

# **PIA – Agenda.**

Laboratorio de Programación Orientada a Objetos.

**Alumno:** Jeremy Uriel Rossell Segura.

**Matrícula:** 2173396.

**Grupo:** 036.

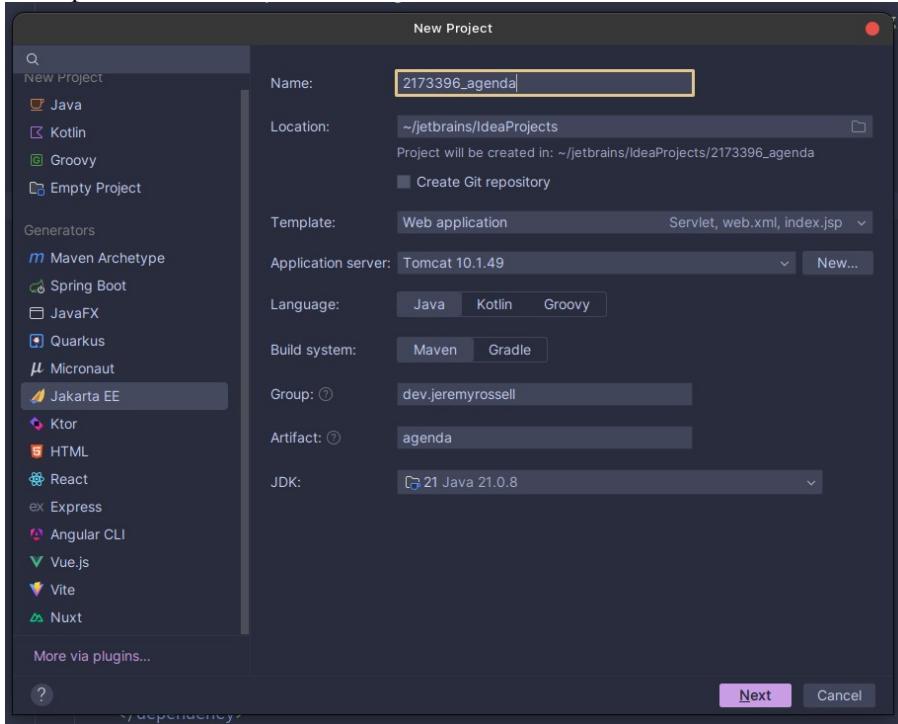
**Horario:** VIERNES 7:00-9:00.

**Materia:** Laboratorio de Programación Orientada a objetos.

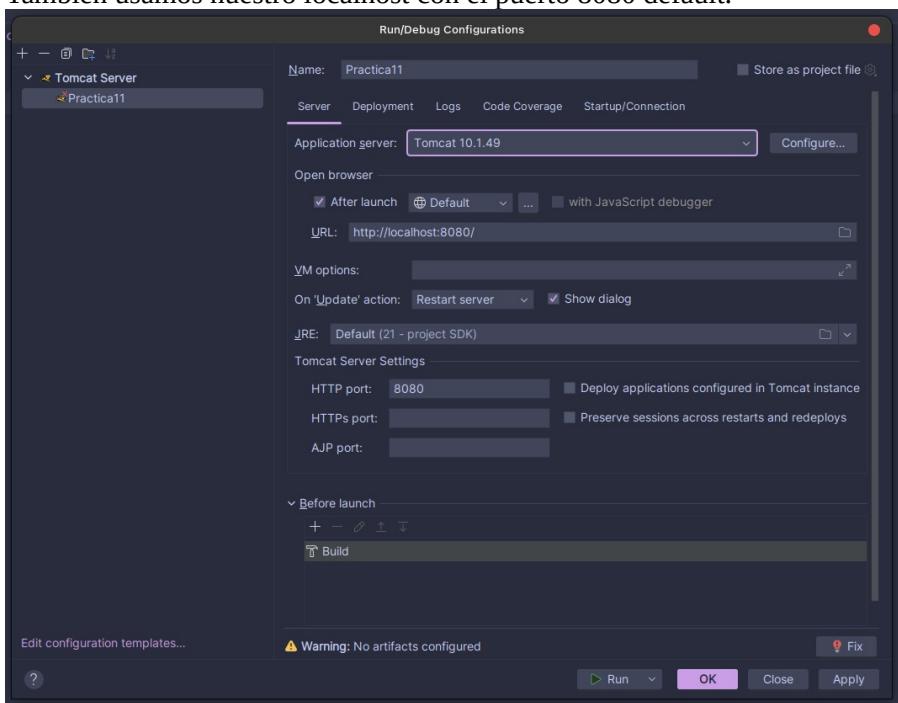
**Docente:** Jorge Alberto Islas Pineda.

# Inicialización.

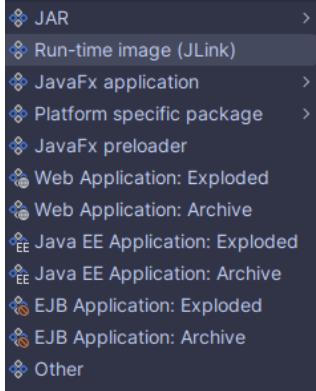
Primero creamos nuestro proyecto IntelliJ con Jakarta EE 11 para Web Application, y usamos Tomcat 10.1.49 como nuestro application server, y elegimos el local. Descargué el tar.gz de la web oficial de Tomcat, y lo guardé en mi directorio `~/jetbrains/Apache/apache-tomcat-10.1.49`, como se aprecia, es directorio con permiso de usuario, esto es porque es mi computadora personal, si quisieramos usarlo en un servidor, ahí no usamos el tar.gz, sino el package manager de nuestra distribución Linux, como Debian es el rey de los servidores, usariamos `sudo apt install tomcat10`, o si hay una versión nueva checar usando `sudo apt update && apt search tomcat`.



Usamos el Tomcat de forma local para no rompernos la cabeza con los puertos, ni configurar el servidor, ni permisos de root ni nada de eso. Simplemente buscamos la carpeta ya extraída en el directorio ya dado. También usamos nuestro localhost con el puerto 8080 default.



En artefactos usamos “Web Application: Exploded”



Ahora rellenamos los campos de groupId con un paquete, si estamos en producción puede ser el nombre del proyecto o empresa, como soy yo en un ámbito académico, usaré “dev.jeremyrossell” como el paquete. Y en artifactId ponemos el nombre de nuestro proyecto “agenda”. El packaging por default debe ser .war.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>dev.jeremyrossell</groupId>
    <artifactId>agenda</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>6.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <version>8.2.0</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
```

Y creamos nuestra base de datos en MySQL Workbench.

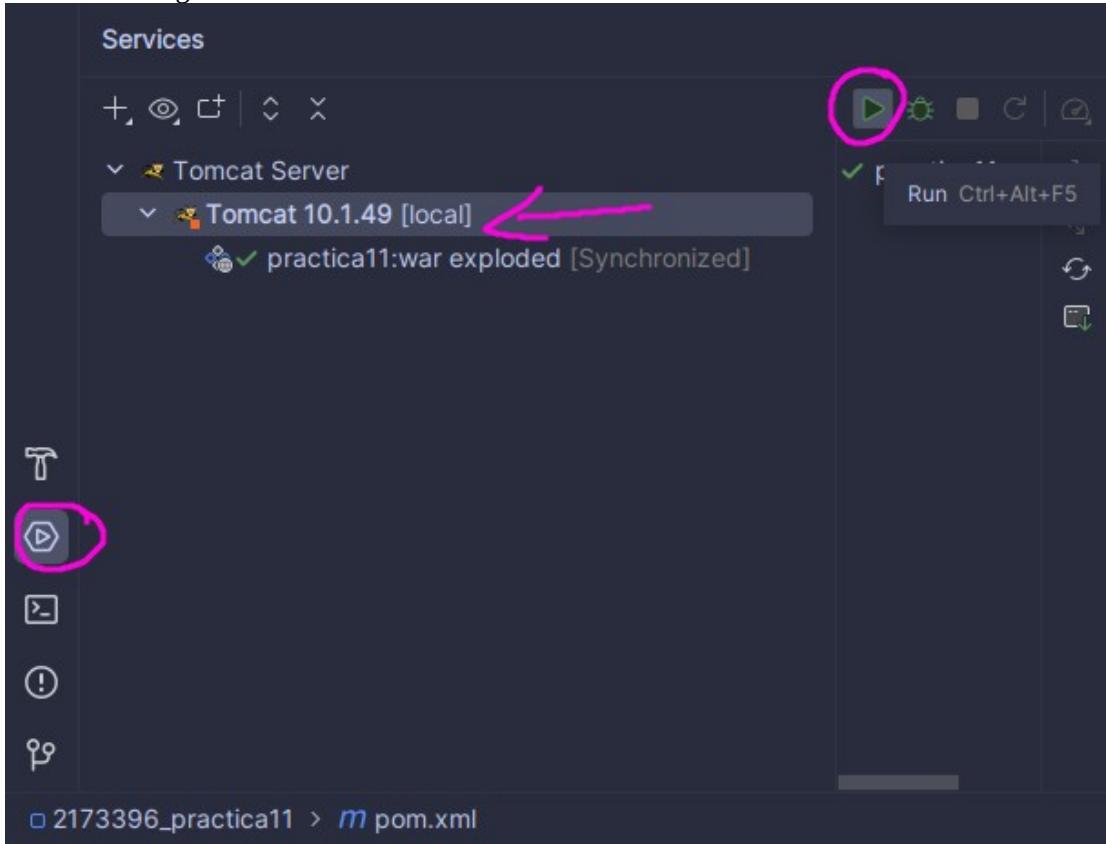
The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree shows the 'agenda\_db' schema selected. In the central pane, a query editor window titled 'Query 1' contains the following SQL code:

```
1 • CREATE DATABASE agenda_db;
2 • USE agenda_db;
3
4 • ◇ CREATE TABLE contactos (
5     id INT AUTO_INCREMENT PRIMARY KEY,
6     nombre VARCHAR(50),
7     apellidoPaterno VARCHAR(50),
8     apellidoMaterno VARCHAR(50),
9     sexo CHAR(1),
10    telefono VARCHAR(20),
11    direccion VARCHAR(100),
12    tipoContacto VARCHAR(20)
13 );
14
15 -- Insertamos un dato de prueba
16 • INSERT INTO contactos (nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto)
17   VALUES ('Juan', 'Perez', 'Lopez', 'M', '555-1234', 'Calle Falsa 123', 'TRABAJO');
```

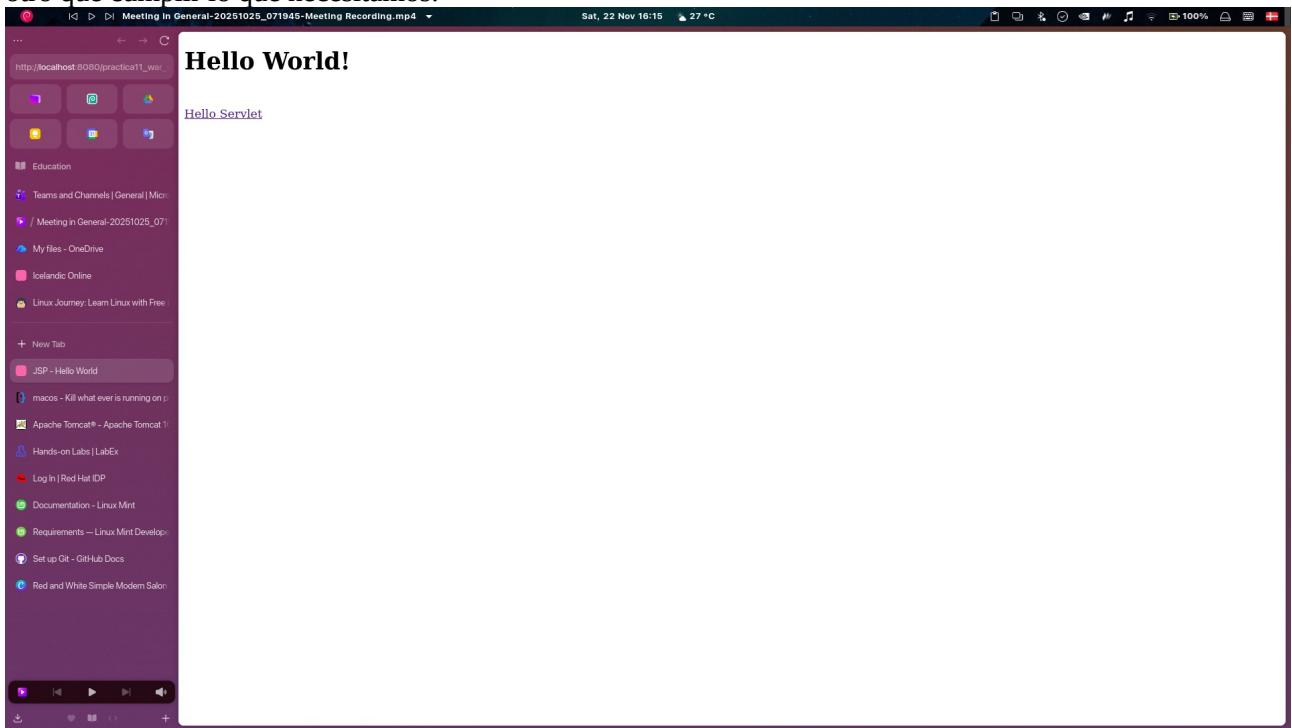
Below the query editor, the 'Action Output' table shows the results of the executed queries:

#	Time	Action	Message	Duration / Fetch
1	00:48:14	CREATE DATABASE agenda_db	1 row(s) affected	0.015 sec
2	00:48:14	USE agenda_db	0 row(s) affected	0.00052 sec
3	00:48:14	CREATE TABLE contactos ( id INT AUTO_INCREMENT P...	0 row(s) affected	0.116 sec
4	00:48:15	INSERT INTO contactos (nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto)	1 row(s) affected	0.014 sec

Una vez chequemos que nuestro pom.xml esté correcto, vamos a la sección de servicios, seleccionamos nuestro Tomcat server, y la instancia local, y le damos a Run para correr nuestro servidor web y probar que todo está en regla hasta el momento.

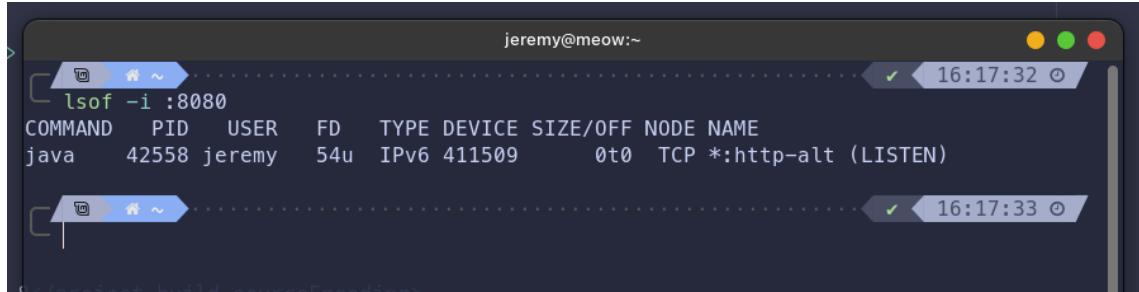


Y como podemos apreciar, si funcionó, este html es uno que se generó por defecto, pero nosotros crearemos otro que cumplir lo que necesitamos.



## Código.

Ya que tenemos nuestro proyecto con todo “bien conectado”, entonces ya iniciamos con lo que queremos. Primero cerramos el proceso del Tomcat, y nos aseguramos que nuestro puerto 8080 del localhost no está siendo usado por nuestra instancia Tomcat en Java o alguna otra, si sale, debemos cerrar bien el servidor, si no sale nada, todo está como debe ser.

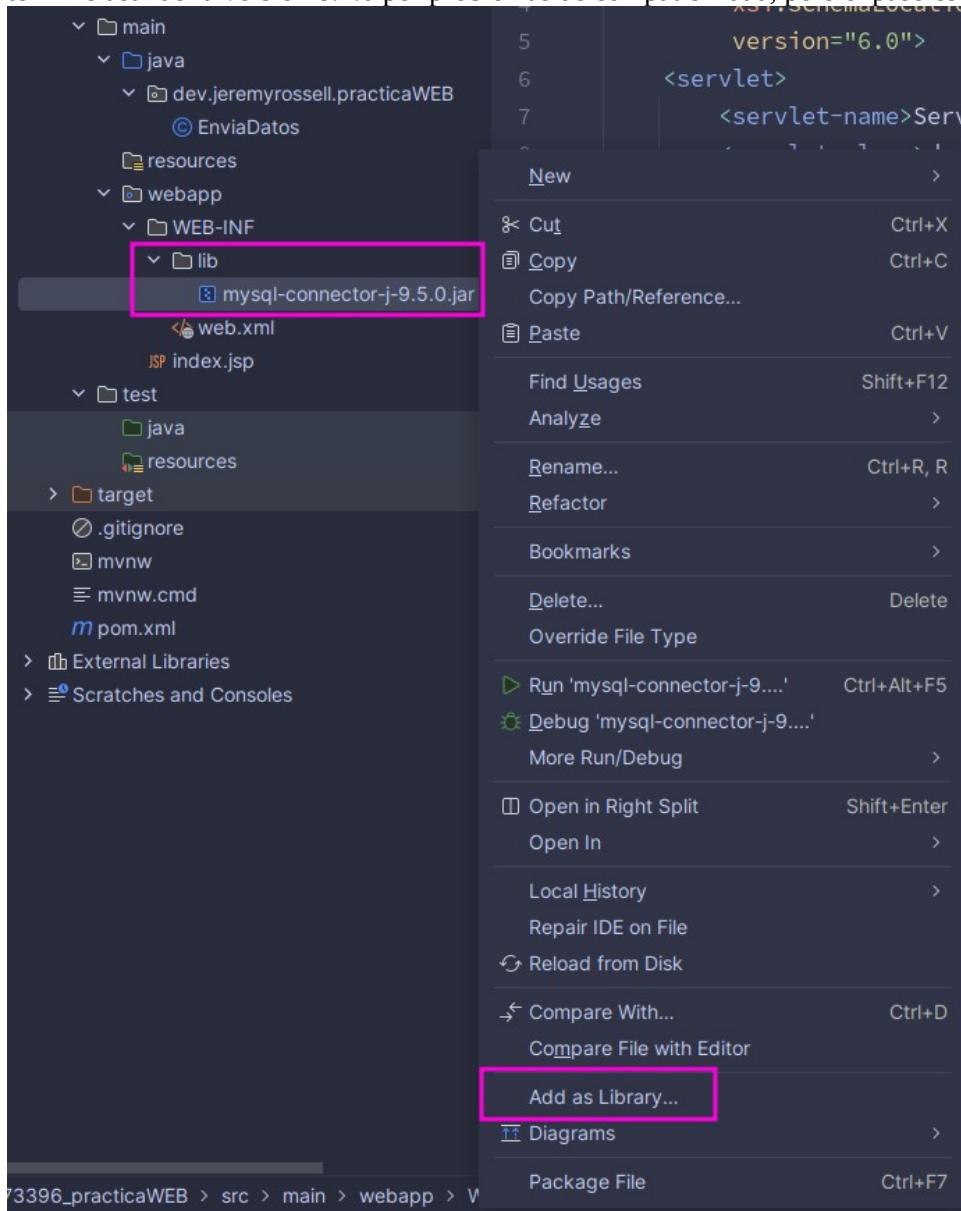


```
jeremy@meow:~
```

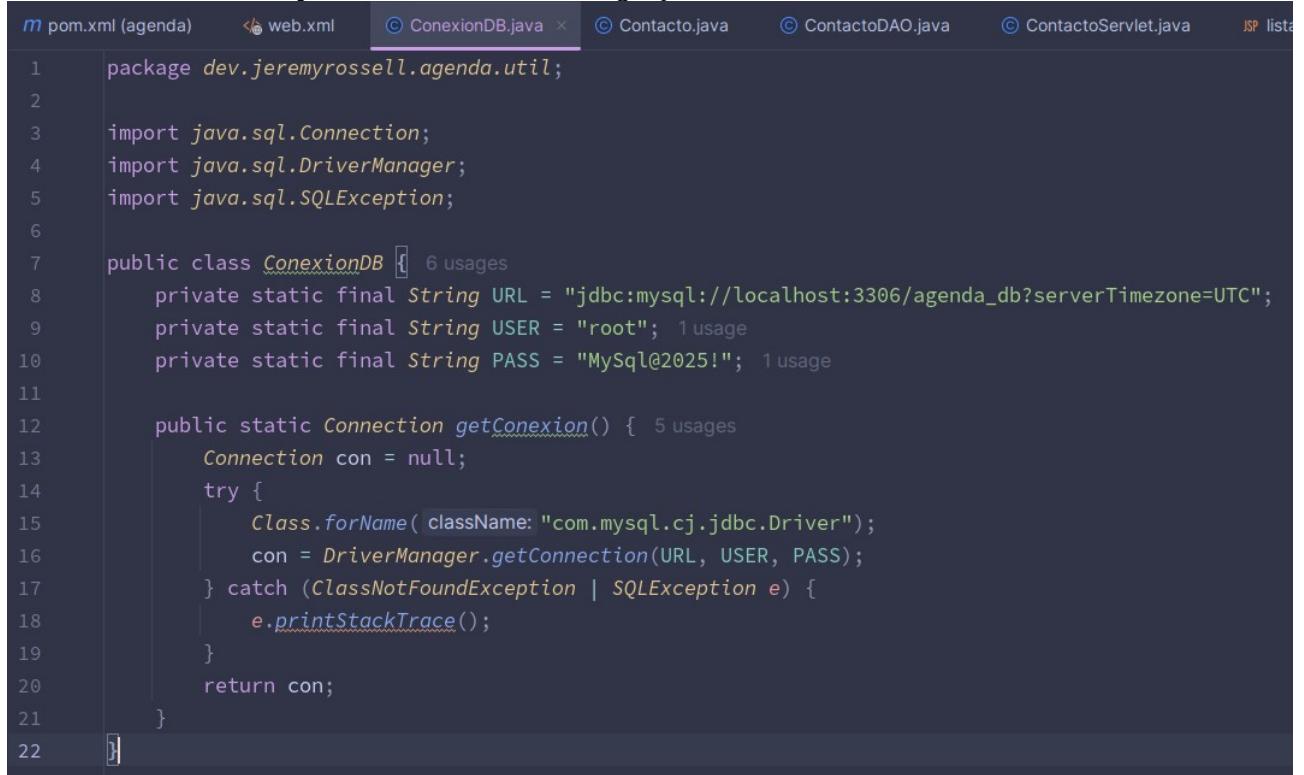
```
lsof -i :8080
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
java 42558 jeremy 54u IPv6 411509      0t0  TCP *:http-alt (LISTEN)
```

```
jeremy@meow:~
```

Como vamos a conectar una base de datos MySQL, de una vez arrastramos el .jar de mysql-connector-j adentro de nuestra carpeta de proyecto, en el directorio src/main/webapp/WEB-INF/lib/, cabe recalcar que terminé usando la versión 8.2.0 por problemas de compatibilidad, pero el paso es el mismo.

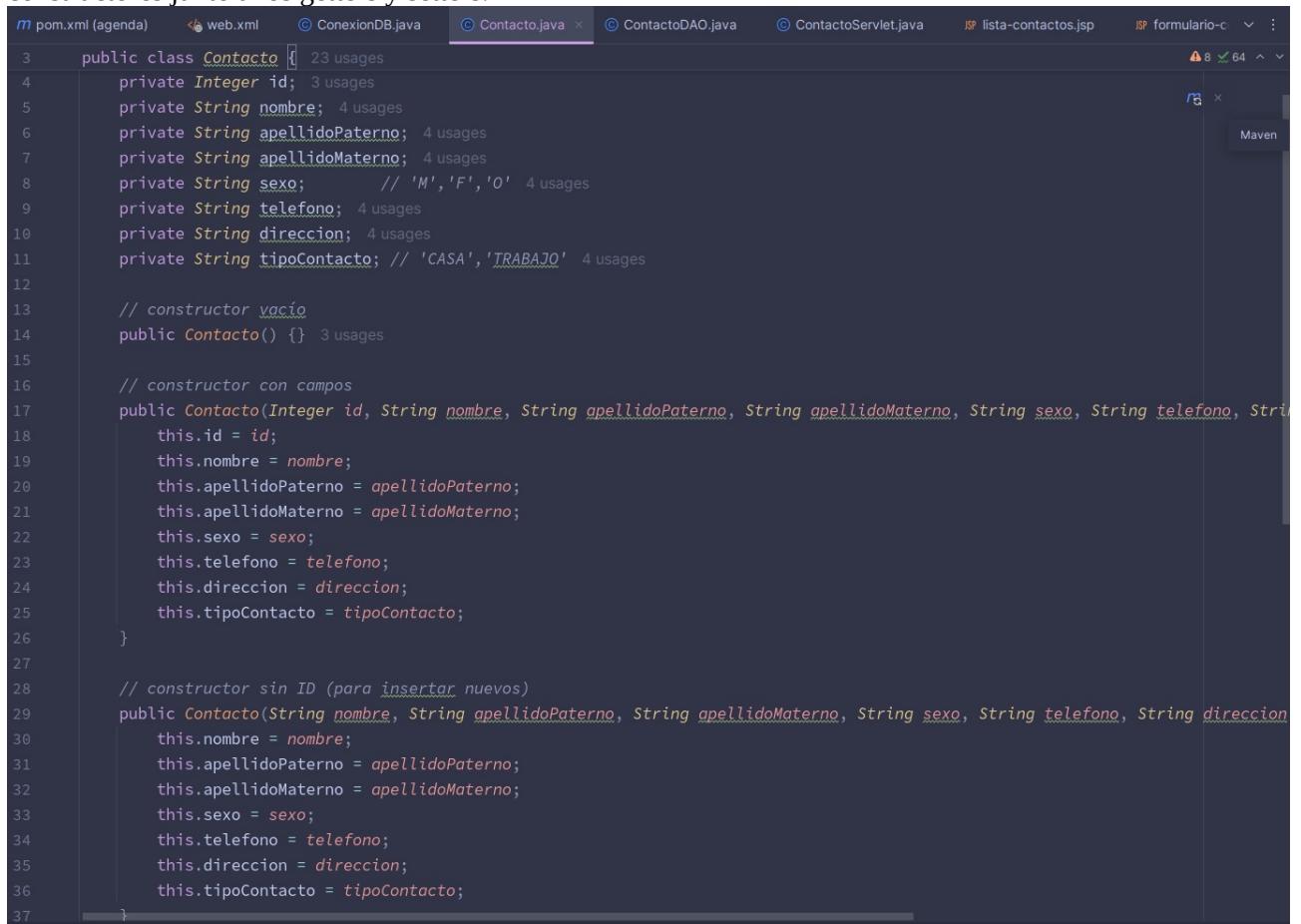


Dentro de **ConexionDB.java**, creamos nuestro código que conectará con nuestra base de datos.



```
1 package dev.jeremyrossell.agenda.util;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class ConexionDB {
8     private static final String URL = "jdbc:mysql://localhost:3306/agenda_db?serverTimezone=UTC";
9     private static final String USER = "root";
10    private static final String PASS = "MySQL@2025!";
11
12    public static Connection getConexion() {
13        Connection con = null;
14        try {
15            Class.forName(className: "com.mysql.cj.jdbc.Driver");
16            con = DriverManager.getConnection(URL, USER, PASS);
17        } catch (ClassNotFoundException | SQLException e) {
18            e.printStackTrace();
19        }
20        return con;
21    }
22 }
```

Dentro de **Contacto.java** definimos los datos que tomaremos de cada persona para la agenda. Creamos los constructores junto a los getters y setters.



```
3 public class Contacto {
4     private Integer id;
5     private String nombre;
6     private String apellidoPaterno;
7     private String apellidoMaterno;
8     private String sexo; // 'M', 'F', 'O'
9     private String telefono;
10    private String direccion;
11    private String tipoContacto; // 'CASA', 'TRABAJO'
12
13    // constructor vacío
14    public Contacto() {}
15
16    // constructor con campos
17    public Contacto(Integer id, String nombre, String apellidoPaterno, String apellidoMaterno, String sexo, String telefono, String direccion) {
18        this.id = id;
19        this.nombre = nombre;
20        this.apellidoPaterno = apellidoPaterno;
21        this.apellidoMaterno = apellidoMaterno;
22        this.sexo = sexo;
23        this.telefono = telefono;
24        this.direccion = direccion;
25        this.tipoContacto = tipoContacto;
26    }
27
28    // constructor sin ID (para insertar nuevos)
29    public Contacto(String nombre, String apellidoPaterno, String apellidoMaterno, String sexo, String telefono, String direccion) {
30        this.nombre = nombre;
31        this.apellidoPaterno = apellidoPaterno;
32        this.apellidoMaterno = apellidoMaterno;
33        this.sexo = sexo;
34        this.telefono = telefono;
35        this.direccion = direccion;
36        this.tipoContacto = tipoContacto;
37    }
}
```

```

m pom.xml (agenda)  web.xml  ConexionDB.java  Contacto.java  ContactoDAO.java  ContactoServlet.java  lista-contactos.jsp  formulario-c  :
3     public class Contacto {  23 usages
29         public Contacto(String nombre, String apellidoPaterno, String apellidoMaterno, String sexo, String telefono, String direccion)
33             this.nombre = nombre;
34             this.apellidoPaterno = apellidoPaterno;
35             this.apellidoMaterno = apellidoMaterno;
36             this.sexo = sexo;
37             this.telefono = telefono;
38             this.direccion = direccion;
39             this.tipoContacto = tipoContacto;
40
41     }
42
43     // getters y setters
44     public Integer getId() { return id; }  5 usages
45     public void setId(Integer id) { this.id = id; }  no usages
46
47     public String getNombre() { return nombre; }  4 usages
48     public void setNombre(String nombre) { this.nombre = nombre; }  no usages
49
50     public String getApellidoPaterno() { return apellidoPaterno; }  4 usages
51     public void setApellidoPaterno(String apellidoPaterno) { this.apellidoPaterno = apellidoPaterno; }  no usages
52
53     public String getApellidoMaterno() { return apellidoMaterno; }  4 usages
54     public void setApellidoMaterno(String apellidoMaterno) { this.apellidoMaterno = apellidoMaterno; }  no usages
55
56     public String getSexo() { return sexo; }  6 usages
57     public void setSexo(String sexo) { this.sexo = sexo; }  no usages
58
59     public String getTelefono() { return telefono; }  4 usages
60     public void setTelefono(String telefono) { this.telefono = telefono; }  no usages
61
62     public String getDireccion() { return direccion; }  4 usages
63     public void setDireccion(String direccion) { this.direccion = direccion; }  no usages
64
65     public String getTipoContacto() { return tipoContacto; }  5 usages
66     public void setTipoContacto(String tipoContacto) { this.tipoContacto = tipoContacto; }  no usages
67
68 }

```

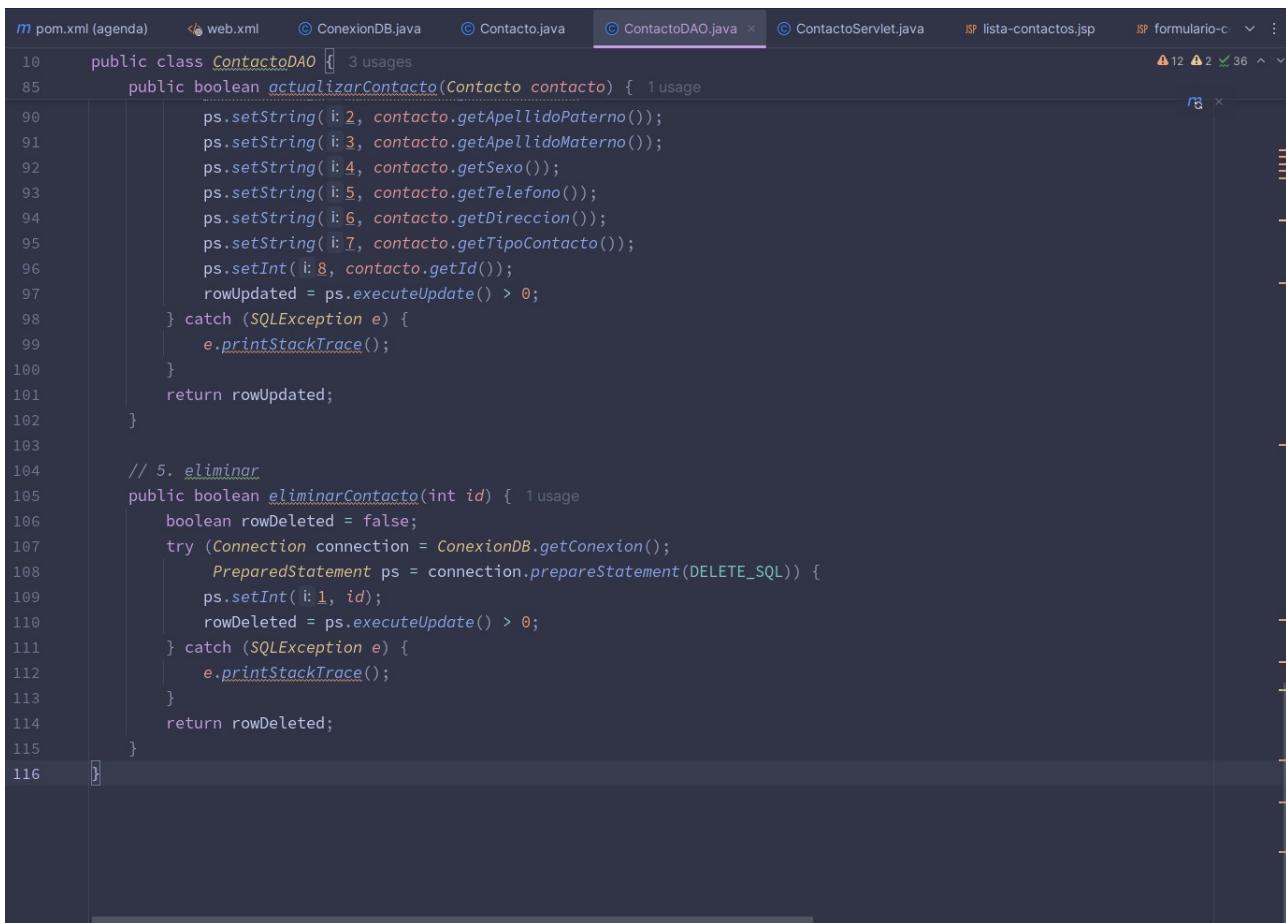
Dentro de **ContactoDAO.java** tenemos nuestro Data Access Object, aquí es donde interactuamos con la base de datos mediante nuestra aplicación web de Tomcat. Como nota, lo subrayado en rojo y amarillo no son errores de sintaxis ni de runtime, son recomendaciones de IntelliJ.

```

m pom.xml (agenda)  web.xml  ConexionDB.java  Contacto.java  ContactoDAO.java  ContactoServlet.java  lista-contactos.jsp  formulario-c  :
1 package dev.jeremyrossell.agenda.dao;
2
3 import dev.jeremyrossell.agenda.model.Contacto;
4 import dev.jeremyrossell.agenda.util.ConexionDB;
5
6 import java.sql.*;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 public class ContactoDAO {  3 usages
11
12     private static final String INSERT_SQL = "INSERT INTO contactos (nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto) VALUES (?, ?, ?, ?, ?, ?, ?)";
13     private static final String SELECT_ALL_SQL = "SELECT * FROM contactos";  1 usage
14     private static final String SELECT_BY_ID_SQL = "SELECT * FROM contactos WHERE id = ?";  1 usage
15     private static final String DELETE_SQL = "DELETE FROM contactos WHERE id = ?";  1 usage
16     private static final String UPDATE_SQL = "UPDATE contactos SET nombre = ?, apellidoPaterno = ?, apellidoMaterno = ?, sexo = ?, telefono = ?, direccion = ?, tipoContacto = ? WHERE id = ?";
17
18     // 1. insertar
19     @
20     public void insertarContacto(Contacto contacto) {  1 usage
21         try (Connection connection = ConexionDB.getConexion();
22               PreparedStatement ps = connection.prepareStatement(INSERT_SQL)) {
23             ps.setString(1, contacto.getNombre());
24             ps.setString(2, contacto.getApellidoPaterno());
25             ps.setString(3, contacto.getApellidoMaterno());
26             ps.setString(4, contacto.getSexo());
27             ps.setString(5, contacto.getTelefono());
28             ps.setString(6, contacto.getDireccion());
29             ps.setString(7, contacto.getTipoContacto());
30             ps.executeUpdate();
31         } catch (SQLException e) {
32             e.printStackTrace();
33         }
34     }
35
36     // 2. seleccionar todos

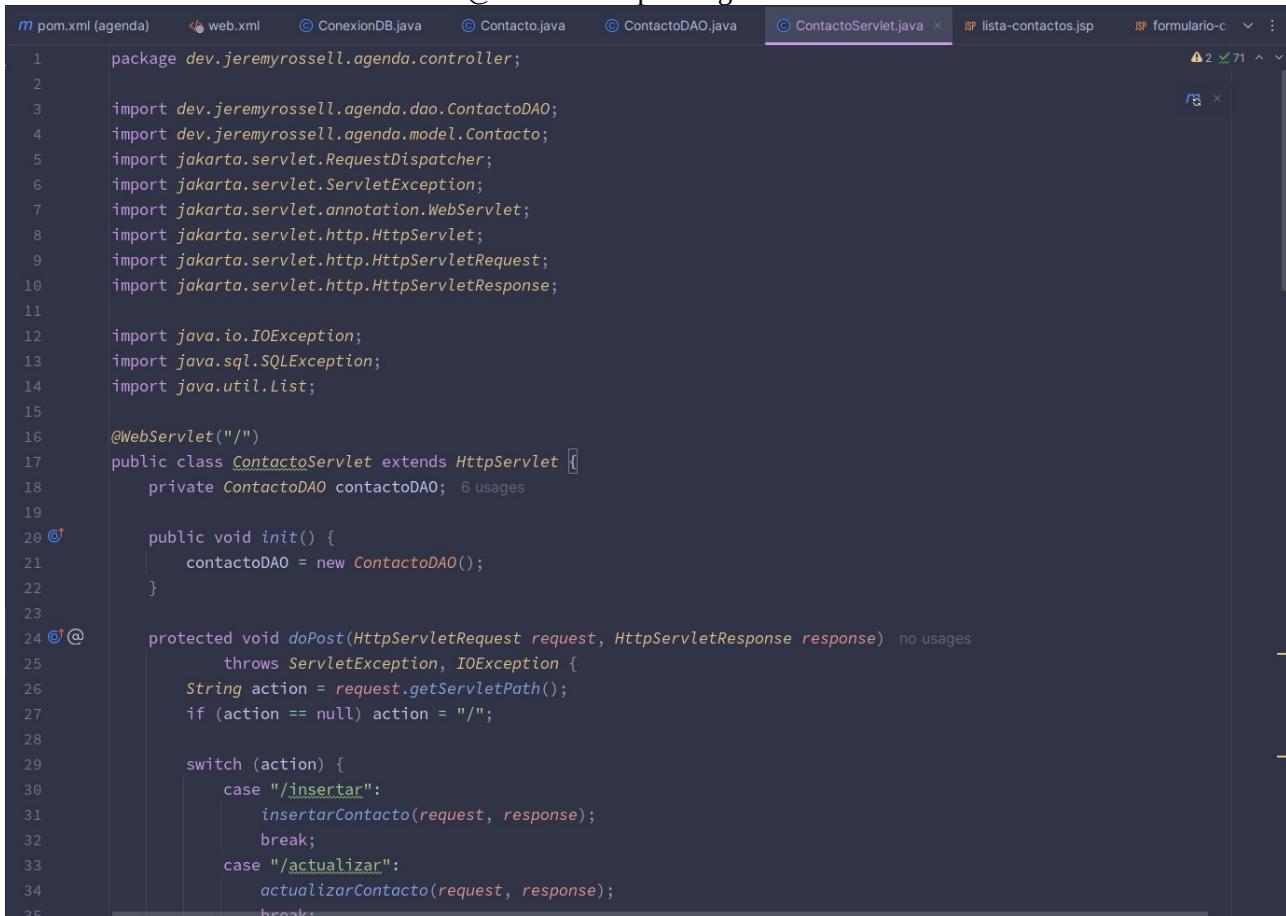
```

```
m pom.xml (agenda)  web.xml  ConexionDB.java  Contacto.java  ContactoDAO.java  ContactoServlet.java  lista-contactos.jsp  formulario-c ... : 10 public class ContactoDAO { 3 usages 35     // 2. seleccionar todos 36     public List<Contacto> seleccionarTodos() { 1 usage 37         List<Contacto> contactos = new ArrayList<>(); 38         try (Connection connection = ConexionDB.getConexion(); 39             PreparedStatement ps = connection.prepareStatement(SELECT_ALL_SQL); 40             ResultSet rs = ps.executeQuery()) { 41             while (rs.next()) { 42                 contactos.add(new Contacto( 43                     rs.getInt(s: "id"), 44                     rs.getString(s: "nombre"), 45                     rs.getString(s: "apellidoPaterno"), 46                     rs.getString(s: "apellidoMaterno"), 47                     rs.getString(s: "sexo"), 48                     rs.getString(s: "telefono"), 49                     rs.getString(s: "direccion"), 50                     rs.getString(s: "tipoContacto") 51                 )); 52             } 53         } catch (SQLException e) { 54             e.printStackTrace(); 55         } 56         return contactos; 57     } 58 59     // 3. seleccionar por ID (para editar) 60     public Contacto seleccionarPorId(int id) { 1 usage 61         Contacto contacto = null; 62         try (Connection connection = ConexionDB.getConexion(); 63             PreparedStatement ps = connection.prepareStatement(SELECT_BY_ID_SQL)) { 64             ps.setInt(i: 1, id); 65             ResultSet rs = ps.executeQuery(); 66             if (rs.next()) { 67                 contacto = new Contacto( 68                     rs.getInt(s: "id"), 69                     rs.getString(s: "nombre"), 70                     rs.getString(s: "apellidoPaterno"), 71                     rs.getString(s: "apellidoMaterno"), 72                     rs.getString(s: "sexo"), 73                     rs.getString(s: "telefono"), 74                     rs.getString(s: "direccion"), 75                     rs.getString(s: "tipoContacto") 76                 ); 77             } 78         } catch (SQLException e) { 79             e.printStackTrace(); 80         } 81         return contacto; 82     } 83 84     // 4. actualizar 85     public boolean actualizarContacto(Contacto contacto) { 1 usage 86         boolean rowUpdated = false; 87         try (Connection connection = ConexionDB.getConexion(); 88             PreparedStatement ps = connection.prepareStatement(UPDATE_SQL)) { 89             ps.setString(i: 1, contacto.getNombre()); 90             ps.setString(i: 2, contacto.getApellidoPaterno()); 91             ps.setString(i: 3, contacto.getApellidoMaterno()); 92             ps.setString(i: 4, contacto.getSexo()); 93             ps.setString(i: 5, contacto.getTelefono()); 94             ps.setString(i: 6, contacto.getDireccion()); 95             ps.setString(i: 7, contacto.getTipoContacto()); 96             ps.setInt(i: 8, contacto.getId()); 97             rowUpdated = ps.executeUpdate() > 0; 98         } catch (SQLException e) { 99             e.printStackTrace();
```



```
10     public class ContactoDAO { 3 usages
11         public boolean actualizarContacto(Contacto contacto) { 1 usage
12             ps.setString( i: 2, contacto.getApellidoPaterno());
13             ps.setString( i: 3, contacto.getApellidoMaterno());
14             ps.setString( i: 4, contacto.getSexo());
15             ps.setString( i: 5, contacto.getTelefono());
16             ps.setString( i: 6, contacto.getDireccion());
17             ps.setString( i: 7, contacto.getTipoContacto());
18             ps.setInt( i: 8, contacto.getId());
19             rowUpdated = ps.executeUpdate() > 0;
20         } catch (SQLException e) {
21             e.printStackTrace();
22         }
23         return rowUpdated;
24     }
25
26     // 5. eliminar
27     public boolean eliminarContacto(int id) { 1 usage
28         boolean rowDeleted = false;
29         try (Connection connection = ConexionDB.getConexion();
30              PreparedStatement ps = connection.prepareStatement(DELETE_SQL)) {
31             ps.setInt( i: 1, id);
32             rowDeleted = ps.executeUpdate() > 0;
33         } catch (SQLException e) {
34             e.printStackTrace();
35         }
36         return rowDeleted;
37     }
38 }
```

Dentro de **ContactoServlet.java** tenemos nuestro servlet, que actúa como un semáforo. Decide qué acción realizar basándose en la URL. Usamos `@WebServlet` para registrarlo sin ensuciar el web.xml.



```
1 package dev.jeremyrossell.agenda.controller;
2
3 import dev.jeremyrossell.agenda.dao.ContactoDAO;
4 import dev.jeremyrossell.agenda.model.Contacto;
5 import jakarta.servlet.RequestDispatcher;
6 import jakarta.servlet.ServletException;
7 import jakarta.servlet.annotation.WebServlet;
8 import jakarta.servlet.http.HttpServlet;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12 import java.io.IOException;
13 import java.sql.SQLException;
14 import java.util.List;
15
16 @WebServlet("/")
17 public class ContactoServlet extends HttpServlet {
18     private ContactoDAO contactoDAO; 6 usages
19
20     public void init() {
21         contactoDAO = new ContactoDAO();
22     }
23
24     protected void doPost(HttpServletRequest request, HttpServletResponse response) no usages
25         throws ServletException, IOException {
26         String action = request.getServletPath();
27         if (action == null) action = "/";
28
29         switch (action) {
30             case "/insertar":
31                 insertarContacto(request, response);
32                 break;
33             case "/actualizar":
34                 actualizarContacto(request, response);
35                 break;
36         }
37     }
38 }
```

```
m pom.xml (agenda)  web.xml  ConexionDB.java  Contacto.java  ContactoDAO.java  ContactoServlet.java  lista-contactos.jsp  formulario-c  :  
17     public class ContactoServlet extends HttpServlet {  
24         protected void doPost(HttpServletRequest request, HttpServletResponse response) no usages  
33             case "/actualizar":  
34                 actualizarContacto(request, response);  
35                 break;  
36             default:  
37                 listarContactos(request, response);  
38                 break;  
39         }  
40     }  
41  
42 @protected void doGet(HttpServletRequest request, HttpServletResponse response) no usages  
43     throws ServletException, IOException {  
44     String action = request.getServletPath();  
45     if (action == null) action = "/";  
46  
47     try {  
48         switch (action) {  
49             case "/nuevo":  
50                 mostrarFormularioNuevo(request, response);  
51                 break;  
52             case "/editar":  
53                 mostrarFormularioEditar(request, response);  
54                 break;  
55             case "/eliminar":  
56                 eliminarContacto(request, response);  
57                 break;  
58             default:  
59                 listarContactos(request, response);  
60                 break;  
61         }  
62     } catch (SQLException ex) {  
63         throw new ServletException(ex);  
64     }  
65 }  
  
m pom.xml (agenda)  web.xml  ConexionDB.java  Contacto.java  ContactoDAO.java  ContactoServlet.java  lista-contactos.jsp  formulario-c  :  
17     public class ContactoServlet extends HttpServlet {  
67     @private void listarContactos(HttpServletRequest request, HttpServletResponse response) 2 usages  
68         throws ServletException, IOException {  
69         List<Contacto> listaContactos = contactoDAO.seleccionarTodos();  
70         request.setAttribute("listaContactos", listaContactos);  
71         RequestDispatcher dispatcher = request.getRequestDispatcher("lista-contactos.jsp");  
72         dispatcher.forward(request, response);  
73     }  
74  
75     @private void mostrarFormularioNuevo(HttpServletRequest request, HttpServletResponse response) 1 usage  
76         throws ServletException, IOException {  
77         RequestDispatcher dispatcher = request.getRequestDispatcher("formulario-contacto.jsp");  
78         dispatcher.forward(request, response);  
79     }  
80  
81     @private void mostrarFormularioEditar(HttpServletRequest request, HttpServletResponse response) 1 usage  
82         throws ServletException, IOException {  
83         int id = Integer.parseInt(request.getParameter("id"));  
84         Contacto contactoExistente = contactoDAO.seleccionarPorId(id);  
85         RequestDispatcher dispatcher = request.getRequestDispatcher("formulario-contacto.jsp");  
86         request.setAttribute("contacto", contactoExistente);  
87         dispatcher.forward(request, response);  
88     }  
89  
90     @private void insertarContacto(HttpServletRequest request, HttpServletResponse response) 1 usage  
91         throws IOException {  
92         String nombre = request.getParameter("nombre");  
93         String apellidoPaterno = request.getParameter("apellidoPaterno");  
94         String apellidoMaterno = request.getParameter("apellidoMaterno");  
95         String sexo = request.getParameter("sexo");  
96         String telefono = request.getParameter("telefono");  
97         String direccion = request.getParameter("direccion");  
98         String tipoContacto = request.getParameter("tipoContacto");  
99  
100        Contacto nuevoContacto = new Contacto(nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto);
```

```

m pom.xml (agenda)  web.xml  C ConexionDB.java  C Contacto.java  C ContactoDAO.java  C ContactoServlet.java  lista-contactos.jsp  formulario-c  :
17     public class ContactoServlet extends HttpServlet {
90         private void insertarContacto(HttpServletRequest request, HttpServletResponse response) 1 usage
96             String telefono = request.getParameter(s: "telefono");
97             String direccion = request.getParameter(s: "direccion");
98             String tipoContacto = request.getParameter(s: "tipoContacto");
99
100            Contacto nuevoContacto = new Contacto(nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto);
101            contactoDAO.insertarContacto(nuevoContacto);
102            response.sendRedirect(location: "list");
103        }
104
105    @
106        private void actualizarContacto(HttpServletRequest request, HttpServletResponse response) 1 usage
107            throws IOException {
108                int id = Integer.parseInt(request.getParameter(s: "id"));
109                String nombre = request.getParameter(s: "nombre");
110                String apellidoPaterno = request.getParameter(s: "apellidoPaterno");
111                String apellidoMaterno = request.getParameter(s: "apellidoMaterno");
112                String sexo = request.getParameter(s: "sexo");
113                String telefono = request.getParameter(s: "telefono");
114                String direccion = request.getParameter(s: "direccion");
115                String tipoContacto = request.getParameter(s: "tipoContacto");
116
117                Contacto contacto = new Contacto(id, nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto);
118                contactoDAO.actualizarContacto(contacto);
119                response.sendRedirect(location: "list");
120            }
121
122    @
123        private void eliminarContacto(HttpServletRequest request, HttpServletResponse response) 1 usage
124            throws IOException, SQLException {
125                int id = Integer.parseInt(request.getParameter(s: "id"));
126                contactoDAO.eliminarContacto(id);
127                response.sendRedirect(location: "list");
}

```

### Así luce nuestra lista-contactos.jsp

```

m pom.xml (agenda)  web.xml  C ConexionDB.java  C Contacto.java  C ContactoDAO.java  C ContactoServlet.java  lista-contactos.jsp  formulario-c  :
1    <%@ page import="java.util.List" %>
2    <%@ page import="dev.jeremyrossell.agenda.model.Contacto" %>
3    <%@ page contentType="text/html;charset=UTF-8" language="java" %>
4
5    <html>
6        <head>
7            <title>Agenda de Contactos</title>
8        </head>
9        <body>
10            <div align="center">
11                <h1>Gestión de Contactos</h1>
12                <h2><a href="nuevo">Agregar Nuevo Contacto</a></h2>
13
14                <table border="1" cellpadding="5">
15                    <tr>
16                        <th>ID</th>
17                        <th>Nombre</th>
18                        <th>A. Paterno</th>
19                        <th>A. Materno</th>
20                        <th>Sexo</th>
21                        <th>Teléfono</th>
22                        <th>Dirección</th>
23                        <th>Tipo</th>
24                        <th>Acciones</th>
25                    </tr>
26
27                    <%
28                        List<Contacto> lista = (List<Contacto>) request.getAttribute("listaContactos");
29                        if (lista != null) {
30                            for (Contacto c : lista) {
31
32                                <td><%= c.getId() %></td>
33                                <td><%= c.getNombre() %></td>
34                                <td><%= c.getApellidoPaterno() %></td>
35                                <td><%= c.getApellidoMaterno() %></td>
36                                <td><%= c.getSexo() %></td>
37
38                            </tr>
39
40                        }
41
42                    </table>
43
44                <hr>
45                <form action="nuevo" method="post">
46                    <table border="1" style="width: 100%; border-collapse: collapse;">
47                        <tr>
48                            <td style="width: 15%;">Nombre:Apellido Paterno:Apellido Materno:Sexo:Teléfono:Dirección:Tipo:

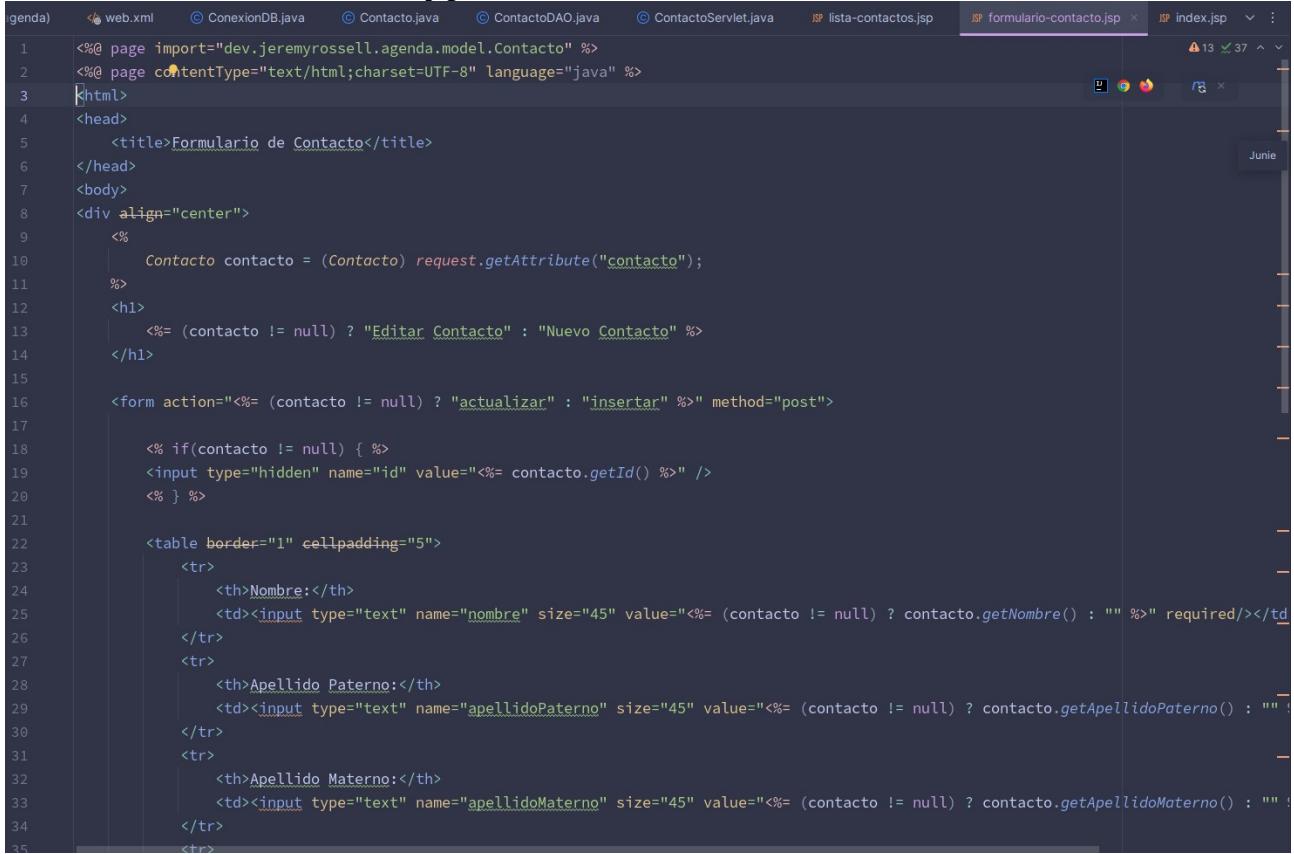
```

```

1   <%@ page import="java.util.List" %>
4   <html>
8   <body>
9   <div align="center">
13  <table border="1" cellpadding="5">
27  |     if (lista != null) {
28  |         for (Contacto c : lista) {
29  |             <tr>
30  |                 <td><%= c.getId() %></td>
31  |                 <td><%= c.getNombre() %></td>
32  |                 <td><%= c.getApellidoPaterno() %></td>
33  |                 <td><%= c.getApellidoMaterno() %></td>
34  |                 <td><%= c.getSexo() %></td>
35  |                 <td><%= c.getTelefono() %></td>
36  |                 <td><%= c.getDireccion() %></td>
37  |                 <td><%= c.getTipoContacto() %></td>
38  |                 <td>
39  |                     <a href="editar?id=<%= c.getId() %>">Editar</a>
40  |                     &ampnbsp&ampnbsp&ampnbsp
41  |                     <a href="eliminar?id=<%= c.getId() %>">Eliminar</a>
42  |                 </td>
43  |             </tr>
44  |             <%
45  |             }
46  |         }
47  |     }
48  |     <%
49  |     </table>
50  | </div>
51  | </body>
52  | </html>

```

## Y así nuestro formulario-contacto.jsp



```

1  <%@ page import="dev.jeremyrossell.agenda.model.Contacto" %>
2  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
3  <html>
4      <head>
5          <title>Formulario de Contacto</title>
6      </head>
7      <body>
8          <div align="center">
9              <%
10             Contacto contacto = (Contacto) request.getAttribute("contacto");
11             %>
12             <h1>
13             <%= (contacto != null) ? "Editar Contacto" : "Nuevo Contacto" %>
14             </h1>
15
16             <form action="<%=(contacto != null) ? "actualizar" : "insertar"%>" method="post">
17
18                 <% if(contacto != null) { %>
19                 <input type="hidden" name="id" value="<%=(contacto.getId())%>" />
20                 <% } %>
21
22                 <table border="1" cellpadding="5">
23                     <tr>
24                         <th>Nombre:</th>
25                         <td><input type="text" name="nombre" size="45" value="<%=(contacto != null) ? contacto.getNombre() : ""%>" required/></td>
26                     </tr>
27                     <tr>
28                         <th>Apellido Paterno:</th>
29                         <td><input type="text" name="apellidoPaterno" size="45" value="<%=(contacto != null) ? contacto.getApellidoPaterno() : ""%>" /></td>
30                     </tr>
31                     <tr>
32                         <th>Apellido Materno:</th>
33                         <td><input type="text" name="apellidoMaterno" size="45" value="<%=(contacto != null) ? contacto.getApellidoMaterno() : ""%>" /></td>
34                     </tr>
35                 </table>

```

```

1 <%@ page import="dev.jeremyrossell.agenda.model.Contacto" %>
3 <html>
5 <body>
7 <div align="center">
9 <form action="<%=(contacto != null) ? "actualizar" : "insertar"%>" method="post">
11 <tr>
12 <td><select name="sexo">
13 <option value="0" <%= (contacto != null && "0".equals(contacto.getSexo())) ? "selected" : "" %>>Otro</option>
14 </select>
15 </td>
16 </tr>
17 <tr>
18 <th>Teléfono:</th>
19 <td><input type="text" name="telefono" size="15" value="<%=(contacto != null) ? contacto.getTelefono() : ""%>" required/>
20 </td>
21 </tr>
22 <tr>
23 <th>Dirección:</th>
24 <td><input type="text" name="direccion" size="45" value="<%=(contacto != null) ? contacto.getDireccion() : ""%>" required/>
25 </td>
26 </tr>
27 <tr>
28 <th>Tipo Contacto:</th>
29 <td>
30 <select name="tipoContacto">
31 <option value="CASA" <%= (contacto != null && "CASA".equals(contacto.getTipoContacto())) ? "selected" : "" %>>Casa</option>
32 <option value="TRABAJO" <%= (contacto != null && "TRABAJO".equals(contacto.getTipoContacto())) ? "selected" : "" %>>Trabajo</option>
33 </select>
34 </td>
35 </tr>
36 <tr>
37 <td colspan="2" align="center">
38 <input type="submit" value="Guardar" />
39 </td>
40 </tr>
41 </table>
42 </form>
43 <h3><a href="list">Volver a la Lista</a></h3>

```

Y para terminar solo agregamos un display name y welcome file a nuestro web.xml para inicializarlo, y la verdad estoy orgulloso, quedó muy limpio el archivo, así las futuras modificaciones serán fáciles.

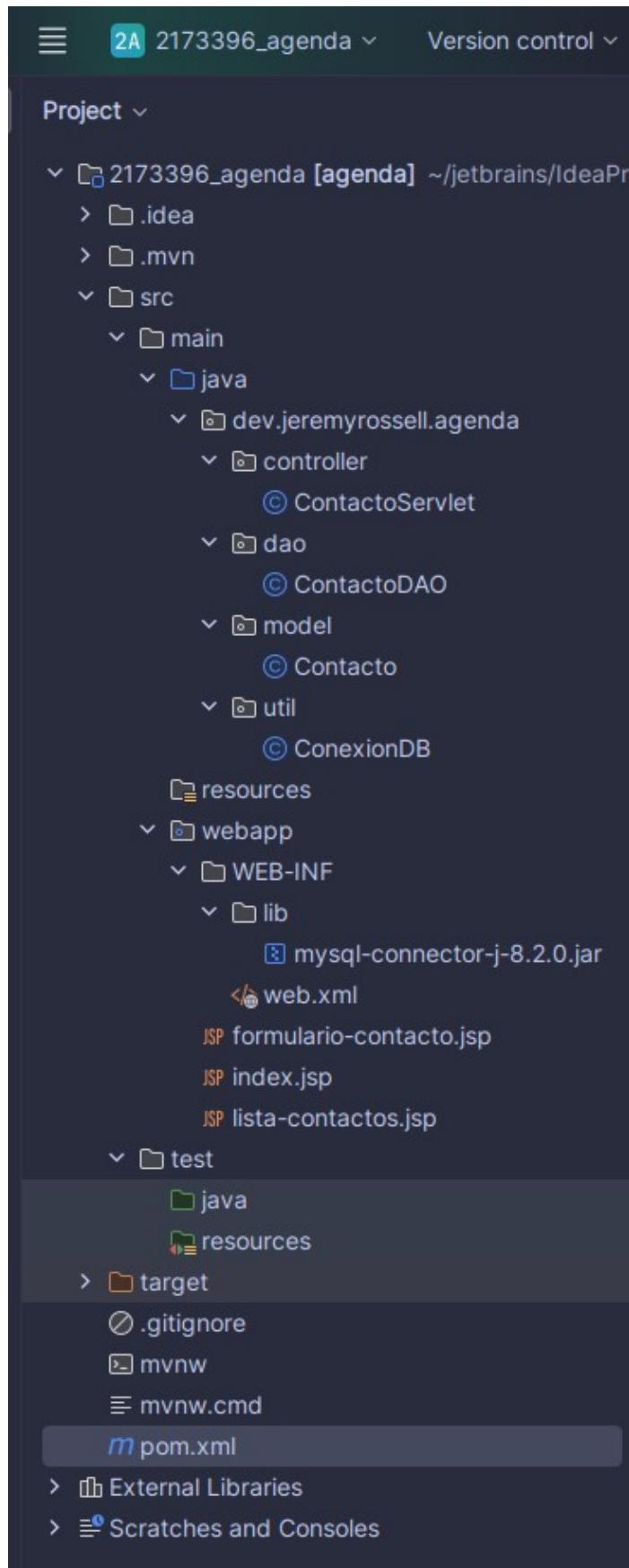
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
5   version="6.0">
6
7   <display-name>Agenda Jakarta</display-name>
8   <welcome-file-list>
9     <welcome-file>lista-contactos.jsp</welcome-file>
10    <welcome-file-list>
11  </web-app>

```

## Archivos.

La estructura de nuestros archivos es la estándar, la que se usa en las convenciones de Tomcat.



# Compilación y prueba.

Ahora viene la prueba de fuego, con esto aseguramos que no solo compila y está depurado, sino que funciona bien como lo quisimos. Arrancamos Tomcat y no nos da errores, todo bien.



The screenshot shows the Eclipse IDE's Services view. A Tomcat Server instance named 'Tomcat 10.1.49 [local]' is selected. The 'agenda:war exploded' folder is expanded, showing several files and folders. To the right, the 'Tomcat Catalina Log' tab is active, displaying deployment logs. The log output includes messages like 'INFO [main] org.apache.catalina.core.StandardEngine.startInternal Starting Servlet engine: [Apache Tomcat/]' and 'INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]'. Deployment details such as artifact exploded paths and deployment times are also present. The status bar at the bottom indicates the file path '2173396\_agenda > src > main > webapp > WEB-INF > web.xml' and the time '10:11 LF UTF-8 4 spaces'.

Luego nos abre nuestra página localhost:8080, y el formulario se ve como esperado



Rellenamos nuestros datos y le damos a Submit.

## Nuevo Contacto

<b>Nombre:</b>	Jorge Alberto
<b>Apellido Paterno:</b>	Isla
<b>Apellido Materno:</b>	Pineda
<b>Sexo:</b>	Masculino
<b>Teléfono:</b>	987654321
<b>Dirección:</b>	Calle #123, Colonia, 112233, Municipio, Estado, País
<b>Tipo Contacto:</b>	Trabajo
<input type="button" value="Guardar"/>	

[Volver a la Lista](#)

Y como era de esperarse, funcionó todo con éxito.

## Gestión de Contactos

## Agregar Nuevo Contacto

ID	Nombre	A. Paterno	A. Materno	Sexo	Teléfono	Dirección	Tipo	Acciones
1	Juan	Perez	Lopez	M	555-1234	Calle Falsa 123	TRABAJO	<a href="#">Editar</a> <a href="#">Eliminar</a>
2	Jorge Alberto	Isla	Pineda	M	987654321	Calle #123, Colonia, 112233, Municipio, Estado, País	TRABAJO	<a href="#">Editar</a> <a href="#">Eliminar</a>

Comprobamos que se ve reflejado a nuestra base de datos, y si.

Se pueden poner más contactos, editarlos, y eliminarlos.

## Gestión de Contactos

## Agregar Nuevo Contacto

ID	Nombre	A. Paterno	A. Materno	Sexo	Teléfono	Dirección	Tipo	Acciones
1	Juan	Perez	Lopez	M	555-1234	Calle Falsa 123	TRABAJO	<a href="#">Editar</a> <a href="#">Eliminar</a>
2	Jorge Alberto	Isla	Pineda	M	987654321	Calle #123, Colonia, 112233, Municipio, Estado, País	TRABAJO	<a href="#">Editar</a> <a href="#">Eliminar</a>
4	pongame	100	profe :D	M	1337-420	papulandia	CASA	<a href="#">Editar</a> <a href="#">Eliminar</a>
5	loquendera	master	666	F	1337-1337	danisoul #666, loquendo city	CASA	<a href="#">Editar</a> <a href="#">Eliminar</a>

Y ya dumpeamos el SQL para tenerlo como evidencia.

```
jeremy@meow:~/Desktop
```

The screenshot shows a terminal window with a dark blue background. At the top, the title bar displays the user name 'jeremy' and the path 'meow:~/Desktop'. The main area of the terminal shows a command being run: 'mysqldump -u root -p agenda\_db > sqldump.sql'. Below the command, the text 'Enter password:' is visible, indicating that the user is prompted for a MySQL password. The bottom status bar shows the command has completed successfully ('✓'), took 22 seconds ('22s'), and the current time is '01:56:59'.

```
mysqldump -u root -p agenda_db > sqldump.sql
```

Enter password:

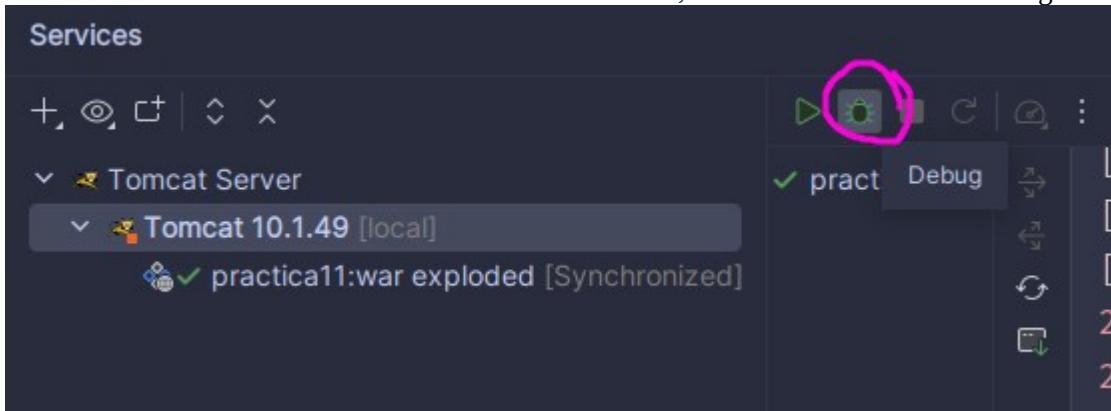
```
mysqldump -u root -p agenda_db > sqldump.sql
```

# Depuración.

Si queremos depurar o debuggear, con IntelliJ solo ponemos breakpoints donde queremos revisar el código.



Y en servicios en vez de arrancar normalmente el Tomcat, lo arrancamos en modo Debug.



Y ya así podremos ver qué hace nuestro programa pasito por pasito.

