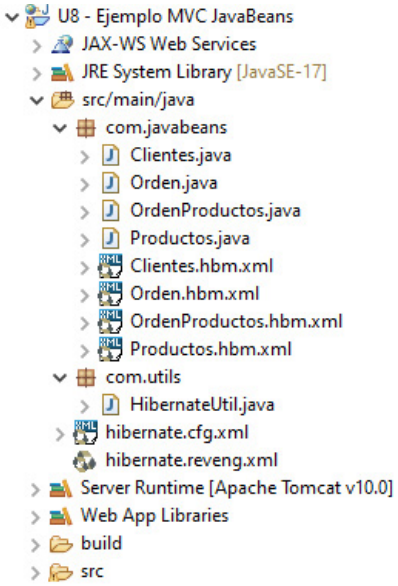


Imagen 7. Ficheros que componente la configuración de hibernate

8.8.3. Creación de servlet controlador

Se creará una clase java con el nombre "SvControladorCliente" y haciendo uso de la super clase *HttpServlet*. La funcionalidad que tendrá el controlador es atender dos acciones, una para mostrar los clientes y otra para dar de alta un nuevo cliente.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String accion = request.getParameter("accion");

    System.out.println("accion: " + accion);

    if(accion != null) {

        switch (accion) {
            case "mostrar": request.getRequestDispatcher("mostrarClientes.jsp").forward(request, response);
                            break;
            case "alta":    request.getRequestDispatcher("altaCliente.jsp").forward(request, response);
                            break;

        }

    } else {
        response.sendRedirect("index.jsp");
    }

}
```

Imagen 8. Código con la funcionalidad del método doGet

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {


    String accion = request.getParameter("accion");

    System.out.println("accion: " + accion);

    if(accion != null) {

        switch (accion) {

            case "alta":    Session session = null;

                            try {
                                String nombre = request.getParameter("nombre");
                                String telefono = request.getParameter("telefono");
                                String email = request.getParameter("email");
                                String direccion = request.getParameter("direccion");

                                session = HibernateUtil.getSessionFactory().openSession();
                                session.beginTransaction();

                                Clientes cliente = new Clientes(nombre, email, telefono, direccion);

                                session.save(cliente);

                                session.getTransaction().commit();

                            } catch (Exception e) {
                                e.printStackTrace(System.err);
                                if(session != null)
                                    session.getTransaction().rollback();
                            }

                            break;

        }

    }

    response.sendRedirect("mostrarClientes.jsp");

}
```

Imagen 9. Código con la funcionalidad del método doPost

Una vez creado el servlet, se procederá a modificar el fichero de contexto web.xml para registrar la configuración del servlet. Se definirá un parámetro llamado "accion" que contendrá las acciones a realizar de mostrar clientes o dar de alta uno nuevo.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd
    http://xmlns.jcp.org/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="5.0">
    <display-name>U8 - Ejemplo MVC JavaBeans</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>ControladorCliente</servlet-name>
        <servlet-class>com.controlador.SvControladorCliente</servlet-class>
        <init-param>
            <param-name>accion</param-name>
            <param-value></param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>ControladorCliente</servlet-name>
        <url-pattern>/controlador</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout> 30 </session-timeout>
    </session-config>
</web-app>
```

Imagen 10. Configuración del fichero web.xml

8.8.4. Uso de JavaBeans asociado a formularios HTML con JSP

Los JavaBeans son las clases que se usan directamente con hibernate, es decir, los mismos POJOs. Podemos hacer uso de los JavaBeans en cualquier clase que sea requerido o incluso en los mismos documentos JSP.

Existen varias formas de emplear los JavaBeans dentro de un documento JSP.

- Usar las propias etiquetas de código incrustado, **por ejemplo** `<% Clientes cliente = new Cliente() %>`, o
- Usar una etiqueta especial para JavaBeans, *jsp:useBean*, **por ejemplo** `<jsp:useBean id="cliente" class="com.javabeans.Clientes"/>`.

Ambas formas son iguales, podremos acceder a sus atributos dentro del documento jsp.

Cuando se usa un objeto dentro de un documento, este solo es válido dentro del contexto del documento, cuando se redirecciona a otro documento jsp o se envía una petición al controlador, el objeto deja de estar disponible.

Si queremos usar el objeto en diferentes partes, es necesario definir el alcance de su ámbito mediante la propiedad *scope* y como valor el rango de *visibilidad, page, request, sesión* o *application*.

Ejemplo:

```
<jsp:useBean id="cliente" scope ="request" class="com.javabeans.Clientes"/>
```

Y para recuperar el objeto desde otro lugar se usará el método getAttribute del request.

```
Clientes cliente = (Clientes) request.getAtribute("cliente")
```

8.8.5. Creación de los JSP

Para crear los ficheros JSP iremos desde la barra de herramientas File – New – JSP File. Se generará un documento base es igual que uno HTML pero con la extensión .jsp

Para el ejemplo de la aplicación MVC se va a disponer de tres ficheros .jsp cada uno de ellos tiene que estar creado dentro de la carpeta webapp del proyecto. A continuación, se muestra en profundidad su funcionalidad.

> `index.jsp`. Es la vista principal, solo tendrá dos enlaces para llamar al controlador indicando la acción a realizar.

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Ventas</title>
    </head>
    <body>
        <p> Que acción quieres realizar </p>
        <table>
            <tr>
                <td><a href="controlador?accion=mostrar">Mostrar clientes</a></td>
            </tr>
            <tr>
                <td><a href="controlador?accion=alta">Alta cliente</a></td>
            </tr>
        </table>

    </body>
</html>
```

Imagen 11. Código de index.jsp

> `mostrarClientes.jsp`. Es la vista que muestra todos los clientes de la base de datos.

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="UTF-8"%>
<%@ page import="java.util.List, org.hibernate.Session, org.hibernate.query.Query,
                 com.utils.HibernateUtil, com.javabeans.Clientes" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Mostrar clientes</title>
    </head>
    <body>
        <%
            Session sesionHibernate = HibernateUtil.getSessionFactory().openSession();
            Query query = sesionHibernate.createQuery("FROM Clientes");
            List<Clientes> clientes = (List<Clientes>) query.getResultList();

            if (clientes != null) {
        %>
            <table>
                <tr>
                    <td>ID</td>
                    <td>NOMBRE</td>
                    <td>TELEFONO</td>
                    <td>EMAIL</td>
                </tr>
                <% for (Clientes cliente: clientes) { %>
                <tr>
                    <td><%=cliente.getId()%></td>
                    <td><%=cliente.getNombre()%></td>
                    <td><%=cliente.getTelefono()%></td>
                    <td><%=cliente.getEmail()%></td>
                </tr>
                <% } %>
            </table>
        <% } else { %>
            <h1> No se han encontrado clientes</h1>
        <% } %>
        <table>
            <tr>
                <td><a href="controlador?accion=alta">Alta cliente</a></td>
            </tr>
        </table>
    </body>
</html>
```

Imagen 12. Código de mostrarClientes.jsp

> `altaCliente.jsp`. Es la vista que permite registrar un nuevo cliente a partir de un formulario.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Alta cliente</title>
    </head>
    <body>
        <form method="post" action="controlador">
            <table>
                <tr>
                    <td>Nombre</td>
                    <td><input name="nombre" type="text"/></td>
                </tr>
                <tr>
                    <td>Telefono</td>
                    <td><input name="telefono" type="text"/></td>
                </tr>
                <tr>
                    <td>Email</td>
                    <td><input name="email" type="text"/></td>
                </tr>
                <tr>
                    <td>Dirección</td>
                    <td><input name="direccion" type="text"/></td>
                </tr>
                <tr>
                    <td><input type="hidden" name="accion" value="alta" /></td>
                    <td><input type="submit" value="Alta" /></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

Imagen 13. Código de altaCliente.jsp

8.8.6. Instalación del servidor de aplicaciones Apache Tomcat

Una vez finalizada la aplicación es necesario desplegarla y probarla. Para ello es necesario disponer de un servidor de aplicaciones, en este caso, se va a usar Apache Tomcat que permite el despliegue de aplicaciones web y es un contenedor de servlets.

Se puede descargar la versión más reciente y estable de su página web oficial, https://tomcat.apache.org. Se usará la distribución core, que viene comprimido todos los ficheros esenciales del servidor. No hay versión executable, tan solo hay que descomprimir todos los ficheros en una carpeta.

Los servidores de aplicaciones funcionan en base a comandos y líneas de consola, nosotros vamos a verlo integrado en el IDE de eclipse. Desde la barra de herramientas iremos a *Windows/Preferences/Server/Runtime Environment* y añadiremos el servidor de Tomcat o cualquier otro que nos interese, indicando en el desplegable el servidor y su versión, y la ruta donde se habia descomprimido su instalación.

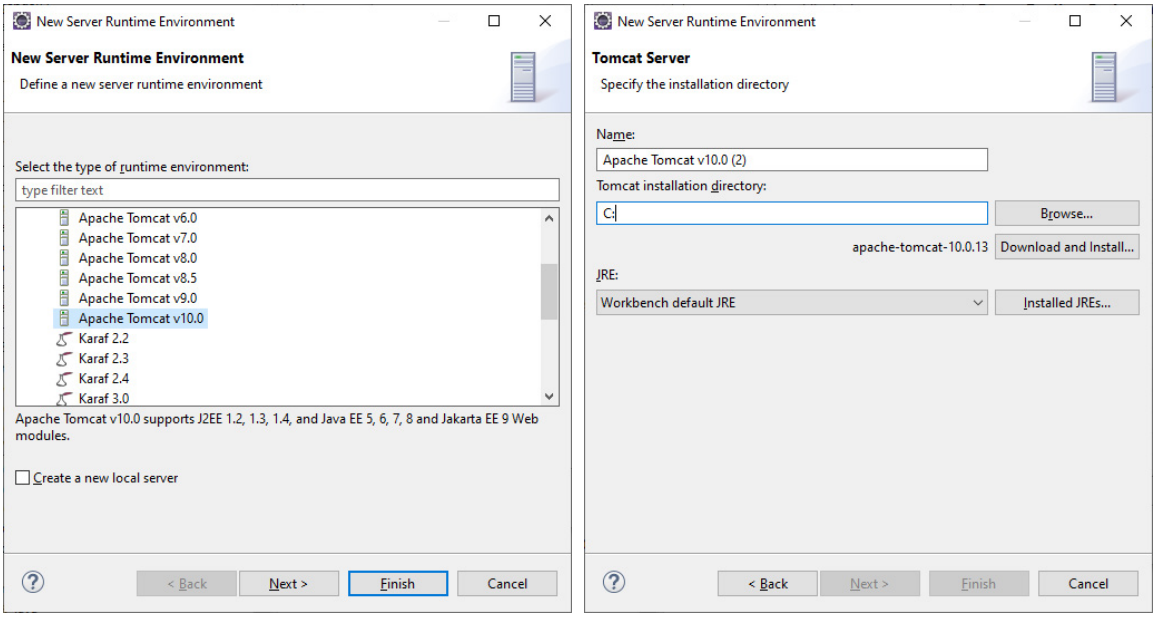


Imagen 14. Asistente de instalación de servidor en Eclipse

Una vez finalizado, nos aparecerá el servidor en la pestaña *Servers*. Desde esta pestaña tendremos el control del servidor, podremos arrancarlo y pararlo, como añadir proyectos para que se desplieguen haciendo click secundario sobre el nombre del servidor y la opción *Add and Remove*.

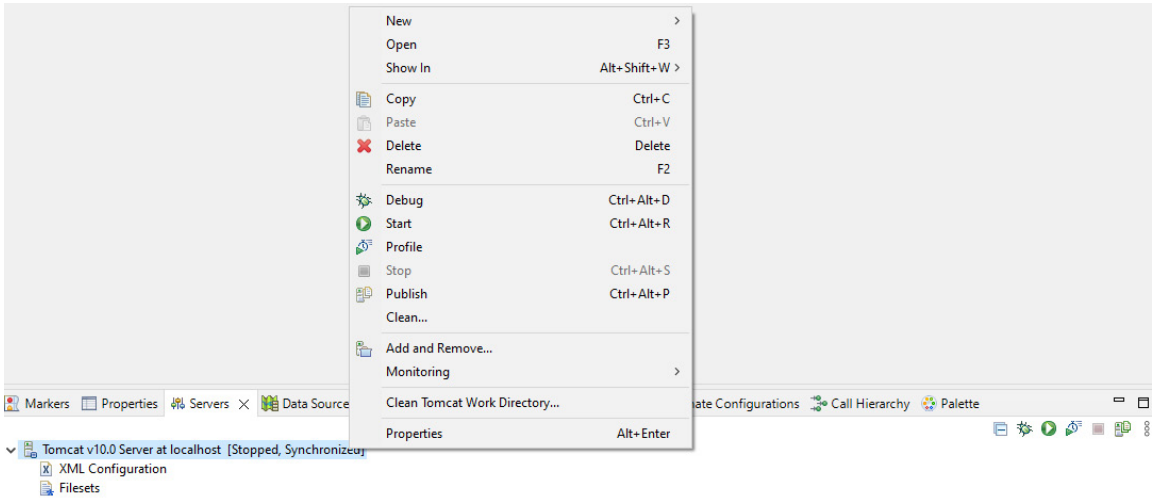


Imagen 15. Añadir proyecto al servidor Tomcat

Una vez añadido el proyecto de la aplicación web MVC y arracando el servidor, podemos probar si funciona, abriendo cualquier navegador y escribiendo http://localhost:8080/"nombre del proyecto".

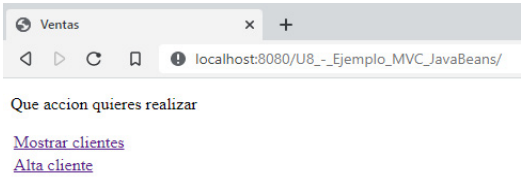


Imagen 16. Funcionamiento de la aplicación web con el servidor

8.9. Enterprise JavaBeans

Con la plataforma de JavaEE se puede hacer uso de JavaBeans o EjB (Enterprise JavaBeans) que se despliegan en un servidor de EjB.

Las principales características que ofrece el uso de EjB son:

- Uso de la API JPA para la persistencia de objetos y JTA para transacciones.
- Permite la sincronización en el acceso a los datos.
- Gestión de las transacciones.
- Control de concurrencia.
- Mecanismos CDI (Context and dependency injection). Uso de anotaciones de Java para acceder fácilmente a servicios básicos.
- Sistema de mensajes con JMS.

Existen tres tipos de EjB:

- **Entidad.** Los objetos entidad representan un modelo de datos que está en la base de datos.
- **Sesión.** Los objetos sesión representa un proceso o acción que gestiona la forma de comunicarse con el servidor.
- **Dirigidos por mensajes.** Los objetos representan un servicio de mensaje JMS. Los clientes no se comunican directamente con estos objetos, si no, que es necesario un mensaje para activarlo, es muy parecido a los eventos.

No todos los servidores de aplicaciones soportan EjB, por ejemplo, Tomcat es un servidor contenedor de servlets, pero no de EjB. En cambio, el servidor GlassFish si permite EjB.

www.universae.com