|  |  |
| --- | --- |
|  | |
| ASIGNATURA: | Programación Orientada a Objetos & Bases de datos |
| PROFESOR: | Mg. Yadira Franco Rocha |
| PERÍODO ACADÉMICO: | 2025 A |
|  | |

****

|  |
| --- |
|  |
| **TÍTULO**:  SIFood – Sistema de Restaurante  **ESTUDIANTE** |
| Juan Lucero |
|  |
|  |
|  |

Contenido

[SIFOOD: Diseño e Implementación de un Sistema de Gestión para Restaurantes con Java Swing y PostgreSQL 3](#_Toc204876272)

[*Palabras Clave—Java Swing, PostgreSQL, Sistema de Gestión, Base de Datos Relacional, Seguridad de Datos, Modelo Cliente-Servidor, pgcrypto.* 3](#_Toc204876273)

[I. INTRODUCCIÓN 3](#_Toc204876274)

[II. REQUERIMIENTOS DEL SISTEMA 4](#_Toc204876275)

[III. ARQUITECTURA DEL SISTEMA 5](#_Toc204876276)

[IV. DISEÑO Y MODELADO DE LA BASE DE DATOS 5](#_Toc204876277)

[V. IMPLEMENTACIÓN DE LÓGICA DE NEGOCIO 7](#_Toc204876278)

[VI. CARACTERÍSTICAS DE SEGURIDAD 7](#_Toc204876279)

[VII. PRUEBAS Y SIMULACIÓN DE PERFILES 7](#_Toc204876280)

[VIII. CONCLUSIONES 7](#_Toc204876281)

[Para el futuro, se tiene pensado añadir modalidad CRUD, que consta en modificar el funcionamiento del mesero, para que tenga la capacidad de cambiar el pedido una vez pase a cocina, eliminar un plato o cambiarlo. 8](#_Toc204876282)

[REFERENCIAS 8](#_Toc204876283)

# SIFOOD: Diseño e Implementación de un Sistema de Gestión para Restaurantes con Java Swing y PostgreSQL

**Juan Lucero**

*Escuela Politécnica Nacional*

*juanluceroandres@gmail.com*

***Resumen***—**Este documento presenta el diseño, desarrollo e implementación de SIFOOD, un sistema de escritorio para la gestión integral de restaurantes. La arquitectura del sistema se basa en un modelo cliente-servidor, utilizando Java Swing para la aplicación de escritorio y PostgreSQL como sistema de gestión de base de datos relacional desplegado en un entorno de nube. El proyecto aborda la necesidad de una solución robusta para la administración de menús, la toma de pedidos, el control visual de mesas y la generación de reportes, enfocándose en la seguridad, la integridad de los datos y la usabilidad. Se detallan el modelado de la base de datos bajo la Tercera Forma Normal (3FN), la implementación de lógica de negocio mediante procedimientos almacenados y triggers, y la aplicación de estrictas políticas de seguridad, incluyendo el hashing de credenciales con pgcrypto y la prevención de ataques de inyección SQL. Finalmente, se valida el sistema mediante la simulación de perfiles profesionales, demostrando el cumplimiento de los requerimientos funcionales y no funcionales establecidos.**

## *Palabras Clave—Java Swing, PostgreSQL, Sistema de Gestión, Base de Datos Relacional, Seguridad de Datos, Modelo Cliente-Servidor, pgcrypto.*

### I. INTRODUCCIÓN

La gestión eficiente de un restaurante requiere la coordinación precisa de múltiples procesos, desde la administración del inventario y el menú hasta la atención al cliente y la facturación. La falta de un sistema informático integrado puede llevar a ineficiencias operativas, errores en la toma de pedidos y una falta de visibilidad sobre el rendimiento del negocio. Para abordar esta problemática, se ha desarrollado SIFOOD, un sistema de escritorio diseñado para optimizar y centralizar las operaciones diarias de un restaurante.

El objetivo principal de este proyecto es desarrollar un sistema de gestión que permita automatizar el flujo de trabajo, desde que un mesero toma un pedido hasta que se genera la factura final. El sistema se ha construido sobre dos pilares tecnológicos: una aplicación cliente robusta desarrollada en Java con la biblioteca Swing para la interfaz gráfica, y una base de datos PostgreSQL segura y escalable alojada en la nube, garantizando alta disponibilidad y acceso remoto.

El sistema implementa un modelo de acceso basado en roles, diferenciando las funcionalidades disponibles para el perfil de **Administrador** y el de **Mesero**, asegurando que cada usuario solo pueda acceder a las herramientas pertinentes a sus responsabilidades.

### II. REQUERIMIENTOS DEL SISTEMA

El desarrollo del sistema SIFOOD se guio por un conjunto de requerimientos funcionales, que definen las acciones que el sistema debe realizar, y no funcionales, que describen sus atributos de calidad.

#### A. Requerimientos Funcionales (RF)

* **Módulo de Autenticación y Seguridad (RF01-RF03):** El sistema debe presentar una pantalla de login que valide usuario y contraseña contra la base de datos. Deben existir los roles de aplicación **Administrador** y **Mesero**, con acceso restringido a los módulos correspondientes.
* **Módulo de Gestión de Mesas (RF04-RF06):** El Administrador debe poder gestionar las mesas del restaurante. El Mesero debe visualizar un mapa de mesas con su estado (Disponible, Ocupada) y poder cambiar dicho estado al iniciar o cerrar un pedido.
* **Módulo de Gestión del Menú (RF07-RF09):** El Administrador debe tener control CRUD (Crear, Leer, Actualizar, Eliminar) sobre los platos del menú, cada uno con nombre, descripción, precio y categoría.
* **Módulo de Gestión de Pedidos (RF10-RF13):** El Mesero debe poder crear un nuevo pedido, asociarlo a una mesa y agregarle productos del menú.
* **Módulo de Facturación (RF14-RF16):** El sistema debe generar una factura detallada para una mesa. El Mesero debe poder registrar el pago, lo que cierra el pedido y actualiza automáticamente el estado de la mesa a "disponible".
* **Módulo de Reportes (RF17-RF18):** El Administrador debe poder generar reportes de consumo diario y visualizar estadísticas como los platos más vendidos.

#### B. Requerimientos No Funcionales (RNF)

* **Usabilidad (RNF01-RNF03):** La interfaz de usuario debe ser intuitiva, con una navegación fluida y proporcionar retroalimentación visual clara al usuario tras realizar acciones.
* **Fiabilidad (RNF04-RNF05):** El sistema debe realizar validaciones de datos en todos los formularios y presentar mensajes de error comprensibles.
* **Rendimiento (RNF06-RNF07):** La carga de datos (menús, mesas) y el guardado de transacciones deben ser operaciones rápidas.
* **Seguridad (RNF08-RNF09):** Las contraseñas deben almacenarse de forma segura (cifradas/hasheadas). Los permisos de los roles deben estar estrictamente separados.

### III. ARQUITECTURA DEL SISTEMA

SIFOOD se ha diseñado siguiendo una arquitectura cliente-servidor de dos capas, una solución clásica y robusta para aplicaciones de escritorio que requieren una gestión centralizada de datos.

* **Capa de Cliente (Frontend):** Consiste en una aplicación de escritorio desarrollada en Java SE, utilizando la biblioteca Swing para la construcción de la interfaz gráfica de usuario (GUI). Esta aplicación es el punto de interacción para los empleados del restaurante (Administradores y Meseros) y es responsable de presentar los datos y capturar las entradas del usuario. Toda la lógica de presentación y la interacción con el usuario reside en esta capa.
* **Capa de Servidor (Backend):** Está compuesta por una base de datos PostgreSQL gestionada en la plataforma en la nube Neon. Esta capa es responsable del almacenamiento, la gestión, la seguridad y la integridad de todos los datos del sistema. La lógica de negocio crítica, como las validaciones cruzadas, los cálculos automáticos y la auditoría, se ha implementado directamente en la base de datos mediante procedimientos almacenados, funciones y triggers para garantizar la coherencia y el rendimiento.

La comunicación entre ambas capas se establece a través de una conexión JDBC (Java Database Connectivity) sobre un canal seguro SSL/TLS, requisito de la plataforma en la nube, garantizando la confidencialidad de los datos en tránsito.

### IV. DISEÑO Y MODELADO DE LA BASE DE DATOS

Un diseño de base de datos robusto es fundamental para la integridad y el rendimiento del sistema. El modelo de SIFood fue diseñado siguiendo los principios de normalización para minimizar la redundancia y mejorar la consistencia de los datos.

#### A. Modelo Entidad-Relación y Normalización

El esquema de la base de datos fue diseñado hasta la **Tercera Forma Normal (3FN)**. Se utilizó la herramienta en línea dbdiagram.io para generar y validar el modelo entidad-relación (ER), asegurando que todas las dependencias transitivas fueran eliminadas y que cada atributo no clave dependiera únicamente de la clave primaria. El modelo incluye relaciones uno a muchos (ej. pedidos y detalle\_pedidos) y una relación muchos a muchos (platos e ingredientes), la cual fue normalizada mediante la tabla de enlace plato\_ingredientes.

#### B. Estructura e Integridad Referencial

La base de datos consta de 12 tablas, incluyendo tablas de catálogo (categorias\_platos, estados\_pedido) y tablas transaccionales (pedidos, usuarios). Se aplicaron rigurosamente las siguientes restricciones para garantizar la integridad de los datos:

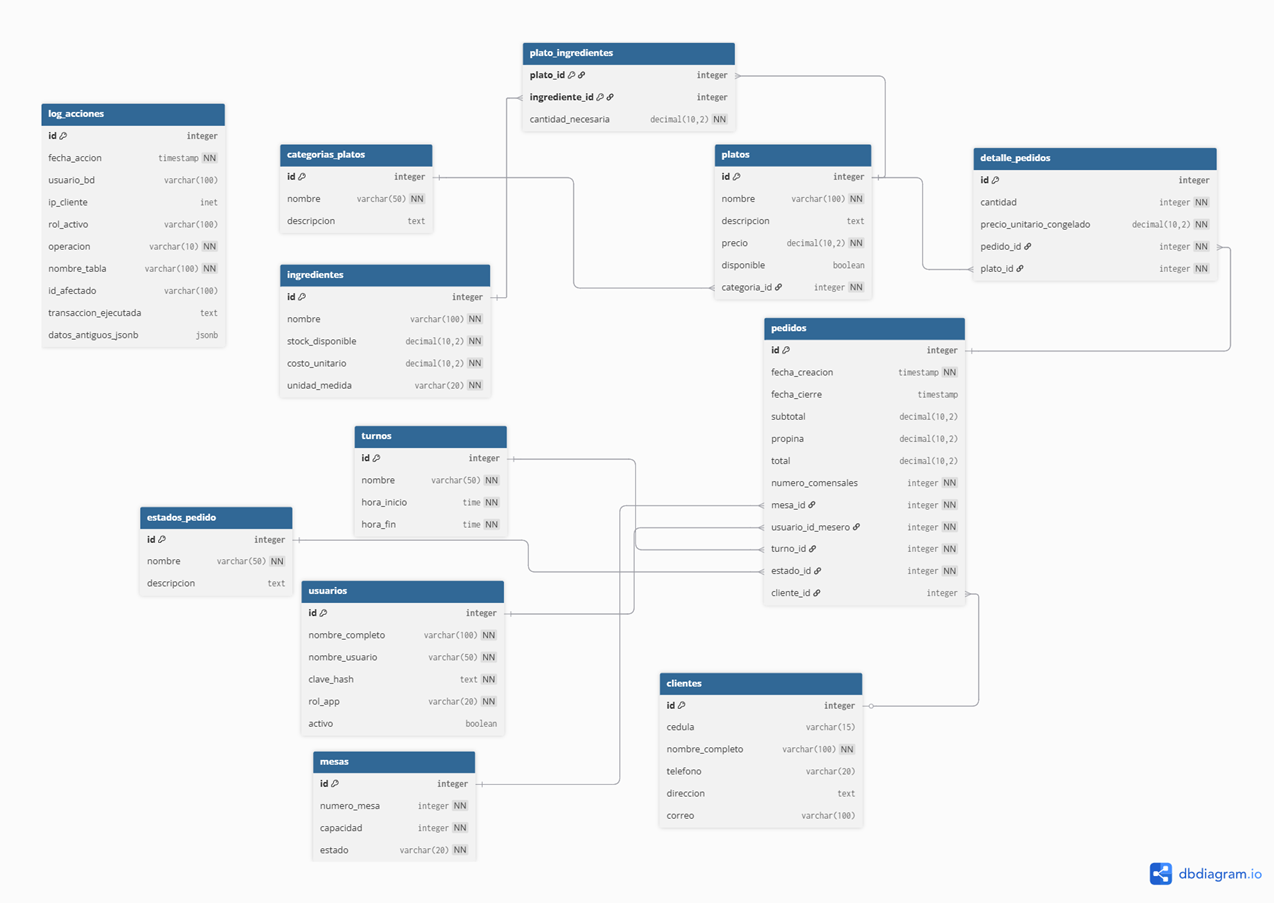
* **Restricciones PRIMARY KEY y UNIQUE:** Para garantizar la unicidad de los registros.
* **Restricción FOREIGN KEY:** Para mantener la integridad referencial entre las tablas. Se utilizaron políticas como ON DELETE CASCADE para asegurar que los registros dependientes se eliminen consistentemente.
* **Restricción CHECK:** Para implementar reglas de negocio a nivel de datos, como asegurar que los precios y el stock no sean negativos.
* **Restricción NOT NULL:** Para asegurar que los campos obligatorios siempre contengan un valor. La tabla se muestra en la Ilustración 1.

Ilustración 1

### V. IMPLEMENTACIÓN DE LÓGICA DE NEGOCIO

Para centralizar las reglas de negocio y mejorar la seguridad, se implementó lógica compleja directamente en la base de datos.

#### A. Procedimientos Almacenados

Se desarrollaron cinco procedimientos almacenados clave, en la Ilustración 2 se destacan:

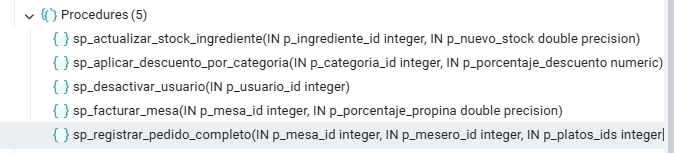


Ilustración 2

* **sp\_registrar\_pedido\_completo**: Encapsula la creación de un pedido en una transacción atómica. Valida que la mesa esté disponible y el mesero activo antes de insertar los registros en las tablas pedidos y detalle\_pedidos, realizando un ROLLBACK automático en caso de error.
* **sp\_facturar\_mesa**: Automatiza el proceso de facturación, calculando el total, actualizando el estado del pedido a "Pagado" y liberando la mesa.

#### B. Funciones y Triggers

Se crearon tres funciones para cálculos específicos, que se muestran en la Ilustración 3 como

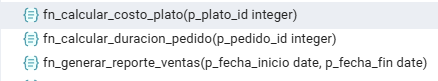


Ilustración 3

fn\_calcular\_costo\_plato() y fn\_generar\_reporte\_ventas(). Adicionalmente, se implementaron dos triggers fundamentales que se visualizan en la siguiente Ilustración 4:

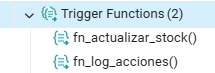


Ilustración 4

1. **Trigger de Control de Stock:** Descuenta automáticamente del inventario los ingredientes utilizados cada vez que se añade un plato a un pedido, previniendo la venta de productos sin stock.
2. **Trigger de Auditoría:** Registra automáticamente cualquier operación de inserción, actualización o eliminación en las tablas críticas (usuarios, pedidos, platos) en una tabla log\_acciones.

### VI. CARACTERÍSTICAS DE SEGURIDAD

La seguridad fue un pilar fundamental en el diseño de SIFood. A continuación en la Ilustración 5 se muestran los roles.



Ilustración 5

* **Control de Acceso Basado en Roles (RBAC):** Se crearon roles a nivel de base de datos (administrador\_db, operador\_app, auditor\_db) con privilegios específicos gestionados mediante GRANT y REVOKE. La aplicación Java se conecta utilizando el rol operador\_app, que tiene los permisos mínimos necesarios para funcionar (principio de mínimo privilegio).
* **Protección de Contraseñas:** Las contraseñas de los usuarios de la aplicación no se almacenan en texto plano. Se utiliza la extensión **pgcrypto** de PostgreSQL para hashearlas con el algoritmo Blowfish (crypt y gen\_salt), un estándar robusto contra ataques de fuerza bruta. En la siguiente Ilustración 6 se muestra la aplicación de pgcrypto en pgAdmin4.

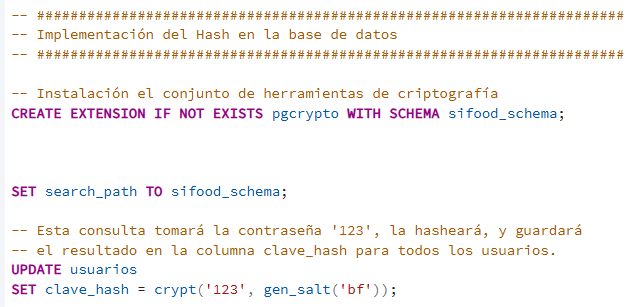


Ilustración 6

* **Prevención de Inyección SQL:** Toda la interacción de la aplicación Java con la base de datos se realiza a través de **PreparedStatement** como en la siguiente Ilustración 7. Este mecanismo parametriza las consultas, tratando todas las entradas del usuario como datos y no como código ejecutable, lo que neutraliza eficazmente los ataques de inyección SQL.

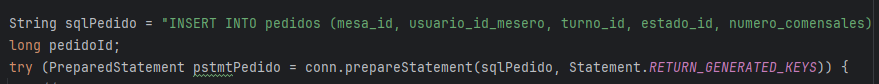


Ilustración 7

Ejemplo conexión insegura:

¡¡¡CÓDIGO INSEGURO Y VULNERABLE - NO USAR!!!

String user = txtUser.getText(); // Supongamos que el usuario escribe: ' OR '1'='1

String pass = new String(txtClave.getPassword());

El programador une los strings para crear la consulta

String sql = "SELECT \* FROM usuarios WHERE nombre\_usuario = '" + user + "' AND clave\_hash = '" + pass + "'";

La consulta resultante sería:

SELECT \* FROM usuarios WHERE nombre\_usuario = '' OR '1'='1' AND clave\_hash = 'password'

La condición '1'='1' siempre es verdadera, por lo que la base de datos devuelve

el primer usuario de la tabla, ¡permitiendo el acceso sin una contraseña válida!

* **Conexión Cifrada:** La comunicación entre la aplicación cliente y la base de datos en la nube está protegida mediante SSL/TLS, que se demuestra en la siguiente Ilustración 8, garantizando la confidencialidad de los datos. Esto se realizó al momento de conectar la base de datos entre NEON Postgre y pgAdmin4.

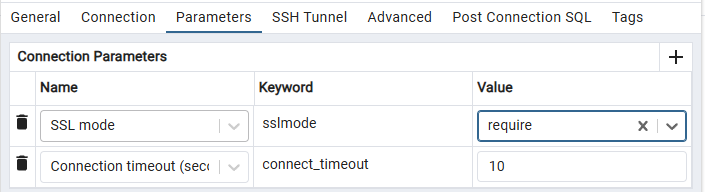


Ilustración 8

### VII. PRUEBAS Y SIMULACIÓN DE PERFILES

Para validar la robustez y seguridad del sistema, se realizaron pruebas simulando los perfiles profesionales involucrados en el ciclo de vida de una base de datos.

* **Administrador de BD:** Se realizaron tareas de mantenimiento como REINDEX y se generó un **backup en caliente** utilizando la herramienta pg\_dump (Ilustración 9).



Ilustración 9

* **Oficial de Seguridad:** Se verificó la correcta aplicación de los privilegios por rol y se consultó la tabla log\_acciones para auditar la actividad. Se demostró la protección de contraseñas consultando la columna clave\_hash. A continuación en la Ilustración 10 se muestra la tabla log\_acciones, y en la Ilustración 11se muestran los usuarios

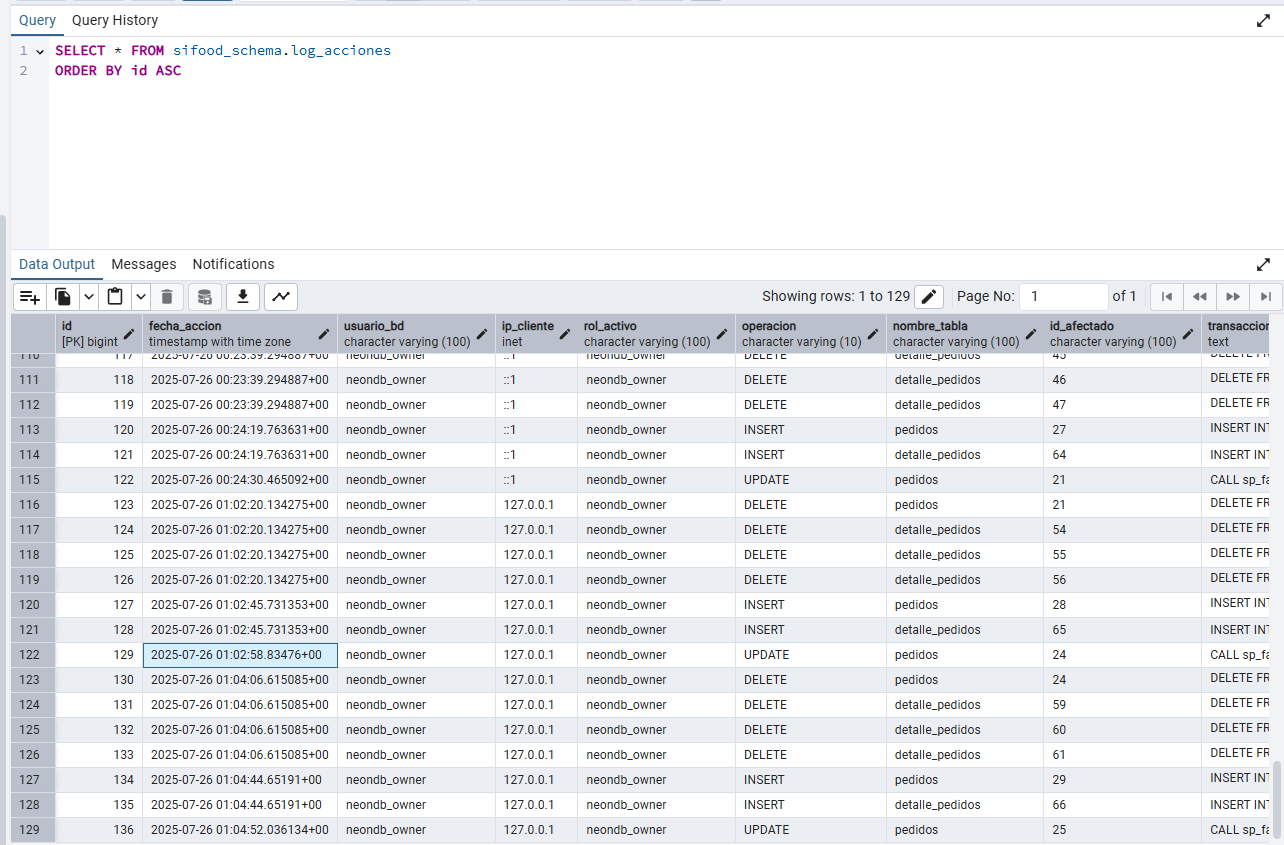


Ilustración 10

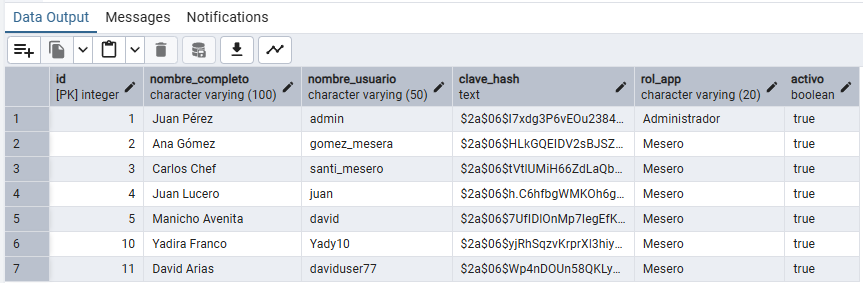


Ilustración 11

* **Desarrollador de Consultas y Usuario Final:** Se ejecutaron las vistas y funciones para validar los reportes, y se realizó una conexión con un rol de privilegios mínimos para confirmar que el acceso a los datos estaba correctamente restringido.

### VIII. CONCLUSIONES

### Para el futuro, se tiene pensado añadir modalidad CRUD, que consta en modificar el funcionamiento del mesero, para que tenga la capacidad de cambiar el pedido una vez pase a cocina, eliminar un plato o cambiarlo.

El proyecto SIFood ha culminado con éxito en el desarrollo de un sistema de gestión para restaurantes funcional, seguro y robusto. La arquitectura cliente-servidor con Java Swing y PostgreSQL en la nube ha demostrado ser una solución eficaz, combinando una interfaz de usuario interactiva con una gestión de datos centralizada y segura.

Se han cumplido todos los objetivos principales, incluyendo la implementación de un sistema de roles, la gestión completa del ciclo de pedidos y la aplicación de múltiples capas de seguridad. El sistema actual sienta una base sólida sobre la cual se pueden construir futuras mejoras, como un módulo de reportes gráficos, una interfaz para la cocina en tiempo real o una aplicación web para reservas en línea.

### REFERENCIAS

[1] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2016.

[2] H. Schildt, *Java: The Complete Reference*, 11th ed. McGraw-Hill Education, 2018.

[3] PostgreSQL Global Development Group, *PostgreSQL 16 Documentation*. The PostgreSQL Global Development Group, 2023. [En línea]. Disponible: <https://www.postgresql.org/docs/16/index.html>

[4] K. Sierra and B. Bates, *Head First Java*, 2nd ed. O'Reilly Media, 2005.

[5] The OWASP Foundation, "SQL Injection Prevention Cheat Sheet". OWASP. [En línea]. Disponible: <https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html>

[6] Neon, Inc., "Neon Documentation". Neon, Inc., 2023. [En línea]. Disponible: <https://neon.tech/docs>