

**Universidad Autónoma de Nuevo León.  
Facultad de Ciencias Físico Matemáticas.**

# **PIA – Agenda.**

Laboratorio de Programación Orientada a Objetos.

**Alumno:** Jeremy Uriel Rossell Segura.

**Matrícula:** 2173396.

**Grupo:** 036.

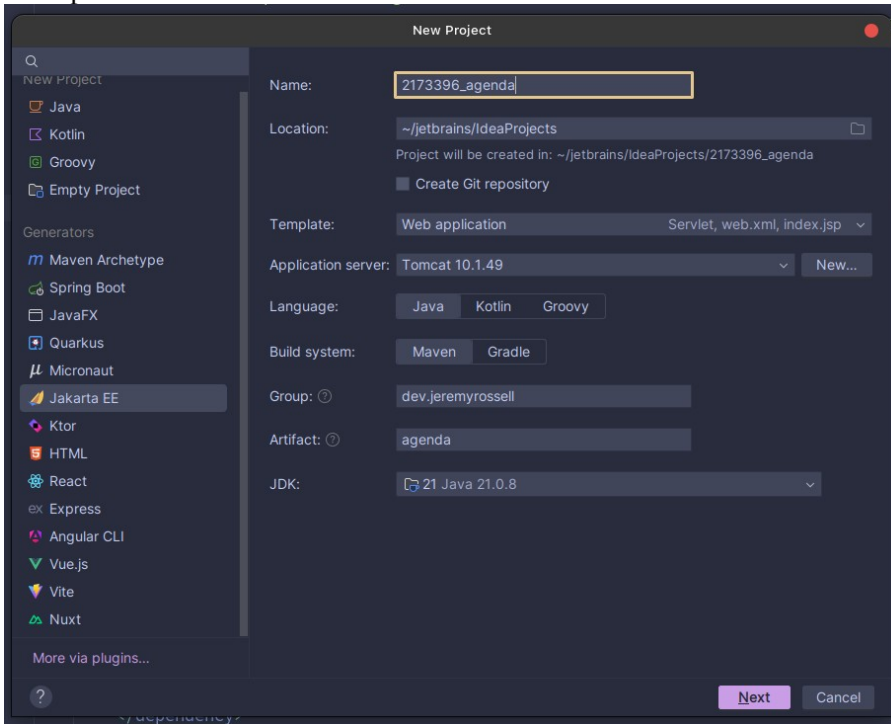
**Horario:** VIERNES 7:00-9:00.

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

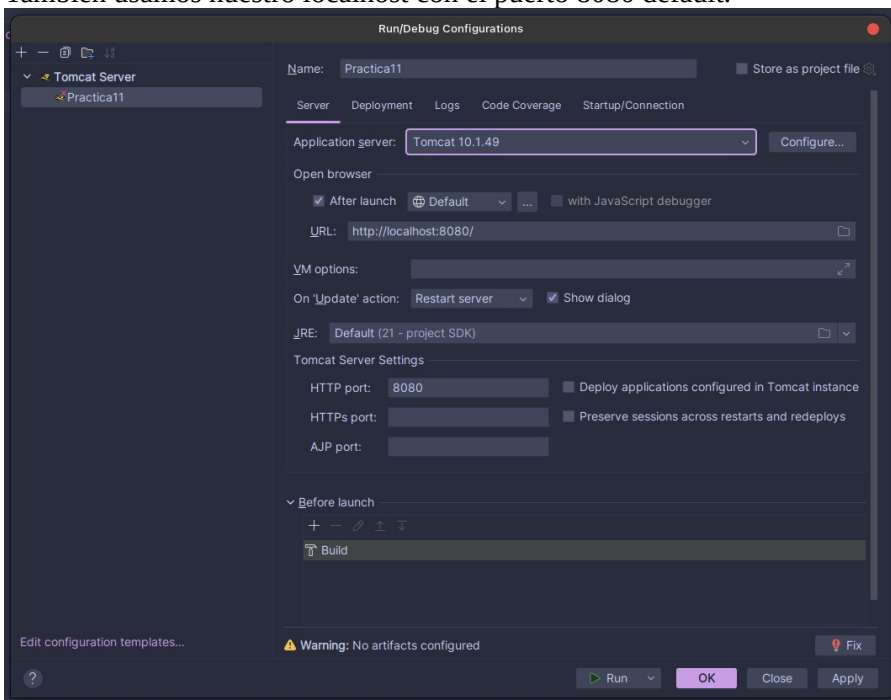
**Docente:** Jorge Alberto Islas Pineda.

# Inicialización.

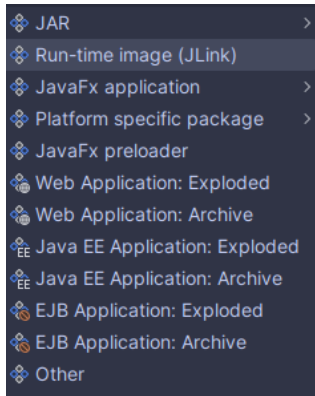
Primero creamos nuestro proyecto IntelliJ con Jakarta EE 11 para Web Application, y usamos Tomcat 10.1.49 como nuestro application server, y elegimos el local. Descargué el tar.gz de la web oficial de Tomcat, y lo guardé en mi directorio ~/jetbrains/Apache/apache-tomcat-10.1.49, como se aprecia, es directorio 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, usaríamos sudo apt install tomcat10, o si hay una versión nueva checar usando sudo apt update && apt search tomcat.



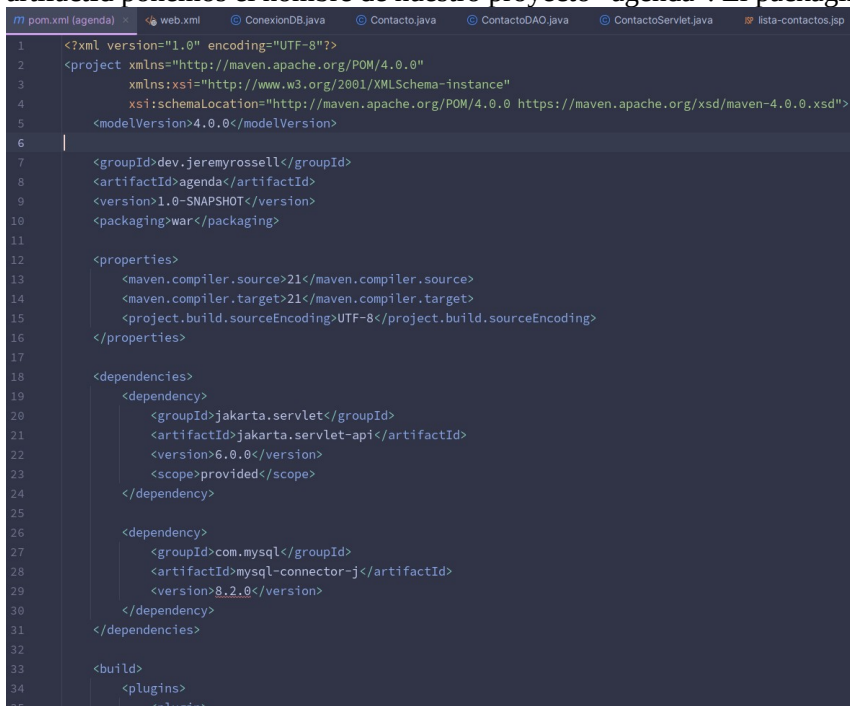
Usamos el Tomcat de forma local para no rompernos la cabeza con los puertos, ni configurar el servidor, ni permisos de root ni nada de eso. Simplemente buscamos la carpeta ya extraída en el directorio ya dado. También usamos nuestro localhost con el puerto 8080 default.



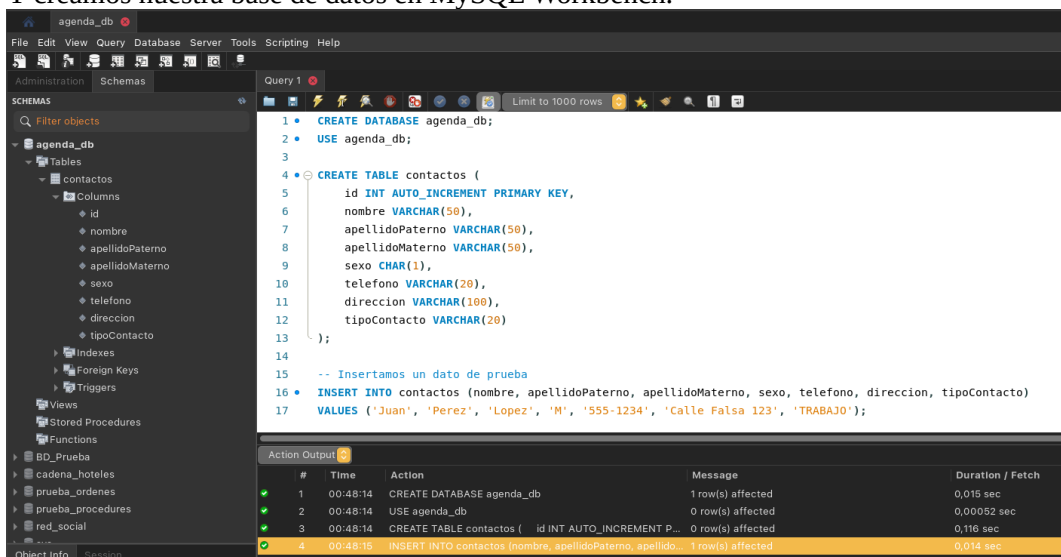
En artefactos usamos “Web Application: Exploded”



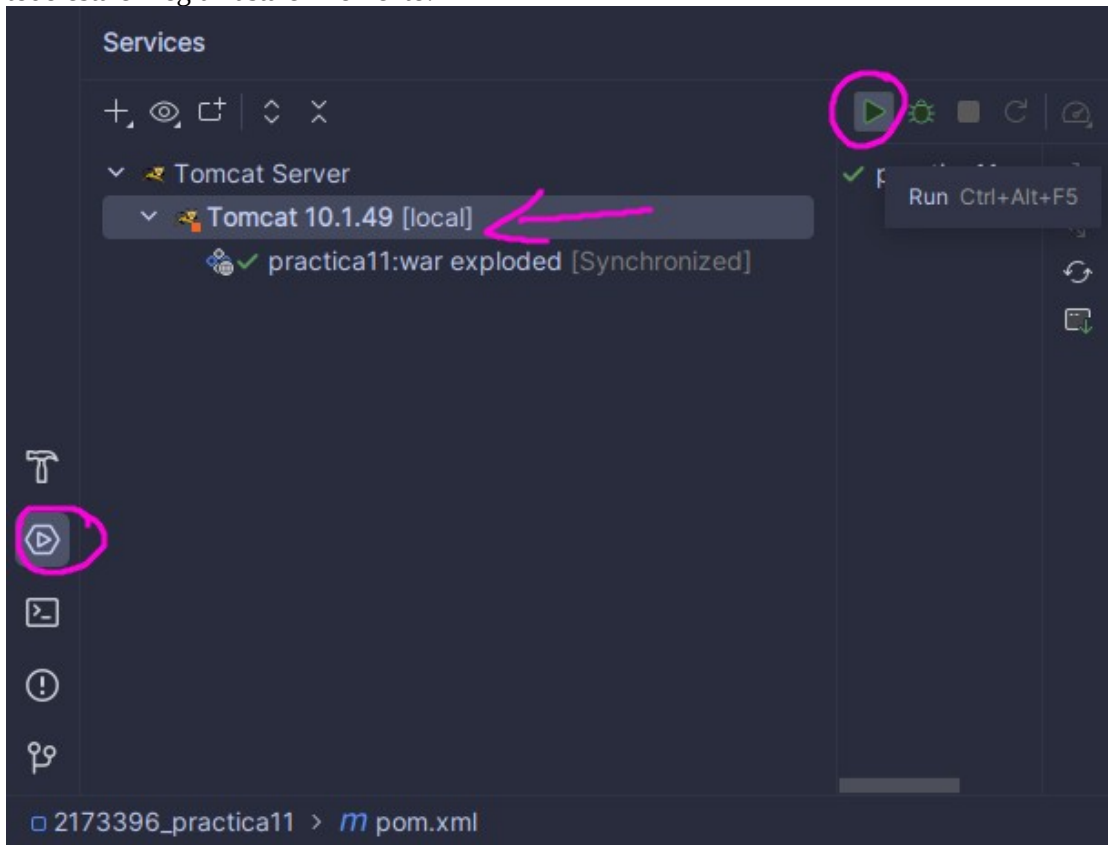
Ahora rellenamos los campos de groupId con un paquete, si estamos en producción puede ser el nombre del proyecto o empresa, como soy yo en un ámbito académico, usaré “dev.jeremyrossell” como el paquete. Y en artifactId ponemos el nombre de nuestro proyecto “agenda”. El packaging por default debe ser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



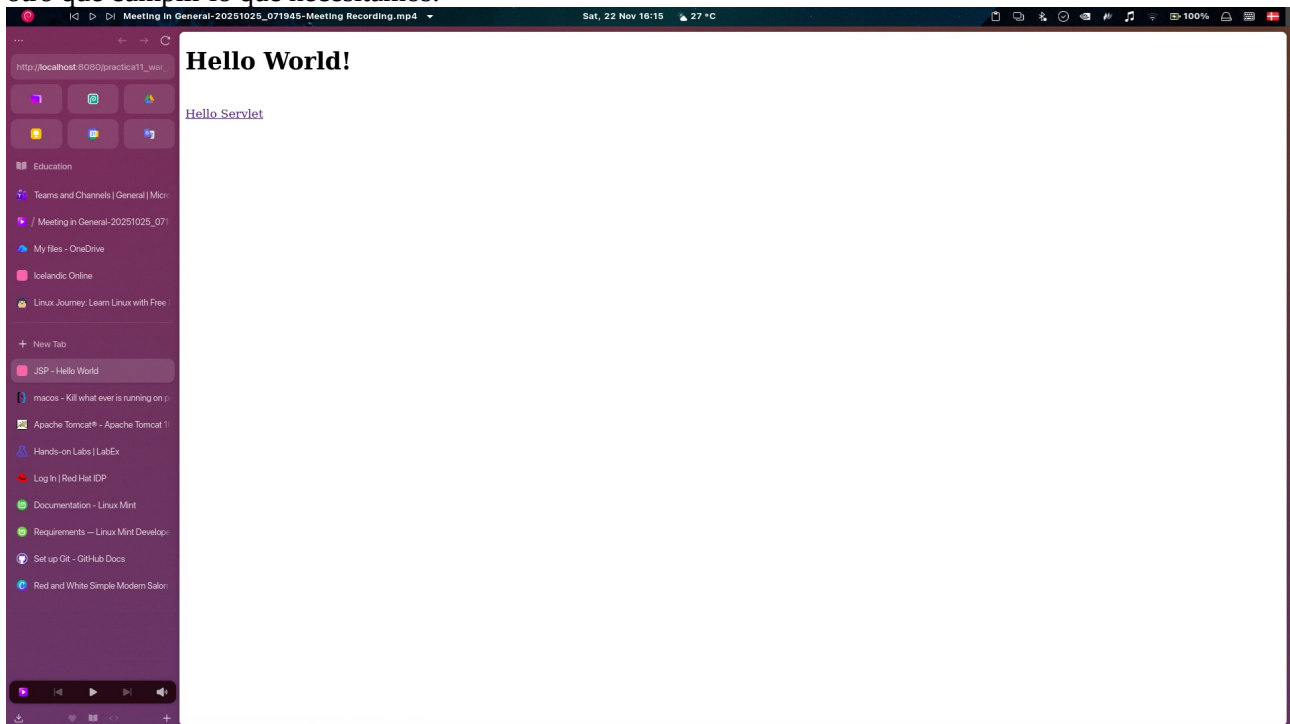
Y creamos nuestra base de datos en MySQL Workbench.



Una vez chequeemos 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.

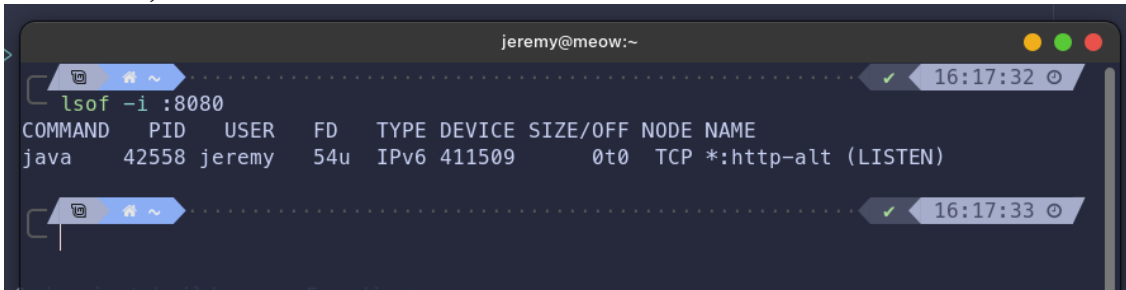


Y como podemos apreciar, si funcionó, este html es uno que se generó por defecto, pero nosotros crearemos otro que cumplir lo que necesitamos.



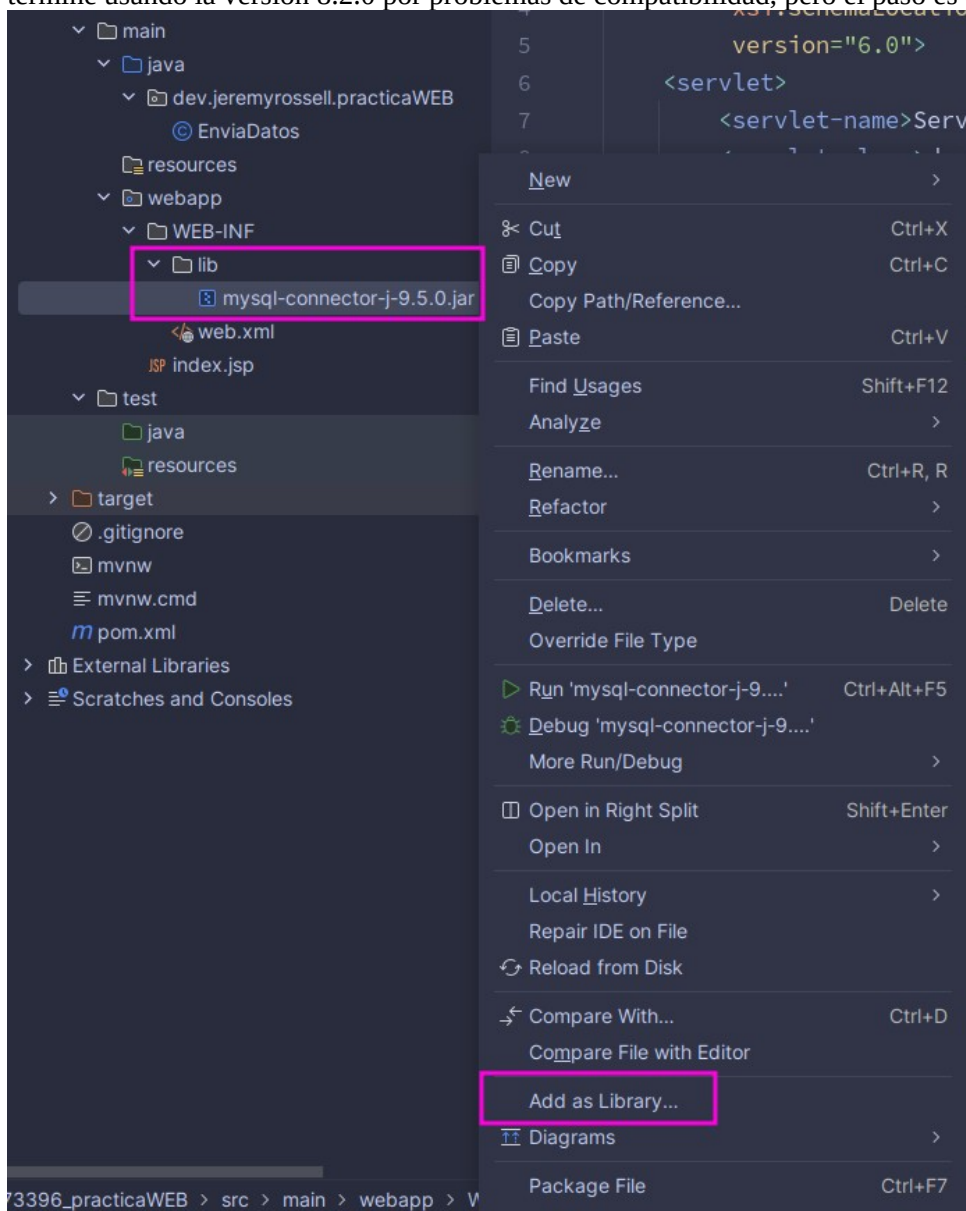
## Código.

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)
```

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.

```
m pom.xml (agenda)  web.xml  ConexionDB.java x  Contacto.java  ContactoDAO.java  ContactoServlet.java  JSP lista
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 { 6 usages
8     private static final String URL = "jdbc:mysql://localhost:3306/agenda_db?serverTimezone=UTC";
9     private static final String USER = "root"; 1 usage
10    private static final String PASS = "MySQL@2025!"; 1 usage
11
12    public static Connection getConnection() { 5 usages
13        Connection con = null;
14        try {
15            Class.forName("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.

```
m pom.xml (agenda)  web.xml  ConexionDB.java  Contacto.java x  ContactoDAO.java  ContactoServlet.java  JSP lista-contactos.jsp  JSP formulario-c  Maven
3 public class Contacto { 23 usages
4     private Integer id; 3 usages
5     private String nombre; 4 usages
6     private String apellidoPaterno; 4 usages
7     private String apellidoMaterno; 4 usages
8     private String sexo; // 'M','F','O' 4 usages
9     private String telefono; 4 usages
10    private String direccion; 4 usages
11    private String tipoContacto; // 'CASA','TRABAJO' 4 usages
12
13    // constructor vacío
14    public Contacto() {} 3 usages
15
16    // constructor con campos
17    public Contacto(Integer id, String nombre, String apellidoPaterno, String apellidoMaterno, String sexo, String telefono, String direccion, String tipoContacto) {
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, String tipoContacto) {
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.sexo = sexo;
34         this.telefono = telefono;
35         this.direccion = direccion;
36         this.tipoContacto = tipoContacto;
37     }
38
39     // getters y setters
40     public Integer getId() { return id; } 5 usages
41     public void setId(Integer id) { this.id = id; } no usages
42
43     public String getNombre() { return nombre; } 4 usages
44     public void setNombre(String nombre) { this.nombre = nombre; } no usages
45
46     public String getApellidoPaterno() { return apellidoPaterno; } 4 usages
47     public void setApellidoPaterno(String apellidoPaterno) { this.apellidoPaterno = apellidoPaterno; } no usages
48
49     public String getApellidoMaterno() { return apellidoMaterno; } 4 usages
50     public void setApellidoMaterno(String apellidoMaterno) { this.apellidoMaterno = apellidoMaterno; } no usages
51
52     public String getSexo() { return sexo; } 6 usages
53     public void setSexo(String sexo) { this.sexo = sexo; } no usages
54
55     public String getTelefono() { return telefono; } 4 usages
56     public void setTelefono(String telefono) { this.telefono = telefono; } no usages
57
58     public String getDireccion() { return direccion; } 4 usages
59     public void setDireccion(String direccion) { this.direccion = direccion; } no usages
60
61     public String getTipoContacto() { return tipoContacto; } 5 usages
62     public void setTipoContacto(String tipoContacto) { this.tipoContacto = tipoContacto; } no usages
63 }
```

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, d
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 =
17
18     // 1. insertar
19     @ public void insertarContacto(Contacto contacto) { 1 usage
20         try (Connection connection = ConexionDB.getConnection();
21             PreparedStatement ps = connection.prepareStatement(INSERT_SQL)) {
22             ps.setString(1, contacto.getNombre());
23             ps.setString(2, contacto.getApellidoPaterno());
24             ps.setString(3, contacto.getApellidoMaterno());
25             ps.setString(4, contacto.getSexo());
26             ps.setString(5, contacto.getTelefono());
27             ps.setString(6, contacto.getDireccion());
28             ps.setString(7, contacto.getTipoContacto());
29             ps.executeUpdate();
30         } catch (SQLException e) {
31             e.printStackTrace();
32         }
33     }
34
35     // 2. seleccionar todos
```

```
m pom.xml (agenda) web.xml ConexionDB.java Contacto.java ContactoDAO.java x 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();
100     }
```

```
m pom.xml (agenda) web.xml ConexionDB.java Contacto.java ContactoDAO.java x ContactoServlet.java lista-contactos.jsp formulario-c
10 public class ContactoDAO { 3 usages
85 public boolean actualizarContacto(Contacto contacto) { 1 usage
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();
100 }
101 return rowUpdated;
102 }
103
104 // 5. eliminar
105 public boolean eliminarContacto(int id) { 1 usage
106     boolean rowDeleted = false;
107     try (Connection connection = ConexionDB.getConnection();
108         PreparedStatement ps = connection.prepareStatement(DELETE_SQL)) {
109         ps.setInt(i: 1, id);
110         rowDeleted = ps.executeUpdate() > 0;
111     } catch (SQLException e) {
112         e.printStackTrace();
113     }
114     return rowDeleted;
115 }
116 }
```

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.

```
m pom.xml (agenda) web.xml ConexionDB.java Contacto.java ContactoDAO.java ContactoServlet.java x lista-contactos.jsp formulario-c
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;
```

```

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 @ throws ServletException, IOException {
43     String action = request.getServletPath();
44     if (action == null) action = "/";
45
46     try {
47         switch (action) {
48             case "/nuevo":
49                 mostrarFormularioNuevo(request, response);
50                 break;
51             case "/editar":
52                 mostrarFormularioEditar(request, response);
53                 break;
54             case "/eliminar":
55                 eliminarContacto(request, response);
56                 break;
57             default:
58                 listarContactos(request, response);
59                 break;
60         }
61     } catch (SQLException ex) {
62         throw new ServletException(ex);
63     }
64 }
65
66 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(s: "listaContactos", listaContactos);
71         RequestDispatcher dispatcher = request.getRequestDispatcher(s: "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(s: "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(s: "id"));
84         Contacto contactoExistente = contactoDAO.seleccionarPorId(id);
85         RequestDispatcher dispatcher = request.getRequestDispatcher(s: "formulario-contacto.jsp");
86         request.setAttribute(s: "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(s: "nombre");
93         String apellidoPaterno = request.getParameter(s: "apellidoPaterno");
94         String apellidoMaterno = request.getParameter(s: "apellidoMaterno");
95         String sexo = request.getParameter(s: "sexo");
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);

```

```

pom.xml (agenda) web.xml ConexionDB.java Contacto.java ContactoDAO.java 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 @ private void actualizarContacto(HttpServletRequest request, HttpServletResponse response) 1 usage
106     throws IOException {
107     int id = Integer.parseInt(request.getParameter(s: "id"));
108     String nombre = request.getParameter(s: "nombre");
109     String apellidoPaterno = request.getParameter(s: "apellidoPaterno");
110     String apellidoMaterno = request.getParameter(s: "apellidoMaterno");
111     String sexo = request.getParameter(s: "sexo");
112     String telefono = request.getParameter(s: "telefono");
113     String direccion = request.getParameter(s: "direccion");
114     String tipoContacto = request.getParameter(s: "tipoContacto");
115
116     Contacto contacto = new Contacto(id, nombre, apellidoPaterno, apellidoMaterno, sexo, telefono, direccion, tipoContacto);
117     contactoDAO.actualizarContacto(contacto);
118     response.sendRedirect(location: "list");
119 }
120
121 @ private void eliminarContacto(HttpServletRequest request, HttpServletResponse response) 1 usage
122     throws IOException, SQLException {
123     int id = Integer.parseInt(request.getParameter(s: "id"));
124     contactoDAO.eliminarContacto(id);
125     response.sendRedirect(location: "list");
126 }
127 }

```

## Así luce nuestra lista-contactos.jsp

```

pom.xml (agenda) web.xml ConexionDB.java Contacto.java ContactoDAO.java 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 <html>
5 <head>
6     <title>Agenda de Contactos</title>
7 </head>
8 <body>
9 <div align="center">
10     <h1>Gestión de Contactos</h1>
11     <h2><a href="nuevo">Agregar Nuevo Contacto</a></h2>
12
13     <table border="1" cellpadding="5">
14         <tr>
15             <th>ID</th>
16             <th>Nombre</th>
17             <th>A. Paterno</th>
18             <th>A. Materno</th>
19             <th>Sexo</th>
20             <th>Teléfono</th>
21             <th>Dirección</th>
22             <th>Tipo</th>
23             <th>Acciones</th>
24         </tr>
25         <%
26             List<Contacto> lista = (List<Contacto>) request.getAttribute("listaContactos");
27             if (lista != null) {
28                 for (Contacto c : lista) {
29                     %>
30                 <tr>
31                     <td><%= c.getId() %></td>
32                     <td><%= c.getNombre() %></td>
33                     <td><%= c.getApellidoPaterno() %></td>
34                     <td><%= c.getApellidoMaterno() %></td>
35                     <td><%= c.getSexo() %></td>

```

[illegible]

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() : "" %>" required/></td>
30                  </tr>
31                  <tr>
32                      <th>Apellido Materno:</th>
33                      <td><input type="text" name="apellidoMaterno" size="45" value="<%= (contacto != null) ? contacto.getApellidoMaterno() : "" %>" required/></td>
34                  </tr>

```

```

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

```

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.

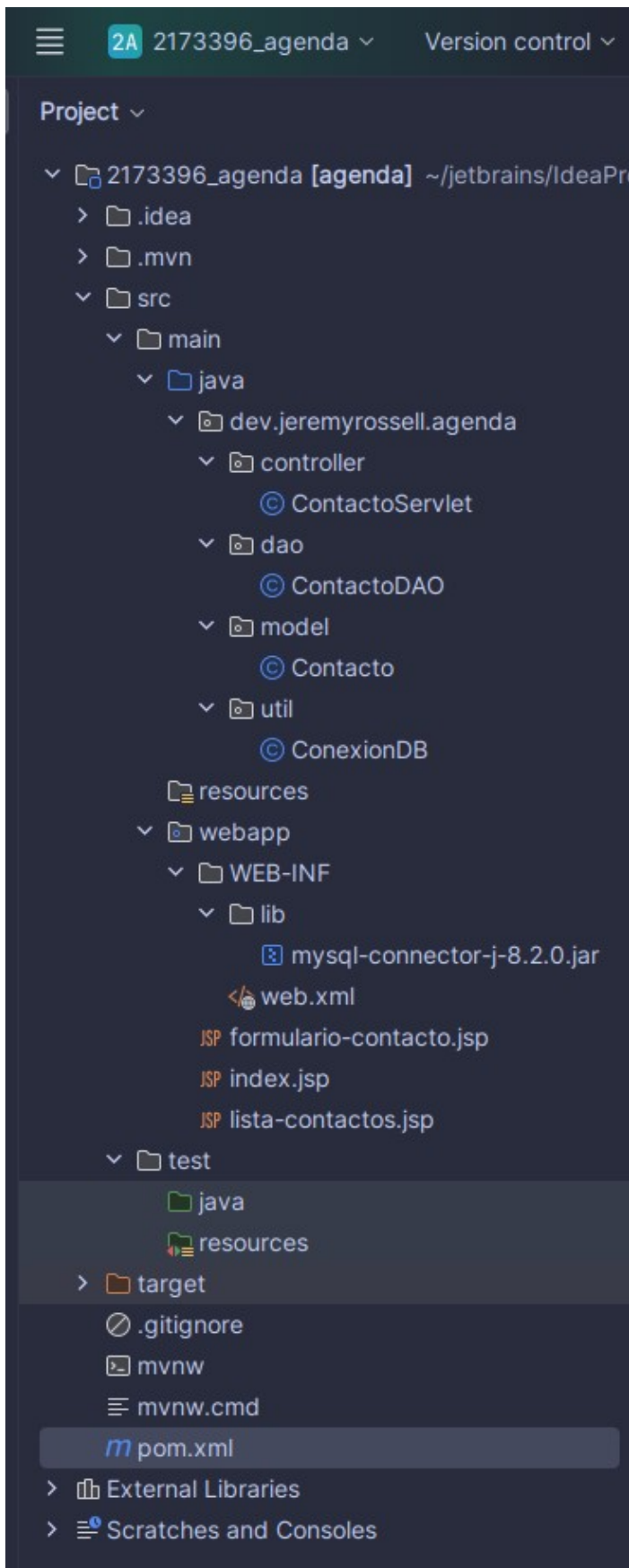
```

pom.xml (agenda) web.xml ConexionDB.java Contacto.java ContactoDAO.java ContactoServlet.java lista-contactos.jsp formulario-contacto.jsp index.jsp
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> <welcome-file-list>
8 <welcome-file>lista-contactos.jsp</welcome-file>
9 </welcome-file-list>
10 </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.



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>                         | <input type="text" value="Jorge Alberto"/>  |
| <b>Apellido Paterno:</b>               | <input type="text" value="Islas"/>  |
| <b>Apellido Materno:</b>               | <input type="text" value="Pineda"/>   |
| <b>Sexo:</b>                           | <input type="text" value="Masculino"/>  |
| <b>Teléfono:</b>                       | <input type="text" value="987654321"/>  |
| <b>Dirección:</b>                      | <input type="text" value="Calle #123, Colonia, 112233, Municipio, Estado, Pais"/> |
| <b>Tipo Contacto:</b>                  | <input type="text" value="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><br><a href="#">Eliminar</a> |
| 2  | Jorge Alberto | Islas      | Pineda     | M    | 987654321 | Calle #123, Colonia, 112233, Municipio, Estado, Pais | TRABAJO | <a href="#">Editar</a><br><a href="#">Eliminar</a> |

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

```
mysql> SHOW agenda_db;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'agenda_db' at line 1

mysql> USE agenda_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM contactos;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | nombre | apellidoPaterno | apellidoMaterno | sexo | telefono | direccion | tipoContacto |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Juan | Perez | Lopez | M | 555-1234 | Calle Falsa 123 | TRABAJO |
| 2 | Jorge Alberto | Islas | Pineda | M | 987654321 | Calle #123, Colonia, 112233, Municipio, Estado, Pais | TRABAJO |
+----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)
```

Se pueden poner más contactos, editarlos, y eliminarlos.

# Gestión de Contactos

## Agregar Nuevo Contacto

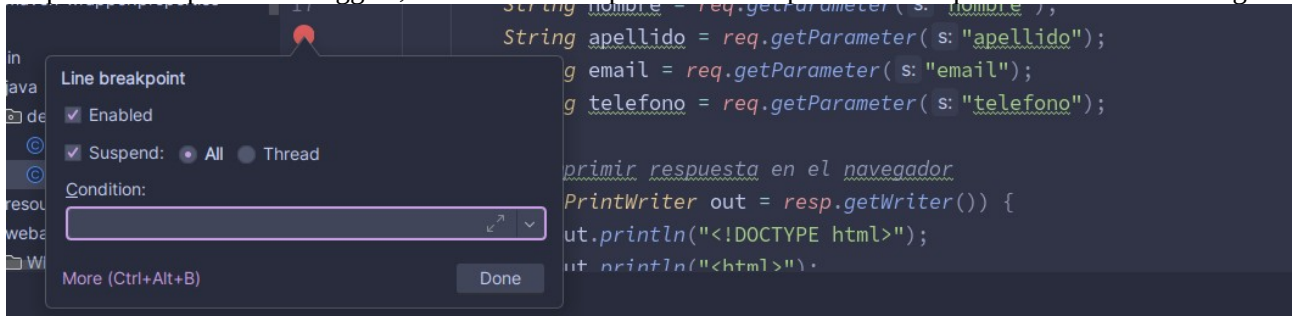
| ID | Nombre        | A. Paterno | A. Materno | Sexo | Teléfono  | Dirección  | Tipo    | Acciones   |
|----|---------------|------------|------------|------|-----------|--|---------|--|
| 1  | Juan          | Perez      | Lopez      | M    | 555-1234  | Calle Falsa 123                             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

Y ya dumpeamos el SQL para tenerlo como evidencia.

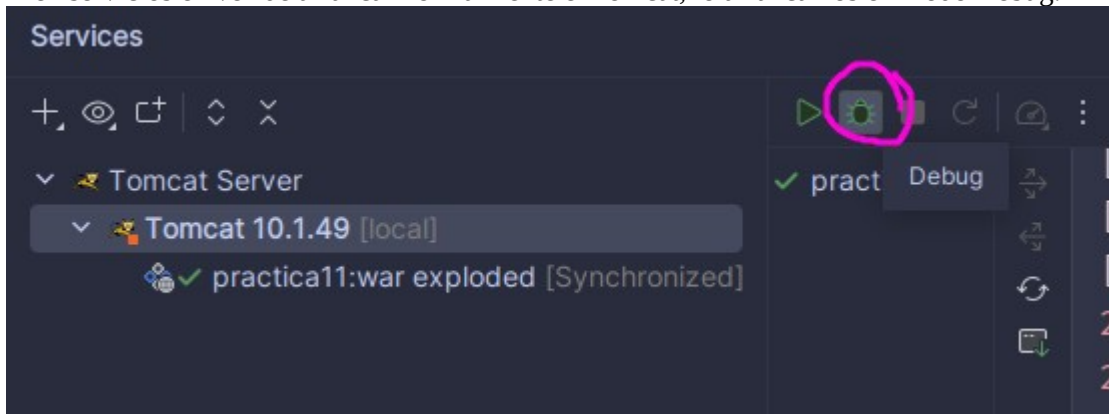
```
mysql> mysqldump -u root -p agenda_db > sqliump.sql
Enter password:
```

## 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.

