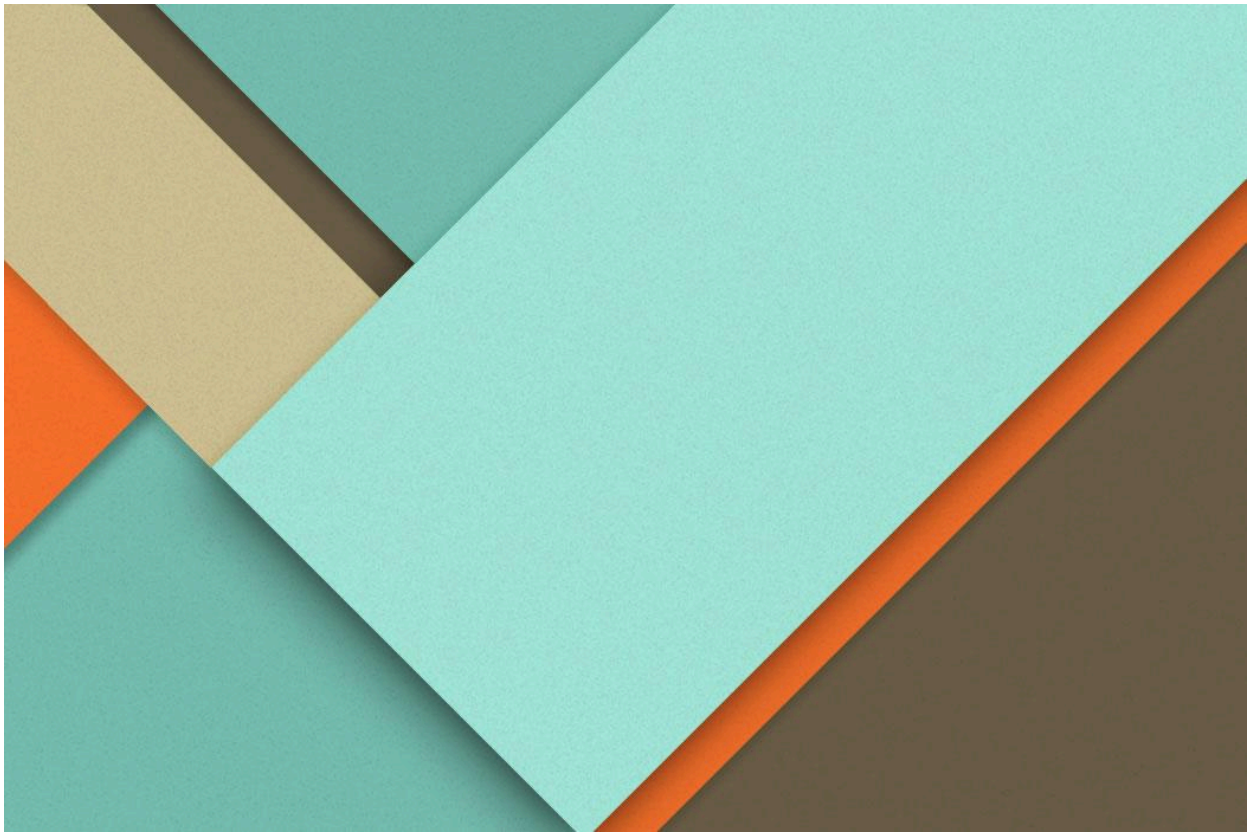


Aplicación de gestión de empleados

12/05/2025

—

Unai Bercianos Martín
Máster en Java / Máster en Desarrollo Full Stack
Academia Atrium

Visión general

Este proyecto consiste en el desarrollo de una aplicación web para la **gestión de empleados**. El objetivo principal es permitir a los administradores y responsables de RRHH llevar un control eficiente de los empleados registrados en la empresa, mediante funcionalidades como la inserción, consulta, modificación y eliminación de datos de los empleados.

La aplicación está desarrollada en **Java**, utilizando el patrón **MVC** junto con el patrón **DAO** para el acceso a datos. La interfaz de usuario se ha construido con **HTML, CSS y JavaScript**, y la persistencia de datos se realiza mediante una base de datos **MySQL**. Además, se ha implementado un sistema de autenticación con contraseñas cifradas mediante **BCrypt**, y se han incluido pruebas unitarias utilizando **JUnit 5 y SQLite** para garantizar el correcto funcionamiento de las operaciones más importantes.

1.- Introducción al proyecto

El presente proyecto consiste en el desarrollo de una aplicación de gestión de empleados orientada a proporcionar un sistema CRUD (Crear, Leer, Actualizar, Eliminar) funcional, fiable y fácilmente ampliable. La aplicación ha sido desarrollada como parte del Trabajo de Fin de Máster y tiene como objetivo aplicar los conocimientos adquiridos durante la formación, utilizando tecnologías estándar de desarrollo Java y bases de datos relacionales.

2.- Objetivos del proyecto

- Desarrollar una aplicación de gestión de empleados con operaciones básicas CRUD.
- Aplicar patrones de diseño como DAO y MVC para garantizar una arquitectura modular.
- Gestionar la persistencia de datos con MySQL.
- Realizar pruebas unitarias usando JUnit 5.
- Mantener una separación clara entre la lógica de negocio y el acceso a datos.
- Documentar el código con Javadoc y elaborar una memoria técnica del sistema.

3.- Análisis de requisitos funcionales

A continuación, se detallan los requisitos funcionales implementados en la aplicación de gestión de empleados:

1. **Gestión de empleados (CRUD)**
    - Permite crear, leer, actualizar y eliminar empleados.
    - Cada empleado está asociado a un departamento y tiene los campos: ID, nombre, email, salario y departamento.
2. **Persistencia de datos**
    - Los datos se almacenan en una base de datos MySQL.
    - Se utiliza JDBC para la conexión y manipulación de datos desde la aplicación Java.
3. **Interfaz web**
    - La aplicación presenta una interfaz web sencilla utilizando HTML, CSS y JavaScript.
    - El usuario puede visualizar todos los empleados en una tabla dinámica, editar sus datos o eliminarlos.
4. **Gestión de usuarios**
    - Implementa un sistema de autenticación con credenciales cifradas mediante BCrypt.
    - Se gestiona al menos un usuario administrador con permisos para acceder a todas las funciones.

-

5. **Control de acceso**
    - El acceso a las páginas está protegido mediante un filtro de autenticación.
    - Solo los usuarios autenticados pueden acceder al sistema.
6. **Patrones de diseño**
    - Se ha implementado el patrón **DAO** para el acceso a datos.
    - Se ha seguido el patrón **MVC** para separar la lógica de negocio, vista y controlador.
7. **Pruebas unitarias**
    - Se han desarrollado pruebas con **JUnit 5 y SQLite** para comprobar el correcto funcionamiento de operaciones críticas como inserción, borrado, actualización y recuperación de empleados.
8. **Documentación técnica**
    - El código está comentado y documentado con **JavaDoc**.
    - Se incluye una guía para desarrolladores.
9. **Control de versiones**
    - Todo el proyecto ha sido gestionado mediante **Git** y está disponible en mi repositorio de GitHub https://github.com/Unaiber/Gestion-de-Empleados

4.- Análisis de requisitos no funcionales

A continuación, se describen los requisitos no funcionales que guían la calidad, seguridad y rendimiento de la aplicación:

1. **Seguridad**
    - Las contraseñas de los usuarios se almacenan cifradas con el algoritmo **BCrypt** para garantizar la protección de credenciales.
    - Se controla el acceso a las páginas mediante un **filtro de autenticación**, evitando que usuarios no autorizados accedan a las funcionalidades internas.
2. **Usabilidad**
    - La interfaz web es clara y accesible, permitiendo gestionar empleados de forma intuitiva mediante una tabla dinámica.
    - Se han utilizado confirmaciones para operaciones críticas como la eliminación de empleados.
3. **Portabilidad**
    - La aplicación se ejecuta en cualquier servidor compatible con **Java y Tomcat**.
    - No requiere instalación compleja y puede adaptarse a distintos entornos.
4. **Mantenibilidad**
    - El código está documentado con **JavaDoc**, comentado, estructurado y dividido en capas (DAO, modelo, controlador).
    - Se utilizan patrones de diseño (DAO, MVC) que facilitan futuras ampliaciones o correcciones.

5. **Fiabilidad**
    - Se han implementado pruebas unitarias con **JUnit 5 y SQLite** para validar operaciones críticas de la base de datos y asegurar el correcto comportamiento del sistema.
6. **Rendimiento**
    - Las operaciones CRUD están optimizadas mediante consultas SQL directas y acceso controlado a la base de datos a través de **conexiones JDBC.**
7. **Compatibilidad**
    - La aplicación es compatible con navegadores actuales (Chrome, Firefox, Edge).
    - El frontend utiliza tecnologías estándar como HTML5, CSS y JavaScript.

5.- Diseño de la arquitectura

La arquitectura del sistema sigue el patrón **MVC (Modelo-Vista-Controlador)**,. La estructura se centra en dos capas principales:

- **Modelo**: representado por las clases `Empleado` y `Usuario`, que definen las entidades del sistema. Estas clases contienen atributos (como nombre, email, sueldo, etc.) y métodos relacionados con la lógica de negocio.

- **DAO (Data Access Object)**: contiene clases como `EmpleadosDAO` y `UsuariosDAO`, que son responsables de gestionar la conexión con la base de datos y realizar operaciones como insertar, eliminar, actualizar o consultar registros.

- **Controlador**: implementado mediante **Servlets**, como `GestionEmpleados` y `GestionUsuarios`, que reciben las peticiones del usuario desde la interfaz web, procesan la lógica correspondiente y redirigen a las vistas.

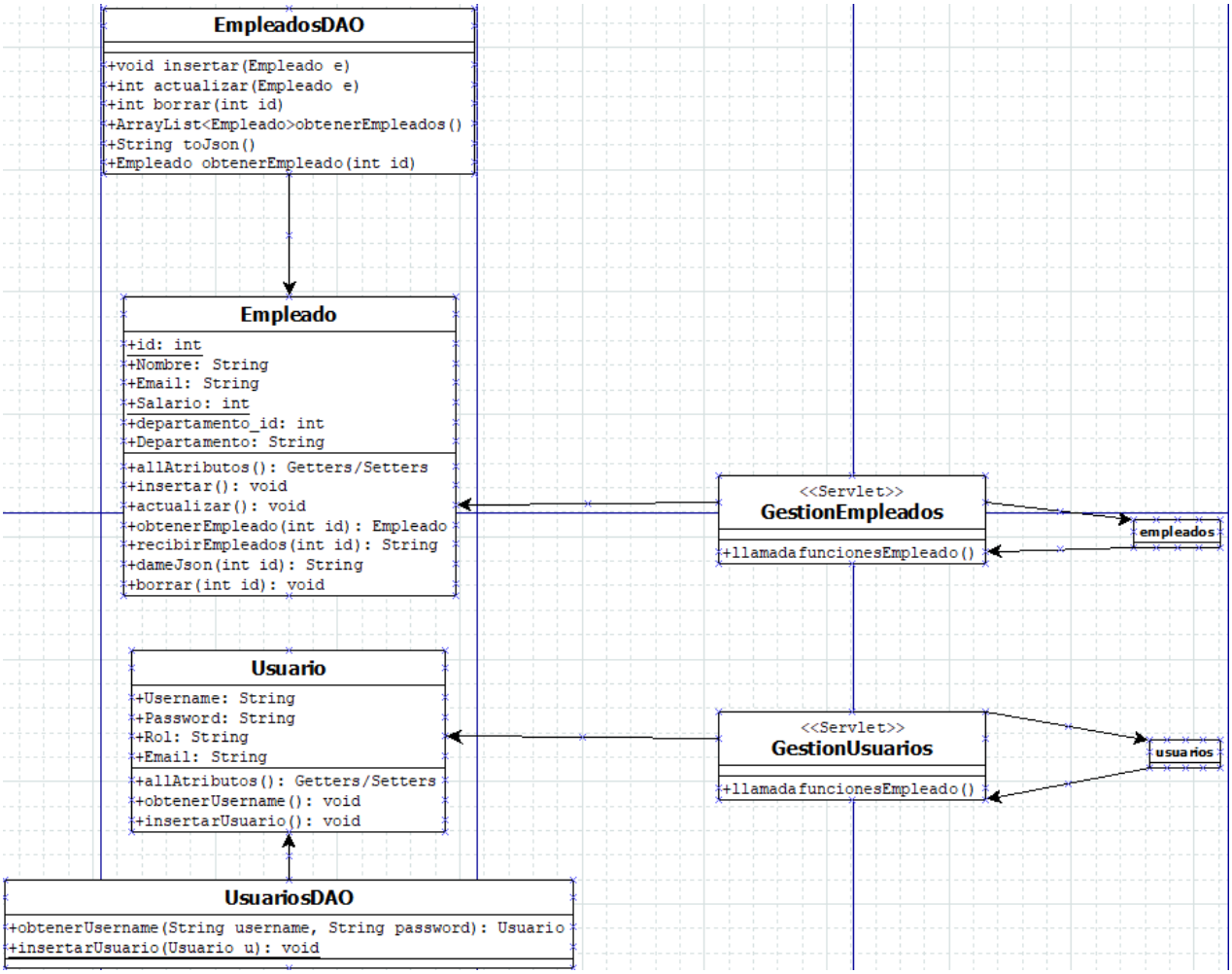
Las clases `Empleado y Usuario` contienen los métodos de negocio y actúan como puente con las clases `EmpleadosDAO y UsuariosDAO` respectivamente, responsables de interactuar con la base de datos.

6.- Persistencia de datos

Se ha utilizado una base de datos **MySQL** para almacenar los registros de empleados. La tabla empleados contiene los campos básicos (id, nombre, email, salario, departamento_id).

Las operaciones de base de datos se han desarrollado usando `PreparedStatement` para garantizar la seguridad frente a inyecciones SQL y mejorar la eficiencia.

Además, durante las pruebas, se ha utilizado una base de datos SQLite en memoria para evitar alteraciones en los datos reales y acelerar la ejecución de los tests.



7.- Pruebas unitarias

Para verificar el correcto funcionamiento del sistema, se han desarrollado pruebas unitarias con **JUnit 5**. Se ha implementado una clase de test `TestEmpleadosDAO_SQLite`, que realiza las siguientes comprobaciones:

- Inserción de empleados.
- Lectura de empleados.
- Actualización de datos.
- Eliminación por ID.

En estos test se prueban, con inserciones manuales, que las funcionalidades funcionen correctamente, sin errores. Debido a que se restringen ciertas alternativas vía cliente, es decir, al tener las funciones muy guiadas respecto a inserción de datos, no se contemplan diferentes test.

Estas pruebas se ejecutan sobre una base de datos SQLite en memoria, aislando el entorno de producción y permitiendo una ejecución repetible y fiable.

8.- Conclusiones

El desarrollo del proyecto ha permitido afianzar conocimientos sobre desarrollo backend en Java, bases de datos, diseño modular y pruebas automatizadas. La arquitectura basada en DAO permite una futura integración sencilla con interfaces web o móviles.

El uso de pruebas unitarias con una base de datos en memoria ha sido clave para garantizar la calidad del código y evitar errores en la lógica de persistencia.

9.- Créditos y experiencia personal

Ahora, hablándoos desde mi faceta personal no académica, quería agradecer a todos los profesores de la Academia Atrium del Máster de Java y Desarrollo Full Stack por su entrega y por contagiarnos el buen rollo en cada clase, además de saber cómo plasmarnos toda la información del curso en un tiempo tan limitado.

En mi caso, habéis conseguido que haya cogido el gusto a la programación después de haberme tirado a la piscina con esta decisión. Habiendo empezado casi desde cero, haber conseguido desarrollar una aplicación web así después de alrededor de seis meses de curso dice mucho de la calidad del profesorado.

Espero que a vosotros, tanto como a mí, os complazca saber que he decidido seguir con mi carrera académica en este ámbito, matriculandome en una FP de Desarrollo de Aplicaciones Multiplataforma que empezaré a cursar a partir de septiembre de este mismo año.