Práctica.- Entorno de desarrollo: Apache Netbeans. Proyecto WebApp

Para conocer cómo funciona el entorno de desarrollo que utilizaremos durante las prácticas, en esta sesión vamos a crear una pequeña aplicación web, en concreto el alta de usuarios, que cubra los conceptos básicos del desarrollo web con Java y nos permita, posteriormente, ir ampliándola a medida que aprendamos nuevos conceptos.

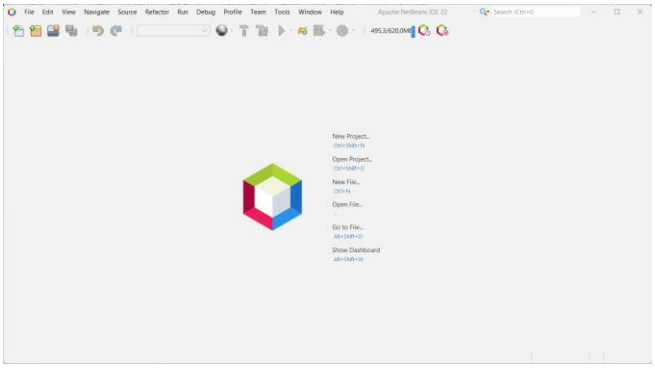
1. Java JDK

Para el desarrollo de aplicaciones con Java necesitamos disponer del JDK (kit de desarrollo de java). Durante este curso utilizaremos la versión JDK 21.

2. Apache Netbeans

Accedemos a netbeans.apache.org, vamos a la sección Download y desde ahí descargamos la versión que deseamos. En este curso, utilizaremos la versión Apache Netbeans 22.

Una vez instalado, ejecuta el programa y aparecerá una pantalla similar a la siguiente:

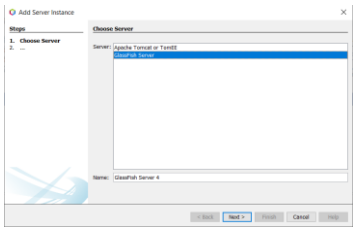
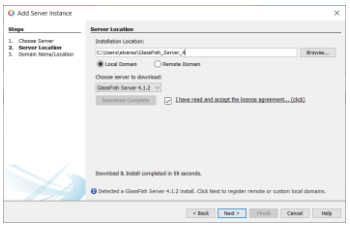
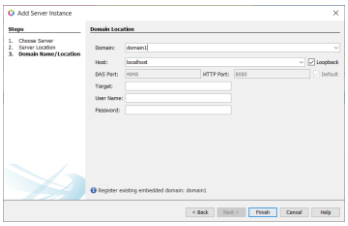


3. Servidor de Aplicaciones: GlassFish

Durante las prácticas utilizaremos GlassFish como servidor de aplicaciones, que además incorpora Apache Derby (Java DB), un sistema gestor de bases de datos ideal para utilizarlo en un entorno de desarrollo; aunque no sería adecuado para utilizarlo en entornos de producción.

En este curso, utilizaremos GlassFish versión 7.0.14.

Para instalar Glassfish, en la pestaña Services (menú Window – Services, si no está visible), hacemos clic con el botón derecho en Servers y hacemos clic en “Add Server…”. En la pantalla que aparece seleccionamos “Glassfish Server”, establecemos un nombre para este servidor (por ejemplo, GlassFish7.0.14) y pulsamos en Next (si es la primera vez que se hace, nos pedirá descargar y activar el plugin para la funcionalidad JEE). En la siguiente pantalla, seleccionamos la ubicación, la versión que deseamos bajar, aceptamos la licencia y pulsamos el botón de descarga. Una vez descargado pulsamos en Next, dejamos los valores por defecto y pulsamos Finish.

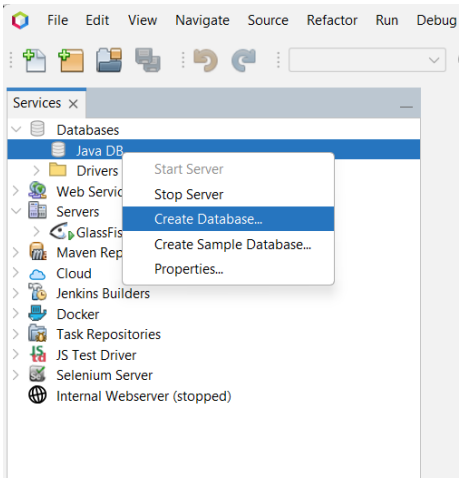




De la misma forma, podemos instalar otros Servidores de Aplicaciones y/o Sistema Gestores de Bases de Datos.

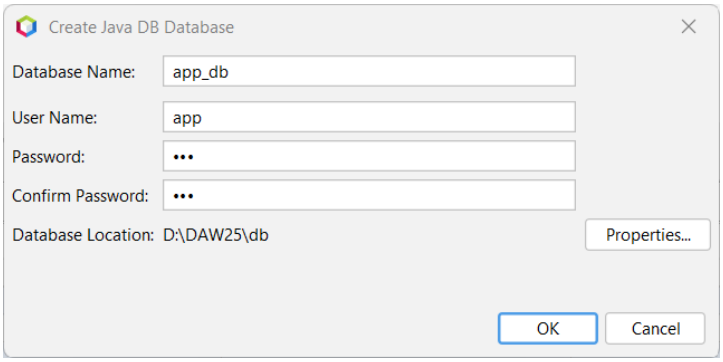
4. Sistema Gestor de Bases de Datos: Apache Derby (Java DB)

Con la instalación de GlassFish, también, se incorpora a Apache Netbeans el SGBD Apache Derby (Java DB) ideal para entornos de desarrollo, pero no para entornos de producción.

Para crear una Base de Datos en este SGBD, accedemos a la pestaña Services (Windows – Services), hacemos Clic derecho en Java DB y elegimos “Create Database…”



Completamos el asistente con los datos deseados:



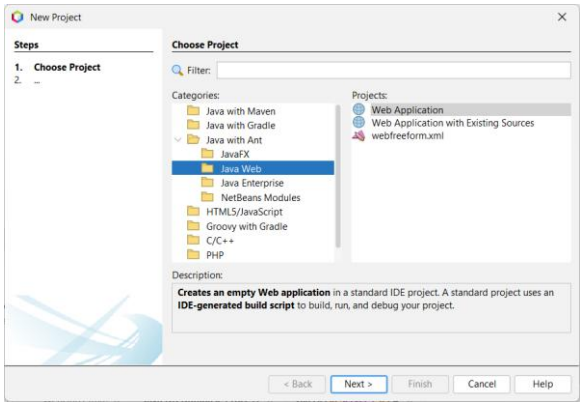
También podemos seleccionar la implementación de Java DB a utilizar y el directorio donde se ubicarán las bases de datos creadas, mediante el botón "Properties…".

4. Primera aplicación Web: WebApp

Apache Netbeans permite crear una variedad de aplicaciones Java utilizando varias herramientas de gestión de proyectos o automatización de compilación, como Ant, Maven y Gradle. En las prácticas de la asignatura utilizaremos Ant para crear nuestras aplicaciones Java Web.

4.1 Crear un proyecto

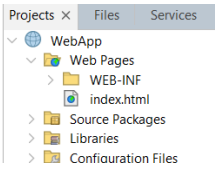
Para crear un nuevo proyecto: File – New Project…



En el asistente, en la sección “Java with Ant”, seleccionamos **Java Web,** en categoría, y **Web Application**, en proyecto, esto crea una aplicación Java web con la estructura adecuada.

En el asistente, pulsamos siguiente, para establecer el nombre del proyecto, el directorio donde se localizan los proyectos, el directorio donde se ubicará este proyecto; así como, si fuese necesario, el directorio donde se ubicarán las librerías. Para este pequeño ejemplo sólo vamos a indicar el nombre del proyecto, por ejemplo, **WebApp**, y pulsamos siguiente.

En la nueva pantalla seleccionamos el servidor de aplicaciones (GlassFish), la versión de JEE y la ruta de contexto (Context Path) de la aplicación (que define la URL donde se localiza la aplicación dentro del servidor). En nuestro caso, cambiamos el Context Path a **/app**, dejamos el resto de los valores por defecto y pulsamos siguiente. Aparecerá una nueva pantalla para seleccionar los frameworks, por si deseamos utilizar alguno (por ahora no vamos a utilizarlos).



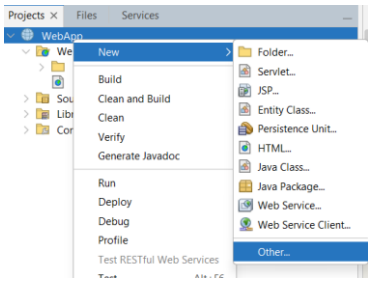
Si todo ha ido bien, en la pestaña Projects aparecerá la estructura de la aplicación, con el fichero index.html creado por defecto, en la parte accesible por los usuarios desde la Internet, y la carpeta WEB-INF, solo accesible dentro del propio servidor. En la pestaña Files podremos ver todos los ficheros reales del proyecto, pero no deberíamos modificarlos directamente.

NOTA: En este momento podemos lanzar el proyecto (con ▷ o Run – Run Project (WebApp) o F6), esto iniciará, si no lo están, el SGBD y el servidor de aplicaciones Glassfish y lanzará el navegador por defecto para mostrar nuestra aplicación (hasta ahora el contenido del fichero index.html).

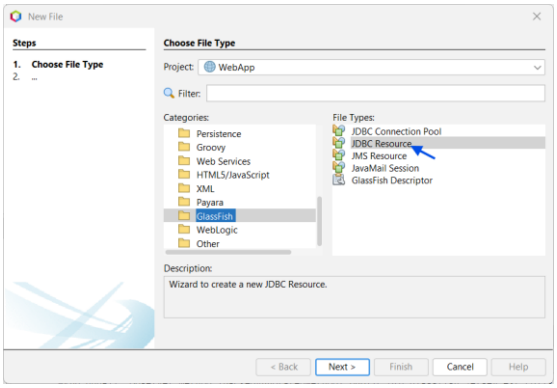
5. Conexión con la BD: Recurso y Pool de Conexiones

Aunque podemos conectar nuestra aplicación a la BD utilizando JBDC tradicional, cuando se trabaja en un entorno Web, en el que se conectarán muchos clientes, es más interesante crear un Pool de Conexiones, que gestionará las conexiones con la BD sin saturar los recursos. Por otro lado, para conectar la aplicación con el Pool necesitamos también crear un recurso.

Para crear el recurso y el pool de conexiones en GlassFish: hacemos clic derecho en el nombre del proyecto (WebApp) y seleccionamos New – Other…,

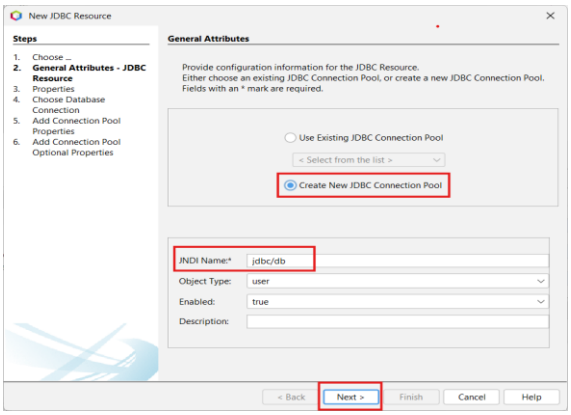


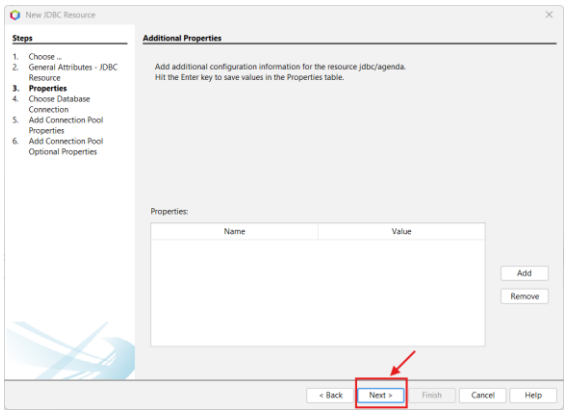
En el asistente, en categoría, buscamos GlassFish y en tipo de fichero elegimos JDBC Resource.

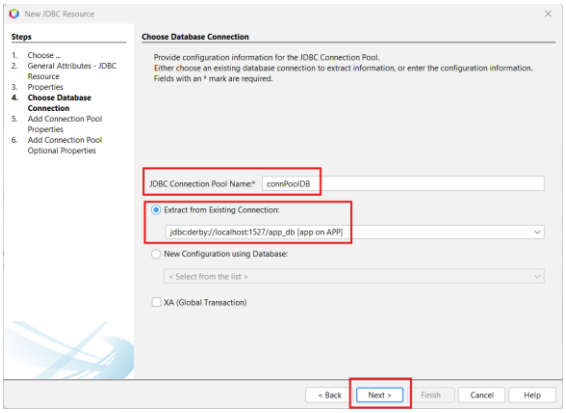


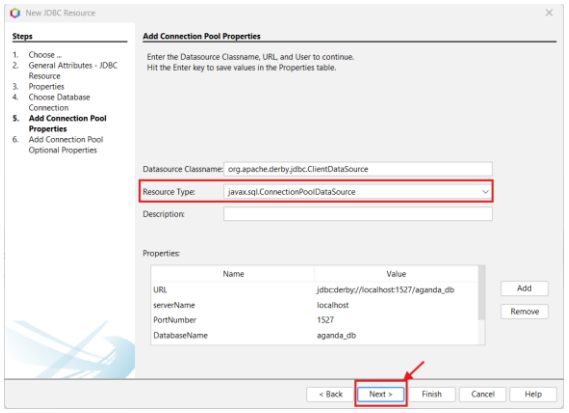
Completamos las diferentes pantallas del asistente: En la primera, marcamos “Create New JDBC Connection Pool”, establecemos el nombre del recurso (por ejemplo, **jdbc/db**), dejamos el resto de opciones por defecto y pulsamos en Next; a continuación, aparecerá la ventana de propiedades adicionales (la dejamos por defecto) y pulsamos en Next. Aaparecerá la ventana para el Pool, establecemos el nombre (por ejemplo, **connPoolDB**), seleccionamos la BD (creada anteriormente, en esta guía app_db) y pulsamos Next. A continuación, en la siguiente ventana, cambiamos el tipo de recurso a javax.sql.ConnectionPoolDataSource y pulsamos Next; finalmente, dejamos por defecto las propiedades adicionales del Pool y pulsamos Finish.

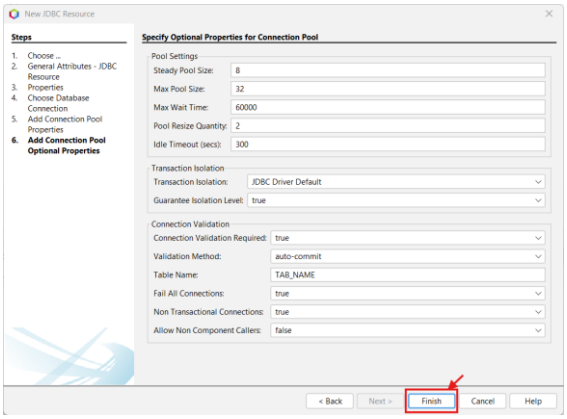
A continuación, se muestran las imágenes del proceso; además de un Bug de Netbeans que debemos resolver para que funcione correctamente.





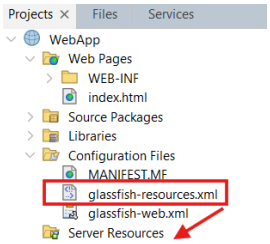


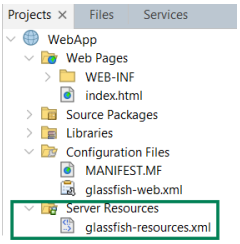




BUG glassfish-resources.xml

Debemos resolver un bug de Apache Netbeans, ya que nos creará el fichero de configuración del recurso y el pool en la carpeta “Configuration Files” pero para que GlassFish cree ambos elementos es necesario moverlo a la carpeta “Server Resources”



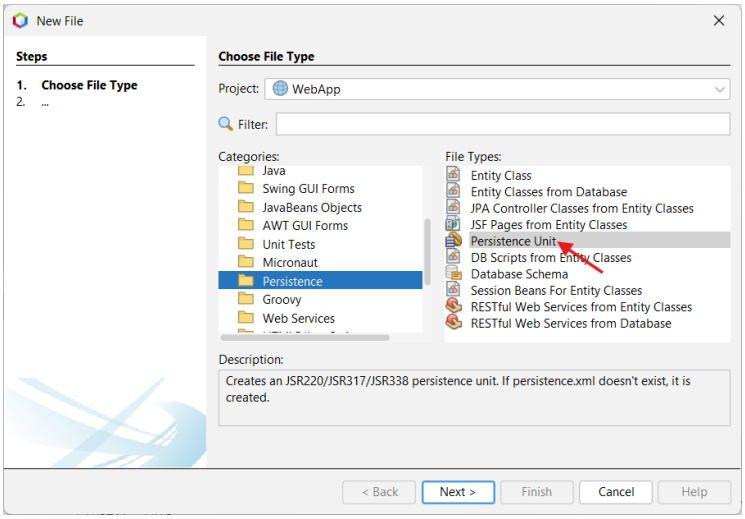


6. Interactuando con los datos: JPA vs JDBC

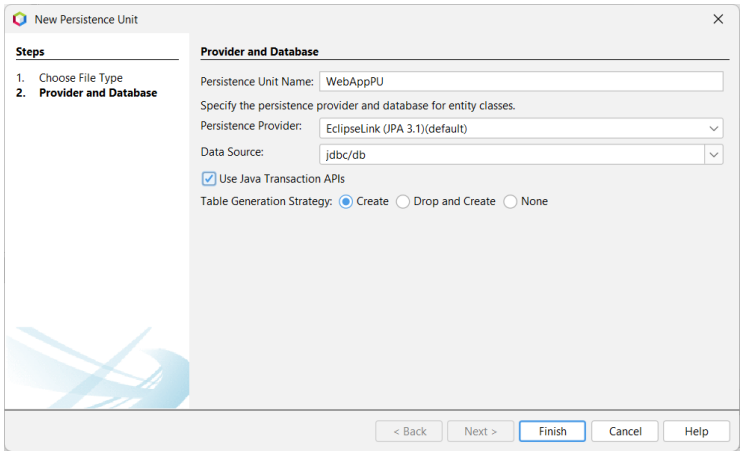
Para insertar y obtener datos de nuestra BD podemos usar JDBC tradicional, pero es más moderno hacer uso de un ORM (Object-Relational Mapping), una nueva capa que nos permite trabajar directamente con los objetos, persistirlos y manejarlos sin necesidad de utilizar sentencias SQL directamente.

Para ello, necesitamos crear una unidad de persistencia, configurarla e inyectar un Gestor de Entidades (Entity Manager) en nuestra aplicación.

Para crear la Unidad de persistencia, clic derecho en el nombre del proyecto – New – Others – Persistence – Persistence Unit.

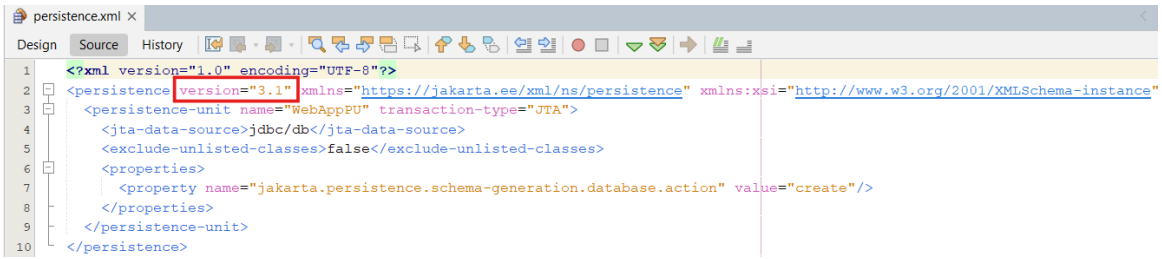


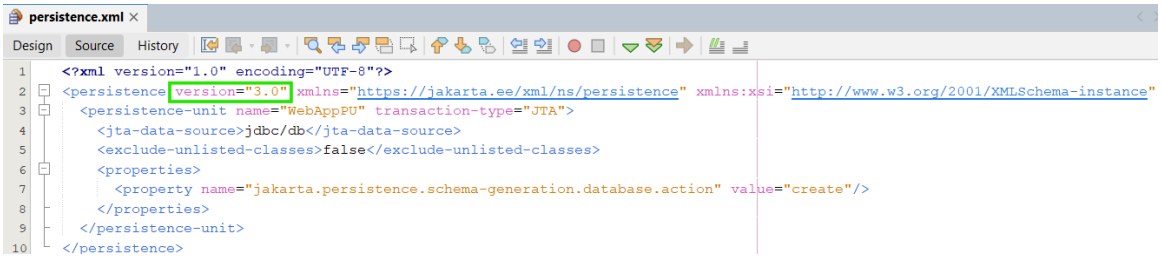
Pulsamos Next y completamos el asistente: establecemos el nombre de la unidad de persistencia (por ejemplo, WebAppPU), el proveedor (aparecerá de forma automática EclipseLink) y elegimos la fuente de datos mediante el recurso creado en el apartado anterior (en esta guía, jdbc/db). Dejamos el resto de las opciones por defecto, incluyendo el API de transacciones y estableciendo la estrategia de generación de tablas en Create.



BUG persistence.xml

Debemos resolver otro bug de Apache Netbeans, en este caso, nos creará un fichero con una versión errónea, por lo que debemos cambiar la versión 3.1 por la 3.0. Para ello, abrimos el fichero persistence.xml, hacemos clic en la pestaña Source, para ver el código XML, y hacemos el cambio:





NOTA: Si, más tarde, haces algún cambio al fichero persistence.xml, asegúrate, de nuevo, que se queda con la versión 3.0; ya que puede volver a cambiar automáticamente a la 3.1 al actualizarlo.

7. Patrón MVC: Model – View - Controller

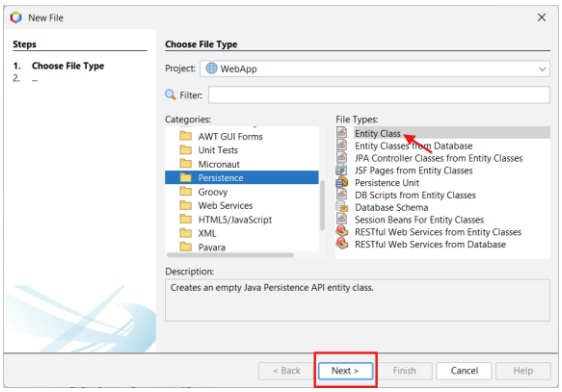
El desarrollo de aplicaciones web con Java permite utilizar Servlets o JSP de forma independiente, lo que se conoce como Model 1; sin embargo, lo ideal es hacer uso de lo mejor de cada una de ellas, utilizándolas de forma conjunta, mediante el Model 2 (patrón MCV).

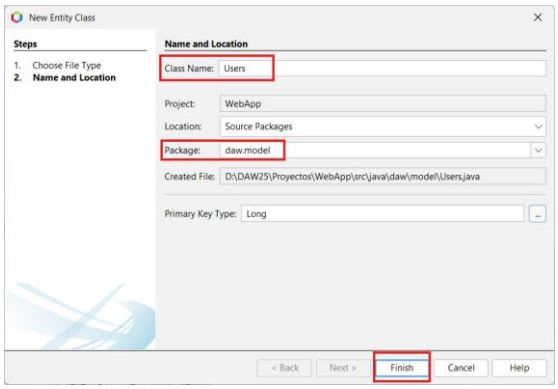
7.1 Modelo (Entity)

Los modelos serán clases java (Java Beans) o entidades (Entity). En este caso, usaremos entidades que serán los objetos que el ORM (o el Entity Manager) persistirá en la BD.

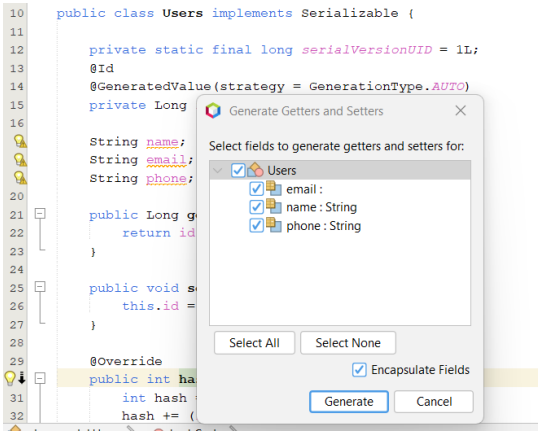
En este ejemplo, vamos a crear la entidad Users en nuestro proyecto.

Para crear una entidad, clic derecho en el nombre del proyecto – New – Others – Persistence – Entity Class.





Esto nos creará la entidad Users con los valores por defecto. Ahora, vamos a editar el código para añadir los atributos que necesitamos, generando los métodos setters y getters de cada uno de ellos. En este caso, añadimos los atributos nombre (name), email (email) y teléfono (phone) para cada usuario y, utilizando el asistente "Insert Code.." (pulsado botón derecho en la zona de edición de código) generamos sus setters y getters.



También, mediante “Insert Code…”, crearemos un constructor vacío y otro con tres de los atributos anteriores, excepto el Id que será generado de forma automática.

```
public Users() {
}

public Users(String name, String email, String phone) {
    this.name = name;
    this.email = email;
    this.phone = phone;
}
```

Además, vamos a cambiar el nombre de la tabla que almacenará los datos de esta entidad (a users, en minúscula); vamos a añadir dos consultas estáticas (named queries) usando JPQL: Users.findAll, para devolver todos los usuarios, y Users.findByName, que devolverá el usuario con el nombre (name) especificado y, finalmente, vamos a modificar la estrategia de generación de la clave primaria a IDENTITY.

NOTA: Recuerda importar todos los paquetes necesarios para los elementos insertados.

```java
@Entity
@Table(name="users")
@NamedQueries({
    @NamedQuery(name="Users.findAll", query="SELECT u FROM Users u"),
    @NamedQuery(name="Users.findByName", query="SELECT u FROM Users u WHERE u.name = :name"),
})
public class Users implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;
    private String phone;

    public Users() {
    }
```

NamedQueries:

```java
@Table(name="users")
@NamedQueries({
    @NamedQuery(name="Users.findAll", query="SELECT u FROM Users u"),
    @NamedQuery(name=" Users.findByName", query="SELECT u FROM Users u WHERE u.name = :name"),
})
```

7.2. Vista

Las vistas serán JSP que, de forma dinámica, generarán código HTML, que puede incluir estilos CSS y código JavaScript.

Si usamos el MVC, las vistas no deben ser accesibles desde el navegador, solo deberían estar disponibles para ser invocadas desde los controladores; por ello, deberían estar incluidas en la carpeta WEB-INF del proyecto, por ejemplo, en una carpeta views organizada según nos interese.

En este ejemplo, vamos a crear dos vistas:

- users.jsp: Para mostrar los usuarios de nuestra aplicación. Esta vista recibirá el listado de los usuarios y usando JSTL mostrará la información.
- formUsers.jsp: Incluirá un formulario HTML para crear usuarios y cuando lo enviemos, si no hay errores, persistirá la información en la BD.

Para crear la carpeta views, clic derecho sobre la carpeta WEB-INF, seleccionar New – Folder y establecer el nombre.

Para crear los JSP, clic derecho sobre la carpeta views, seleccionar New – JSP y establecer el nombre. A continuación, se muestra el código de las dos vistas.

Vista formUsers.jsp

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Web App</title>
    </head>
    <body>
        <h1>Web App</h1>
        <h3>Añadir Usuario</h3>

        <form action="/app/user/save" method="POST">
            <label for="name">Nombre:</label>
            <input id="name" type="text" name="name"><br />
            <label for="email">Correo:</label>
            <input id="email" type="text" name="email"><br />
            <label for="phone">Teléfono: </label>
            <input id="phone" type="text" name="phone"><br />

            <input type="submit" value="Guardar" />
        </form>

        <a href="/app/users">Inicio</a>

    </body>
</html>
```

Vista users.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Wen App</title>
    </head>
    <body>
        <nav> | <a href="/app/user/new">Crear Nuevo Usuario</a> | </nav>
        <h1>Web App</h1>

        <c:if test="${!empty requestScope.users}">
            <table>
                <tr>
                    <th>ID</th>
                    <th>Nombre</th>
                    <th>Correo</th>
                    <th>Teléfono</th>
                </tr>
                <c:forEach var="user" items="${requestScope.users }" >
                    <tr>
                        <td>${user.id}</td>
                        <td>${user.name}</td>
                        <td>${user.email}</td>
                        <td>${user.phone}</td>
                    </tr>
                </c:forEach>
            </table>
        </c:if>
        <c:if test="${empty requestScope.users}">
            <p>Oops! No hay Usuarios todavía!</p>
        </c:if>

    </body>
</html>
```

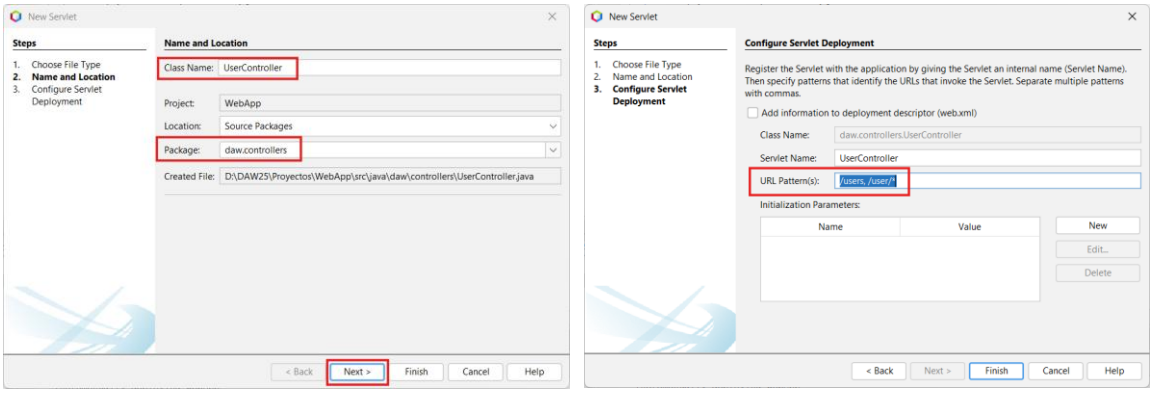
7.3 Controlador

Los controladores serán Servlets que atenderán diferentes acciones en función de las URLs que los invoquen. En este caso, crearemos un Servlet (Controlador) que atenderá las URLs:

http://host:8080/app/users (GET) Para mostrar los usuarios
http://host:8080/app/user/new (GET) Para mostrar el form de usuario para crearlo
http://host:8080/app/user/save (POST) Para guardar (Persistir) el usuario en la BD

Para crear el Servlet, clic derecho sobre el proyecto, seleccionar New – Servlet y completar el asistente:

- Establecemos el nombre (UserController), el paquete (daw.controllers) y en la siguiente pantalla las URLs mediante las que será invocado (/users, /user/*).



A continuación, vamos a editar el código para realizar las acciones que deseamos:

1. Añadimos un atributo para el gestor de entidades (Entity Manager), este será inyectado a partir de la Unidad de Persistencia creada anteriormente (AgendaPU).
2. Añadimos un atributo para gestionar las transacciones, anotándolo como Resource para que también sea inyectado.
3. Opcionalmente, muy adecuado para analizar la seguridad de la aplicación, añadimos un atributo para mantener un log de nuestra aplicación.

```
@WebServlet(name = "UserController", urlPatterns = {"/users", "/user/*"})
public class UserController extends HttpServlet {

    @PersistenceContext(unitName = "WebAppPU")
    private EntityManager em;
    @Resource
    private UserTransaction utx;

    private static final Logger Log = Logger.getLogger(UserController.class.getName());
```

```
@PersistenceContext(unitName = "WebAppPU")
private EntityManager em;
@Resource
private UserTransaction utx;

private static final Logger Log = Logger.getLogger(UserController.class.getName());
```

NOTA: Recuerda importar las librerías necesarias. Para Logger es necesario importar las librerías import java.util.logging.Logger e import java.util.logging.Level.

En este ejercicio, vamos a eliminar el método processRequest, dejando los métodos doGet y doPost, en los que insertaremos el código.

Método doGet():

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

    String vista;

    String accion = "/users";
    if (request.getServletPath().equals("/user")) {
        if (request.getPathInfo() != null) {
            accion = request.getPathInfo();
        } else {
            accion = "error";
        }
    }
    switch (accion) {
        case "/users" -> {
            List<Users> lu;
            TypedQuery<Users> q = em.createNamedQuery("Users.findAll", Users.class);
            lu = q.getResultList();
            request.setAttribute("users", lu);
            vista = "users";
        }
        case "/new" -> {
            vista = "formUsers";
        }
        default -> {
            vista = "error";
        }
    }

    RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/views/" + vista + ".jsp");
    rd.forward(request, response);
}
```

request.getServletPath() Devuelve la parte de la URL correspondiente al Servlet

request.getPathInfo() Devuelve la parte de la URL siguiente al Servlet

URL	getServletPath()	getPathInfo()
http://localhost:8080/app/users	/users	null
http://localhost:8080/app/user/new	/user	/new

En función de la URL, definimos la acción a realizar:

- Para /users la acción consultará los objetos Users disponibles en la BD y delegará la petición en la vista users.jsp a la cual se le pasará la lista con los usuarios, para que los muestre mediante JSTL.
- Para /user/new se delegará la petición a la vista formUsers.jsp
- Finalmente, hemos incluido una vista error.jsp para contemplar las acciones no permitidas. (Puedes crear esta vista en /WEB-INF/views con el contenido de error).

Si analizamos el formulario, veremos que la acción (action) invoca a la URL user/save mediante una petición POST, que se implementa en el método doPost().

Método doPost():

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

    String accion = request.getPathInfo();

    if (accion.equals("/save")) {

        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        try {

            if (name.isEmpty() || email.isEmpty() || phone.isEmpty()) {
                throw new NullPointerException();
            }
            Users u = new Users(name, email, phone);
            save(u);
            response.sendRedirect("http://localhost:8080/app/users");

        } catch (Exception e) {
            request.setAttribute("msg", "Error: datos no válidos");
            RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/views/error.jsp");
            rd.forward(request, response);
        }

    } else {
        RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/views/error.jsp");
        rd.forward(request, response);
    }

}
```

Este método atiende peticiones POST y en el código implementamos la acción /user/save; en caso de cualquier otra petición delegaremos en la vista error.

En esta acción, en primer lugar, obtenemos los parámetros enviados desde el formulario, comprobamos que los datos cumplen con los requisitos (**NOTA**: en este caso, tan solo comprobamos que no estén en blanco, pero por cuestiones de seguridad, es necesario chequear de forma minuciosa que los datos cumplen con las políticas de seguridad); creamos un objeto de tipo Users y lo persistimos mediante el Entity Manager. Para ello, se ha creado el método save():

```
public void save(Users u) {
    Long id = u.getId();
    try {
        utx.begin();
        if (id == null) {
            em.persist(u);
            Log.log(Level.INFO, "New User saved");
        } else {
            Log.log(Level.INFO, "User {0} updated", id);
            em.merge(u);
        }
        utx.commit();
    } catch (Exception e) {
        Log.log(Level.SEVERE, "exception caught", e);
        throw new RuntimeException(e);
    }
}
```

El método save(), recibe un objeto de tipo Users y mediante el Entity Manager, si el objeto es nuevo, esto es no tiene aún un ID, lo persiste en la BD mediante em.persist(obj), pero si ya está en la BD y, por lo tanto, tenemos un ID, entonces lo actualiza em.merge(obj) con los nuevos valores.

Finalmente, la acción si todo ha ido bien redirige la petición a una nueva URL o si ha habido alguna Excepción delega la petición en la vista error.

7.4 Mejorando las vistas: CSS y JS

NOTA: este contenido debe estar en Web Pages, no puede estar dentro de la carpeta /WEB-INF ya que debe ser accesible desde Internet.

Crear una hoja de estilo CSS:

- Crear una carpeta, llamada css, para guardar los estilos, mediante clic derecho sobre la carpeta Web Pages, seleccionar New – Folder y completar el asistente con el nombre de la carpeta.
- Crear la hoja de estilos, main.css, con clic derecho sobre la carpeta css, creada anteriormente, y selecciona New – Other – Web – Cascading Style Sheet… y completar el asistente
- Incluir el siguiente contenido en el fichero main.css:

```
body {
    margin: 50px;
}

h1 { color: red; }

label {
    display: inline-block;
    width: 4em;
}

table {
    font-size: 20px;
}

th {
    background-color: black;
    color: white;
}


tr:nth-child(odd) {
    background-color: darkblue;
}

tr:nth-child(even) {
    background-color: darkgray;
}

.derecha {
    text-align: right;
}
```

Crear un fichero JavaScript para incluir la funcionalidad en el cliente:

- Crear una carpeta, llamada js, mediante clic derecho sobre la carpeta Web Pages, seleccionar New – Folder y completar el asistente con el nombre de la carpeta.
- Crear el fichero JavaScript, functions.js, con clic derecho sobre la carpeta js, creada anteriormente, y selecciona New – Other – Web – JavaScript File… y completar el asistente
- Incluir el siguiente contenido en el fichero functions.js:

```
let f = document.getElementById("formulario");
f.addEventListener("submit", checkData);

function checkData(evt) {

    let ok = true;

    let name = document.getElementById("name").value;

    if (name === "") {
        alert("El nombre es obligatorio");
        evt.preventDefault();
        ok = false;
    }

    return ok;
}
```

Modificar las vistas para incluir la hoja de estilos y el fichero JavaScript:

Vista formUsers.jsp

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Web App</title>
        <link href="/app/css/main.css" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <h1>Wep App</h1>
        <h3>Añadir Usuario</h3>

        <form id="formulario" action="/app/user/save" method="POST">

            <label for="name">Nombre:</label>
            <input id="name" type="text" name="name"><br />
            <label for="email">Correo:</label>
            <input id="email" type="text" name="email"><br />
            <label for="phone">Teléfono: </label>
            <input id="phone" type="text" name="phone"><br />

            <input type="submit" value="Guardar" />

        </form>

        <a href="/app/users">Inicio</a>

        <script src="/app/js/functions.js"></script>

    </body>
</html>
```

Vista users.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Web App</title>
        <link href="/app/css/main.css" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <nav> | <a href="/app/user/new">Crear Nuevo Usuario</a> | </nav>
        <h1>Wep App</h1>

        <c:if test="${!empty requestScope.users}">
            <table>
                <tr>
                    <th>ID</th>
                    <th>Nombre</th>
                    <th>Correo</th>
                    <th>Teléfono</th>
                </tr>
                <c:forEach var="user" items="${requestScope.users}" >
                    <tr>
                        <td>${user.id}</td>
                        <td>${user.name}</td>
                        <td>${user.email}</td>
                        <td class="derecha">${user.phone}</td>
                    </tr>
            </c:forEach>
        </table>
    </c:if>
    <c:if test="${empty requestScope.users}">
        <p>Oops! No hay Usuarios todavía!</p>
    </c:if>

    <script src="/app/js/functions.js"></script>

</body>
</html>
```