UNIVERSIDAD DE LAS FUERZAS ARMADAS “ESPE”

Departamento de Ciencias de la Computación(DCCO)
Curso de Ingeniería de Software

Programación Web

Proyecto Gestión de Compras

NRC: 27815

TERCERA UNIDAD:

Klever Jami

08/02/2026

ÍNDICE

1. Introducción 2
2. Visión y Alcance 3
3. Arquitectura del sistema 3
4. Funcionamiento General del Sistema..... 4
5. Conclusiones 5

1. Introducción

Este proyecto consiste en desarrollar un sistema web completo para gestionar el proceso de compras de una empresa. La aplicación permite administrar tres entidades principales: Proveedores (quienes suministran productos), Bodegas (lugares donde se almacenan) e Ingresos (registro de recepción de mercancía que vincula proveedor y bodega).

El sistema fue implementado usando arquitectura MVC con Java EE, lo que separa claramente la lógica de negocio (Modelo), la presentación (Vista) y el control de flujo (Controlador). Esta separación hace que el código sea más mantenible, escalable y fácil de entender.

La aplicación se conecta a una base de datos MySQL mediante JDBC para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre cada una de las entidades. La interfaz de usuario está construida con JSP y JSTL, y cuenta con un diseño moderno y responsivo mediante CSS personalizado.

2. Objetivos

Objetivo General

Desarrollar una aplicación web funcional que permita gestionar de forma eficiente el registro de proveedores, bodegas e ingresos de inventario, aplicando el patrón de diseño MVC y tecnologías Java EE estándar.

Objetivos Específicos

- Diseñar e implementar la base de datos relacional con las tres tablas principales y sus relaciones.
- Construir la capa de Modelo con clases Java que representen las entidades y gestionen el acceso a datos mediante JDBC
- Desarrollar la capa de Controlador con Servlets que manejen las peticiones HTTP y coordinen el flujo de la aplicación
- Crear la capa de Vista con páginas JSP y JSTL para presentar la información al usuario de forma clara
- Implementar operaciones CRUD completas para las tres entidades (Proveedor, Bodega, Ingreso)
- Aplicar un diseño visual consistente y moderno utilizando CSS personalizado
- Desplegar la aplicación en el servidor GlassFish y verificar su correcto funcionamiento

3. Marco Teórico

3.1 Patrón MVC

El patrón MVC es una arquitectura de software que separa la aplicación en tres componentes principales:

Modelo: Representa la lógica de negocio y los datos de la aplicación. En nuestro caso, incluye las clases Java que representan las entidades (Proveedor, Bodega, Ingreso) y las clases DAO que gestionan la conexión a la base de datos y las consultas SQL.

Vista: Es la capa de presentación que muestra los datos al usuario. Está implementada con páginas JSP que utilizan JSTL para mostrar información dinámica.

Controlador: Actúa como intermediario entre el Modelo y la Vista. Los Servlets reciben las peticiones HTTP del navegador, invocan los métodos del Modelo necesarios, y deciden qué Vista mostrar según el resultado.

Esta separación tiene múltiples ventajas: el código es más organizado, cada componente tiene una responsabilidad clara, los cambios en la interfaz no afectan la lógica de negocio, y varios desarrolladores pueden trabajar en paralelo en diferentes capas.

3.2 JDBC

JDBC es la API estándar de Java para conectarse a bases de datos relacionales. Permite ejecutar consultas SQL desde código Java y procesar los resultados.

En este proyecto, JDBC se utiliza en la clase `DatabaseConnection` para establecer la conexión con MySQL, y en las clases DAO para ejecutar sentencias SQL como `INSERT`, `SELECT`, `UPDATE` y `DELETE`. El uso de `PreparedStatement` previene inyecciones SQL y mejora el rendimiento al reutilizar consultas precompiladas.

3.3 JSP y JSTL

JSP: Permite incrustar código Java dentro de HTML para generar páginas web dinámicas. Sin embargo, mezclar mucho código Java en las vistas hace difícil el mantenimiento.

JSTL: Es una librería de etiquetas que reemplaza el código Java embebido por etiquetas XML más limpias y legibles. En este proyecto se usa `c:forEach` para iterar sobre listas de registros, `c:if` y `c:choose` para condicionales, y `c:url` para construir URLs correctamente con el context path del servidor.

4. Desarrollo del Proyecto

4.1 Base de Datos MySQL

La base de datos `gestion_compras` contiene tres tablas principales con las siguientes características:

Tabla: proveedor

Almacena la información de los proveedores que suministran productos.

Campo	Tipo	Descripción
idProveedor	INT (PK, AUTO_INCREMENT)	Identificador único
nombre	VARCHAR(120)	Nombre del proveedor
ruc	VARCHAR(13)	RUC del proveedor

Table 1: Estructura de la tabla proveedor

Tabla: bodega

Registra los lugares físicos donde se almacena la mercancía.

Campo	Tipo	Descripción
idBodega	INT (PK, AUTO_INCREMENT)	Identificador único
nombre	VARCHAR(120)	Nombre de la bodega
ubicacion	VARCHAR(200)	Dirección o ubicación

Table 2: Estructura de la tabla bodega

Tabla: ingreso

Registra cada entrada de mercancía, vinculando proveedor y bodega mediante llaves foráneas.

Campo	Tipo	Descripción
idIngreso	INT (PK, AUTO_INCREMENT)	Identificador único
fecha	DATE	Fecha del ingreso
cantidad	INT	Cantidad recibida
idProveedor	INT (FK)	Referencia a proveedor
idBodega	INT (FK)	Referencia a bodega

Table 3: Estructura de la tabla ingreso

Las relaciones están definidas con FOREIGN KEY y ON DELETE CASCADE, lo que significa que si se elimina un proveedor o bodega, automáticamente se eliminan los ingresos asociados.

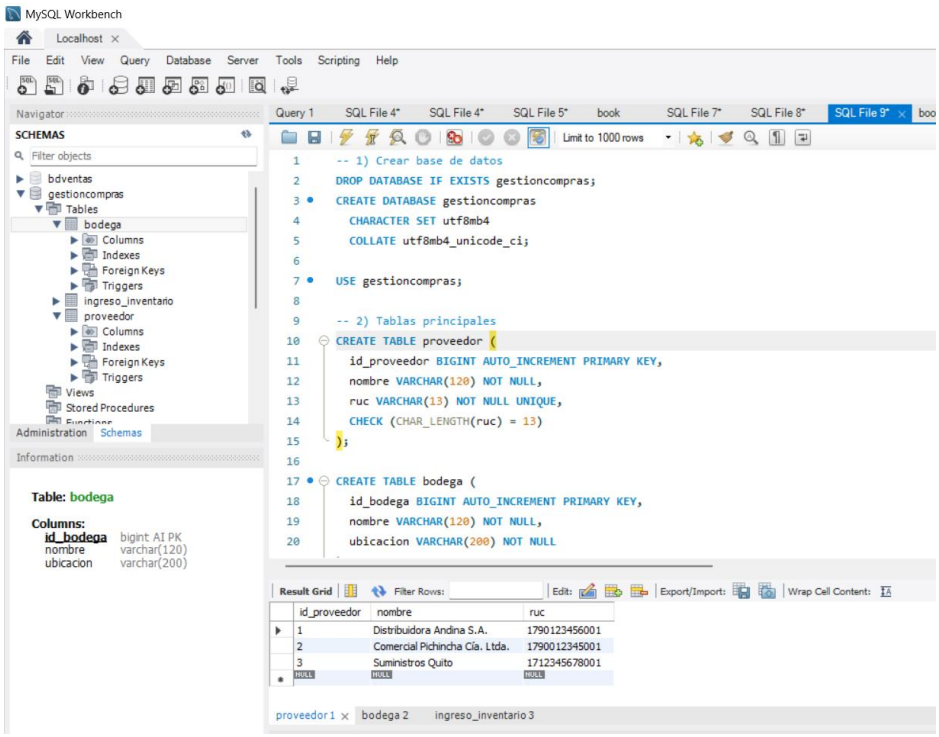


Imagen 1: Script SQL completo de creación de la base de datos, mostrando las tres tablas con sus campos, tipos de datos, claves primarias y foráneas

4.2 Estructura del Proyecto Maven

El proyecto sigue la estructura estándar de Maven para aplicaciones web Java EE:

- **src/main/java:** Contiene todo el código Java (paquetes modelo, dao, controlador)

- **src/main/webapp:** Contiene los recursos web (JSP, CSS, archivos de configuración)
- **src/main/webapp/WEB-INF:** Páginas JSP protegidas (no accesibles directamente por URL)
- **src/main/webapp/assets:** Archivos estáticos como CSS e imágenes
- **pom.xml:** Archivo de configuración Maven con las dependencias del proyecto

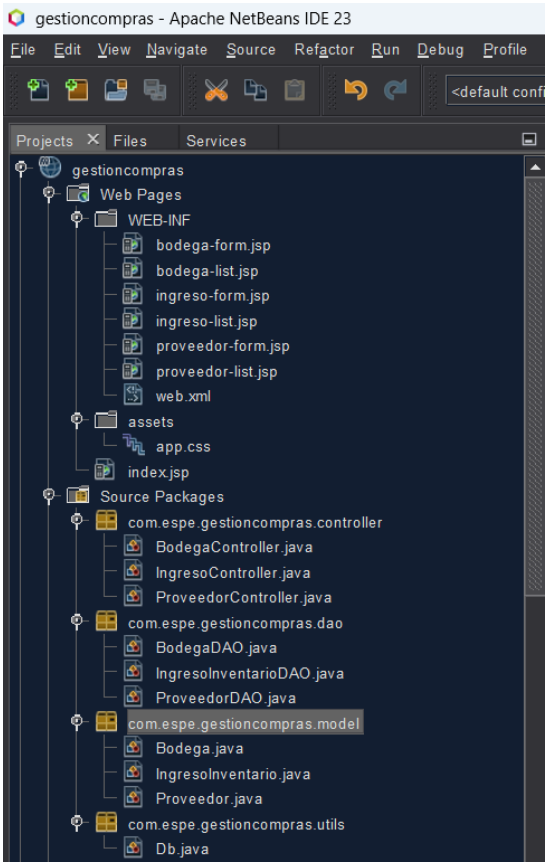


Imagen 2: Estructura completa del proyecto en NetBeans, mostrando todos los paquetes y las carpetas webapp, WEB-INF, assets

4.3 Configuración del Proyecto (pom.xml)

El archivo pom.xml define las dependencias necesarias para el proyecto. Las principales son:

- **Jakarta Servlet API:** Para crear Servlets que actúen como controladores
- **Jakarta JSP API:** Para renderizar páginas JSP
- **Jakarta JSTL:** Para usar etiquetas JSTL en las vistas
- **MySQL Connector:** Driver JDBC para conectar con MySQL

También se configura el compilador Java para usar la versión 17 y se especifica que es una aplicación web empaquetada como WAR.

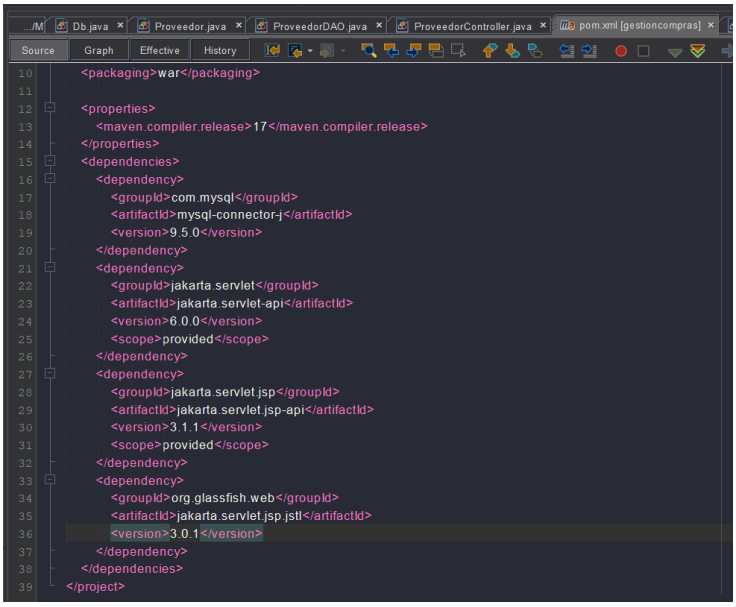


Imagen 3: Archivo pom.xml completo.

4.4 Capa de Modelo

Las clases de entidad que representan cada tabla de la base de datos. Cada clase tiene:

- Atributos privados correspondientes a las columnas de la tabla
- Constructor vacío (necesario para algunos frameworks)
- Constructor con parámetros para inicializar objetos fácilmente
- Métodos getter y setter para acceder a los atributos

Proveedor.java: Tiene `idProveedor`, `nombre` y `ruc`. Los getters y setters permiten leer y modificar estos valores desde otras capas.

Bodega.java: Tiene `idBodega`, `nombre` y `ubicacion`.

Ingreso.java: Además de `idIngreso`, `fecha`, `cantidad`, `idProveedor` e `idBodega`, también tiene atributos adicionales como `proveedorNombre` y `bodegaNombre` para mostrar información completa en las vistas sin hacer consultas adicionales.

Imagen 4: Clase Proveedor.java completa.

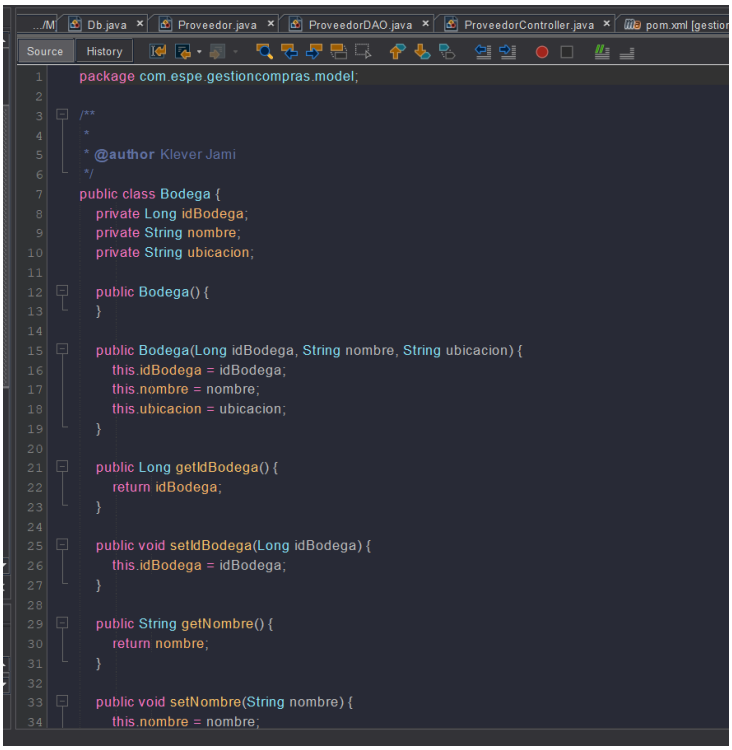
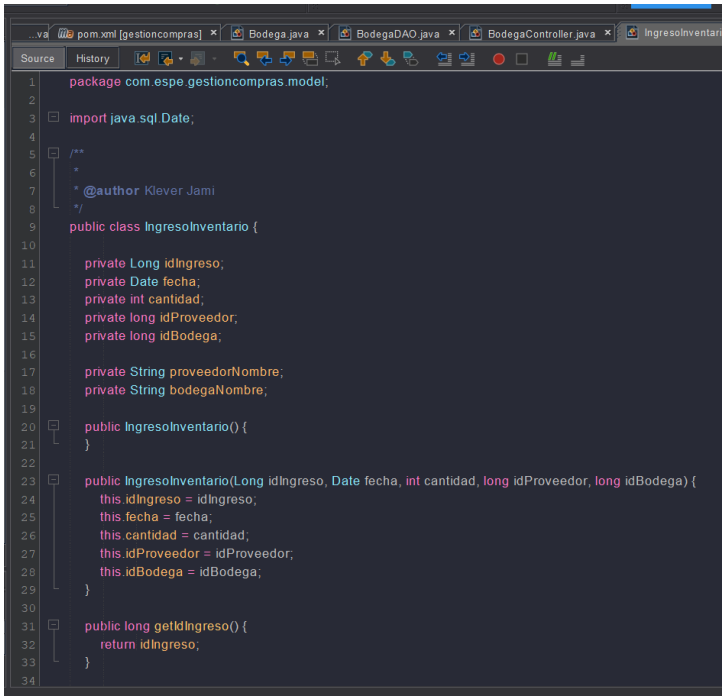


Imagen 5: Clase Bodega.java completa



```java
package com.espe.gestioncompras.model;

import java.sql.Date;

/**
 *
 * @author Klever Jami
 */
public class IngresoInventario {

    private Long idIngreso;
    private Date fecha;
    private int cantidad;
    private long idProveedor;
    private long idBodega;

    private String proveedorNombre;
    private String bodegaNombre;

    public IngresoInventario() {
    }

    public IngresoInventario(Long idIngreso, Date fecha, int cantidad, long idProveedor, long idBodega) {
        this.idIngreso = idIngreso;
        this.fecha = fecha;
        this.cantidad = cantidad;
        this.idProveedor = idProveedor;
        this.idBodega = idBodega;
    }

    public long getIdIngreso() {
        return idIngreso;
    }
```

Imagen 6: Clase Ingreso.java completa.

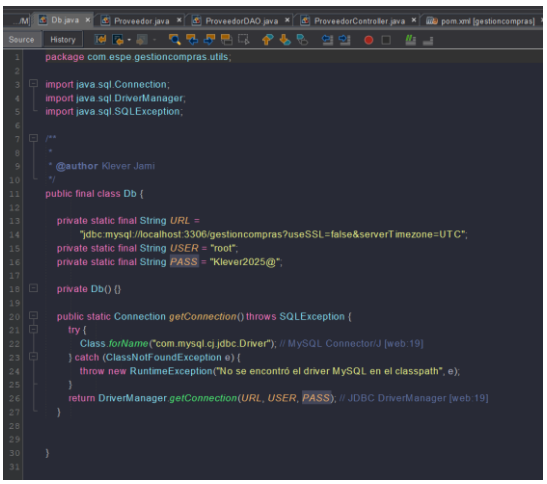
4.5 Conexión a Base de Datos

Esta clase es fundamental porque centraliza la lógica de conexión a MySQL. Utiliza el patrón Singleton para asegurar que solo exista una instancia de conexión en toda la aplicación, evitando múltiples conexiones innecesarias.

La clase tiene:

- Constantes con la URL de conexión, usuario y contraseña de MySQL
- Un método estático `getConnection()` que crea y devuelve una conexión usando DriverManager
- Manejo de excepciones para capturar errores de conexión

Todas las clases DAO llaman a `DatabaseConnection.getConnection()` para obtener la conexión antes de ejecutar sus consultas SQL.



```java
package com.espe.gestioncompras.utils;

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

/**
 *
 * @author Klever Jami
 */
public final class Db {

    private static final String URL =
            "jdbc:mysql://localhost:3306/gestioncompras?useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";
    private static final String PASS = "Klever2025@";

    private Db() {}

    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); // MySQL Connector/J [web:19]
        } catch (ClassNotFoundException e) {
            throw new RuntimeException("No se encontró el driver MySQL en el classpath", e);
        }
        return DriverManager.getConnection(URL, USER, PASS); // JDBC DriverManager [web:19]
    }

}
```

Imagen 7: Clase Data base Connection

4.6 Capa DAO

Las clases DAO encapsulan todas las operaciones de base de datos. Cada entidad tiene su propia clase DAO con métodos para las operaciones CRUD:

ProveedorDAO.java

- listar(): Ejecuta SELECT * FROM proveedor y devuelve una lista de objetos Proveedor
- obtenerPorId(int id): Busca un proveedor específico por su ID
- insertar(Proveedor p): Ejecuta INSERT INTO proveedor con los datos del objeto
- actualizar(Proveedor p): Ejecuta UPDATE proveedor SET ... para modificar un registro existente
- eliminar(int id): Ejecuta DELETE FROM proveedor WHERE idProveedor = ?

BodegaDAO.java

Tiene métodos similares adaptados a la tabla bodega: listar(), obtenerPorId(), insertar(), actualizar(), eliminar().

IngresoDAO.java

Además de los métodos CRUD básicos, el método listar() es más complejo porque ejecuta un SELECT con JOIN para obtener los nombres del proveedor y bodega asociados a cada ingreso. Esto permite mostrar información completa en la vista sin hacer consultas adicionales.

Todas las consultas usan PreparedStatement con parámetros ? para prevenir inyecciones SQL. Los resultados se procesan con ResultSet, recorriendo cada fila y creando objetos Java correspondientes.

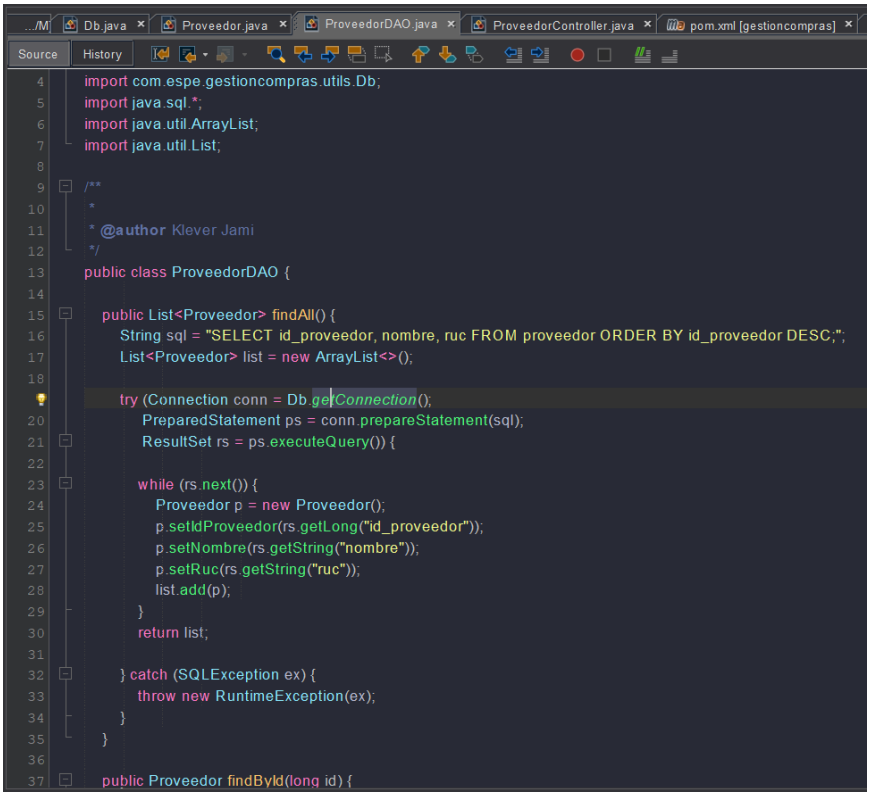


Imagen 8: ProveedorDAO

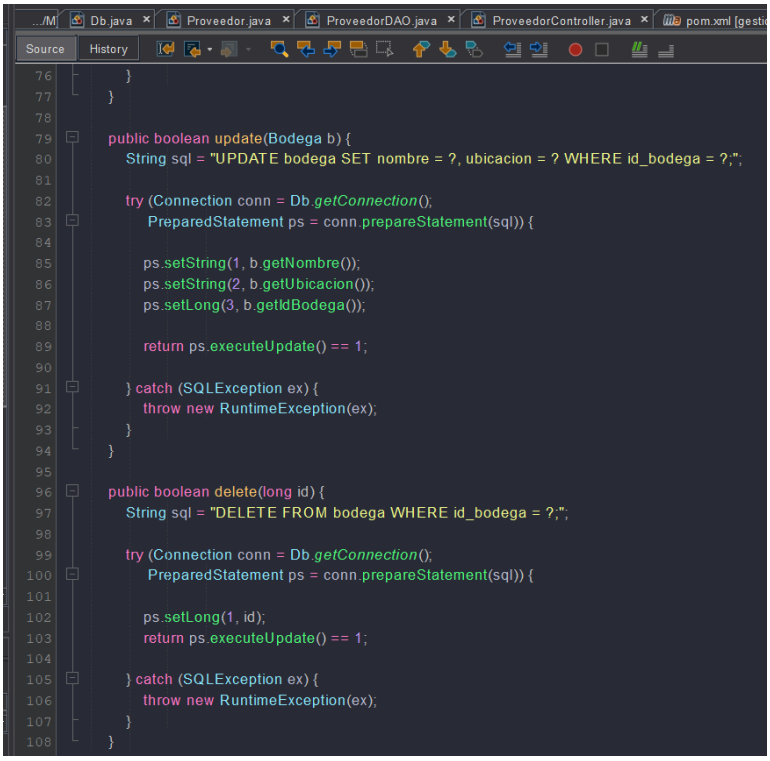


Imagen 9: BodegaDAO

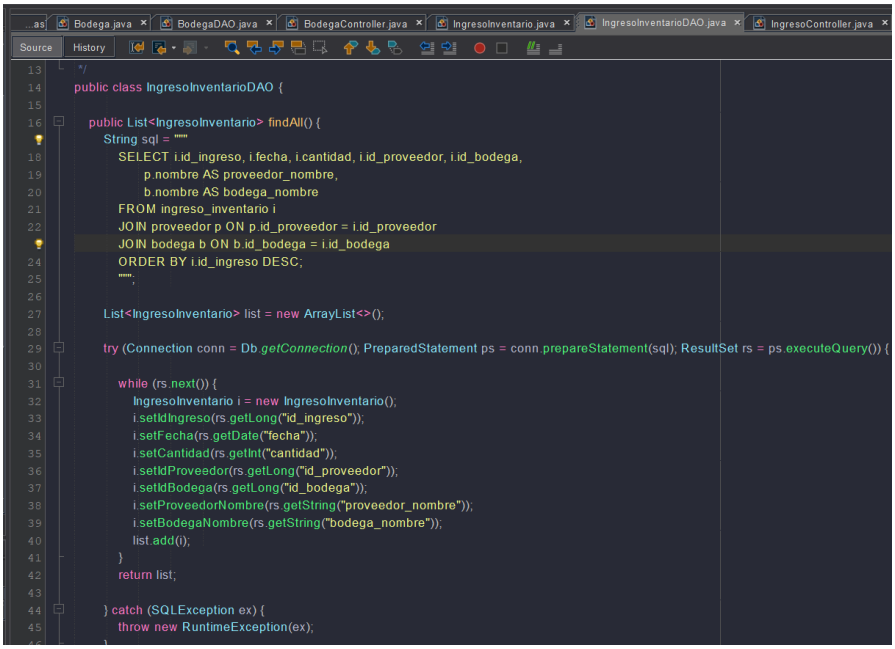


Imagen 10: IngresoDAO método listar() con JOIN para obtener nombres de proveedor y bodega

4.7 Capa de Controlador - Servlets

Los Servlets son el cerebro de la aplicación. Reciben las peticiones HTTP del navegador, determinan qué operación realizar según los parámetros recibidos, invocan los métodos del DAO necesarios, y redirigen a la vista apropiada.

Cada entidad tiene su propio Servlet:

Proveedor

Está mapeado a la URL /proveedores mediante la anotación @WebServlet. El método doGet() maneja las peticiones GET (listar, mostrar formulario de nuevo, editar, eliminar) y el método doPost() maneja los POST (guardar datos del formulario).

Según el parámetro action recibido, el servlet decide qué hacer:

- Sin action o action=list: Llama a dao.listar(), guarda la lista en el request con setAttribute(), y forward a proveedor-list.jsp
- action=new: Forward directo a proveedor-form.jsp con formulario vacío
- action=edit: Obtiene el id del parámetro, llama a dao.obtenerPorId(), guarda el proveedor en el request, y forward a proveedor-form.jsp con datos precargados
- action=delete: Llama a dao.eliminar() y redirige a la lista
- action=insert (POST): Lee los datos del formulario, crea objeto Proveedor, llama a dao.insertar(), redirige a la lista
- action=update (POST): Similar a insert pero llama a dao.actualizar()

Bodega e Ingreso

Siguen la misma lógica que Proveedor, adaptada a sus respectivas entidades. IngresoServlet tiene lógica adicional: antes de mostrar el formulario (new o edit), carga las listas de proveedores y bodegas disponibles para poblar los select del formulario.

4.8 Capa de Vista

Las páginas JSP son la interfaz que ve el usuario. Utilizan JSTL para mostrar datos dinámicos y mantener el código limpio.

Estructura común de todas las JSP:

- Directivas `page` y `taglib` al inicio para configurar encoding y habilitar JSTL
- Declaración de variables con `c:url` para construir URLs dinámicas (adaptadas al context path)
- Sección topbar con título y menú de navegación
- Contenido principal dentro de un contenedor con clase CSS
- Enlace a la hoja de estilos `assets/app.css` usando `c:url`

index.jsp

Es la página principal que se carga al acceder a la aplicación. Muestra un panel de bienvenida con tres tarjetas (tiles) que enlazan a cada módulo (Proveedores, Bodegas, Ingresos). Cada tarjeta tiene un título, descripción breve e ícono representativo.

proveedor-list.jsp

Muestra una tabla con todos los proveedores registrados. Utiliza `c:choose` y `c:when` para verificar si la lista está vacía y mostrar un mensaje, o iterar con `c:forEach` sobre la lista de proveedores. Cada fila de la tabla tiene botones "Editar" y "Eliminar" que construyen URLs con `c:url` incluyendo el action y el id correspondiente.

proveedor-form.jsp

Formulario para crear o editar un proveedor. Usa una variable `isUpdate` (calculada con `c:set`) para determinar si está en modo creación o edición. Si es edición, precarga los valores en los inputs con `value="${proveedor.nombre}"` y muestra el ID en un pill informativo. El formulario se envía por POST al servlet con el action correspondiente (insert o update).

bodega-list.jsp y bodega-form.jsp

Exactamente la misma estructura que las páginas de proveedor, adaptadas a los campos de bodega (nombre y ubicación).

ingreso-list.jsp

Similar a las otras listas, pero la tabla tiene más columnas (fecha, cantidad, proveedor, bodega). Los nombres de proveedor y bodega vienen ya cargados desde el DAO gracias al JOIN en la consulta SQL.

ingreso-form.jsp

Formulario más complejo porque tiene campos de fecha, cantidad (input type="number"), y dos selects dinámicos. Los selects se pueblan iterando con c:forEach sobre las listas de proveedores y bodegas cargadas por el servlet. En modo edición, usa c:if para marcar como selected la opción que corresponde al proveedor/bodega actual del ingreso.

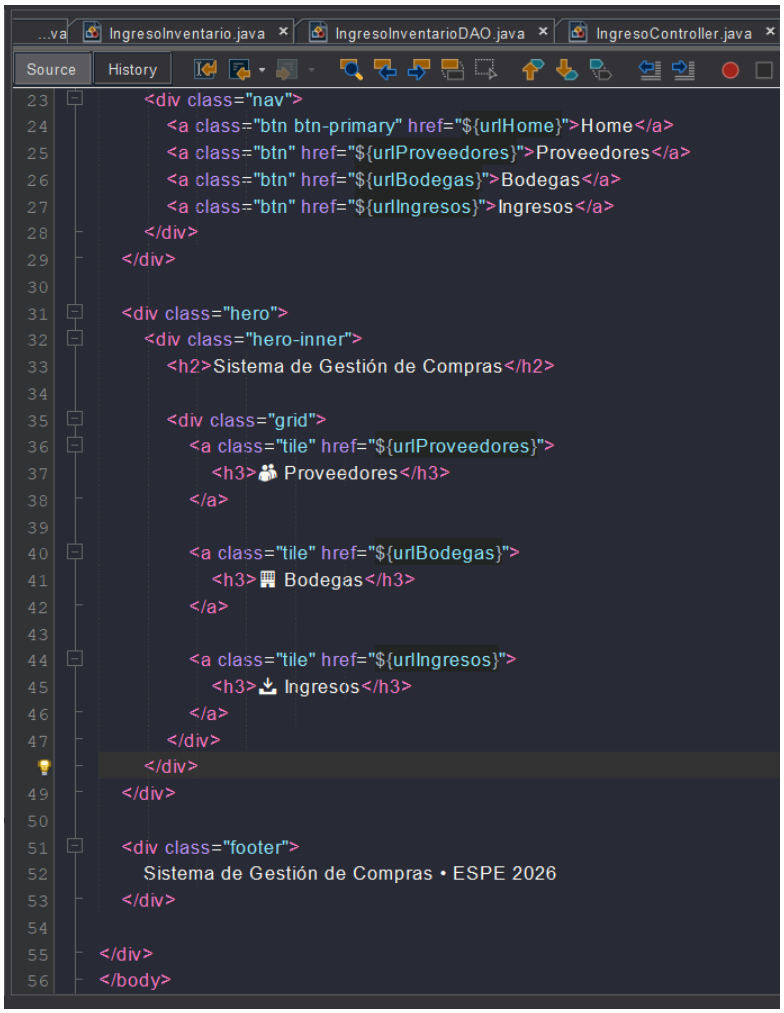


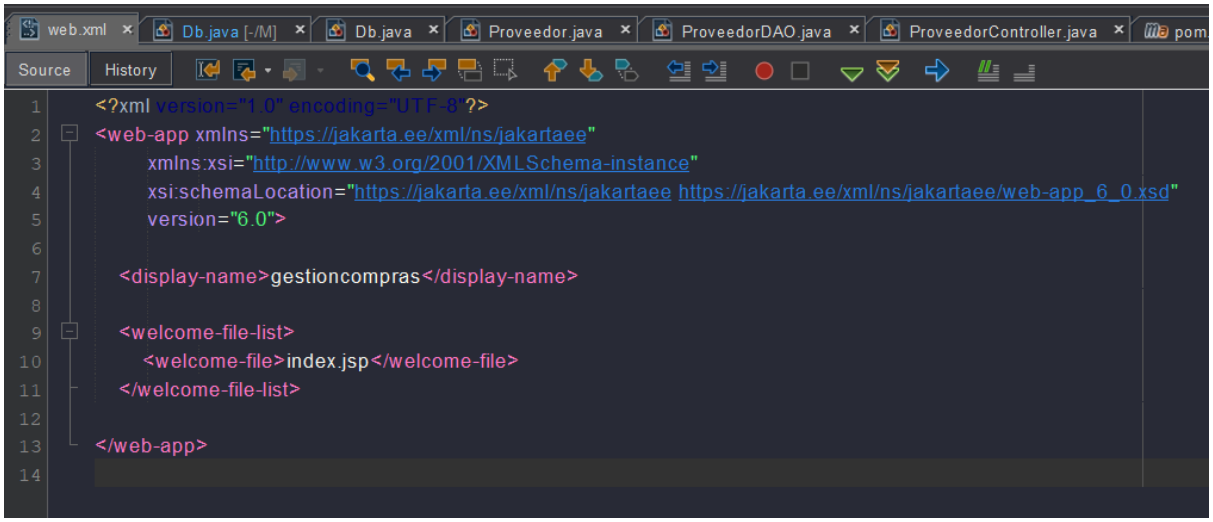
Imagen 11: Código de index.jsp estructura del hero.

4.9 Configuración Web

El archivo web.xml es el descriptor de despliegue de la aplicación. Define configuraciones globales como:

- welcome-file-list: Especifica que index.jsp es la página de inicio cuando se accede a la raíz de la aplicación

En aplicaciones modernas con Servlet 3.0+, muchas configuraciones se pueden hacer con anotaciones (como @WebServlet), pero el welcome-file-list sigue siendo necesario en web.xml.



Iamgen 12: Archivo web.xml

4.10 Diseño Visual

El archivo CSS contiene todos los estilos de la aplicación, implementando un diseño moderno con:

- Variables CSS en :root para colores, sombras y espaciados (facilita cambios globales)
- Fondo degradado sutil con radial-gradient para dar profundidad
- Topbar con backdrop-filter: blur() para efecto de vidrio esmerilado
- Botones con estados hover y active bien definidos
- Cards con sombras suaves y bordes redondeados
- Tablas con filas alternadas (:nth-child(even)) y hover effect
- Formularios con inputs que tienen focus ring azul para accesibilidad
- Grid responsivo para la página de inicio que se adapta a móviles
- Clases utilitarias para flexbox, espaciado, alineación

El diseño es completamente responsivo gracias a media queries que adaptan el layout en pantallas pequeñas.

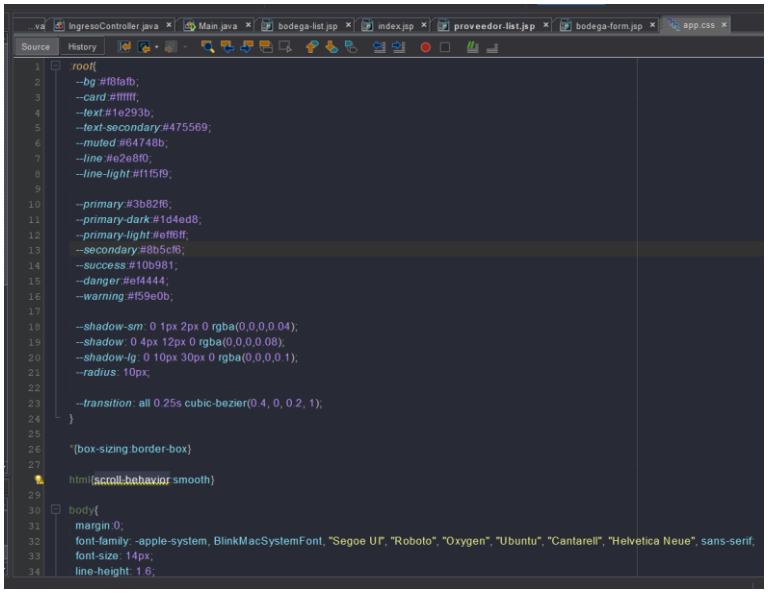


Imagen 13: Sección de variables CSS (:root) mostrando los colores y valores definidos

5. Flujo de Operación del Sistema

5.1 Caso de Uso:

1. El usuario accede a la URL `/Home`
2. El navegador envía una petición GET al servidor GlassFish
3. GlassFish invoca el método `doGet()` de ProveedorServlet
4. El servlet detecta que no hay parámetro action (o action=list)
5. Crea una instancia de ProveedorDAO y llama al método `listar()`
6. ProveedorDAO se conecta a MySQL, ejecuta `SELECT * FROM proveedor`
7. Los resultados se convierten en objetos Proveedor y se devuelven en una List
8. El servlet guarda la lista en el request con `setAttribute("proveedores", lista)`
9. Hace forward a `proveedor-list.jsp`
10. La JSP itera sobre la lista con `c:forEach` y genera la tabla HTML
11. El HTML final se envía al navegador y el usuario ve la lista de proveedores

5.2 Caso de Uso: Crear Nuevo Proveedor

1. El usuario hace clic en el botón "Nuevo" en la lista de proveedores
2. Se accede a `/proveedores?action=new`
3. ProveedorServlet detecta `action=new` en `doGet()`
4. Establece `formAction = "insert"` y hace forward a `proveedor-form.jsp`
5. La JSP muestra un formulario vacío (modo creación)
6. El usuario completa los campos (nombre, RUC) y envía el formulario

7. El navegador hace POST a /proveedores
8. ProveedorServlet recibe la petición en doPost()
9. Lee los parámetros del formulario con request.getParameter()
10. Crea un objeto Proveedor con esos datos
11. Llama a dao.insertar(proveedor)
12. ProveedorDAO ejecuta INSERT INTO proveedor en MySQL
13. El servlet redirige con response.sendRedirect() a /proveedores (lista)
14. El usuario ve la lista actualizada con el nuevo proveedor

5.3 Caso de Uso: Editar Proveedor

1. El usuario hace clic en "Editar" en la fila de un proveedor
2. Se accede a /proveedores?action=edit&id=X
3. ProveedorServlet detecta action=edit en doGet()
4. Obtiene el id del parámetro y llama a dao.obtenerPorId(id)
5. ProveedorDAO ejecuta SELECT * FROM proveedor WHERE idProveedor = ?
6. El proveedor obtenido se guarda en el request con setAttribute("proveedor", p)
7. Establece formAction = "update" y hace forward a proveedor-form.jsp
8. La JSP muestra el formulario con los campos precargados (modo edición)
9. El usuario modifica los campos y envía el formulario
10. POST a /proveedores con action=update e idProveedor oculto
11. ProveedorServlet en doPost() lee todos los parámetros incluyendo el id
12. Crea objeto Proveedor con los datos actualizados y llama a dao.actualizar(proveedor)
13. ProveedorDAO ejecuta UPDATE proveedor SET ... WHERE idProveedor = ?
14. Redirige a la lista y el usuario ve los cambios reflejados

5.4 Caso de Uso: Eliminar Proveedor

1. El usuario hace clic en "Eliminar" en la fila de un proveedor
2. JavaScript muestra un confirm() preguntando si está seguro
3. Si confirma, se accede a /proveedores?action=delete&id=X
4. ProveedorServlet detecta action=delete en doGet()
5. Llama a dao.eliminar(id)
6. ProveedorDAO ejecuta DELETE FROM proveedor WHERE idProveedor = ?
7. Si había ingresos asociados, se eliminan automáticamente por ON DELETE CASCADE

8. El servlet redirige a la lista
9. El usuario ve la lista sin el proveedor eliminado

5.5 Caso de Uso: Crear Ingreso

Este caso es más complejo porque involucra relaciones con otras tablas:

1. El usuario hace clic en "Nuevo" en la lista de ingresos
2. Se accede a `/ingresos?action=new`
3. IngresoServlet detecta `action=new` en `doGet()`
4. Antes de mostrar el formulario, carga las listas de proveedores y bodegas disponibles llamando a sus respectivos DAOs
5. Guarda ambas listas en el request con `setAttribute()`
6. Hace forward a `ingreso-form.jsp`
7. La JSP genera dos selects poblados con las listas usando `c:forEach`
8. El usuario completa fecha, cantidad, selecciona proveedor y bodega
9. POST a `/ingresos` con todos los campos
10. IngresoServlet en `doPost()` lee los parámetros, convierte la fecha de String a Date
11. Crea objeto Ingreso con todos los datos y llama a `dao.insertar(ingreso)`
12. IngresoDAO ejecuta `INSERT INTO ingreso` con los IDs del proveedor y bodega
13. Redirige a la lista de ingresos
14. La lista muestra el nuevo ingreso con los nombres completos de proveedor y bodega (gracias al JOIN)

6. Resultados y Capturas de Pantalla

6.1 Página de Inicio (Home)

La página principal presenta un diseño limpio con fondo degradado sutil y tres tarjetas que funcionan como accesos directos a cada módulo del sistema.

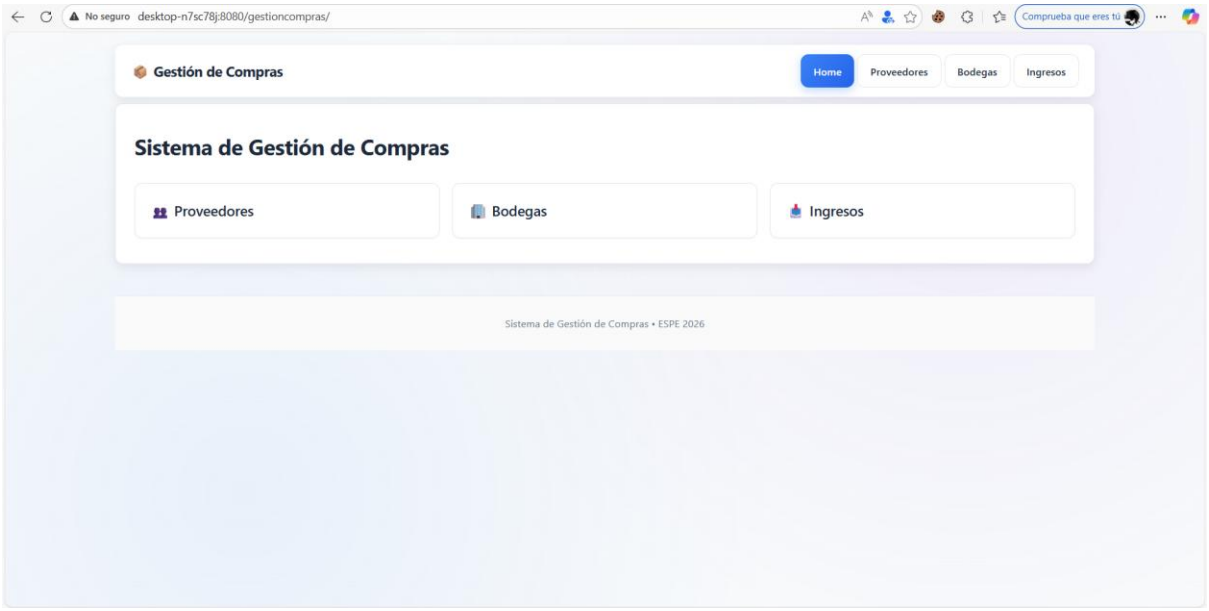


Imagen 14: página de inicio

6.2 Módulo de Proveedores

Lista de Proveedores

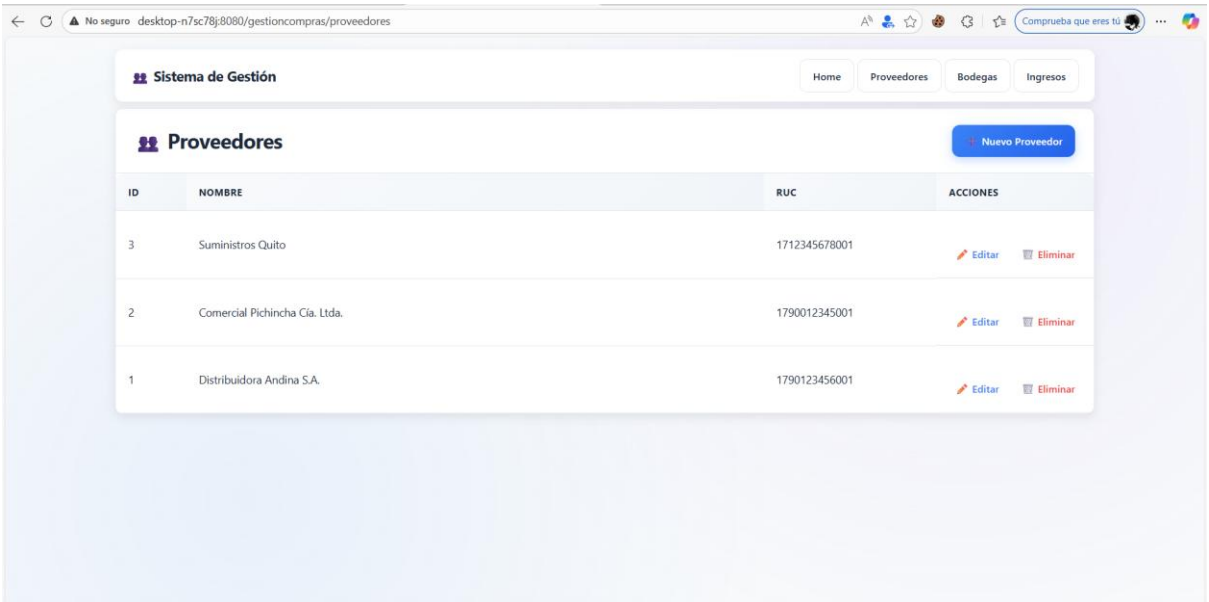


Imagen 15: Página de lista de proveedores mostrando la tabla con columnas ID, Nombre, RUC y Acciones.

Formulario Nuevo Proveedor

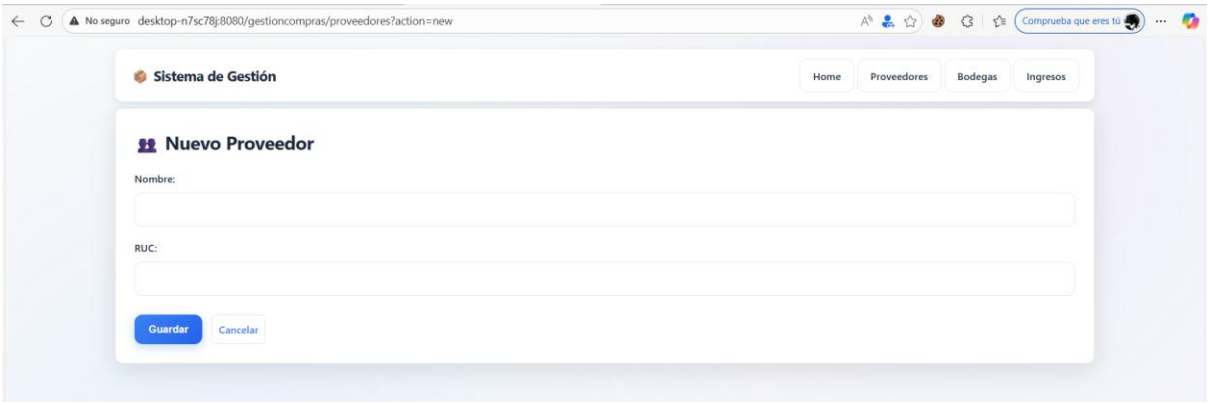


Imagen 16: Formulario de nuevo proveedor vacío, mostrando los campos "Nombre" y "RUC" sin datos, el título "Nuevo Proveedor", y los botones "Guardar" y "Cancelar"

Formulario Editar Proveedor

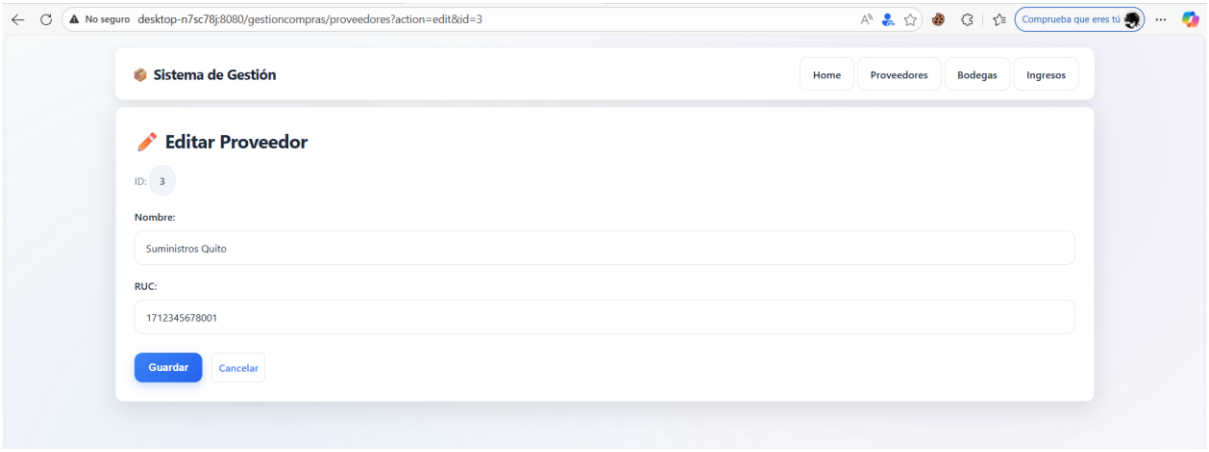


Imagen 17: Formulario de edición de proveedor con datos precargados, mostrando el título "Editar Proveedor", el ID en un pill gris, los campos con valores existentes y los botones de acción

6.3 Módulo de Bodegas

Lista de Bodegas

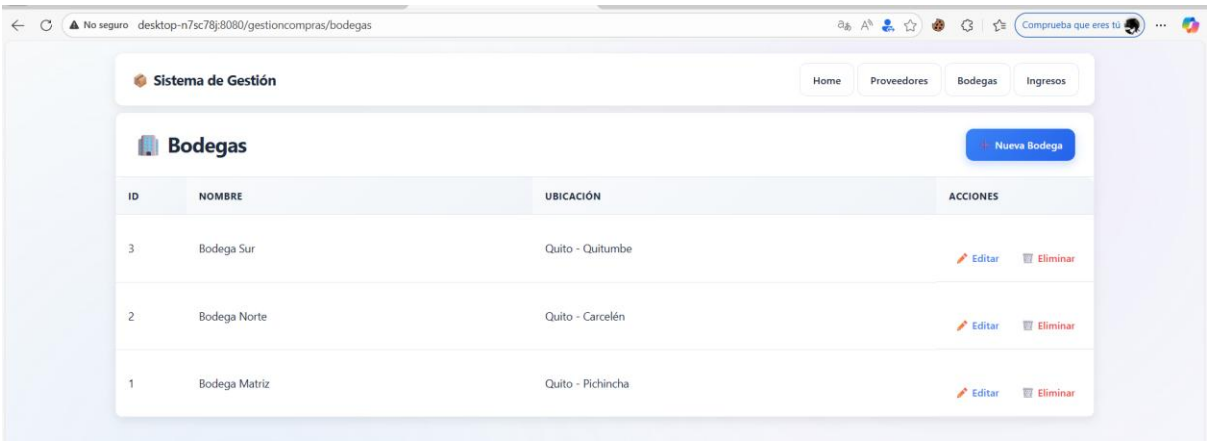


Imagen 18: Lista de bodegas mostrando la tabla con columnas ID, Nombre, Ubicación y Acciones.

Formulario Nuevo Bodega

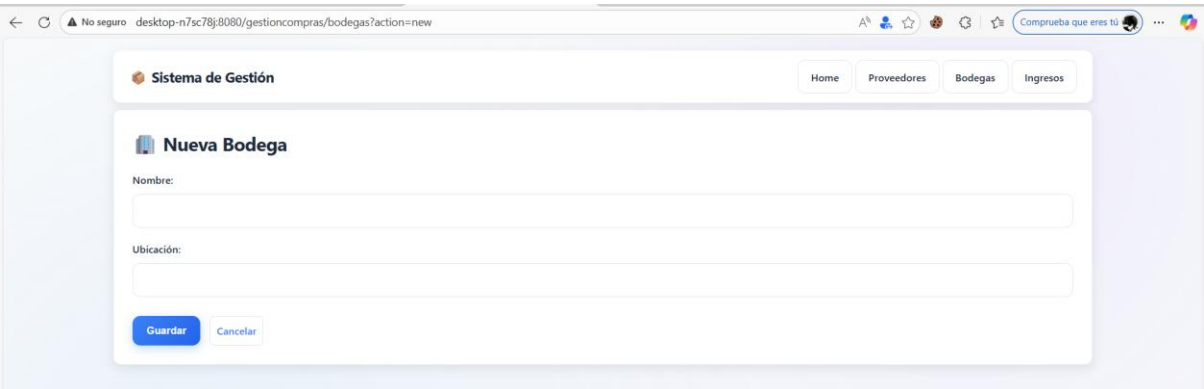


Imagen 19: Formulario de nueva bodega vacío, con campos "Nombre" y "Ubicación"

Formulario Editar Bodega

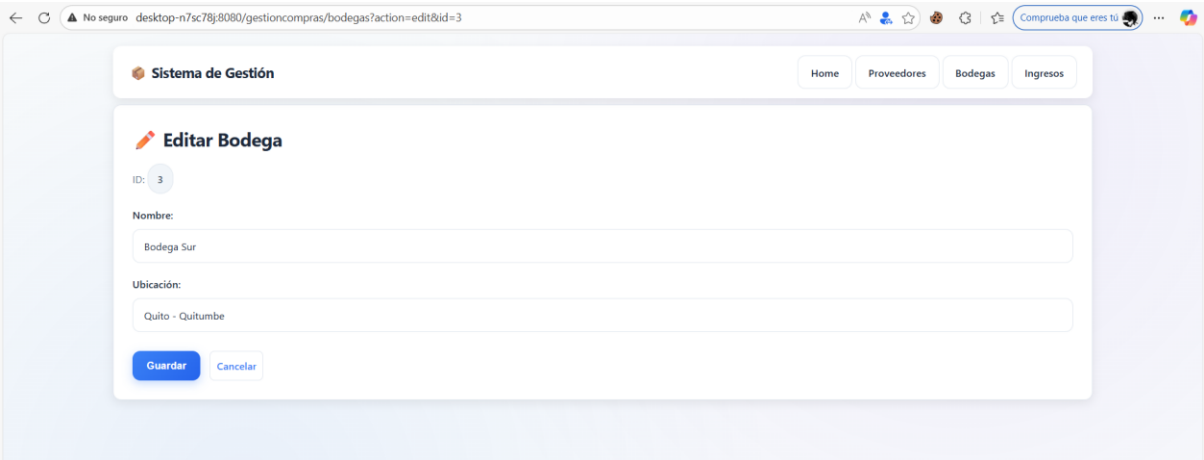


Imagen 20: Formulario de edición de bodega con datos precargados

6.4 Módulo de Ingresos

Lista de Ingresos

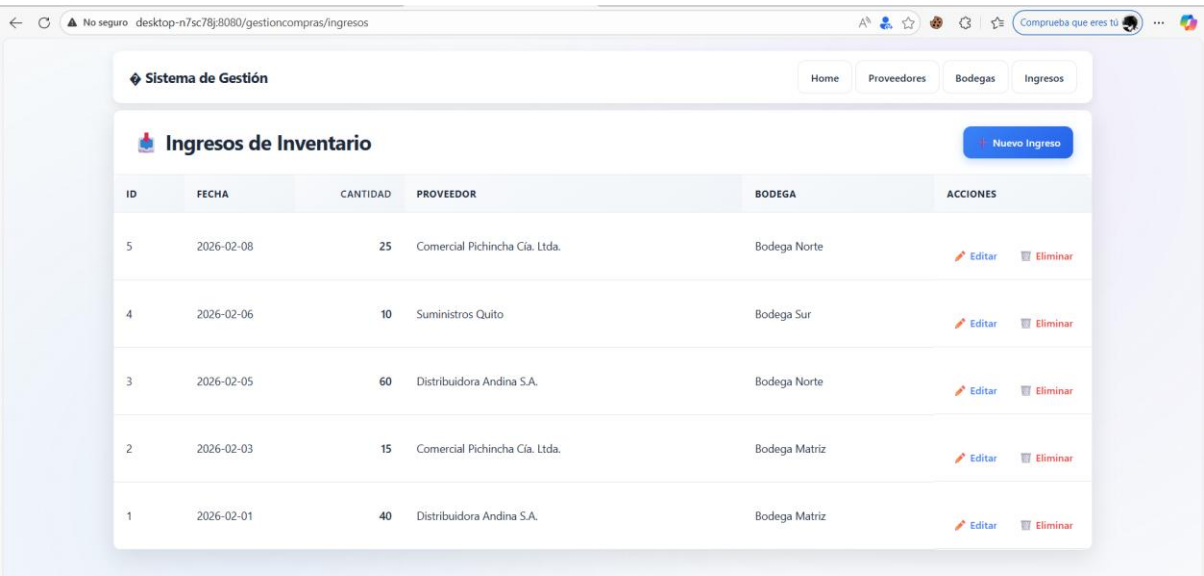


Imagen 21: Lista de ingresos mostrando la tabla con columnas ID, Fecha, Cantidad, Proveedor, Bodega y Acciones.

Formulario Nuevo Ingreso

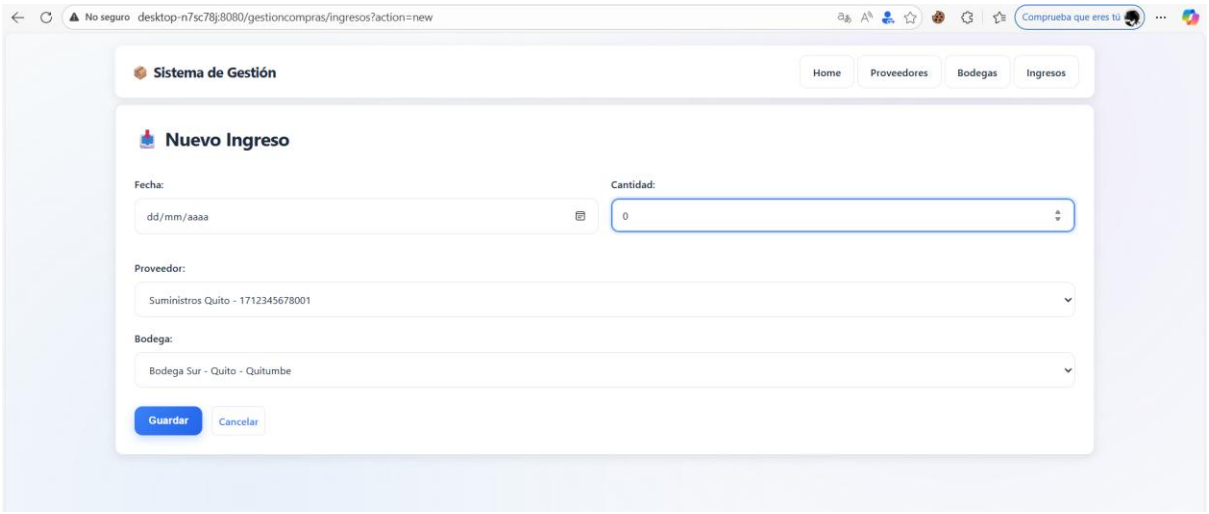


Imagen 22: Formulario de nuevo ingreso mostrando los campos de fecha, cantidad, y los dos selects desplegables (Proveedor y Bodega)

Formulario Editar Ingreso

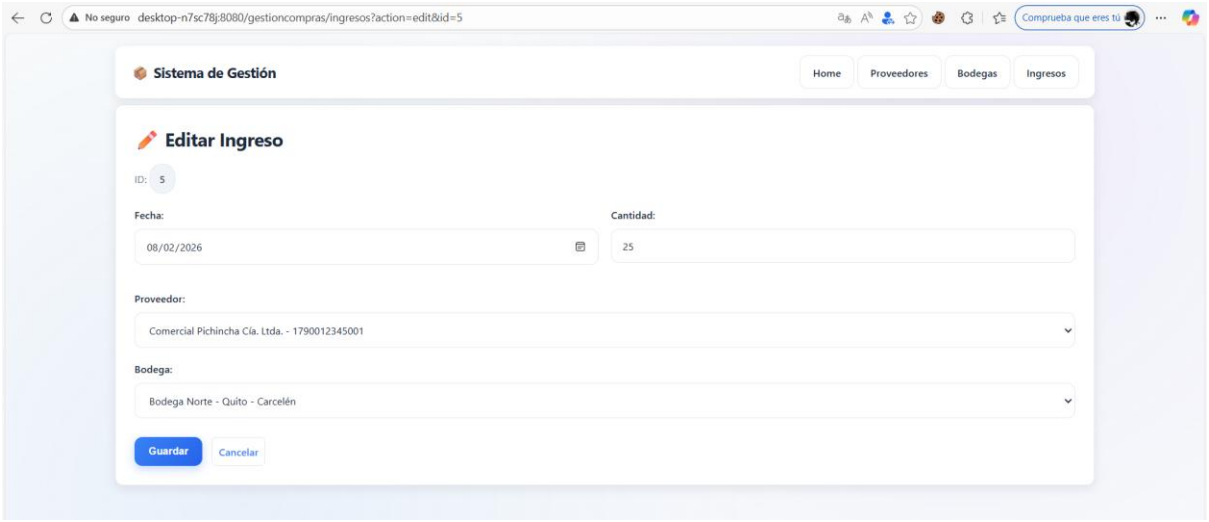


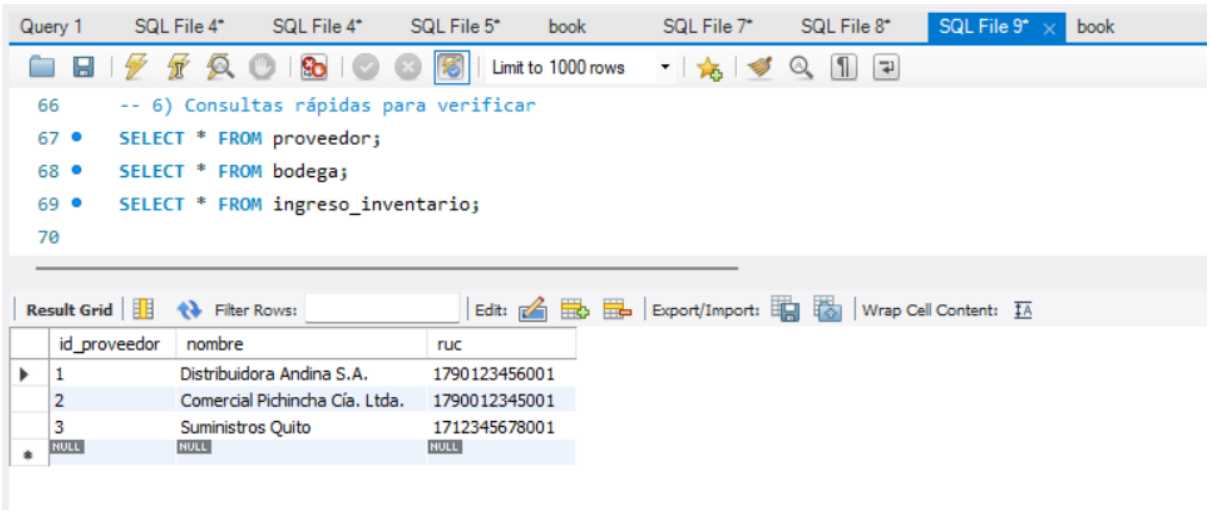
Imagen 23: Formulario de edición de ingreso con datos precargados.

6.5 Base de Datos MySQL

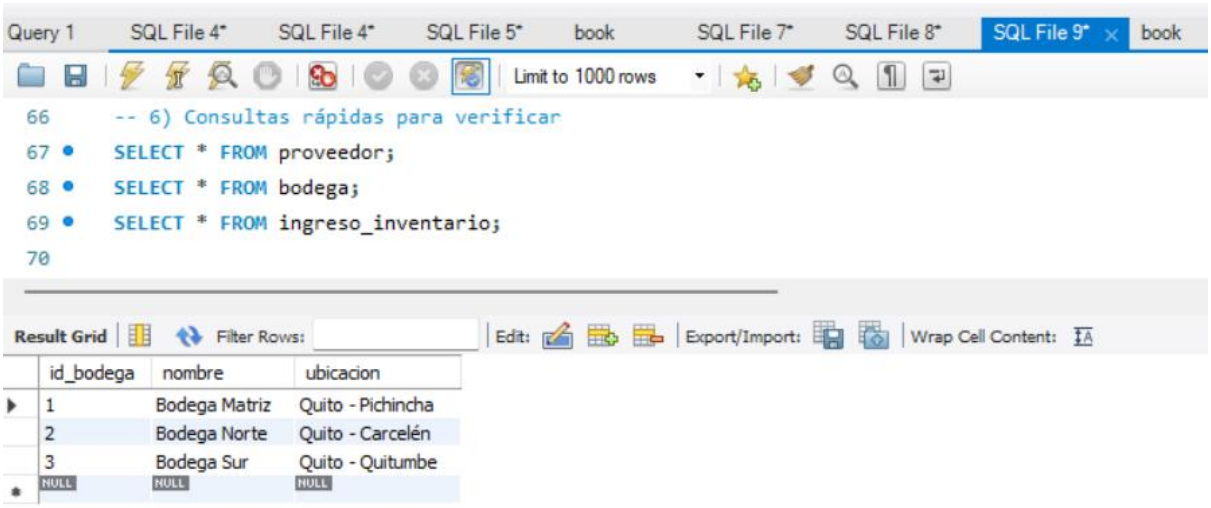
Tablas Creadas

Imagen 24: MySQL Workbench lista de las tres tablas (proveedor, bodega, ingreso) en la base de datos gestioncompras

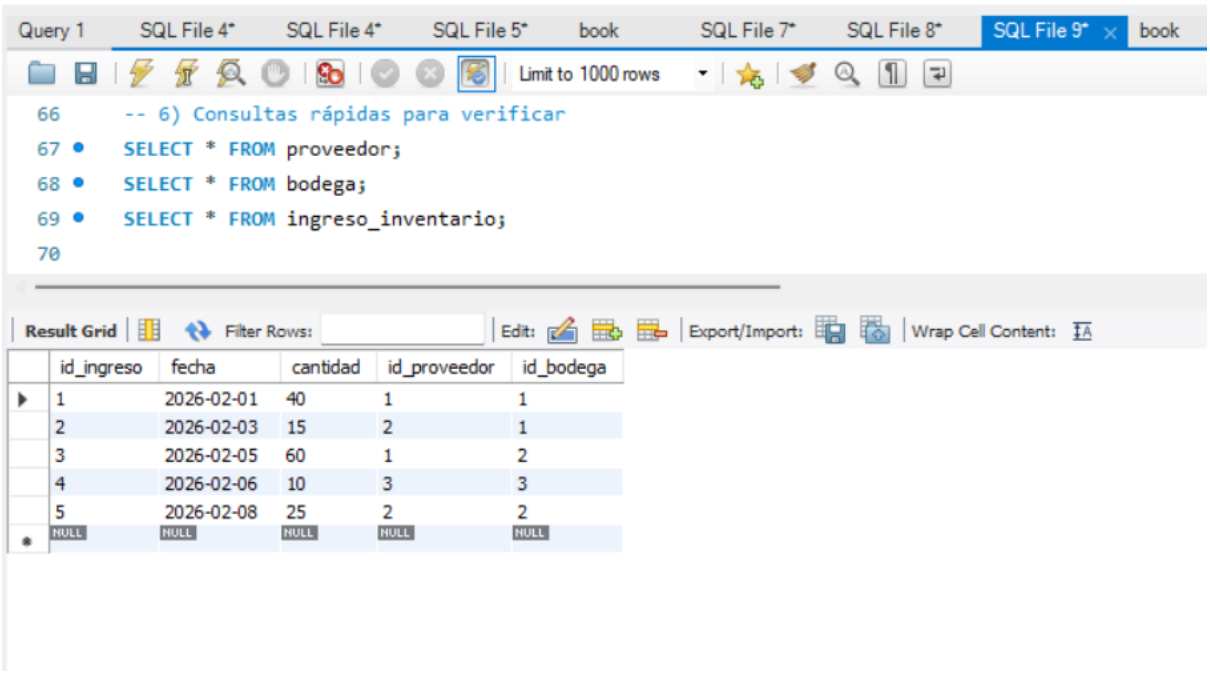
Datos en Tabla Proveedor



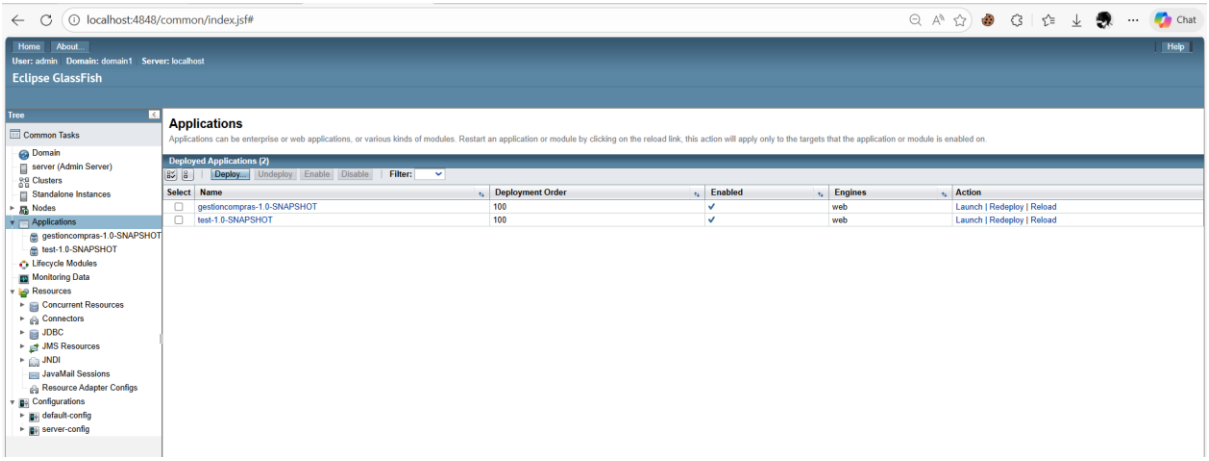
Datos en Tabla Bodega



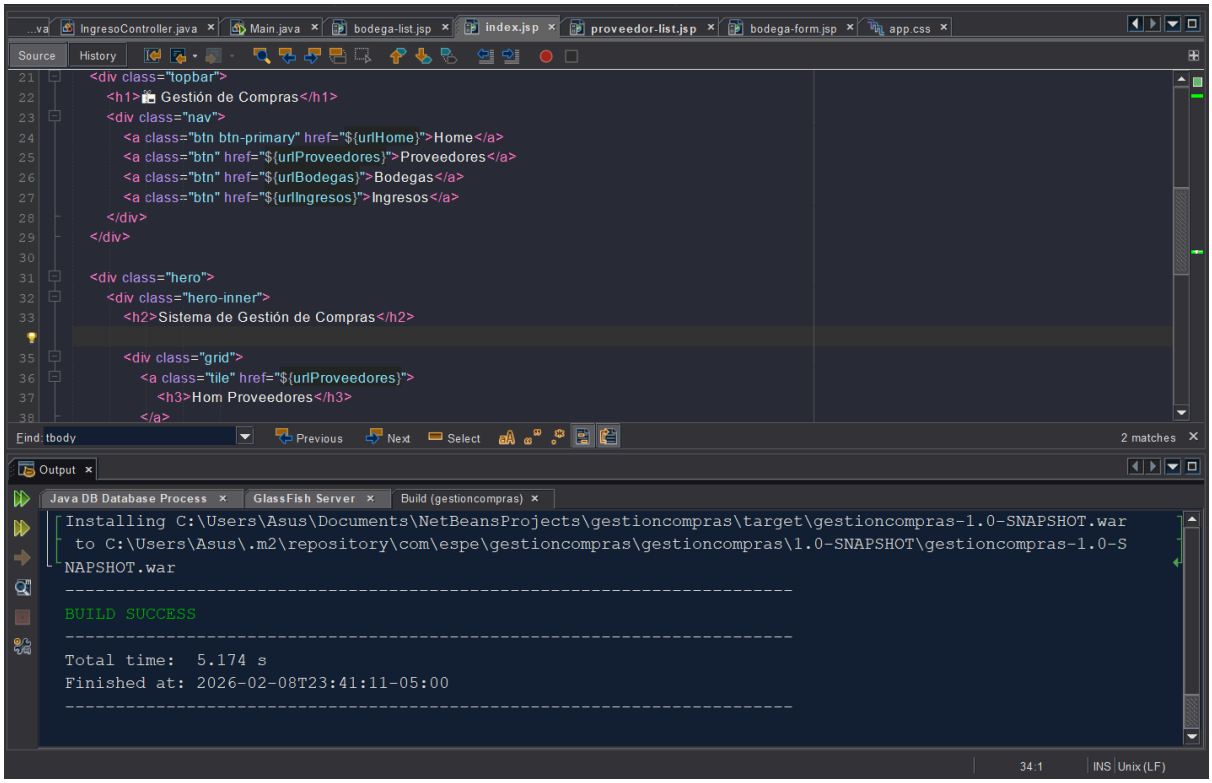
Datos en Tabla Ingreso con JOIN



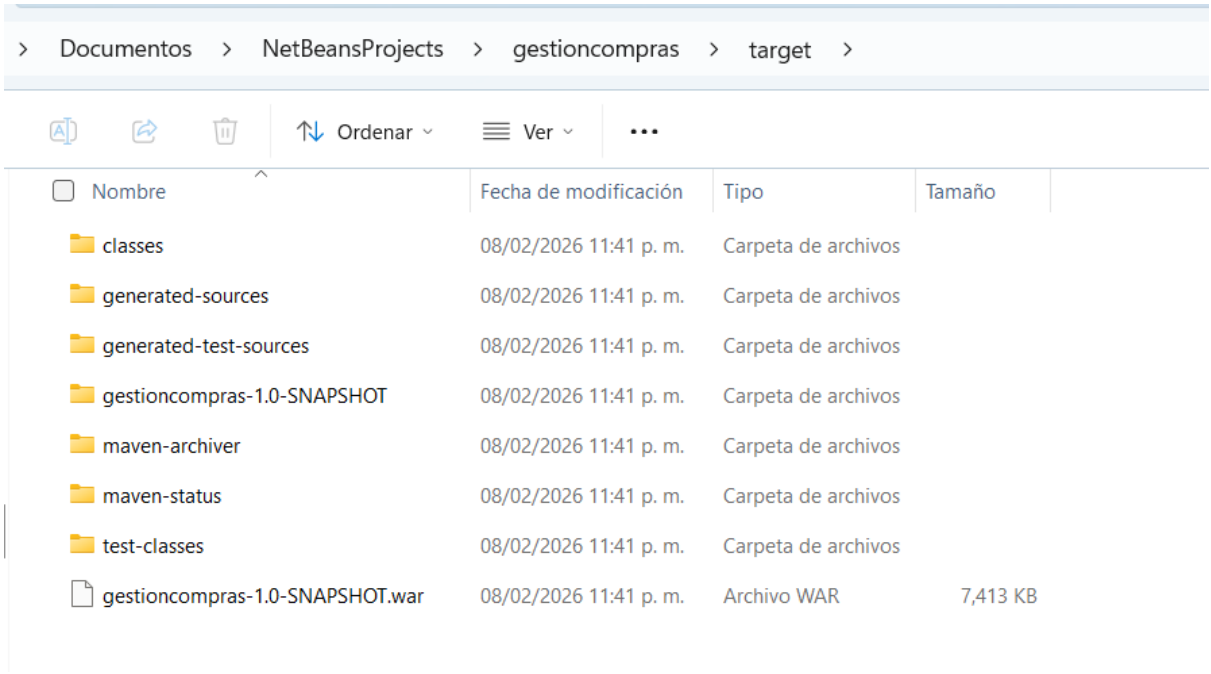
6.6 Despliegue en GlassFish



Captura de la consola de administración de GlassFish (http://localhost:4848) mostrando la aplicación gestioncompras-1.0-SNAPSHOT desplegada en la sección Applications



Captura del log de compilación de Maven mostrando "BUILD SUCCESS" y la generación del archivo WAR



7. Conclusiones

El proyecto logró implementar exitosamente un sistema web completo de gestión de compras utilizando arquitectura MVC. Se desarrollaron las tres capas de forma independiente pero coordinada: el Modelo maneja toda la lógica de acceso a datos mediante JDBC, la Vista presenta la información de forma clara con JSP y JSTL, y el Controlador orquesta el flujo de la aplicación mediante Servlets.

Las operaciones CRUD funcionan correctamente para las tres entidades (Proveedor, Bodega, Ingreso), incluyendo validaciones básicas y manejo de relaciones mediante llaves foráneas. La interfaz de usuario es moderna, consistente y fácil de usar gracias al diseño CSS personalizado.

Este proyecto demuestra la efectividad del patrón MVC y las tecnologías Java EE para desarrollar aplicaciones web empresariales robustas y mantenibles. El sistema resultante es funcional, tiene un diseño moderno y cumple con los estándares de la industria.

La experiencia ganada con este proyecto proporciona una base sólida para desarrollar aplicaciones más complejas en el futuro, y los conceptos aprendidos (MVC, JDBC, DAO, JSTL) son transferibles a otros frameworks y tecnologías modernas como Spring Boot, JPA/Hibernate, Thymeleaf, etc.