Universidad Autónoma de Nuevo León

Facultad de Ciencias

Físico Matemáticas

Laboratorio de POO

Documentación de sistema: Agenda Web con Java Servlet MVC

Nombres:

Jorge Castillo Acosta 2132842

Lugar: Monterrey, NL. Fecha 21/11/2025

Contenido

1.- INTRODUCCIÓN

El presente proyecto consiste en el desarrollo de una **aplicación web de Agenda de Contactos**, implementada con tecnología Java utilizando **Servlets**, el patrón de diseño **Modelo–Vista–Controlador (MVC)** y una base de datos **MySQL** para el almacenamiento persistente de la información. La aplicación permite gestionar de manera sencilla y organizada un catálogo de contactos mediante operaciones CRUD (Crear, Leer, Actualizar y Eliminar), accesibles desde cualquier navegador web.

Para lograr su funcionamiento, el sistema se construyó siguiendo una arquitectura por capas. El **Modelo** contiene la clase Contacto, encargada de representar cada registro de la agenda. La capa **DAO (Data Access Object)** gestiona todas las operaciones SQL hacia la base de datos mediante JDBC, a través de una clase de conexión dedicada (DBConnection). La capa **Controlador** está compuesta por un Servlet principal (ContactoServlet), encargado de procesar las solicitudes HTTP, controlar el flujo de navegación y coordinar las acciones entre la interfaz y la base de datos. Finalmente, la capa **Vista** está formada por páginas JSP (lista.jsp y form.jsp), responsables de mostrar al usuario la información de forma dinámica.

El proyecto se desplegó en el servidor de aplicaciones **Apache Tomcat 9**, lo que permitió ejecutar la lógica Java y atender las peticiones desde el navegador. Asimismo, se diseñó la estructura de la base de datos y se generó un *dump* para facilitar la importación del esquema. La correcta integración entre todas las tecnologías permitió construir una aplicación web funcional, intuitiva y modular, adecuada para comprender el flujo completo del desarrollo web con Java y MVC.

2. OBJETIVOS

2.1 Objetivo General

Desarrollar una aplicación web con Java Servlet MVC capaz de gestionar contactos almacenados en una base de datos MySQL.

2.2 Objetivos Específicos

- Implementar el patrón MVC en una aplicación web.
- Utilizar Servlets como controladores para manejar peticiones HTTP.

- Crear vistas dinámicas utilizando JSP.
- Implementar conexión a MySQL mediante JDBC.
- Desarrollar un CRUD completo (Crear, Leer, Actualizar, Eliminar).
- Empaquetar la aplicación y desplegarla en Apache Tomcat.

PLANTEAMIENTO DEL PROBLEMA

En la actualidad, la gestión de información personal o profesional, como los datos de contactos, continúa siendo una necesidad fundamental tanto en entornos laborales como académicos. Sin embargo, muchas personas siguen utilizando métodos tradicionales como libretas físicas, notas dispersas o archivos de texto para almacenar nombres, correos electrónicos y teléfonos. Estos métodos son propensos a **errores**, **pérdida de información**, duplicidad de registros, falta de organización y dificultades para realizar búsquedas rápidas o actualizaciones.

Por otro lado, aunque existen aplicaciones de agenda avanzadas, muchas de ellas requieren instalación, conexión permanente a servicios externos o interfaces complejas que resultan poco accesibles para usuarios principiantes o para entornos educativos donde se busca comprender la estructura interna del software. Surge entonces la necesidad de contar con una **aplicación web ligera, funcional y fácil de usar**, que permita gestionar contactos de manera eficiente sin depender de herramientas externas o plataformas cerradas.

Dado este contexto, se plantea el problema de desarrollar un sistema que permita administrar una agenda digital mediante un mecanismo claro y estructurado, utilizando tecnologías abiertas y ampliamente utilizadas en el desarrollo profesional: **Java**, **Servlets**, **JSP**, **MVC**, **MySQL** y **Apache Tomcat**. La falta de integración entre estas tecnologías y un enfoque adecuado de arquitectura podría ocasionar aplicaciones difíciles de mantener, extensiones limitadas o errores en la manipulación de datos.

Por ello, el proyecto debe resolver cómo construir una herramienta que:

- Organice y almacene la información de contactos de forma segura y persistente.
- Permita realizar operaciones básicas de gestión (crear, consultar, modificar y eliminar) sin complicaciones.

- Aplique el patrón MVC para separar la lógica, la presentación y el acceso a datos, facilitando la escalabilidad y el mantenimiento.
- Se ejecute de manera eficiente en un servidor web real como Apache Tomcat.
- Conecte correctamente con una base de datos MySQL mediante JDBC.

En síntesis, el problema a resolver es la creación de una **agenda web funcional y estructurada**, que permita gestionar contactos de forma ordenada, confiable y accesible desde cualquier navegador, aprovechando las herramientas y buenas prácticas del desarrollo web en Java.

ALCANCE DEL PROYECTO

El presente proyecto abarca el desarrollo completo de una aplicación web básica para la gestión de contactos, implementada utilizando Java, Servlets, el patrón de diseño Modelo–Vista–Controlador (MVC) y una base de datos MySQL. El alcance del sistema se limita a las funcionalidades esenciales que permiten demostrar el funcionamiento integral de un CRUD dentro de una arquitectura web estructurada.

Funcionalidades Incluidas dentro del Alcance

El sistema desarrollado contempla las siguientes capacidades:

1. Gestión de Contactos

- Registrar nuevos contactos mediante un formulario web.
- Mostrar una lista dinámica de todos los contactos almacenados en la base de datos.
- Editar la información de un contacto existente.
- Eliminar contactos de manera permanente.
- Validación básica de campos requeridos (por ejemplo, nombre obligatorio).

2. Arquitectura Web con Java

- Implementación completa del patrón MVC:
    - **Modelo:** Clase Contacto que representa la entidad principal.
    - **DAO:** Manejo de la lógica de acceso a datos mediante SQL y JDBC.

    - **Controlador:** ContactoServlet que gestiona peticiones GET y POST.
    - **Vistas:** Páginas JSP para mostrar formularios y listas.
- Separación adecuada de responsabilidades entre cada capa.

3. Conexión a Base de Datos MySQL

- Creación del esquema agenda y su tabla contactos.
- Uso de JDBC para realizar consultas, inserciones, actualizaciones y eliminaciones.
- Manejo correcto de la conexión mediante la clase DBConnection.
- Integración del driver JDBC dentro del entorno web y Tomcat.

4. Despliegue en Servidor Apache Tomcat

- Ejecución de la aplicación como proyecto web en Tomcat 9.
- Configuración de rutas, mapeos de servlets y estructura de proyecto web.
- Empaquetamiento de librerías necesarias dentro de WEB-INF/lib.

5. Diseño Visual Básico

- Interfaz sencilla y funcional en JSP.
- Formularios claros para registrar o editar contactos.
- Tabla estilizada para la visualización de la lista de contactos.

Funcionalidades Fuera del Alcance

Para mantener el proyecto dentro de los requisitos del curso, se dejaron fuera las siguientes características:

- Autenticación o roles de usuario (inicio de sesión, permisos, etc.).
- Interfaz gráfica avanzada o diseño responsivo profesional.
- API REST o servicios externos.
- Exportación/importación de contactos desde archivo.
- Seguridad avanzada con filtros, tokens o cifrado.

- Módulos adicionales (agenda de eventos, recordatorios, categorías, etc.).

Resumen del Alcance

El proyecto cumple con el objetivo principal de construir una aplicación web funcional utilizando Servlets, MVC y conexión a MySQL, demostrando un flujo completo de desarrollo web: desde el diseño del modelo hasta la interacción con la base de datos y el despliegue en un servidor real.

5. ANÁLISIS DEL SISTEMA

El análisis del sistema permite identificar los elementos clave que intervienen en el funcionamiento de la aplicación, así como los requerimientos necesarios para garantizar que el desarrollo cumpla con los objetivos planteados. En esta sección se describen los actores involucrados y los requerimientos funcionales y no funcionales que delimitan el comportamiento y las características que el sistema debe presentar.

5.1 Actores del Sistema

Usuario del Sistema (Administrador Único)

Es la persona que interactúa con la aplicación a través del navegador web. En este proyecto no existe un sistema de usuarios múltiples ni autenticación; por lo tanto, el único actor tiene acceso completo a todas las funciones:

- Registrar nuevos contactos
- Consultar la lista completa
- Modificar información existente
- Eliminar contactos

Este actor representa al administrador general de la agenda.

5.2 Requerimientos Funcionales

Los requerimientos funcionales describen las acciones específicas que la aplicación debe ser capaz de realizar para cumplir con su propósito:

RF1. Registrar un nuevo contacto

El sistema debe permitir capturar nombre, correo electrónico y teléfono, y almacenar el registro en la base de datos.

RF2. Consultar la lista de contactos

El sistema debe mostrar en pantalla todos los contactos almacenados, ordenados de manera organizada.

RF3. Editar un contacto existente

El usuario debe poder seleccionar un contacto, modificar sus datos y actualizar la información en la base de datos.

RF4. Eliminar un contacto

El sistema debe permitir borrar un contacto de forma permanente a través de la interfaz.

RF5. Validar datos antes de guardarlos

El sistema debe verificar que los campos esenciales, como el nombre, estén completos antes de permitir guardar o actualizar registros.

5.3 Requerimientos No Funcionales

Los requerimientos no funcionales especifican las características de calidad, rendimiento y restricciones técnicas del sistema:

RNF1. Ejecución en servidor Apache Tomcat

La aplicación debe ser desplegada en un servidor Apache Tomcat 9 o compatible, permitiendo su ejecución mediante Servlets.

RNF2. Base de datos MySQL

Toda la información debe almacenarse en una base de datos MySQL utilizando JDBC para su acceso.

RNF3. Interfaz sencilla y comprensible

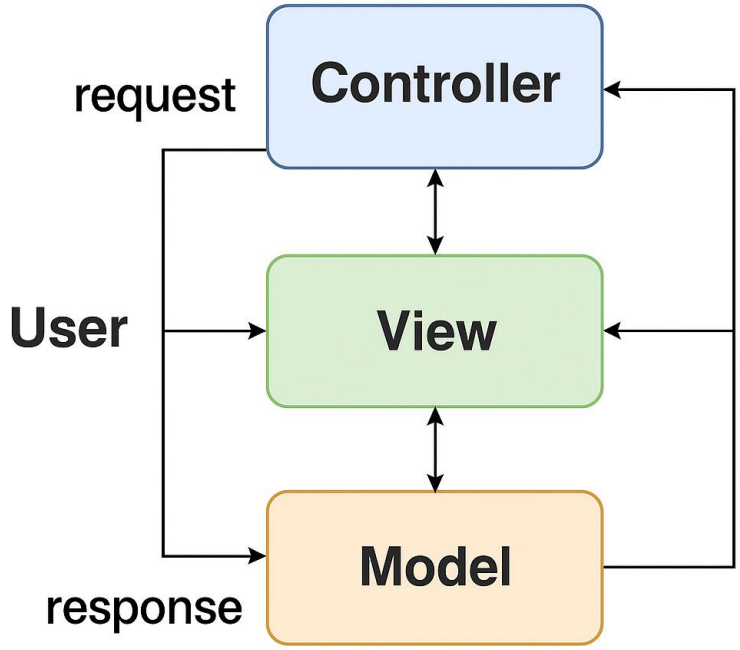
Las páginas JSP deben presentar una interfaz clara, funcional y fácil de usar, incluso para usuarios sin experiencia técnica.

RNF4. Arquitectura MVC obligatoria

El código debe aplicar el patrón Modelo–Vista–Controlador, manteniendo separadas las responsabilidades de lógica de negocio, presentación y control.

6. DISEÑO DEL SISTEMA

6.1 Arquitectura MVC



El diagrama representa la arquitectura **Modelo–Vista–Controlador (MVC)** utilizada en el proyecto. En este modelo, el **Usuario** interactúa directamente con la **Vista**, la cual muestra la interfaz gráfica (JSP). Cuando el usuario realiza una acción, como enviar un formulario, la Vista envía la solicitud al **Controlador** (el Servlet), encargado de procesarla y determinar qué operación debe ejecutarse. El Controlador se comunica con el **Modelo**, que contiene la lógica de negocio y el acceso a datos mediante el DAO y la base de datos MySQL. Una vez que el

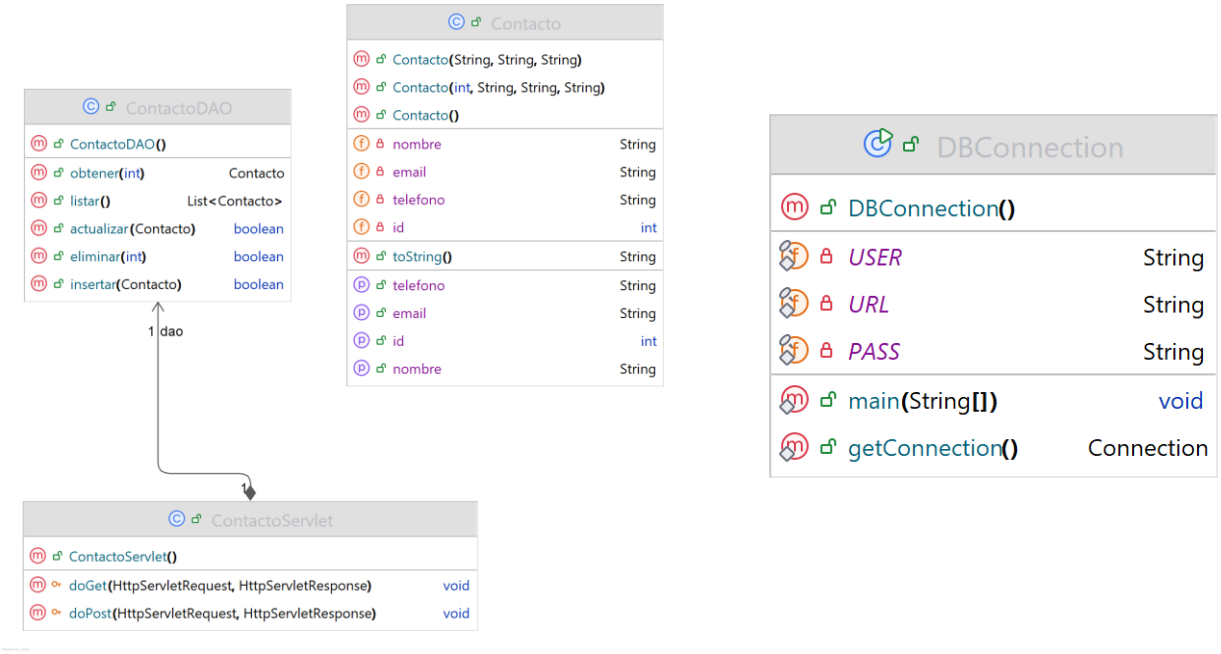
Modelo devuelve la información solicitada, el Controlador actualiza la Vista correspondiente, y esta finalmente muestra la respuesta al usuario.

En resumen, el diagrama ilustra cómo MVC separa responsabilidades:

- **Modelo** gestiona los datos.
- **Vista** muestra la información al usuario.
- **Controlador** coordina la comunicación entre ambos.

Esta estructura mejora la organización del código, facilita el mantenimiento del sistema y permite ampliar la aplicación de manera modular.

6.2 Diagrama UML



Descripción del Diagrama de Clases del Proyecto

El diagrama de clases representa la estructura principal del sistema y muestra cómo se organizan las clases y cómo se relacionan entre sí dentro del proyecto de Agenda Web implementado con Java y el patrón Modelo–Vista–Controlador (MVC).

Clase Contacto (Modelo)

La clase **Contacto** representa la entidad principal del sistema. Contiene los atributos básicos de un contacto: id, nombre, email y telefono. Además, incluye constructores, métodos *getters* y *setters* para manipular sus datos. Esta clase funciona como el **Modelo** dentro del patrón MVC, ya que encapsula la información que se almacena y recupera desde la base de datos.

Clase ContactoDAO (Acceso a Datos)

La clase **ContactoDAO** es responsable de la comunicación directa con la base de datos y contiene todos los métodos necesarios para realizar operaciones CRUD sobre la tabla contactos.
Entre sus métodos se encuentran:

- listar() para obtener todos los registros,
- insertar(Contacto c),
- actualizar(Contacto c),
- eliminar(int id),
- obtener(int id).

Esta clase utiliza objetos Contacto y se conecta a la base de datos mediante la clase DBConnection. Su función es separar la lógica de acceso a datos del resto del sistema, siguiendo el principio de responsabilidad única.

Clase DBConnection (Conexión a la Base de Datos)

La clase **DBConnection** administra la creación de conexiones hacia la base de datos MySQL utilizando JDBC. Contiene la URL de conexión, el usuario y la contraseña, además de un método getConnection() que devuelve una conexión lista para ser usada por el DAO. Esta clase centraliza la conexión a la base de datos y evita duplicación de código.

Clase ContactoServlet (Controlador)

El **ContactoServlet** actúa como el **Controlador** dentro del patrón MVC. Procesa las solicitudes HTTP enviadas desde las vistas JSP y determina qué acción ejecutar según los parámetros recibidos. Utiliza a la clase ContactoDAO para realizar operaciones sobre la base de datos y luego envía la información resultante a las vistas (lista.jsp o form.jsp) mediante atributos de solicitud.
Gestiona operaciones como:

- Mostrar la lista de contactos,
- Abrir el formulario de nuevo contacto,
- Editar un contacto existente,
- Eliminar un registro,
- Guardar cambios mediante POST.

Relaciones entre las clases

- ContactoDAO depende de **DBConnection** para obtener la conexión a MySQL.
- ContactoDAO utiliza instancias de **Contacto** para enviar y recibir datos de la base de datos.
- ContactoServlet utiliza **ContactoDAO** para ejecutar operaciones CRUD y administra la comunicación entre el usuario y el modelo.
- Las páginas JSP no aparecen en el diagrama como clases, pero se comunican con el Servlet a través de atributos de solicitud.

7. BASE DE DATOS

La base de datos es un componente fundamental para el funcionamiento de la aplicación, ya que almacena de forma persistente todos los contactos registrados por el usuario. En esta sección se describe la estructura del esquema utilizado, el diseño de la tabla principal y el script SQL necesario para recrear la base de datos.

7.1 Descripción del Esquema

El proyecto utiliza un esquema llamado **agenda**, el cual contiene una única tabla denominada **contactos**. Esta tabla almacena la información de cada contacto junto con un identificador único generado automáticamente.

Este esquema se creó en el sistema gestor de bases de datos **MySQL**, utilizando el motor de almacenamiento **InnoDB** y la codificación **utf8mb4**, garantizando compatibilidad con caracteres internacionales.

7.2 Diseño de la Tabla

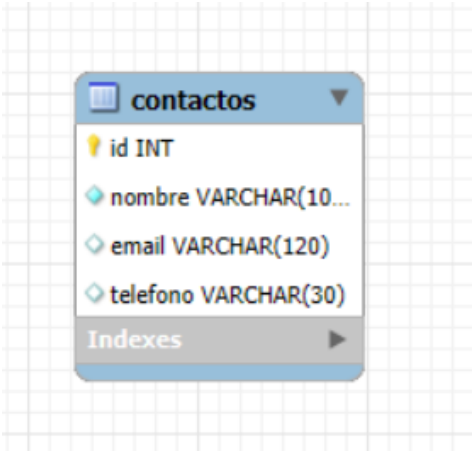
La tabla **contactos** está compuesta por los siguientes campos:

Campo	Tipo	Descripción
id	INT, PK, AI	Identificador numérico único del contacto
nombre	VARCHAR(100)	Nombre del contacto
email	VARCHAR(150)	Correo electrónico del contacto
telefono	VARCHAR(20)	Número telefónico del contacto

Características importantes:

- id usa **AUTO_INCREMENT**, por lo que no se ingresa manualmente.
- Los campos permiten valores NULL excepto nombre, que es obligatorio en la aplicación.
- No se usa clave foránea ya que el proyecto maneja un único recurso.

7.3 Diagrama de la Tabla



7.4 Script SQL de Creación del Esquema

```sql
CREATE DATABASE IF NOT EXISTS agenda
  DEFAULT CHARACTER SET utf8mb4
  DEFAULT COLLATE utf8mb4_unicode_ci;

USE agenda;

CREATE TABLE IF NOT EXISTS contactos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(150),
    telefono VARCHAR(20)
);
```

7.5 Datos de Ejemplo

```sql
INSERT INTO contactos (nombre, email, telefono) VALUES
('Ada Lovelace', 'ada@example.com', '555-0101'),
('Alan Turing',  'alan@example.com', '555-0102');
```

7.6 Dump de la Base de Datos

El siguiente dump fue generado mediante la herramienta *MySQL Workbench – Data Export*. Este archivo puede utilizarse para restaurar la base de datos completa del sistema en otro entorno de desarrollo o servidor.

```sql
CREATE DATABASE  IF NOT EXISTS `agenda` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;
USE `agenda`;
-- MySQL dump 10.13  Distrib 8.0.43, for Win64 (x86_64)
--
-- Host: localhost    Database: agenda
-- ------------------------------------------------------
-- Server version   8.0.43

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `contactos`
--

DROP TABLE IF EXISTS `contactos`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `contactos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `nombre` varchar(100) NOT NULL,
```

```
  `nombre` varchar(100) NOT NULL,
  `email` varchar(120) DEFAULT NULL,
  `telefono` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `contactos`
--

LOCK TABLES `contactos` WRITE;
/*!40000 ALTER TABLE `contactos` DISABLE KEYS */;
INSERT INTO `contactos` VALUES (1,'Ada Lovelace','ada@example.com','555-0101'),(2,'Alan Turing','alan@example.com','555-0102'),(3,'JORGE','titorrocastillo10@gmail.com','744428
/*!40000 ALTER TABLE `contactos` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2025-11-20 15:12:38
```

8. IMPLEMENTACIÓN

8.1 Estructura del Proyecto

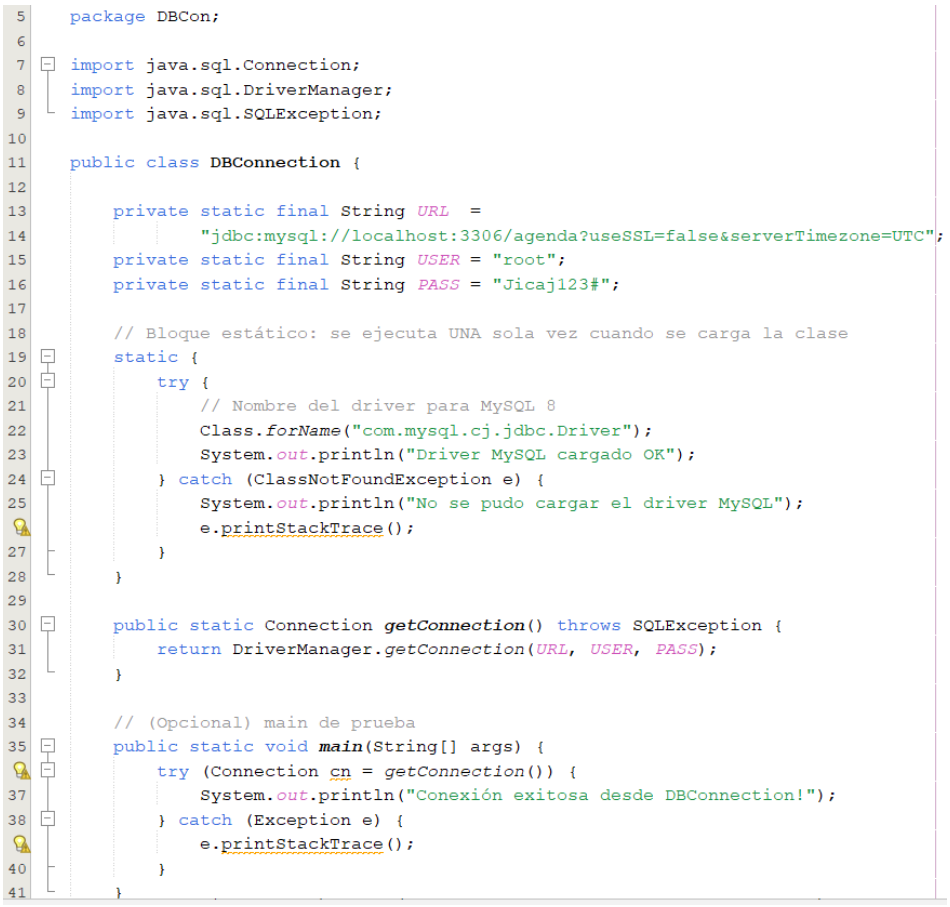
1. Clase Contacto (Modelo)

Esta clase representa la estructura de un contacto dentro del sistema. Contiene los atributos principales —*id, nombre, email y teléfono*— junto con sus métodos constructores, getters y setters. Esta clase funciona como el modelo de datos dentro del patrón MVC, ya que define la información que se captura, procesa y almacena en la base de datos. En esta sección se muestra el código completo de la clase, evidenciando cómo se encapsulan los datos del contacto y cómo se garantiza su manejo ordenado dentro del programa.

```java
package modelo;

public class Contacto {
    private int id;
    private String nombre;
    private String email;
    private String telefono;

    public Contacto() {}

    public Contacto(int id, String nombre, String email, String telefono) {
        this.id = id;
        this.nombre = nombre;
        this.email = email;
        this.telefono = telefono;
    }

    public Contacto(String nombre, String email, String telefono) {
        this.nombre = nombre;
        this.email = email;
        this.telefono = telefono;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getTelefono() { return telefono; }
    public void setTelefono(String telefono) { this.telefono = telefono; }

    @Override
    public String toString() {
        return "Contacto{id=" + id + ", nombre='" + nombre + "', email='" + email + "', telefono='" + telefono + "'}";
    }
}

```

2. Clase DBConnection (Conexión a la Base de Datos)

La clase DBConnection es responsable de establecer la conexión entre la aplicación y la base de datos MySQL. Contiene la URL de conexión, el usuario, la contraseña y un método principal (getConnection()) que devuelve una conexión activa mediante JDBC. También cuenta con un bloque estático que registra el driver de MySQL, asegurando que Tomcat puede reconocerlo al ejecutar el sistema.
Esta sección muestra el script completo que permite la comunicación entre el servidor Tomcat y la base de datos, lo cual es fundamental para que el CRUD funcione correctamente.



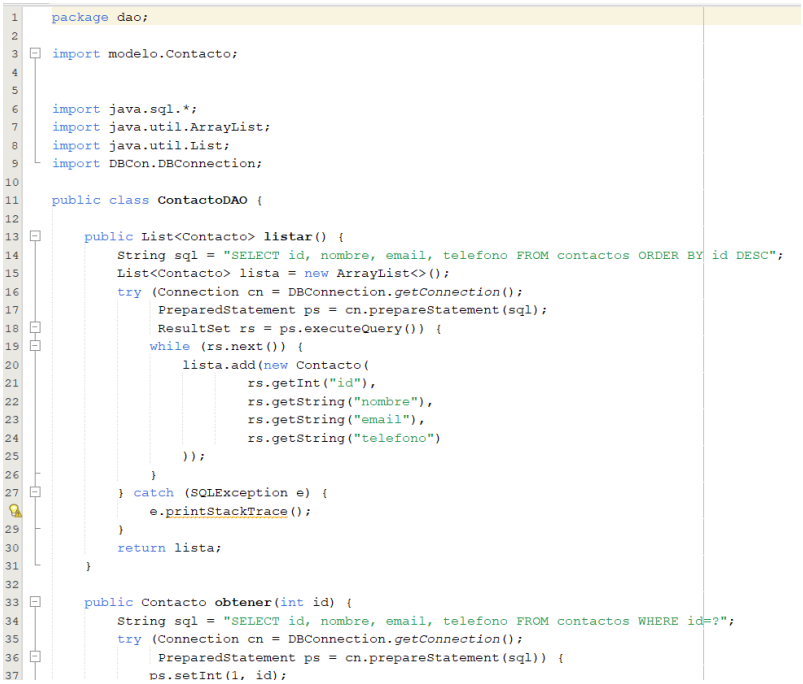
```java
package DBCon;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {

    private static final String URL  =
            "jdbc:mysql://localhost:3306/agenda?useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";
    private static final String PASS = "Jicaj123#";

    // Bloque estático: se ejecuta UNA sola vez cuando se carga la clase
    static {
        try {
            // Nombre del driver para MySQL 8
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Driver MySQL cargado OK");
        } catch (ClassNotFoundException e) {
            System.out.println("No se pudo cargar el driver MySQL");
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASS);
    }

    // (Opcional) main de prueba
    public static void main(String[] args) {
        try (Connection cn = getConnection()) {
            System.out.println("Conexión exitosa desde DBConnection!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Clase ContactoDAO (Capa de Acceso a Datos)

El ContactoDAO implementa toda la lógica relacionada con la base de datos. En esta clase se encuentran los métodos que ejecutan operaciones CRUD reales mediante SQL:

- listar() → devuelve todos los contactos
- insertar() → agrega un nuevo contacto
- actualizar() → modifica un contacto existente
- eliminar() → elimina un registro
- obtener() → busca un contacto por ID

Esta capa permite que la lógica de la aplicación permanezca separada del acceso a datos, manteniendo una arquitectura modular. En esta parte se muestran las capturas de cada uno de los métodos, junto con su estructura interna.



```java
package dao;

import modelo.Contacto;


import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import DBCon.DBConnection;

public class ContactoDAO {

    public List<Contacto> listar() {
        String sql = "SELECT id, nombre, email, telefono FROM contactos ORDER BY id DESC";
        List<Contacto> lista = new ArrayList<>();
        try (Connection cn = DBConnection.getConnection();
             PreparedStatement ps = cn.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                lista.add(new Contacto(
                        rs.getInt("id"),
                        rs.getString("nombre"),
                        rs.getString("email"),
                        rs.getString("telefono")
                ));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return lista;
    }

    public Contacto obtener(int id) {
        String sql = "SELECT id, nombre, email, telefono FROM contactos WHERE id=?";
        try (Connection cn = DBConnection.getConnection();
             PreparedStatement ps = cn.prepareStatement(sql)) {
            ps.setInt(1, id);
```

```java

    public Contacto obtener(int id) {
        String sql = "SELECT id, nombre, email, telefono FROM contactos WHERE id=?";
        try (Connection cn = DBConnection.getConnection();
             PreparedStatement ps = cn.prepareStatement(sql)) {
            ps.setInt(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return new Contacto(
                            rs.getInt("id"),
                            rs.getString("nombre"),
                            rs.getString("email"),
                            rs.getString("telefono")
                    );
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    public boolean insertar(Contacto c) {
        String sql = "INSERT INTO contactos (nombre, email, telefono) VALUES (?,?,?)";
        try (Connection cn = DBConnection.getConnection();
             PreparedStatement ps = cn.prepareStatement(sql)) {
            ps.setString(1, c.getNombre());
            ps.setString(2, c.getEmail());
            ps.setString(3, c.getTelefono());
            return ps.executeUpdate() == 1;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
',

     public boolean actualizar(Contacto c) {
         String sql = "UPDATE contactos SET nombre=?, email=?, telefono=? WHERE id=?";
         try (Connection cn = DBConnection.getConnection();
              PreparedStatement ps = cn.prepareStatement(sql)) {
             ps.setString(1, c.getNombre());
             ps.setString(2, c.getEmail());
             ps.setString(3, c.getTelefono());
             ps.setInt(4, c.getId());
             return ps.executeUpdate() == 1;
         } catch (SQLException e) {
             e.printStackTrace();
             return false;
         }
     }

     public boolean eliminar(int id) {
         String sql = "DELETE FROM contactos WHERE id=?";
         try (Connection cn = DBConnection.getConnection();
              PreparedStatement ps = cn.prepareStatement(sql)) {
             ps.setInt(1, id);
             return ps.executeUpdate() == 1;
         } catch (SQLException e) {
             e.printStackTrace();
             return false;
         }
     }
}
```

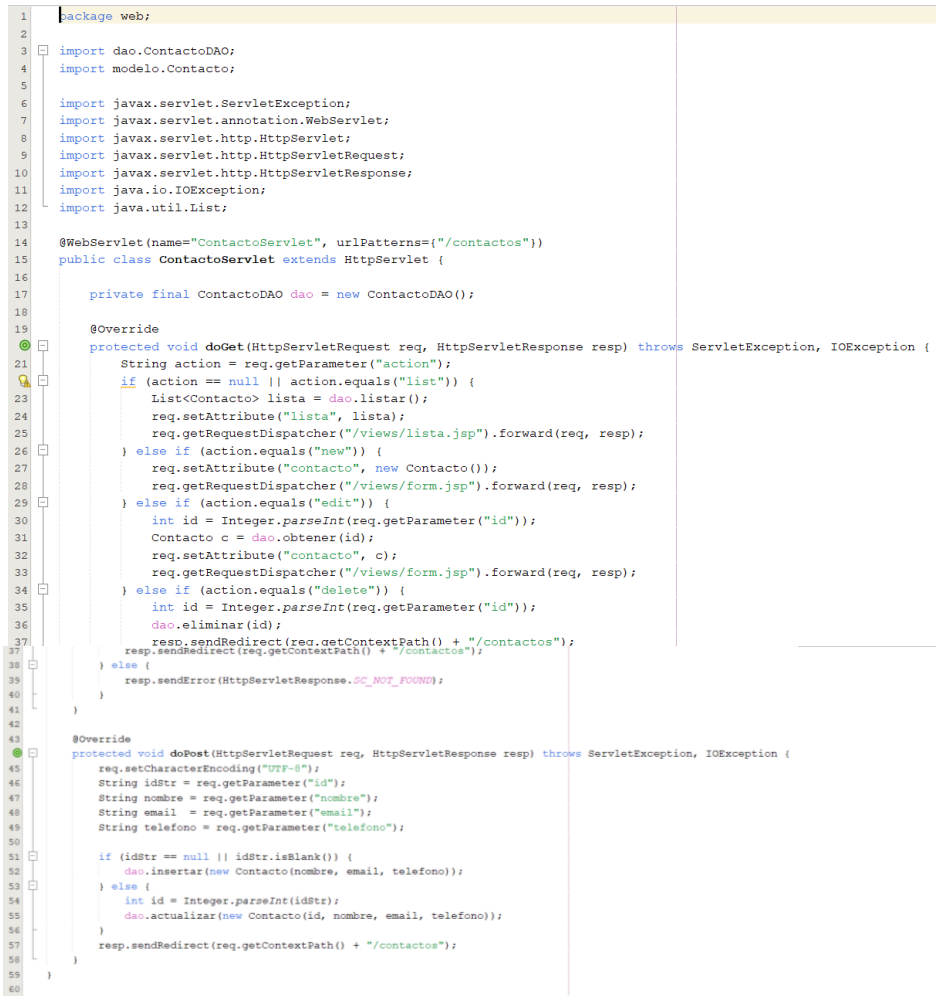
4. Clase ContactoServlet (Controlador – MVC)

El ContactoServlet actúa como **controlador principal del sistema**, procesando todas las peticiones HTTP que llegan desde el navegador.
En el método doGet se manejan las acciones de listar, crear, editar y eliminar contactos.
En el método doPost se gestionan las operaciones de guardado y actualización en la base de datos.

El servlet coordina la comunicación entre:

- la capa de acceso a datos (DAO)
- el modelo (Contacto)
- y las vistas (JSP)

En esta sección se presenta el código del servlet, demostrando cómo dirige el flujo de información entre las distintas capas del sistema.



```java
package web;

import dao.ContactoDAO;
import modelo.Contacto;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet(name="ContactoServlet", urlPatterns={"/contactos"})
public class ContactoServlet extends HttpServlet {

    private final ContactoDAO dao = new ContactoDAO();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String action = req.getParameter("action");
        if (action == null || action.equals("list")) {
            List<Contacto> lista = dao.listar();
            req.setAttribute("lista", lista);
            req.getRequestDispatcher("/views/lista.jsp").forward(req, resp);
        } else if (action.equals("new")) {
            req.setAttribute("contacto", new Contacto());
            req.getRequestDispatcher("/views/form.jsp").forward(req, resp);
        } else if (action.equals("edit")) {
            int id = Integer.parseInt(req.getParameter("id"));
            Contacto c = dao.obtener(id);
            req.setAttribute("contacto", c);
            req.getRequestDispatcher("/views/form.jsp").forward(req, resp);
        } else if (action.equals("delete")) {
            int id = Integer.parseInt(req.getParameter("id"));
            dao.eliminar(id);
            resp.sendRedirect(req.getContextPath() + "/contactos");
            resp.sendRedirect(req.getContextPath() + "/contactos");
        } else {
            resp.sendError(HttpServletResponse.SC_NOT_FOUND);
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        String idStr = req.getParameter("id");
        String nombre = req.getParameter("nombre");
        String email  = req.getParameter("email");
        String telefono = req.getParameter("telefono");

        if (idStr == null || idStr.isBlank()) {
            dao.insertar(new Contacto(nombre, email, telefono));
        } else {
            int id = Integer.parseInt(idStr);
            dao.actualizar(new Contacto(id, nombre, email, telefono));
        }
        resp.sendRedirect(req.getContextPath() + "/contactos");
    }
}
```

5. Vistas JSP: lista.jsp y form.jsp (Capa de Presentación)

Las vistas JSP conforman la interfaz gráfica del sistema y muestran la información al usuario.

lista.jsp

Muestra todos los contactos registrados. Incluye una tabla donde se renderiza dinámicamente la información proveniente del Servlet y botones para editar, crear o eliminar registros.

form.jsp

Contiene el formulario utilizado para agregar o editar un contacto. Envía los datos al Servlet mediante método POST, e incluye controles básicos de validación en los campos visibles.

En esta parte se muestran las capturas del código correspondiente a ambas vistas, evidenciando cómo se generan interfaces dinámicas usando JSP y atributos del request.

lista.jsp:

```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Agenda - Contactos</title>
  <style>
    body{font-family: Arial, sans-serif; max-width: 900px; margin: 2rem auto;}
    table{border-collapse: collapse; width:100%;}
    th, td{border:1px solid #ccc; padding:8px;}
    .topbar{display:flex; justify-content:space-between; align-items:center; margin-bottom:1rem;}
    a.button{padding:8px 12px; border:1px solid #444; text-decoration:none;}
  </style>
</head>
<body>
  <div class="topbar">
    <h1>Contactos</h1>
    <a class="button" href="${pageContext.request.contextPath}/contactos?action=new">Nuevo</a>
  </div>

  <c:choose>
    <c:when test="${empty lista}">
      <p>No hay contactos aún.</p>
    </c:when>
    <c:otherwise>
      <table>
        <thead>
        <tr><th>ID</th><th>Nombre</th><th>Email</th><th>Teléfono</th><th>Acciones</th></tr>
        </thead>
        <tbody>
        <c:forEach var="c" items="${lista}">
          <tr>
            <td>${c.id}</td>
            <td>${c.nombre}</td>
            <td>${c.email}</td>
            <td>${c.telefono}</td>
```

```jsp
  </div>

  <c:choose>
    <c:when test="${empty lista}">
      <p>No hay contactos aún.</p>
    </c:when>
    <c:otherwise>
      <table>
        <thead>
        <tr><th>ID</th><th>Nombre</th><th>Email</th><th>Teléfono</th><th>Acciones</th></tr>
        </thead>
        <tbody>
        <c:forEach var="c" items="${lista}">
          <tr>
            <td>${c.id}</td>
            <td>${c.nombre}</td>
            <td>${c.email}</td>
            <td>${c.telefono}</td>
            <td>
              <a href="${pageContext.request.contextPath}/contactos?action=edit&id=${c.id}">Editar</a>
              |
              <a href="${pageContext.request.contextPath}/contactos?action=delete&id=${c.id}" onclick="return confirm('¿Eliminar?');">Elimi
            </td>
          </tr>
        </c:forEach>
        </tbody>
      </table>
    </c:otherwise>
  </c:choose>
</body>
</html>
```

form.jsp:

```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Contacto</title>
  <style>
    body{font-family: Arial, sans-serif; max-width: 700px; margin: 2rem auto;}
    form{display:grid; gap:.6rem;}
    label{font-weight:bold;}
    input[type=text], input[type=email]{padding:.5rem; width:100%;}
    .row{display:grid; grid-template-columns: 1fr 1fr; gap: 1rem;}
  </style>
</head>
<body>
  <h1><%= request.getAttribute("contacto") == null || ((modelo.Contacto)request.getAttribute("contacto")).getId() == 0 ? "Nuevo" : "Editar"
  <%
    modelo.Contacto c = (modelo.Contacto) request.getAttribute("contacto");
    if (c == null) { c = new modelo.Contacto(); }
  %>
  <form method="post" action="${pageContext.request.contextPath}/contactos">
    <input type="hidden" name="id"
       value="<%= (c.getId() == 0 ? "" : c.getId()) %>"/>
    <label>Nombre</label>
    <input type="text" name="nombre" value="<%= c.getNombre() == null ? "" : c.getNombre() %>" required/>
    <div class="row">
      <div>
        <label>Email</label>
        <input type="email" name="email" value="<%= c.getEmail() == null ? "" : c.getEmail() %>"/>
      </div>
      <div>
        <label>Teléfono</label>
        <input type="text" name="telefono" value="<%= c.getTelefono() == null ? "" : c.getTelefono() %>"/>
      </div>
    </div>
    <div>
```

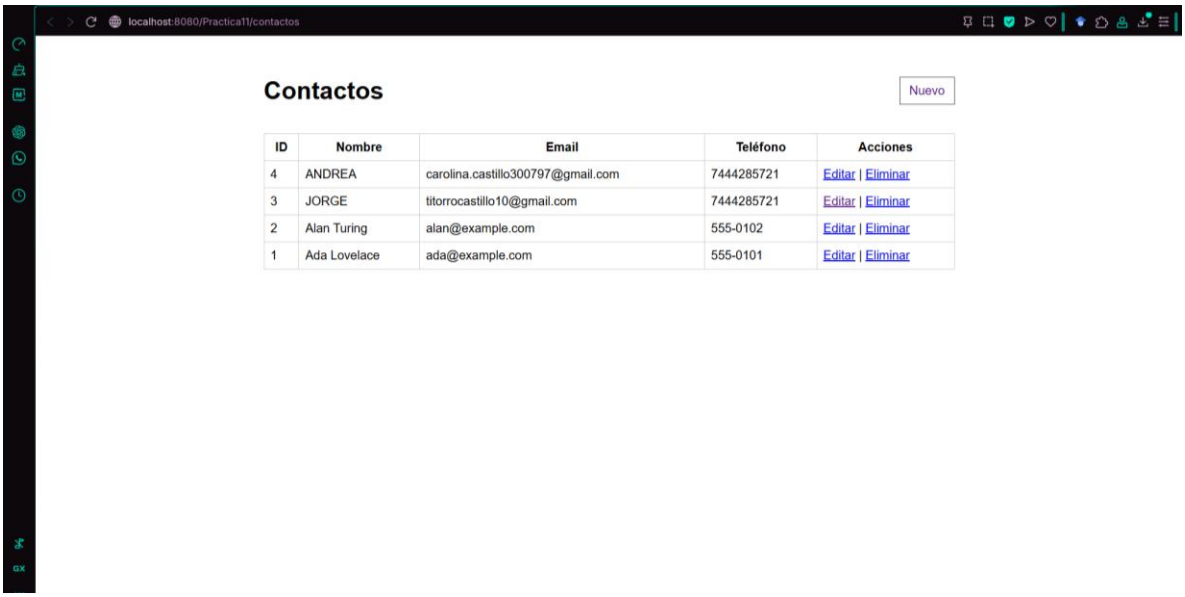
6. Archivo web.xml (Configuración del Proyecto Web)

El archivo web.xml define la configuración base del proyecto dentro de WEB-INF. Aunque el proyecto también utiliza anotaciones con @WebServlet, este archivo sirve como referencia de la estructura del despliegue, el mapeo del servlet, las rutas internas y cualquier configuración adicional del contenedor web.
En esta sección se presenta la estructura del archivo XML tal como se genera dentro del proyecto.

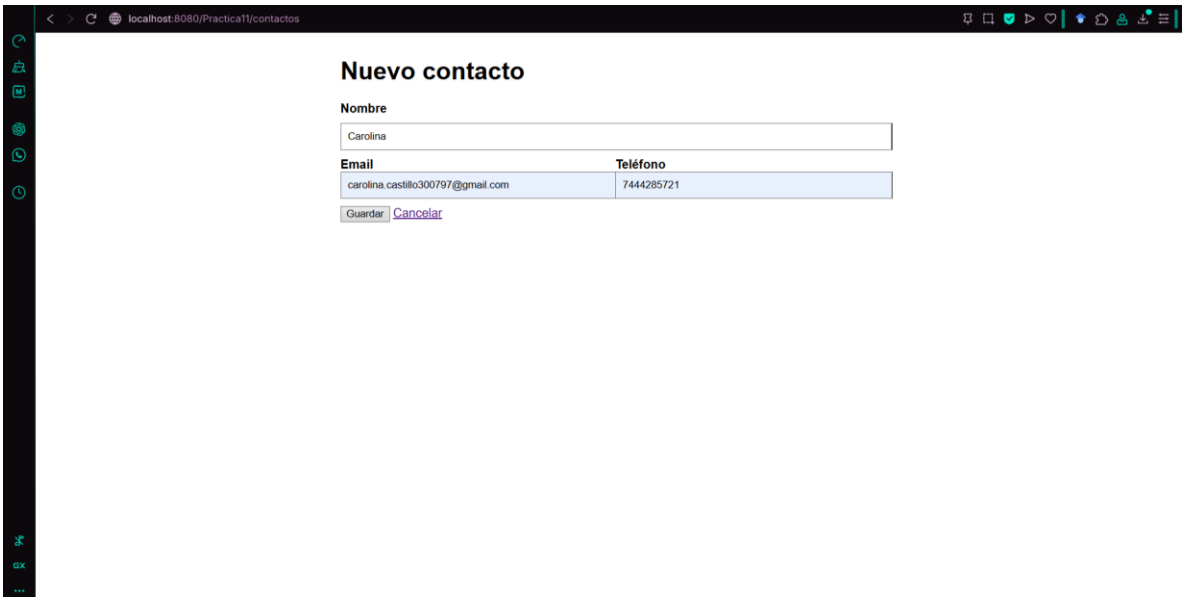
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <display-name>Practica11</display-name>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>

```

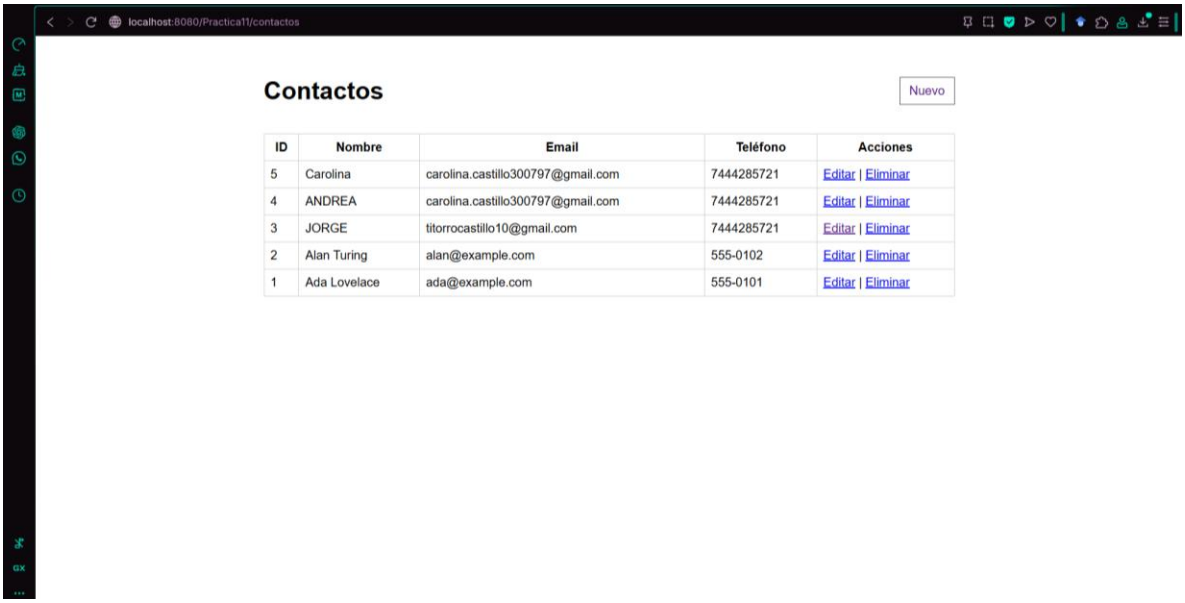
9. PRUEBA DE FUNCIONAMIENTO

Ejecución del proyecto (Mostrar contactos):



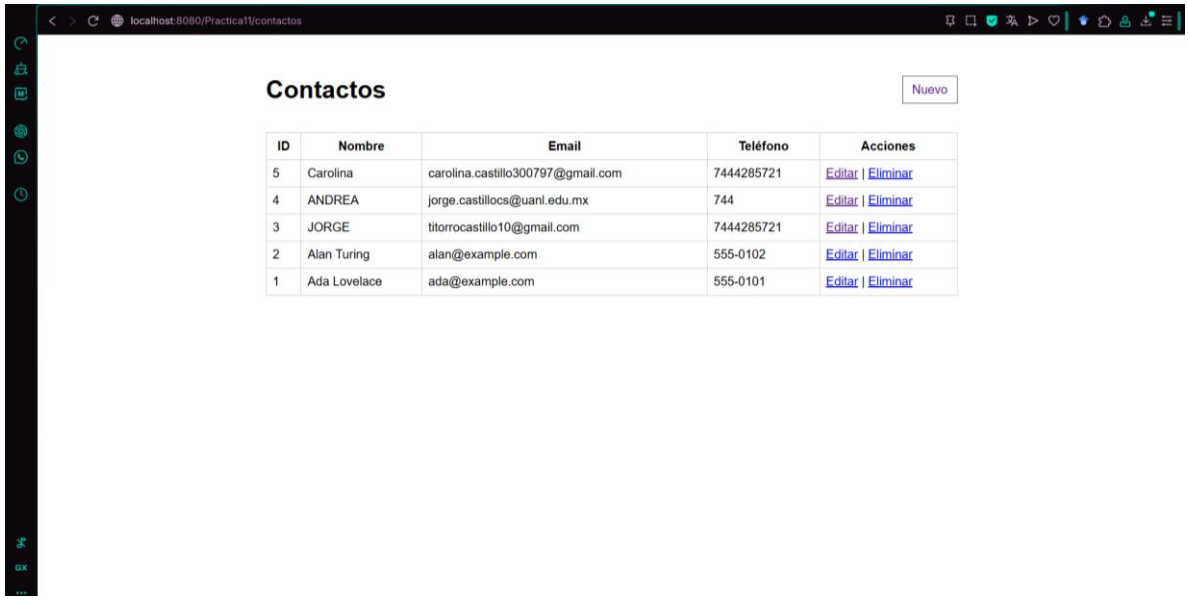
Insertar contacto:



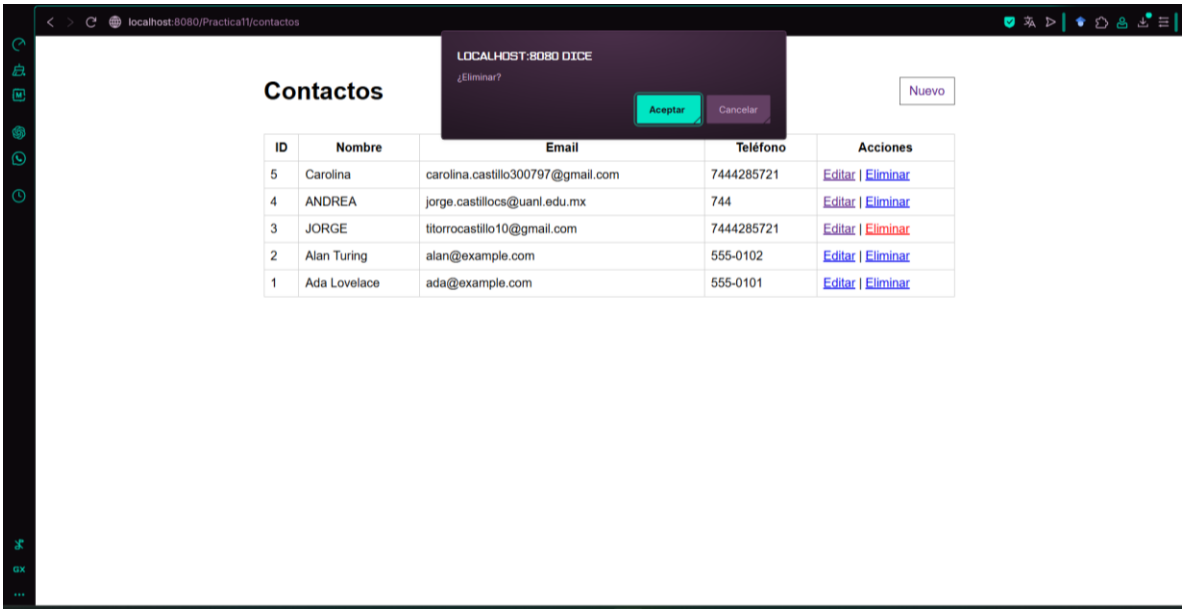


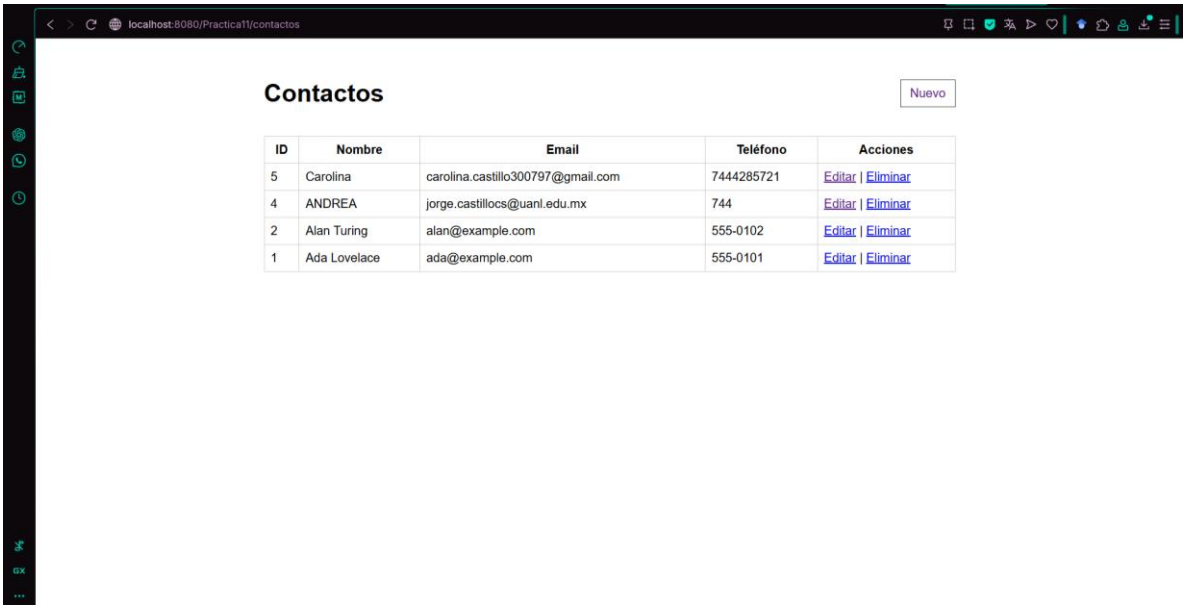
Editar contacto:


localhost:8080/Practica11/contactos
Editar contacto
Nombre
ANDREA
Email
jorge.castillocs@uanl.edu.mx
Teléfono
744
7444285721
Guardar
Cancelar


localhost:8080/Practica11/contactos
Contactos
Nuevo
ID
Nombre
Email
Teléfono
Acciones
5
Carolina
carolina.castillo300797@gmail.com
7444285721
Editar | Eliminar
4
ANDREA
jorge.castillocs@uanl.edu.mx
744
Editar | Eliminar
3
JORGE
titorrocastillo10@gmail.com
7444285721
Editar | Eliminar
2
Alan Turing
alan@example.com
555-0102
Editar | Eliminar
1
Ada Lovelace
ada@example.com
555-0101
Editar | Eliminar

Eliminar contacto:





10.CONCLUSIÓN

El desarrollo de este proyecto permitió comprender de manera práctica el funcionamiento de una aplicación web construida con tecnologías esenciales del ecosistema Java, tales como Servlets, JSP, JDBC y el patrón de diseño Modelo–Vista–Controlador (MVC). A través de la implementación de una agenda digital con operaciones CRUD completas, fue posible integrar todos los componentes necesarios para crear un sistema funcional, estructurado y mantenible.

Durante el proceso se reforzó el uso de principios fundamentales de la programación orientada a objetos, la importancia de separar responsabilidades mediante capas (Modelo, DAO, Controlador y Vista), y la correcta administración de conexiones a la base de datos mediante JDBC. Asimismo, la experiencia de configurar un servidor de aplicaciones real como Apache Tomcat permitió entender cómo se despliegan aplicaciones web en un entorno profesional.

La construcción de la base de datos en MySQL, junto con el uso de un dump para su respaldo, aseguró un almacenamiento persistente, organizado y recuperable de los contactos. De igual manera, la generación de las vistas en JSP y su comunicación con el Servlet reforzaron el ciclo de petición–respuesta dentro de una arquitectura web.

En conjunto, este proyecto no solo logró cumplir con los objetivos planteados, sino que proporcionó una visión integral del proceso de construcción de una aplicación web en Java, desde la estructura del código hasta la interacción con el usuario final. La experiencia obtenida sienta una base sólida para futuros desarrollos más complejos y escalables dentro del área del desarrollo web y la ingeniería de software.

11. Referencias

Caules, C. Á. (2024, 24 septiembre). *El modelo vista controlador y sus responsabilidades*. Arquitectura Java. https://www.arquitecturajava.com/el-modelo-vista-controlador-y-sus-responsabilidades

IBM Operational Decision Manager. (s. f.). https://www.ibm.com/docs/es/odm/8.12.0?topic=api-http-methods

Design patterns: Data access object. (s. f.). https://www.oracle.com/java/technologies/data-access-object.html

12. Repositorio GitHub

https://github.com/J-Cazz/Agenda_Servlet_PIA/tree/ed9bc31b735f71b3d519a5613557ae224098721b