

UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC

Facultad de Ingenierías

Escuela Académico Profesional de Ingeniería Informática y Sistemas



INGENIERÍA DE SOFTWARE I

Proyecto de Software

Software de gestión de pedidos de Pizzería

Docente:

Julio Cesar Lloclli Champi

Presentado por:

- Cristian Olivera Chávez
- Giovanni Ttito Ccaccasto
- Yhury Cristiam Anampa Quispe
- Amílcar Virtu Loayza
- Luis Fernando Juarez Peña

Abancay, Perú

Año 2024

AGRADECIMIENTOS

"Deseo expresar mi más sincero agradecimiento a todas las personas que hicieron posible la realización de este Software. En primer lugar, agradecer al Ing. Julio César Lloelli Champi, por su valiosa orientación, paciencia y conocimientos compartidos a lo largo de este proyecto. Sus sugerencias y retroalimentación fueron fundamentales para el desarrollo y mejora de este trabajo."

También quiero agradecer a nuestras familias, quienes han sido mi mayor apoyo y fuente de motivación en cada etapa de nuestra carrera. Su constante aliento y confianza en nosotros han sido determinantes para llegar hasta aquí.

Asimismo, agradezco a nuestros compañeros de estudio, quienes compartieron esta travesía académica con nosotros. Sus aportes, debates y colaboración mutua enriquecieron mi experiencia y contribuyeron a la culminación de esta monografía.

Finalmente, quiero agradecer a todas las instituciones y personas que, de alguna manera, brindaron su apoyo, recursos y acceso a información relevante para llevar a cabo este software.

ÍNDICE

I. INTRODUCCIÓN.....	5
II. MISION Y VISION DE LA EMPRESA.....	6
2.1 Misión.....	6
2.2 Visión.....	6
III. PLANTEAMIENTO DEL PROBLEMA.....	7
3.1 Definición.....	7
3.2 Descripción e impacto del problema.....	7
3.1 Objetivos del proyecto.....	8
3.1.1 Objetivo general.....	8
3.1.2 Objetivo específico.....	9
IV. JUSTIFICACIÓN DEL PROYECTO.....	10
4.1 Beneficios Esperados:.....	10
4.2 Beneficiarios del Proyecto ¿Quienes se beneficiaran?.....	11
V. ALCANCE DEL PROYECTO.....	12
5.1 Incluye.....	12
5.2 No incluye.....	15
VI. MODELO EN CASCADA.....	16
6.1 Análisis.....	16
6.1.1 Introducción.....	16
6.1.1.1 Contexto.....	16
6.1.1.2 Propósito.....	16
6.1.1.3 Alcance.....	16
6.1.1.4 Definiciones.....	17
6.1.2 Descripción General:.....	17
6.1.2.1 Perspectiva del Producto.....	17
6.1.2.2 Funcionalidades.....	17
6.1.2.3 Características del Usuario.....	18
6.1.2.4 Cuestionario De Requerimientos De Software.....	18
6.1.6 Estudio de factibilidad de Software.....	21
6.1.6.1 Factibilidad técnica.....	21
7.1.6.2 Costos de Infraestructura.....	25
6.1.6.2.1. Salarios de los Personales.....	30
6.1.6.2.2. Capacitación de Personal.....	30
6.1.6.3 Costos Iniciales de Implementación.....	30
6.1.6.4 Costos de Mantenimiento y Soporte.....	31
6.1.6.5 Total Estimado de Costos de software.....	31
6.1.3 Requisitos Específicos:.....	34
6.1.3.1 Requisitos Funcionales:.....	34
6.1.3.2 Requisitos No Funcionales.....	37
6.1.4 Modelado de negocios (ARCHITEC):.....	40
6.1.5 Validación y Gestión de Requisitos:.....	47

6.2 Fase de Diseño:.....	48
6.2.1 Base de datos.....	48
6.2.1.1 Diseño de base de datos modelo Entidad-Relación (ER).....	48
6.2.1.2 Diseño de base de datos modelo lógico.....	48
6.2.1.3 Diseño de base de datos modelo físico.....	49
6.2.1.4 Diccionario de base de datos.....	51
6.2.2 Diseño de la interfaz de usuario.....	52
6.3 Fase de Implementación:.....	59
VII. CRONOGRAMA DEL PROYECTO.....	94
VIII. CONCLUSIÓN.....	95
IX. ANEXOS.....	96
X. REFERENCIAS BIBLIOGRÁFICAS.....	103

I. INTRODUCCIÓN

El desarrollo de un sistema de gestión de pedidos para una pizzería nos permite aplicar nuestros conocimientos en ingeniería de software a problemas reales. Además, buscamos crear una herramienta que mejore la eficiencia operativa de estos negocios, respondiendo a la creciente necesidad de soluciones tecnológicas.

Las pizzerías requieren una coordinación precisa para evitar errores y mejorar el servicio al cliente. Nuestro sistema busca automatizar y optimizar estos procesos, permitiendo al personal centrarse en la calidad del servicio mientras el software se encarga de las tareas repetitivas.

El desarrollo del sistema incluye varias etapas, como análisis de requisitos, diseño, codificación, pruebas y despliegue. Estas fases nos permiten aplicar metodologías modernas de programación y desarrollo ágil, fomentando habilidades clave como el trabajo en equipo y la toma de decisiones.

Además de optimizar procesos, nuestro proyecto se centra en la experiencia del usuario, diseñando una interfaz intuitiva y fácil de usar. Esto asegura que el sistema sea eficiente y accesible, facilitando el trabajo de los empleados sin necesidad de conocimientos técnicos avanzados.

II. MISIÓN Y VISIÓN DE LA EMPRESA

2.1 Misión

Nuestra misión es deleitar a nuestros clientes con pizzas artesanales hechas con ingredientes frescos y de la mejor calidad, brindando un servicio personalizado y amable. Nos esforzamos por crear una atmósfera donde cada cliente se sienta como en casa, disfrutando de un servicio rápido, eficiente y siempre con una sonrisa. Trabajamos con pasión y dedicación para superar las expectativas de nuestros clientes, convirtiendo cada visita en una experiencia memorable.

2.2 Visión

Nos vemos como la pizzería de referencia en nuestra comunidad, reconocida por nuestra dedicación a la excelencia culinaria y el servicio al cliente. Buscamos expandirnos de manera sostenible, manteniendo siempre nuestros altos estándares de calidad y convirtiéndonos en un punto de encuentro querido por los amantes de la buena comida y el ambiente acogedor.

III. PLANTEAMIENTO DEL PROBLEMA

3.1 Definición

En el contexto de la gestión y operación de una pizzería, la eficiencia y precisión son cruciales para asegurar la satisfacción del cliente y la rentabilidad del negocio. Actualmente, la pizzería enfrenta varios problemas operativos debido al uso de métodos manuales en la gestión de pedidos y cobros. Estos problemas no solo afectan la experiencia del cliente sino que también aumentan la carga de trabajo del personal y reducen la eficiencia operativa. Este planteamiento del problema describe las principales ineficiencias y errores en el sistema actual y propone la necesidad de una solución automatizada.

3.2 Descripción e impacto del problema

En la ciudad de Abancay la pizzería Adriana realiza sus procesos operacionales del sistema de manera manual, este método genera varias ineficiencias y problemas, como errores frecuentes en la toma de pedidos, demoras en el procesamiento de pagos, y dificultades para mantener un control preciso y actualizado de las operaciones diarias.

1. Control Manual de Pedidos

- **Descripción:** Los mozos utilizan métodos manuales, como papel y lápiz, para tomar y gestionar pedidos.
- **Impacto:**
 - **Errores en Pedidos:** Errores humanos al anotar pedidos pueden resultar en entregas incorrectas.
 - **Retrasos:** El tiempo necesario para escribir y transmitir los pedidos a la cocina causa retrasos en el servicio.
 - **Dificultad en el seguimiento:** La gestión manual dificulta el seguimiento y actualización del estado de los pedidos.

2. Ineficiencia en la Realización de Cobros y Cierre de Mesa

- **Descripción:** El cajero realiza cobros manualmente, utilizando calculadoras y registros en papel.
- **Impacto:**
 - **Largas Esperas:** Los clientes enfrentan largas esperas para pagar, afectando su experiencia.

- **Errores de Cobro:** Errores en el cálculo o registro del pago pueden generar pérdidas financieras y problemas de conciliación de caja.
- **Baja Rotación:** El tiempo adicional necesario para cada transacción reduce la rotación de mesas y, por ende, los ingresos potenciales.

3. Errores al Realizar Pedidos

- **Descripción:** La falta de un sistema automatizado lleva a errores al tomar los pedidos.
- **Impacto:**
 - **Cliente Insatisfecho:** Pedidos incorrectos o incompletos causan insatisfacción y quejas.
 - **Retrabajo:** Los errores en los pedidos pueden requerir rehacer la comida, aumentando los costos y el tiempo de servicio.
 - **Ineficiencia:** Aumenta la carga de trabajo tanto para el personal de servicio como para la cocina.

4. Mala Optimización del Tiempo en el Sistema de Funciones de la Pizzería

- **Descripción:** La falta de un sistema eficiente de gestión de funciones resulta en una mala utilización del tiempo del personal.
- **Impacto:**
 - **Desperdicio de Recursos:** Mozos y cajeros dedican más tiempo del necesario a tareas que podrían automatizarse.
 - **Productividad baja:** La falta de optimización reduce la productividad general del equipo.
 - **Insatisfacción del Personal:** El manejo ineficiente del tiempo puede llevar a estrés y desmotivación entre los empleados.

3.1 Objetivos del proyecto

3.1.1 Objetivo general

El objetivo general del proyecto es desarrollar un software de escritorio para una pizzería que optimice la gestión de pedidos y cobros, reduciendo la ineficiencia y los errores asociados con el control manual.

3.1.2 Objetivo específico

Desarrollar un módulo para mozos que permita registrar y gestionar pedidos de manera digital, reduciendo errores en la toma de pedidos en un 75%.

Este sistema incluirá módulos específicos para mozos y cajeros, mejorando la precisión en la toma de pedidos, la eficiencia en la realización de cobros. Además, se enfocará en optimizar el tiempo de las funciones operativas de la pizzería, proporcionando una solución integral que aumente la productividad y la satisfacción del cliente.

Implementar un módulo para cajeros que optimice el proceso de cobro, aumentando la eficiencia y reduciendo el tiempo necesario para realizar transacciones, así como minimizar errores en el cálculo y registro de pagos.

Crear una funcionalidad de cierre de mesa automatizado que facilite a los cajeros la reconciliación de ventas diarias, asegurando una contabilidad precisa y eficiente al final de cada jornada laboral.

Proveer un sistema de seguimiento y control en tiempo real de los pedidos y cobros, permitiendo a los gerentes supervisar y ajustar las operaciones de la pizzería en función de la demanda y las necesidades del negocio.

IV. JUSTIFICACIÓN DEL PROYECTO

4.1 Beneficios Esperados:

- **Reducción de Errores en la Toma de Pedidos:** Usar un sistema automatizado para tomar pedidos ayuda a reducir errores, mejorar la eficiencia y aumentar la satisfacción del cliente, lo que es fundamental para que el negocio sea exitoso y rentable.
- **Eficiencia en la Realización de Cobros:** Reducir los procesos de cobros, de manera más eficiente y precisa, mejorando la experiencia del cliente y fortaleciendo la gestión financiera.
- **Incremento de la Productividad Operativa:** Automatizar los procesos operativos en una pizzería no solo aumenta la productividad, sino que también mejora la eficiencia, reduce costos y mejora la calidad del servicio, lo que resulta en mayor satisfacción tanto para los clientes como para el personal.
- **Aumento de la Satisfacción del Cliente:** Automatizar procesos clave no solo mejora la eficiencia operativa, sino que también tiene un impacto positivo directo en la satisfacción del cliente. Esto contribuye a la fidelización, a una buena reputación y a un aumento en las ventas, asegurando el éxito y crecimiento del negocio.
- **Análisis y Reportes Detallados por Mesa:** Automatizar el análisis y los reportes detallados por mesa proporciona una ventaja competitiva importante, permitiendo gestionar el negocio de manera más eficiente y efectiva. Tomar decisiones informadas y rápidas, optimizar operaciones y mejorar la satisfacción del cliente lleva a un negocio más rentable y sostenible.
- **Seguridad y Cumplimiento:** Automatizar procesos relacionados con la seguridad y el cumplimiento protege contra riesgos y asegura que se cumplan las regulaciones, mejorando la eficiencia operativa y la confianza del cliente. Implementar estas soluciones es esencial para la gestión exitosa y sostenible del negocio.

4.2 Beneficiarios del Proyecto ¿Quienes se beneficiaran?.

- Los propietarios del restaurante de pizza:

Al poner en marcha proyectos que mejoren la eficacia operativa, la experiencia del cliente, la seguridad y protección, el cumplimiento y el impacto financiero del negocio y la capacidad de ajuste, los propietarios de una pizzería son los principales beneficiarios. Esos proyectos no solo aumentan la rentabilidad y la solidez del negocio; garantizan el crecimiento sostenible a largo plazo y fortalecen la posición en el mercado.

- Mozo y Cajero.

Los mozos y cajeros de la pizzería Adriana se benefician de la eficacia operativa y la experiencia del cliente, y los proyectos de impacto financiero y capacidad de ajuste que mejoran el cumplimiento, al tiempo que les resultan fáciles de cumplir y de avanzar. Estos proyectos no solo permiten mejor experiencia de servicio, mayor satisfacción y lealtad del cliente, sino que también mejoran la competitividad en el mercado.

- Comunidad y Entorno Local

La pizzería Adriana está comprometida con su comunidad, entorno local y el ambiente que ofrece. Al adoptar prácticas sostenibles, la pizzería no solo fortalece su posición en el mercado, sino que también construye relaciones duraderas y positivas con la comunidad.

V. ALCANCE DEL PROYECTO

5.1 Incluye

1. Desarrollo del Software:
 - Creación de un software de escritorio específico para la gestión de una pizzería.
 - Implementación de módulos específicos para mozos y cajeros.
2. Módulo para Inicio de sesión
 - Contiene dos formularios para ingresar correo y contraseña.
 - Contiene botón para hacer click e iniciar sesión.
3. Módulo para agregar salas y mesas
 - contiene dos formularios para agregar el número de salas y mesas dentro de ellas.
4. Módulo para seleccionar sala.
 - Contiene la opción de seleccionar una sala y dentro de ellas contengan las mesas numeradas.
5. Módulo para seleccionar mesa
 - Una vez seleccionada la sala podremos seleccionar la mesa en la que queremos agregar productos.
6. Módulo para seleccionar producto
 - En este módulo se seleccionan los productos que ofrece la pizzería.
7. Modulo para agregar producto.
 - En este módulo se agrega uno o más productos a la mesa seleccionada anteriormente.
8. Módulo para la cuenta de usuarios (datos del trabajador)
 - En este módulo se observa los datos del trabajador y la contraseña de manera encriptada.
 - La contraseña es de más de 8 caracteres.
9. Módulo para Verificar Datos de la empresa (Pizzeria Adriana)
 - En este módulo se ven los datos de la empresa y se pueden editar acorde a las necesidades, por si cambia de número de teléfono, dirección.
10. modulo para ver el historial de pedidos

- En este módulo se ven el estado del historial de cada pedido, es decir si la mesa aún está ocupada o desocupada, además podrás verificar los precios de cada una de ellas.

11. módulo para cierre de mesa

- En este módulo podrás cerrar la mesa cuando el cliente haya pagado su cuenta, además aquí incluirá un reporte de nota de venta en formato PDF descargable, en donde quedará registrado el monto total a pagar que será entregado al cliente antes de pagar.

12. Módulo para Mozos:

- Funcionalidad para iniciar sesión con Usuario y Contraseña.
- Funcionalidad para tener acceso a la pantalla principal.
- Funcionalidad para accesos a las mesas y ver si están o no disponibles.
- Funcionalidad para registrar y gestionar pedidos de manera digital.
- Interfaz amigable para facilitar la toma de pedidos .
- Funcionalidad para añadir, cancelar, eliminar productos de un pedido.
- Funcionalidad de poder ver el reporte de cada mesa con los detalles.
- Funcionalidad para poder imprimir la nota de reporte por mesa.
- Funcionalidad para imprimir nota de venta en formato PDF. La nota de venta incluirá los siguientes datos:



Pizzería Adriana
Ruc: 2074435572
Teléfono: 957847894
Dirección: Av. Nuñez 118, 1ra cuadra - Abancay

Atendido: Pablito
Nº de pedido: 4
Fecha y Hora: 2024-04-21 19:52:25

CANTIDAD	PRODUCTO	PREC. UNITARIO	PREC. TOTAL
1	PIZZA HAWAIANA	25.00	25.00
1	GASEOSA COCA COLA 2L	12.00	12.00

TOTAL A PAGAR: S/.37.00

Cancelación

Firma

13. Módulo para Cajeros:

- Funcionalidad para iniciar sesión con Usuario y Contraseña.
- Funcionalidad para tener acceso a la pantalla principal.
- Funcionalidad para accesos a las mesas y ver si están o no disponibles.
- Funcionalidad para registrar y gestionar pedidos de manera digital.
- Interfaz amigable para facilitar la toma de pedidos .
- Funcionalidad para añadir, cancelar, eliminar productos de un pedido.
- Funcionalidad de poder ver el reporte de cada mesa con los detalles.
- Funcionalidad para poder imprimir la nota de reporte por mesa.
- Funcionalidad para optimizar el proceso de cobro.
- Herramientas para registrar y calcular pagos de manera precisa.
- Funcionalidad de cierre de mesa al realizar el cobro para la reactivación de mesas automatizado.

14. Sistema de Seguimiento y Control en Tiempo Real:

- Herramientas para supervisar pedidos y cobros en tiempo real.

15. Análisis de Tiempos y Tareas:

- Herramientas de análisis para identificar ineficiencias operativas.
- Funcionalidades para mejorar la distribución del trabajo entre mozos y cajeros.
- Optimización del uso del tiempo operativo.

16. Capacitación y Soporte:

- Capacitación inicial para mozos y cajeros sobre el uso del sistema.
- Documentación y manuales de usuario.
- Soporte técnico durante el periodo de implementación y un tiempo determinado post-implementación.

5.2 No incluye

1. Desarrollo de Hardware:

- Provisión de equipos como computadoras

2. Desarrollo de Funcionalidades Adicionales:

- Características que no estén especificadas en los objetivos específicos del proyecto.
- Modificaciones personalizadas posteriores a la implementación inicial que no estén incluidas en el alcance original.

3. Soporte Continuo Indefinido:

- Soporte técnico y actualizaciones del sistema más allá de la entrega.

4. Mantenimiento del Software:

- Mantenimiento del software más allá del periodo de garantía establecido.
- Adaptaciones y mejoras del sistema fuera del periodo de mantenimiento incluido en el contrato.

VI. MODELO EN CASCADA

6.1 Análisis

6.1.1 Introducción

6.1.1.1 Contexto

En la ciudad de Abancay, muchas pizzerías realizan sus procesos de manera manual, incluyendo la toma de pedidos, el procesamiento de pagos y el cierre de mesa. Esta práctica manual lleva a ineficiencias significativas y un alto riesgo de errores, afectando la precisión en los pedidos, la eficiencia en los cobros y la satisfacción del cliente. La necesidad de modernizar estos procesos es evidente para mejorar la operatividad y la rentabilidad de las pizzerías.

6.1.1.2 Propósito

El propósito de este documento es detallar los requerimientos para el desarrollo de un software de escritorio que optimice la gestión de pedidos y cobros en una pizzería, reduciendo la ineficiencia y los errores asociados con el control manual. Este software buscará proporcionar una solución integral que aumente la productividad y la satisfacción del cliente mediante la implementación de módulos específicos para mozos, cajeros y gerentes.

6.1.1.3 Alcance

El proyecto abarca el desarrollo de un software de escritorio con las siguientes funcionalidades principales:

- Módulo de Mozos: Digitalización y gestión de pedidos.
- Módulo de Cajeros: Optimización del proceso de cobro.
- Sistema de Seguimiento y Control en Tiempo Real: Monitoreo y ajuste de operaciones.

El software estará diseñado para su uso en pizzerías de Abancay, adaptándose a las necesidades específicas de este entorno y mejorando la eficiencia operativa de estos establecimientos.

6.1.1.4 Definiciones

- Mozos: Personal encargado de tomar los pedidos y atender a los clientes en el local.
- Cajeros: Personal encargado de procesar los pagos y llevar el control de las transacciones.
- Gerentes: Personal responsable de supervisar y gestionar las operaciones diarias de la pizzería.
- Cierre de Mesa: Proceso de reconciliación de las ventas y pagos al final del consumo del cliente.
- Comandas: Hojas de papel donde se anotan los pedidos de los clientes para su posterior preparación en la cocina.

6.1.2 Descripción General:

6.1.2.1 Perspectiva del Producto

El proyecto se centra en el desarrollo de un software de escritorio diseñado específicamente para optimizar la gestión operativa de una pizzería en Abancay. Este software reemplazará los procesos manuales actuales con soluciones digitales que mejorarán la eficiencia, reducirán los errores y proporcionarán un mejor control sobre las operaciones diarias.

6.1.2.2 Funcionalidades

1. Módulo de Mozos:

- Registro digital de pedidos con capacidad de modificación y cancelación.
- Comunicación instantánea de pedidos a la cocina.
- Actualización en tiempo real del estado de los pedidos.
- Reducción significativa de errores en la toma de pedidos.

2. Módulo de Cajeros:

- Procesamiento eficiente de pagos con soporte para múltiples métodos (efectivo, tarjetas).
- Registro automático de transacciones y generación de recibos.
- Gestión de descuentos y promociones.
- Integración con sistemas de punto de venta (POS) para agilizar el proceso de cobro.

3. Sistema de Seguimiento y Control en Tiempo Real:

- Monitoreo en tiempo real del estado de pedidos y transacciones.
- Alertas automáticas para incidencias críticas (por ejemplo, pedidos demorados).

6.1.2.3 Características del Usuario

- Mozos: Personal operativo encargado de tomar y gestionar los pedidos de los clientes.
- Cajeros: Encargados del proceso de cobro y gestión financiera en el punto de venta.

6.1.2.4 Cuestionario De Requerimientos De Software

1. ¿Cuál es la actividad económica principal de la empresa?

La empresa se dedica principalmente a la elaboración y venta de pizzas y otros productos alimenticios relacionados.

Composición de la Empresa

2. ¿Cómo está compuesta la empresa?

La empresa en términos de estructura organizativa, está compuesta incluyendo el número de empleados, roles (mozos, cajeros, cocineros).

Manejo de Información

3. ¿Qué tipo de información manejan?

Información sobre pedidos, transacciones de ventas, inventario de ingredientes, horarios de empleados, datos de clientes, y reportes financieros.

4. ¿Quiénes tienen acceso a la información?

Personal autorizado como cajeros, gerentes y administradores.

Archivado y Actualización de la Información

5. ¿Qué modalidades utilizan para archivar la información?

Métodos de archivado como bases de datos digitales, registros en papel, sistemas de gestión de documentos, etc.

6. ¿Cada cuánto actualizan los registros de la información almacenada?

Frecuencia de actualización de los registros, por ejemplo, diariamente, semanalmente, en tiempo real, etc.

Problemática y Solución

7. ¿Cuál es la problemática que buscan solucionar a través del sistema de información?

Problemas como ineficiencias en la gestión de pedidos y cobros, errores en la toma de pedidos y en el cálculo de pagos, dificultades en el cierre de caja, y la falta de un seguimiento en tiempo real de las operaciones.

Acceso al Sistema de Información

8. ¿Quiénes tendrán acceso al sistema de información?

Personal autorizado como mozos y cajeros.

Preguntas Específicas por Módulo

Módulo de Mozos

1. ¿Cómo registran actualmente los mozos los pedidos de los clientes?

Actualmente al registrar los pedidos lo hacemos de manera manual utilizando un bolígrafo y papel lo cual implica que a veces cometemos errores de transcripción , pérdida de los pedidos y hasta confusiones lo que conlleva a un error en el pedido.

2. ¿Cuáles son los errores más comunes que ocurren durante la toma de pedidos?

Al momento de realizar un pedido un cliente, los empleados a veces se equivocan en los pedidos a las mesas, como también a los clientes esto a veces se debe a la aglomeración o saturación de clientes, por ello es fundamental un sistema en la pizzeria.

3. ¿Cómo se comunican actualmente los pedidos a la cocina?

Los pedidos son comunicados en comandas escritas a mano, a los encargados de cocina para su preparación y en el orden de llegada. Las comandas son papeles bond cortados en medidas pequeñas, ahí contiene la lista de pedidos de las mesas numeradas.

Módulo de Cajeros

1. ¿Cómo se manejan actualmente los pagos y transacciones en la pizzería?

Los pagos: se realizan de manera manual haciendo el uso de hojas y calculadoras.

Las transacciones: lo realizamos con un registro manual, para concretar cada pedido este al final se registra en un cuaderno para cuadrar caja al finalizar el día laboral.

2. ¿Cuánto tiempo toma en promedio procesar una transacción?

No está definido, debido a la variada cantidad de pedidos que realizan los clientes, además este se tiene que registrar en un cuaderno para finalizar cada pedido, en promedio terminar una transacción dura 3 minutos a 4 minutos.

3. ¿Qué tipos de errores son comunes durante el proceso de cobro?

Los errores más comunes en el proceso de cobro son las posibles equivocaciones en el cálculo de las cuentas ya que se hace de manera manual lo que conlleva a obtener pérdidas de dinero innecesarias.

6.1.6 Estudio de factibilidad de Software

6.1.6.1 Factibilidad técnica

Mediante esta factibilidad se establece si el sistema propuesto puede desarrollarse con los recursos técnicos con que cuenta el equipo de desarrollo; esto se hace considerando la disponibilidad de los recursos existentes en términos de hardware, software y recurso humano, o sea la existencia de la tecnología y el conocimiento necesario para establecer que sea factible técnicamente el desarrollo del proyecto.

6.1.6.1.1 Sistema Operativo

Este elemento es de los más importantes ya que debe cumplir con las características de estabilidad, administración, velocidad, facilidad de uso, seguridad, multiusuario y escalabilidad para soportar la instalación del sistema informático y a la vez brindar velocidad de conexión a las bases de datos y seguridad a los usuarios.

Se presentan a continuación diferentes sistemas operativos que cumplen con las características necesarias, para el buen funcionamiento del sistema propuesto.

Opciones:

ITEM	DESCRIPCIÓN	PRECIO ÚNICO POR PC
1	WINDOWS SERVER 2021	S/ 699,00
2	LINUX	GNU A GPL
3	MAC OS	INSTALACION DE FABRICA

TABLA N°1: Podemos visualizar los tipos de sistemas operativos a usar en el proyecto junto al costo de usabilidad

Elección: Windows 11, porque destaca por su amplia compatibilidad con software, flexibilidad de hardware y familiaridad de interfaz, a menudo a un costo inicial más bajo. En contraste, macOS ofrece una integración profunda con el ecosistema de Apple y un enfoque en la experiencia del usuario, mientras que Linux, aunque es gratuito y tiene una comunidad activa, puede requerir más habilidades técnicas y tiene una compatibilidad más limitada con ciertos programas.

7.1.6.1.2 Lenguaje de Desarrollo

El lenguaje de desarrollo debe cumplir con las siguientes características:

- Soporte a gran cantidad de bases de datos
- Facilidad de desarrollo de sistemas
- En continua mejora
- Fácil de administrar
- Estable y ampliamente utilizado

Se presentan a continuación diferentes lenguajes de desarrollo que cumplen con las características arriba mencionadas.

ITEM	DESCRIPCIÓN	PRECIO
1	JAVA	GNU AGPL
2	PYTHON	GNU AGPL
3	C#	GNU AGPL

Tabla N°2: Se observa los lenguajes de programación junto con los precios.

Elección : Java es ideal para un software de pizzería debido a su robustez, escalabilidad y portabilidad. Su capacidad para crear aplicaciones empresariales robustas asegura que el sistema pueda manejar múltiples operaciones simultáneas, como la gestión de pedidos, inventarios y pagos

6.1.6.1.3 Software de creación de base de datos

Este es un factor muy importante ya que determinará la manera en que se guardará la información, la velocidad de procesamiento, respaldo de los datos y la seguridad.

El sistema gestor de base de datos debe cumplir con las siguientes características:

- Estable
- Seguro
- Escalable
- Soporte de grandes cantidades de información
- Conexión con diferentes lenguajes de programación vía ODBC
- En continua mejora

ITEM	DESCRIPCIÓN	CANTIDAD	PRECIO	PRECIO
			MENSUAL	ANUAL
1	SQLITE	1	2400.00	28800.00
2	MYSQL	1	2500.00	30000.00
3	MONGODB	1	1800.00	21600.00
4	SQL SERVER	1	1000.00	12000.00

Tabla N°3: Se observan las bases de datos consideradas para usar en el proyecto junto al precio Mensual y Anual.

Elección : MySQL es una excelente elección para tu proyecto debido a su capacidad de integración con tecnologías de Microsoft, como .NET, lo que asegura una experiencia de desarrollo fluida. Aunque existen otras opciones como SQLite, que no es escalable, o MongoDB, que no es ideal para transacciones ACID, MySQL ofrece un equilibrio sólido entre rendimiento, escalabilidad y características avanzadas de seguridad. Además, su facilidad de uso y herramientas de gestión eficaces lo hacen una opción confiable para aplicaciones de nivel empresarial, combinando eficiencia y robustez en la gestión de bases de datos.

6.1.6.1.4 Servidores

Un servidor es una computadora diseñada para procesar, gestionar y almacenar datos, así como para proporcionar servicios a otras computadoras, conocidas como clientes, dentro de una red. Los servidores son fundamentales en la infraestructura tecnológica, ya que permiten la centralización y administración eficiente de recursos y aplicaciones. Los servidores pueden ser físicos, como los que se detallan a continuación, o virtuales, ejecutados en entornos de virtualización.

OPCIÓN	MODELO	ESPECIFICACIONES	COSTO UNITARIO	CANTIDAD	COSTO TOTAL
1	Xampp	Intel Xeon E-2224G, 16GB RAM, 1TB HDD	GNU AGPL	5	GNU AGPL
2	HP ProLiant ML30 G10	Intel Xeon E-2124, 32GB RAM, 2TB HDD	S/ 1.800,00	5	S/ 9.000,00
3	Lenovo ThimkSystem st50	Intel Xeon E-2124, 32GB RAM, 1TB SSD	S/ 2.500,00	5	S/ 12.500,00

Tabla N°4: Representa los modelos de los servidores que son considerados a utilizar con sus respectivas especificaciones, costo unitario, cantidad y el costo total.

Elección: Xampp es un paquete de software gratuito bajo la licencia GNU AGPL que incluye Apache, MariaDB, PHP y Perl, ideal para el desarrollo y pruebas de aplicaciones web. Es fácil de instalar y usar, lo que permite a los desarrolladores configurar rápidamente un entorno de servidor local. Su gran comunidad y abundante documentación facilitan la resolución de problemas y la implementación de soluciones.

7.1.6.2 Costos de Infraestructura

Las características de los equipos de cómputo con que se dispone actualmente para el desarrollo del sistema informático, se muestran a continuación:

EQUIPO	ELEMENTO	CAPACIDAD
Laptop1	Memoria RAM	8 GB
	Disco Solido	256 GB
	Procesador	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
	CD ROM	Sí
Laptop 2	Memoria RAM	16 GB
	Disco Duro	1TB
	Procesador	AMD RYZEN 7-7300

	CD ROM	Sí
Laptop 3	Memoria RAM	12 GB
	Disco Duro	512 GB
	Procesador	Intel Core I5 Duo
	CD ROM	Sí
Laptop4	Memoria RAM	4 GB
	Disco Duro	256 GB
	Procesador	Intel Core I3 Duo
	CD ROM	Sí
LAPTOP PRINCIPAL	Memoria RAM	32 GB
	Disco Duro	1TB
	Procesador	Intel Core i9-14900HX de 14 ^a generación
	Tarjeta Gráfica	NVIDIA GeForce RTX 4080
	Mouse, Teclado	Sí
	CD ROM	Sí

Tabla N°5: Se presentan los equipos de desarrollo que se utilizaran en el proyecto y las especificaciones de cada uno de ellos.

DESARROLLADORES	HERRAMIENTAS DE TRABAJO	COSTO
Cristian Olivera Chávez	Computadora de escritorio/laptop	4000
	Teclado y mouse:	150
	Silla ergonómica	600

	Software de desarrollo (IDE, licencias)	300
	Conexión a internet (mensual)	100
	Almacenamiento en la nube (mensual)	50
Yhury Cristian Anampa Quispe	Computadora de escritorio/laptop	3000
	Teclado y mouse:	150
	Silla ergonómica	800
	Software de desarrollo (IDE, licencias)	300
	Conexión a internet (mensual)	120
	Almacenamiento en la nube (mensual)	70
Luis Fernando Juarez Peña	Computadora de escritorio/laptop	3600

	Teclado y mouse:	200
	Silla ergonómica	750
	Software de desarrollo (IDE, licencias)	250
	Conexión a internet (mensual)	100
	Almacenamiento en la nube (mensual)	50
Giovanni totto Ccaccasto	computadora de escritorio/laptop	5000
	Teclado y mouse:	150
	Silla ergonómica	800
	Software de desarrollo (IDE, licencias)	300
	Conexión a internet (mensual)	120
Amilcar Virto Loayza	Computadora de escritorio/laptop	4500

	Teclado y mouse:	150
	Silla ergonómica	800
	Software de desarrollo (IDE, licencias)	300
	Conexión a internet (mensual)	120
COSTO TOTAL:		20880

Tabla N°6: Representa las herramientas de trabajo de cada uno de los desarrolladores junto con los precios

6.1.6.2.1. Salarios de los Personales

ITEM	NOMBRES	RANGO	SALARIO MENSUAL	SALARIO ANUAL
1	Cristian Olivera Chavez	Junior	1500	18000
2	Amilcar Virtu Loayza	Senior	3000	36000
3	Yhury Cristian Anampa	Backend	2000	24000
4	Luis Fernando Juarez Peña	Junior	1500	18000
5	Giovanni Tito Ccaccasto	Backend	2000	24000
TOTAL			10000	120000

Tabla N°7: Se observa los salarios personales de cada uno de los desarrolladores conjuntamente con el salario mensual y anual de acuerdo al rango respectivo.

6.1.6.2.2. Capacitacion de Personal

OPCIÓN	DETALLE	ESPECIFICACIONES	COSTO UNITARIO	CANTIDAD	COSTO TOTAL
1	Curso en Línea	Curso de 40 horas	200	5	1000
2	Taller Presencial	Curso de 3 días	1000	5	5000
3	Curso Mixto	Curso en línea más sesiones prácticas	500	5	2500

Tabla N°7: Se observa los detalles de la capacitación personal de los desarrolladores junto al costo de cada curso

6.1.6.3 Costos Iniciales de Implementación

ITEM	MODELO SELECCIONADO	COSTO ANUAL	CANTIDAD	COSTO TOTAL
Sistema operativo	Windows Server 2021	699	5	3495
Lenguaje de desarrollo	Java	GNU AGPL	5	GNU AGPL
Base de Datos	SQL Server	12000	5	60000
Servidores	Xampp	GNU AGPL	5	GNU AGPL
Capacitación del Personal por hora	Taller Mixto	-	5	2500
Infraestructura	-	-	5	20880
TOTAL DE COSTOS INICIALES				86875

Tabla N°8: Identifica los costos de Implementación de los elegidos para el desarrollo del proyecto

6.1.6.4 Costos de Mantenimiento y Soporte

ITEM	DETALLE	COSTO ANUAL
Salarios	5 desarrolladores	120000
Licencias de Software	Renovación anual de Office 365	900

Mantenimiento de Servidores	Mantenimiento y actualización	10000
Mantenimiento de Red	Soporte y mantenimiento	5000
Infraestructura	Alquiler de oficina, servicios públicos, internet, otras herramientas	50000
Otros Costos Operativos	Suministros de oficina, marketing, seguros	55800
Costos Totales Operativos		241700

Tabla N°9: Se observa los costos de mantenimiento y soporte a detalle del proyecto junto al costo anual de mantenimiento.

6.1.6.5 Total Estimado de Costos de software

TOTAL DE COSTOS INICIALES				110255
Costos Totales Operativos				241700
TOTAL ESTIMACIÓN DE COSTOS				351955

Tabla N°10: Identifica el total estimado de costos para la realización del software.

7.1.6.6 Ingresos Anuales Proyectados

FUENTES DE INGRESO	cantidad	costo
venta de pizzas personales	10800	216000
venta de pizzas medianas	14400	432000
venta de pizzas personales	18000	720000
Otros Ingresos	bebidas, extras	180000
Total Ingresos		1548000

Tabla N°11: Se aprecia todos los ingresos anuales proyectados de la empresa.

7.1.6.7. Inversión inicial de pizzeria

Descripción	Compra/Alquiler
horno	17300
muebles	80230
utensilios de mesa	10000
equipos electrónicos	28410
letrero/publicidad	600
máquina de cocina	115000
total	-251540

Tabla N°12: Identifica la inversión inicial de la empresa.

7.1.6.8. Inversión variable-anual de pizzeria

Descripción	costo
personales	36000
servicios básicos	15000
alquiler local	54000
ingredientes	602050
total	707050

Tabla N°13: Se observa la inversión variable-anual de la empresa

7.1.6.6. Análisis de flujo de caja anual

MOVIMIENTO	ANALISIS DE FLUJO DE CAJA										
	0	1	2	3	4	5	6	7	8	9	10
INGRESOS		15480 00									
INVERSIÓN INICIAL	-251540										
INVERSIÓN VARIABLE-ANUAL	-707050	-7070 50									
VENTA DE AMASADOR A						10690					
VENTA DE LICUADOR A						7810					
VENTA DE EQUIPOS ELECTRÓNICOS						14205					

VENTA DE MUEBLES						40115					
MANTENIMINETO DE HORNO						-5000					
COSTO DE SOFTWARE	-351955										
MANTENIMIENTO DE LETRERO			-300		-300		-300		-300		-300
FLUJO DE CAJA		84095 0	84065 0	84095 0	84065 0	90877 0	84065 0	84095 0	84065 0	84095 0	84065 0
FLUJO DE CAJA ACUMULADO	-131054 5	84095 0	16816 00	25225 50	33632 00	42719 70	51126 20	59535 70	67942 20	76351 70	84758 20

Tabla N°14: Análisis de flujo de caja del proyecto.

7.1.6.7. Análisis de viabilidad a 10 años.

inversión	1310545		VAN	649016,001 2
tasa de descuento	90%		TIR	15%
nro de periodos o años	10			

Tabla N°15: Análisis de viabilidad a 10 años de la empresa

Tabla N°16. Análisis del VAN y TIR de la empresa

inversión	-131054 5	84095 0	168160 0	25225 50	33632 00	42719 70	51126 20	59535 70	67942 20	76351 70	84758 20
		1,90	3,61	6,86	13,03	24,76	47,05	89,39	169,84	322,69	613,11
inversión	-131054 5	44260 5,26	465817 ,17	36777 2,27	25807 0,46	17252 8,24	10867 3,06	66604, 30	40004, 68	23661, 17	13824, 38

Tabla N°17: Inversión total para la realización del proyecto.

TASA DE DESCUENTO	VAN
0%	45.341.125,00
10%	23.305.074,53
20%	13.159.051,97
30%	7.983.389,43
40%	5.103.901,16
50%	3.379.773,04

60%	2.281.278,49
70%	1.543.719,78
80%	1.026.104,41
90%	649.016,00
100%	365.474,30

Tabla N°18: Tasa de descuento y análisis VAN

6.1.3 Requisitos Específicos:

6.1.3.1 Requisitos Funcionales:

RF 100: Requisitos funcionales				
Identificación del Requerimiento	Nombre del Requerimiento	Características	Descripción del Requerimiento	Prioridad del Requerimiento
Área de Gestión de Pedidos				
RF101	Registro de Pedidos	- Permitir a los mozos registrar los pedidos de manera digital	El sistema debe permitir a los mozos registrar los pedidos de forma digital, incluyendo detalles específicos como tamaño e ingredientes.	Alta
RF102	Consulta de Pedidos	- Consultar el estado de los pedidos en tiempo real	Permite a mozos y cajeros consultar el estado de los pedidos en tiempo real, mostrando su estado (pendiente, en preparación, listo, entregado).	Alta
RF103	Modificación de Pedidos	- Modificar pedidos antes de ser preparados	Permite modificar pedidos antes de ser preparados, incluyendo la opción de	Alta

			ajustar detalles como ingredientes o cantidades.	
RF104	Listado de Pedidos	- Listado filtrado por estado	Proporciona un listado de todos los pedidos realizados, con opciones para filtrar por estado (pendiente, en preparación, listo, entregado) y tiempo de espera.	Alta
RF105	Cancelación de Pedidos	- Cancelación de pedidos y registro del motivo	Permite cancelar pedidos en caso de errores o solicitudes del cliente, registrando el motivo de la cancelación.	Alta
RF106	Confirmación de Pedido	- Solicitar confirmación del pedido antes de enviarlo a la cocina	Muestra un resumen del pedido para que el mozo lo confirme antes de enviarlo a la cocina.	Alta
RF Para Módulos Generales				
RF107	Inicio de Sesión	- Formularios para correo y contraseña - Botón para iniciar sesión	Contiene formularios para ingresar correo y contraseña, y un botón para iniciar sesión en el sistema.	Alta
RF108	Agregar Salas y Mesas	- Formularios para agregar número de salas y mesas	Permite agregar el número de salas y mesas dentro de ellas mediante formularios específicos.	Alta
RF109	Selección de Sala	- Opción para seleccionar sala y	Permite seleccionar una sala y ver las mesas numeradas dentro de ella.	Alta

		visualizar mesas enumeradas		
RF110	Selección de Mesa	- Selección de mesa dentro de una sala	Permite seleccionar una mesa específica dentro de la sala previamente seleccionada para agregar productos.	Alta
RF111	Selección de Producto	- Visualización de productos del menú	Permite seleccionar productos disponibles en el menú de la pizzería.	Alta
RF112	Agregar Producto	- Agregar uno o más productos a una mesa	Permite agregar uno o más productos a la mesa seleccionada, incluyendo ajustes de cantidad.	Alta
RF113	Datos del Trabajador	- Visualización de datos y contraseña encriptada	Muestra los datos del trabajador y la contraseña encriptada, asegurando que la contraseña tenga más de 8 caracteres.	Media
RF114	Datos de la Empresa	- Visualización y edición de datos de la empresa	Permite ver y actualizar los datos de la empresa, como número de teléfono y dirección.	Media
RF115	Historial de Pedidos	- Visualización del estado de pedidos y precios	Muestra el estado de cada pedido y verifica los precios, incluyendo el estado de las mesas (ocupada o desocupada).	Alta
RF116	Cierre de Mesa	- Cierre de mesa y reporte en formato PDF	Permite cerrar la mesa cuando el cliente ha pagado, generando un	Alta

			reporte de nota de venta en formato PDF descargable.	
RF117	Funcionalidades para Mozos	- Inicio de sesión - Gestión de mesas y pedidos - Generación de reportes	Incluye inicio de sesión, acceso a mesas, registro y gestión de pedidos, interfaz amigable, y generación de reportes y notas de venta.	Alta
RF118	Funcionalidades para Cajeros	- Inicio de sesión - Gestión de mesas y pedidos - Optimización del proceso de cobro - Cierre automatizado de mesas	Incluye funcionalidades para el manejo de mesas y pedidos, optimización del cobro, y cierre automatizado de mesas.	Alta

Tabla N°19: Representa los requisitos funcionales especificando el nombre de requerimiento, características, descripción del requerimiento y la prioridad del requerimiento.

6.1.3.2 Requisitos No Funcionales

Identificación del Requerimiento	Nombre del Requerimiento	Descripción del Requerimiento	Prioridad del Requerimiento
RNF101	Usabilidad	Interfaz intuitiva y fácil de usar para todos los roles (mozos, cajeros, gerentes) con entrenamiento mínimo requerido.	Alta

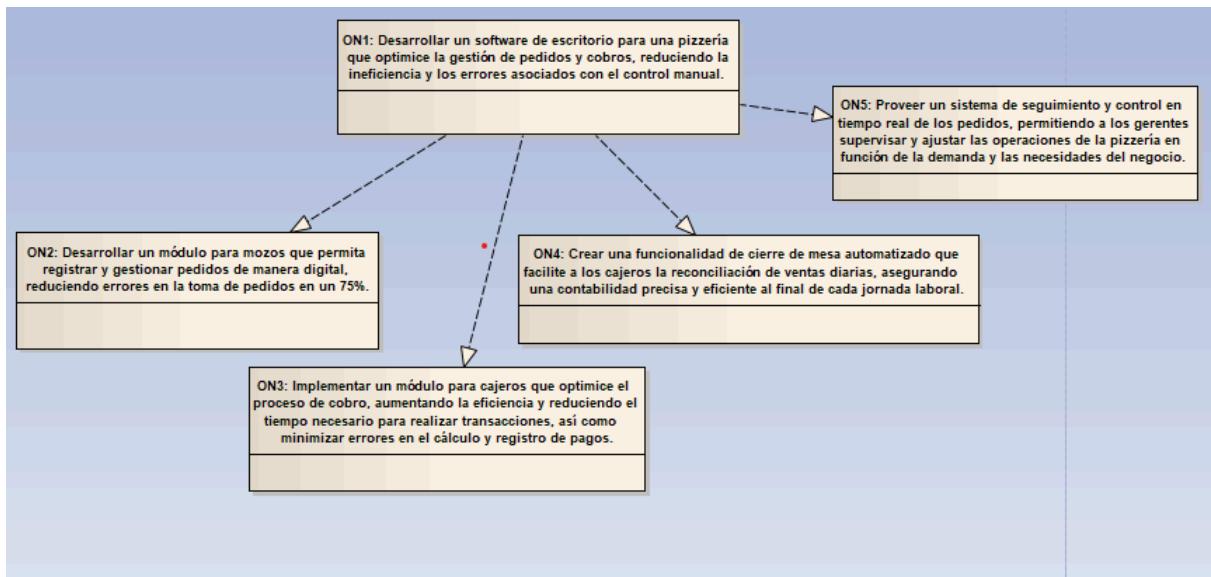
RNF102	Rendimiento	Respuesta rápida del sistema incluso en momentos de alta demanda (por ejemplo, horas pico).	Alta
RNF103	Seguridad	Cumplimiento con estándares de seguridad de datos y protección de información sensible de clientes y transacciones.	Alta
RNF104	Disponibilidad	Disponibilidad del sistema 24/7 con mínimo tiempo de inactividad programada para mantenimiento.	Alta
RNF105	Escalabilidad	Capacidad para manejar el crecimiento del negocio y la adición de nuevas funcionalidades sin afectar el rendimiento.	Alta
RNF106	Compatibilidad	Compatibilidad con diversos dispositivos y sistemas operativos para asegurar	Alta

		accesibilidad desde múltiples plataformas.	
RNF107	Documentación	Documentación completa y actualizada del software para soporte técnico y entrenamiento del personal.	Media
RNF108	Cumplimiento Legal	Cumplimiento con regulaciones locales y nacionales relacionadas con la gestión de restaurantes y protección de datos.	Alta

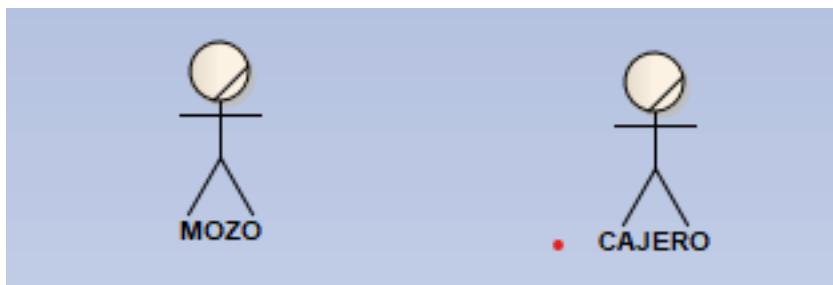
Tabla N°20. Representa los requisitos No Funcionales identificando la identificación del requerimiento, nombre de requerimiento, descripción del requerimiento y prioridad del requerimiento.

6.1.4 Modelado de negocios (ARCHITEC):

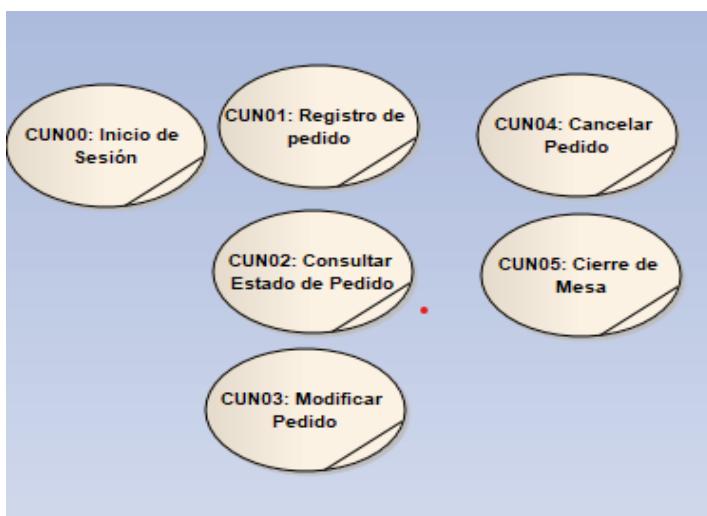
6.1.4.1 Objetivos de negocio :



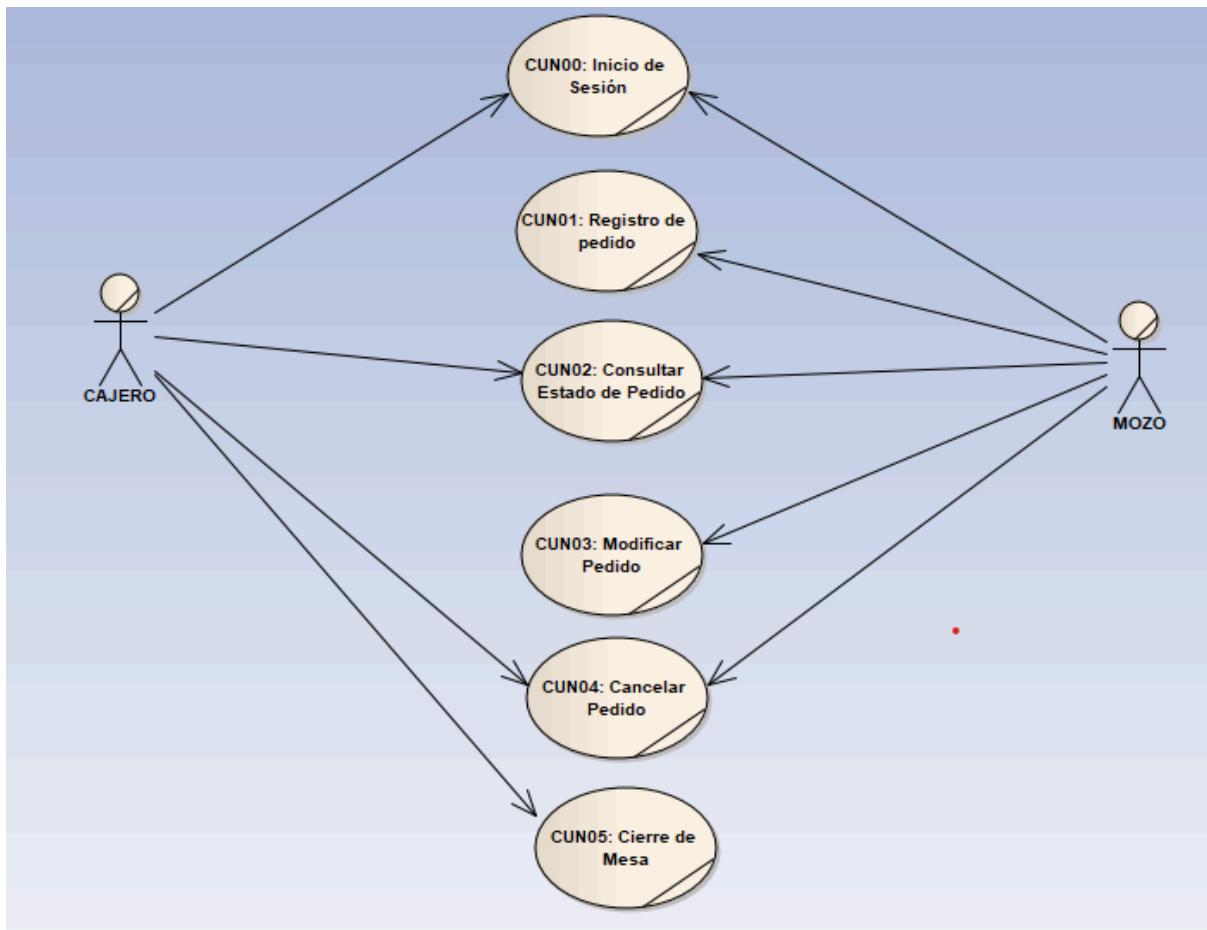
6.1.4.2 Actores de negocio:



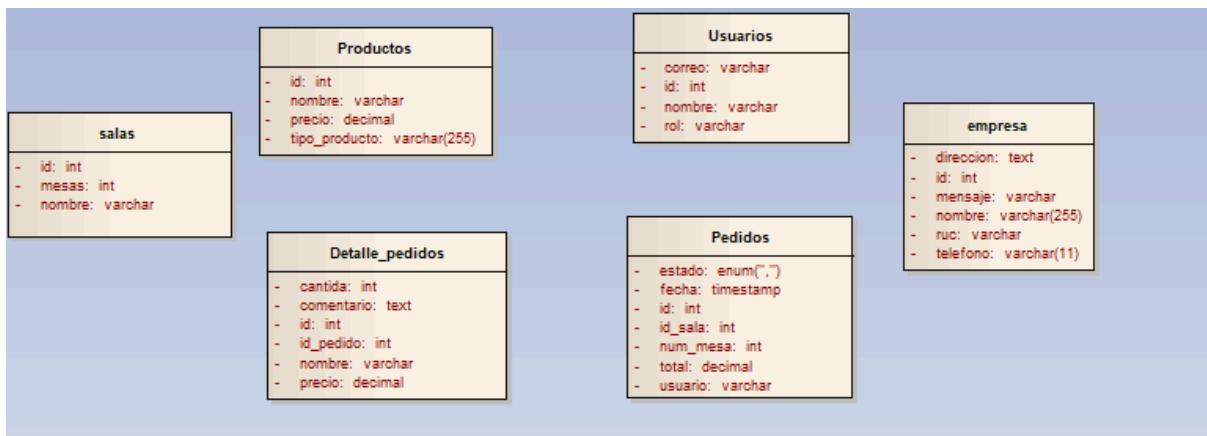
6.1.4.3 Casos de uso de negocio:



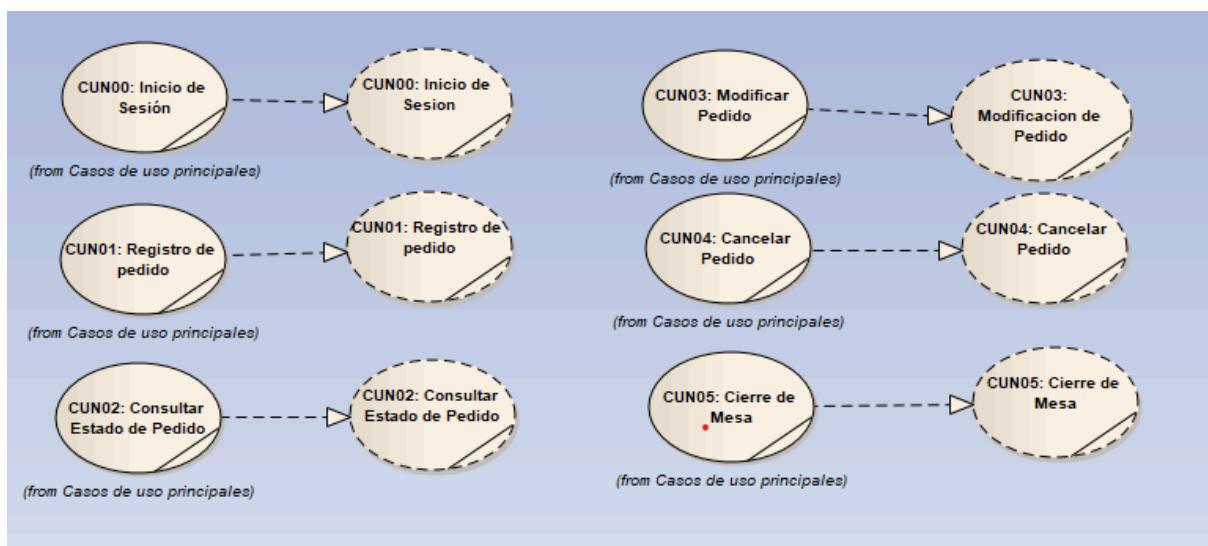
6.1.2.4 Modelado de caso de uso de negocio (MCUN):



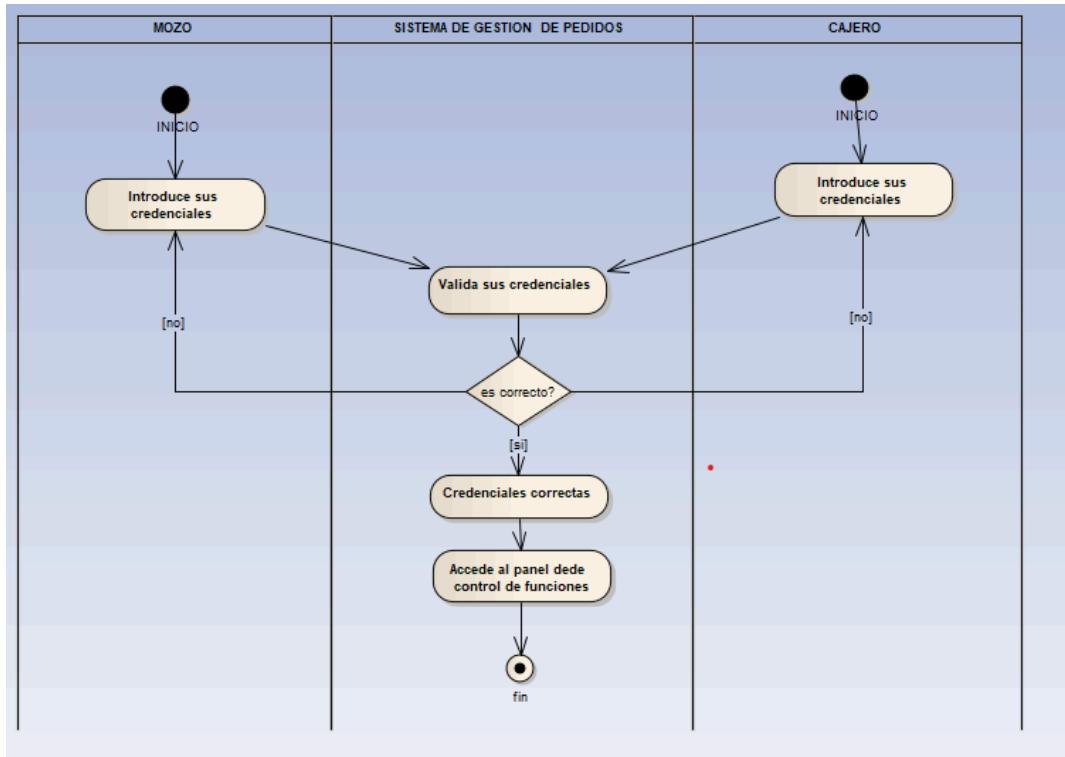
6.1.2.5 Entidades de negocio:

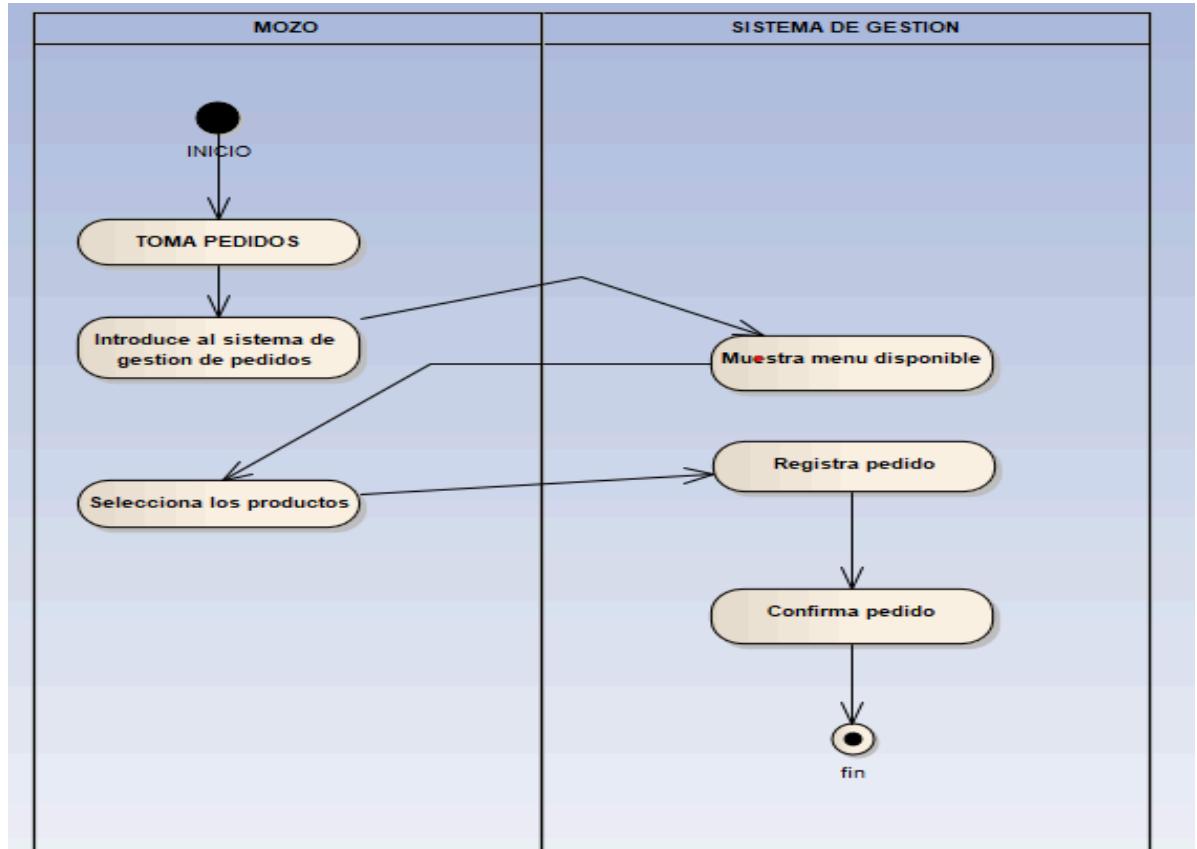


6.1.2.6 Realización de casos de uso de negocio :

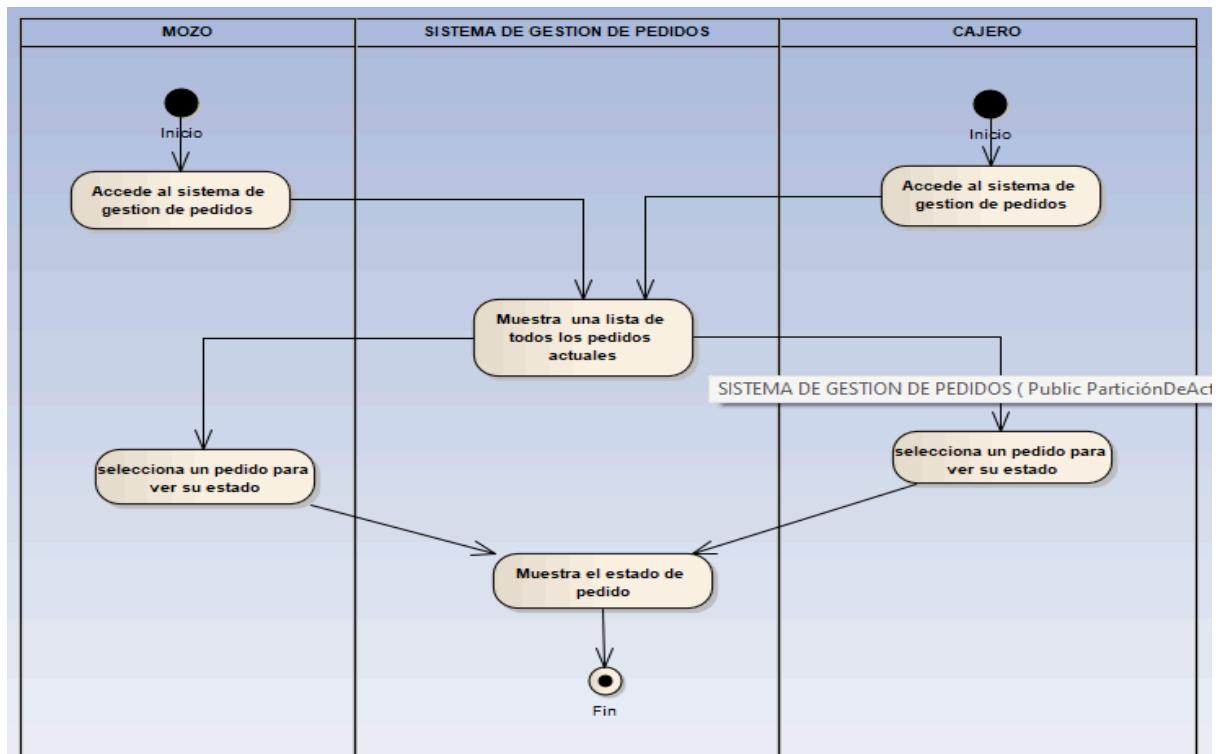


6.1.2.6.1 Inicio de sesión:

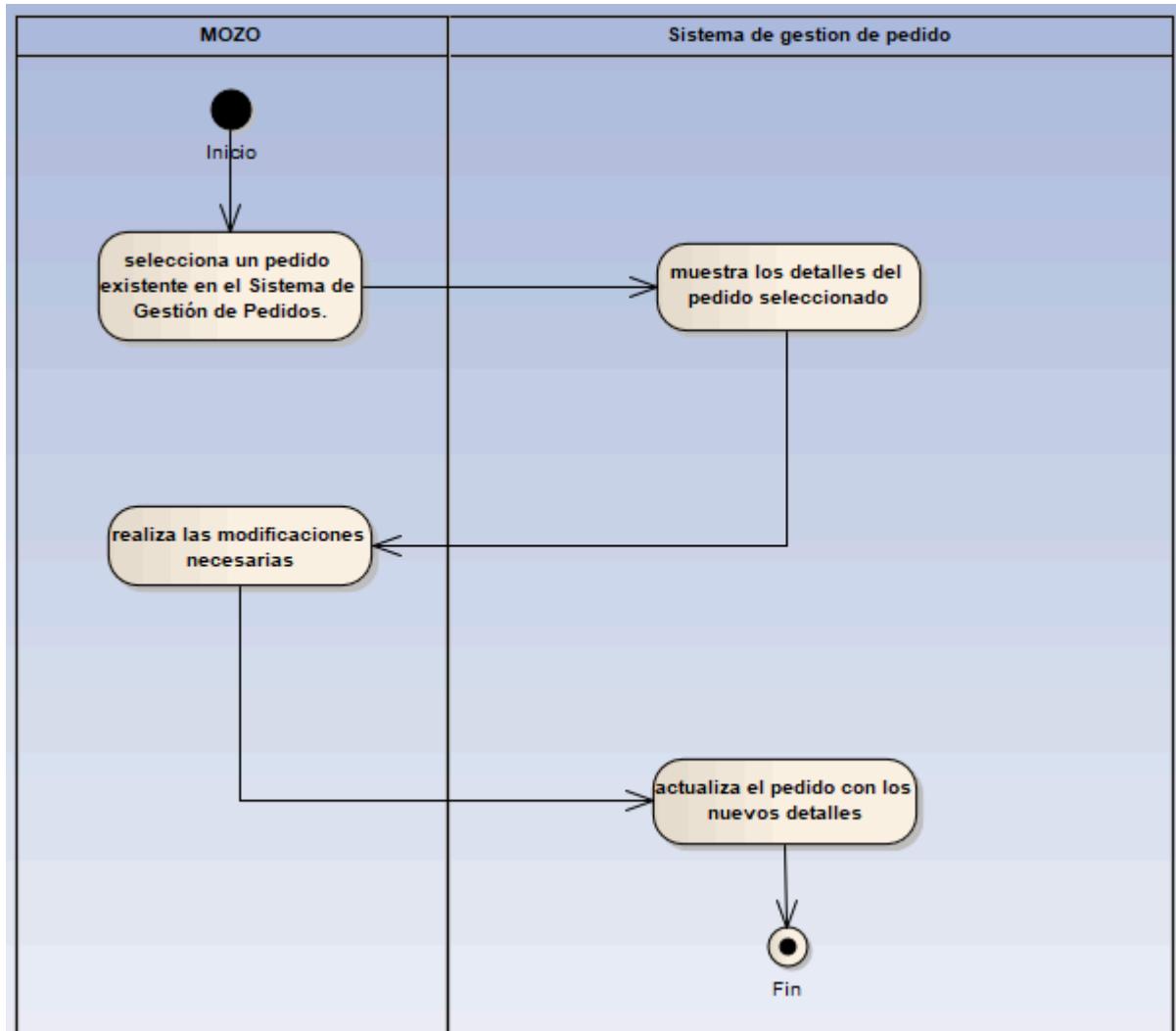




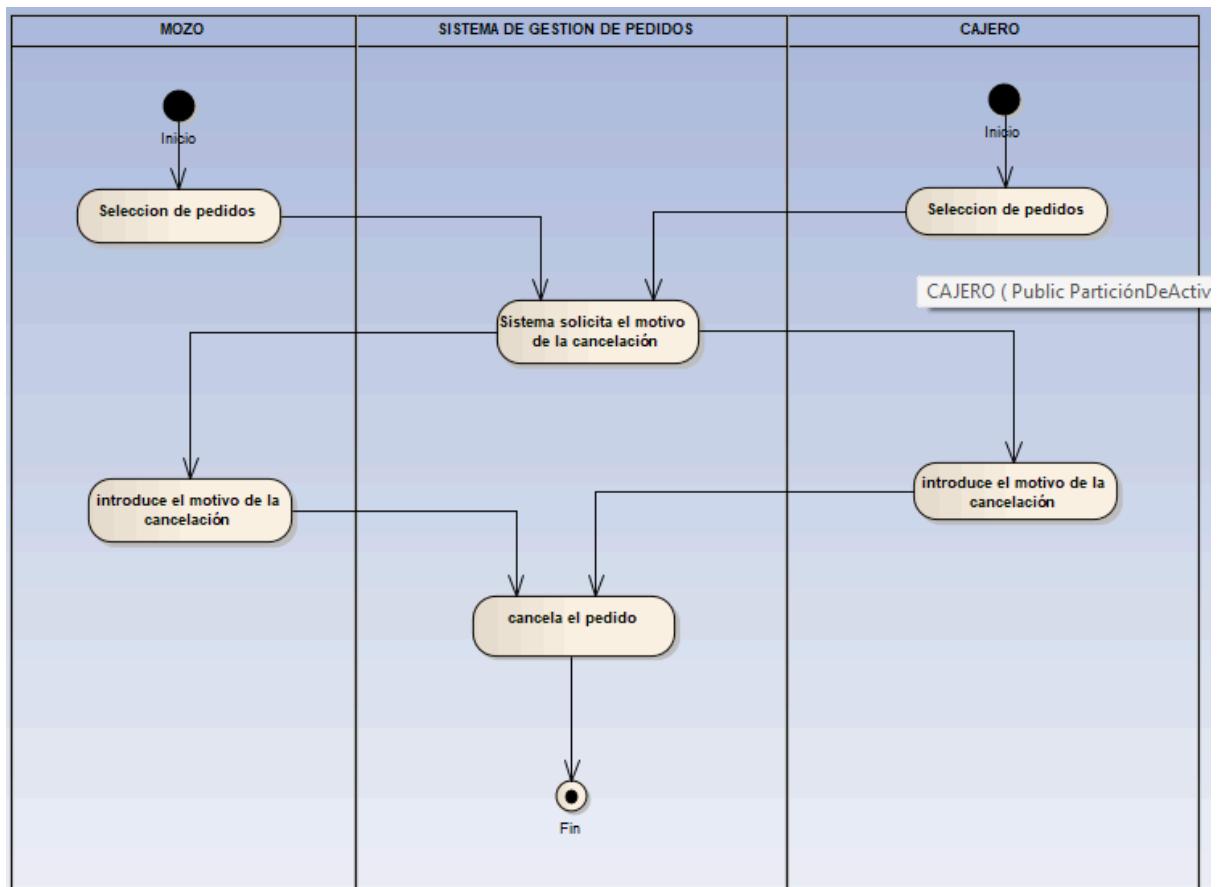
6.1.2.6.3 Consultar Estado de Pedido



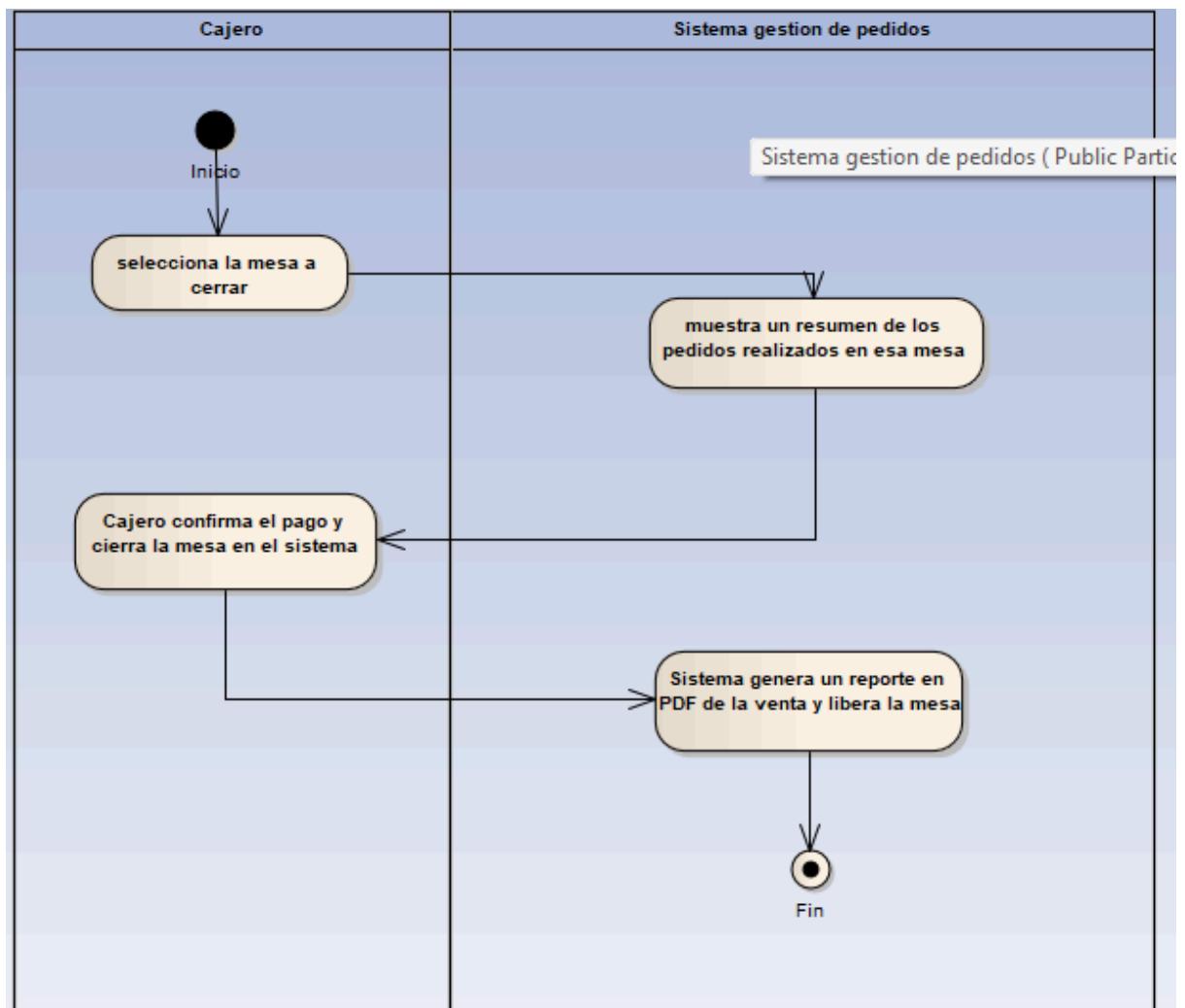
6.1.2.6.4 Modificación de Pedido



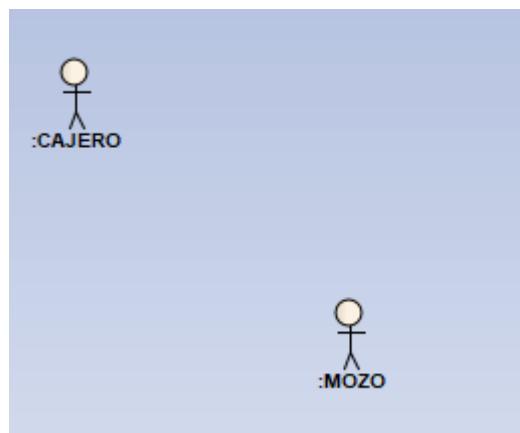
6.1.2.6.5 Cancelar Pedido



6.1.2.6.6 Cierre de Mesa



6.1.4.7 Trabajadores de negocio



6.1.5 Validación y Gestión de Requisitos:

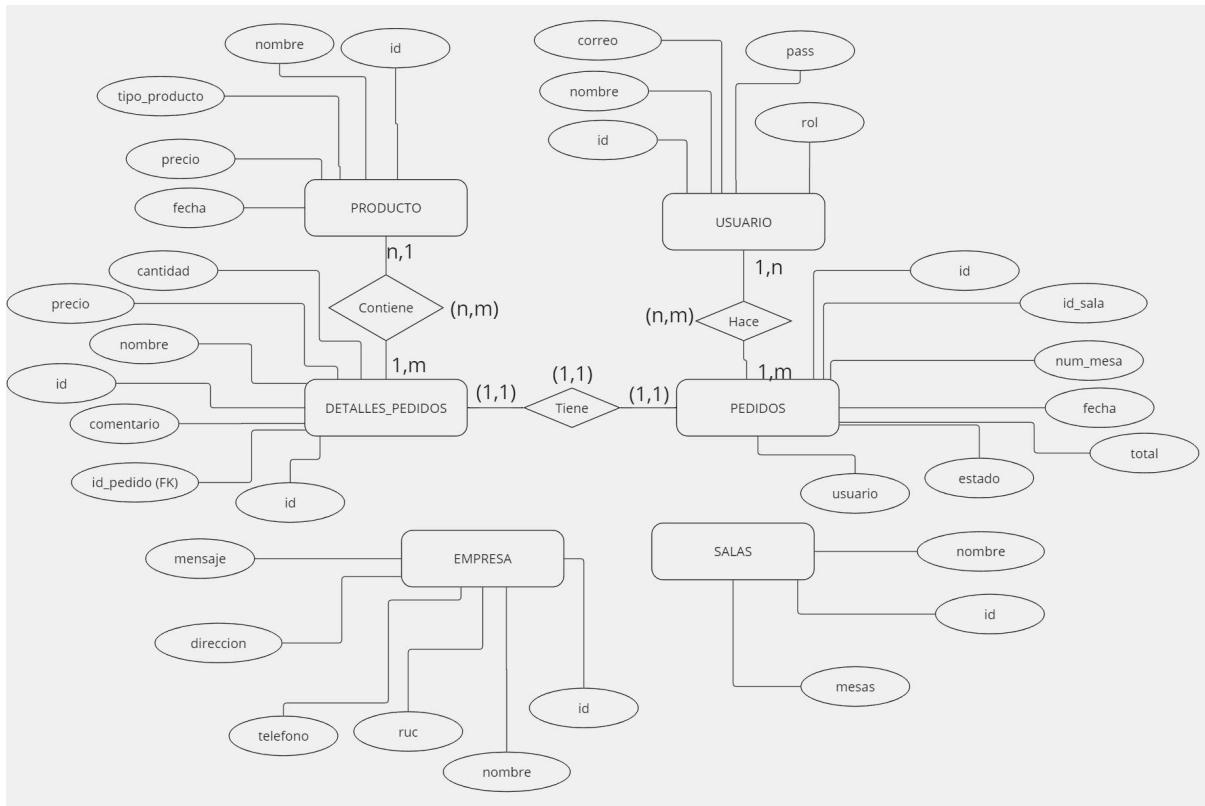
	Identificación del Requerimiento	Validación	Criterios de Aceptación	Estado
RNF:000	RNF001	Revisado	Interfaz evaluada por usuarios Feedback positivo de usabilidad	Aprobado
	RNF002	Revisado	Pruebas de rendimiento realizadas - Respuesta rápida bajo carga	Aprobado
	RNF003	Revisado	Auditoría de seguridad Cumplimiento con estándares	Aprobado
	RNF004	Revisado	Pruebas de disponibilidad Mínimo tiempo de inactividad	Aprobado
	RNF005	Revisado	Pruebas de escalabilidad sistema soporta crecimiento	Aprobado
	RNF006	Revisado	Pruebas en múltiples dispositivos y SO Acceso desde diversas plataformas	Aprobado
	RNF007	Revisado	Documentación completa Actualizaciones regulares	Aprobado
	RNF008	Revisado	Cumplimiento con regulaciones Auditoría legal	Aprobado

Tabla N°21: Representa la validación y gestión de los requerimientos no funcionales, identificando el requerimiento, validación, los criterios de aceptación y el estado en el que se encuentra.

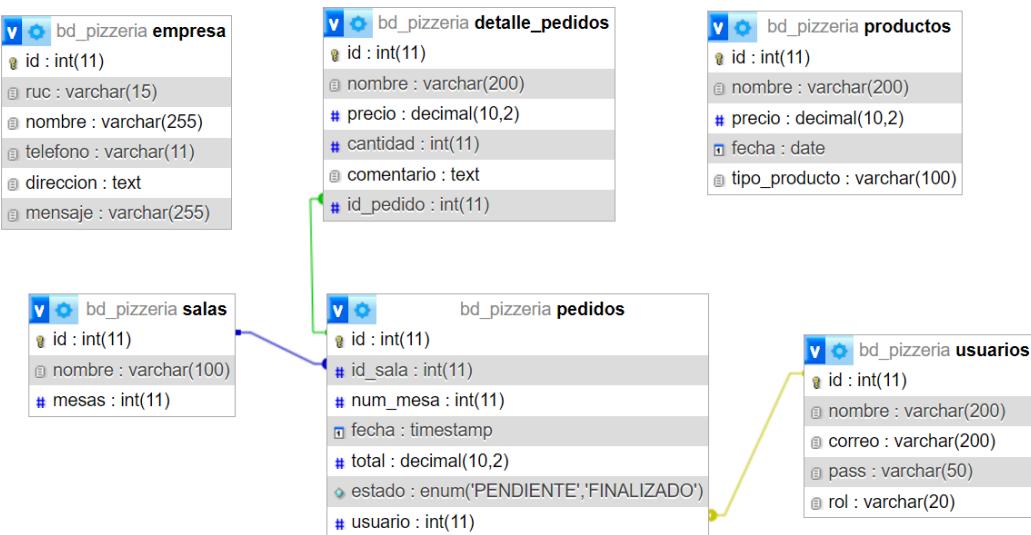
6.2 Fase de Diseño:

6.2.1 Base de datos

6.2.1.1 Diseño de base de datos modelo Entidad-Relación (ER).



6.2.1.2 Diseño de base de datos modelo lógico.



6.2.1.3 Diseño de base de datos modelo físico.

1. Crear la tabla salas

```
CREATE TABLE salas (
    id INT(11) NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    mesas INT(11) NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci;
```

2. Crear la tabla usuarios (debe ser creada antes que la tabla pedidos)

```
CREATE TABLE usuarios (
    id INT(11) NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(200) NOT NULL,
    correo VARCHAR(200) NOT NULL,
    pass VARCHAR(50) NOT NULL,
    rol VARCHAR(20) NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci;
```

3. Crear la tabla pedidos

```
CREATE TABLE pedidos (
    id INT(11) NOT NULL AUTO_INCREMENT,
    id_sala INT(11) NOT NULL,
    num_mesa INT(11) NOT NULL,
    fecha TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    total DECIMAL(10,2) NOT NULL,
    estado ENUM('PENDIENTE', 'FINALIZADO') NOT NULL DEFAULT
'PENDIENTE',
    usuario INT(11) NOT NULL,
```

```

PRIMARY KEY (id),
KEY id_sala (id_sala),
KEY usuario (usuario),
CONSTRAINT usuario_fk FOREIGN KEY (usuario) REFERENCES usuarios(id),
CONSTRAINT pedidos_ibfk_1 FOREIGN KEY (id_sala) REFERENCES salas(id)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci;

```

4. Crear la tabla detalle_pedidos

```

CREATE TABLE detalle_pedidos (
    id INT(11) NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(200) NOT NULL,
    precio DECIMAL(10,2) NOT NULL,
    cantidad INT(11) NOT NULL,
    comentario TEXT DEFAULT NULL,
    id_pedido INT(11) NOT NULL,
    PRIMARY KEY (id),
    KEY id_pedido (id_pedido),
    CONSTRAINT detalle_pedidos_ibfk_1 FOREIGN KEY (id_pedido)
    REFERENCES pedidos(id)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci;

```

5. Crear la tabla empresa

```

CREATE TABLE config (
    id INT(11) NOT NULL AUTO_INCREMENT,
    ruc VARCHAR(15) NOT NULL,
    nombre VARCHAR(255) NOT NULL,
    telefono VARCHAR(11) NOT NULL,
    direccion TEXT NOT NULL,
    mensaje VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci;

```

6. Crear la tabla productos

```
DROP TABLE IF EXISTS productos;
CREATE TABLE platos (
    id INT(11) NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(200) NOT NULL,
    precio DECIMAL(10,2) NOT NULL,
    fecha DATE DEFAULT NULL,
    tipo_producto varchar(100),
    PRIMARY KEY (id)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci;
```

6.2.1.4 Diccionario de base de datos.

a) Tabla detalle_pedidos

Columna	Tipo	Nulo	Predeterminado	Comentarios			
id (<i>Primaria</i>)	int(11)	No		identificador de un detalle de pedido			
nombre	varchar(200)	No		indica el nombre del producto pedido			
precio	decimal(10,2)	No		indica el precio por unidad			
cantidad	int(11)	No		es la cantidad de producto pedido			
comentario	text	Sí	NULL	detalle de como quiere el producto			
id_pedido	int(11)	No		identificador de la tabla pedido			

Índices

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	id	0	A	No	
id_pedido	BTREE	No	No	id_pedido	0	A	No	

b) Tabla empresa

Columna	Tipo	Nulo	Predeterminado	Comentarios			
id (<i>Primaria</i>)	int(11)	No		identificador de la empresa			
ruc	varchar(15)	No		indica el ruc de la empresa			
nombre	varchar(255)	No		indica el nombre de la empresa			
telefono	varchar(11)	No		indica el telefono de la empresa			
direccion	text	No		indica la direccion de la empresa			
mensaje	varchar(255)	No		indica el slogan de la empresa			

Índices

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	id	0	A	No	

c) Tabla pedidos

Columna	Tipo	Nulo	Predeterminado	Comentarios			
id (<i>Primaria</i>)	int(11)	No		es el codigo o identificador de un pedido			
id_sala	int(11)	No		indica el numero de sala e identificador de sala			
num_mesa	int(11)	No		indica el numero de mesa en que se realiza el pedido			
fecha	timestamp	No	current_timestamp()	es la fecha de realizacion de pedido			
total	decimal(10,2)	No		es el total a pagar			
estado	enum('PENDIENTE', 'FINALIZADO')	No	PENDIENTE	indica si el producto esat pendiente en entregar o finalizado			
usuario	int(11)	No		indica el codigo de trabajador quien atiende a la mesa			

Índices

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	id	0	A	No	
id_sala	BTREE	No	No	id_sala	0	A	No	
usuario	BTREE	No	No	usuario	0	A	No	

d) Tabla productos

Columna	Tipo	Nulo	Predeterminado	Comentarios			
id (<i>Primaria</i>)	int(11)	No		es el codigo que identifica el producto			
nombre	varchar(200)	No		es el nombre del producto que se ofrece			
precio	decimal(10,2)	No		indica el precio unitario de cada producto que se ofrece			
fecha	date	Sí	NULL	indica la fecha			
tipo_producto	varchar(100)	Sí	NULL	indica que tipo de producto es, pizza o bebida			

Índices

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	id	0	A	No	

e) Tabla salas

Columna	Tipo	Nulo	Predeterminado	Comentarios			
id (<i>Primaria</i>)	int(11)	No		es el identificador de una sala			
nombre	varchar(100)	No		es el nombre de la sala, puede ser por pisos o patios, etc			
mesas	int(11)	No		indica el identificador de mesas o numero de mesas			

Índices

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	id	0	A	No	

f) Tabla usuarios

Columna	Tipo	Nulo	Predeterminado	Comentarios			
id (<i>Primaria</i>)	int(11)	No		es el identificador de un trabajador o de quien interactua con el sistema			
nombre	varchar(200)	No		es el nombre del usuario quien interactua con el sistema			
correo	varchar(200)	No		es el correo de quien interactua con el sistema, con este podra iniciar sesion			
pass	varchar(50)	No		es la contraseña para poder iniciar sesion			
rol	varchar(20)	No		es el rol de quien interactua con el sistema			

Índices

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo	Comentario
PRIMARY	BTREE	Sí	No	id	0	A	No	

6.2.2 Diseño de la interfaz de usuario.

El diseño de la interfaz gráfica del usuario se realizó en la página online de Figma.

Figma es una herramienta de diseño colaborativo basada en la nube que permite a los equipos crear, prototipar y compartir interfaces de usuario y experiencias visuales en tiempo real. Facilita la colaboración simultánea, lo que la hace ideal para equipos de

diseño distribuidos. Su integración con otras herramientas y su facilidad de uso la convierten en una opción popular para diseñadores y desarrolladores.

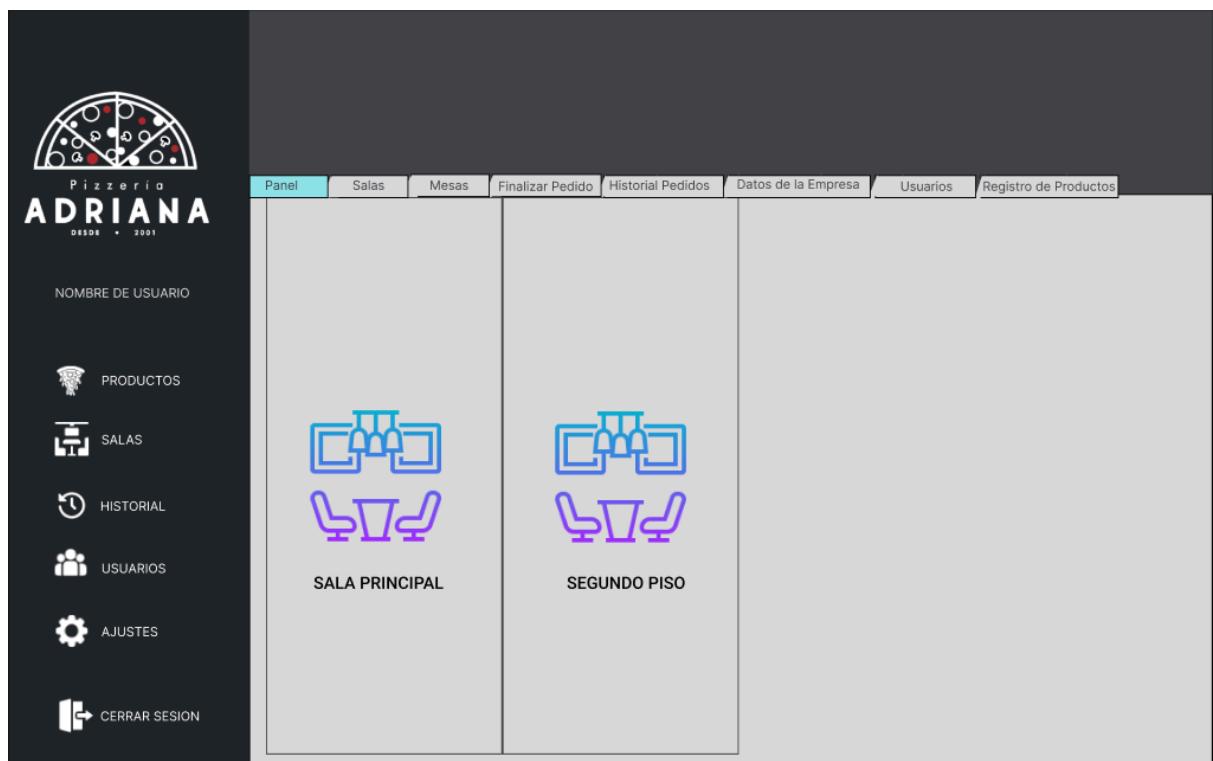
6.2.2.1 Login de Usuario

Inicio de sesión del usuario



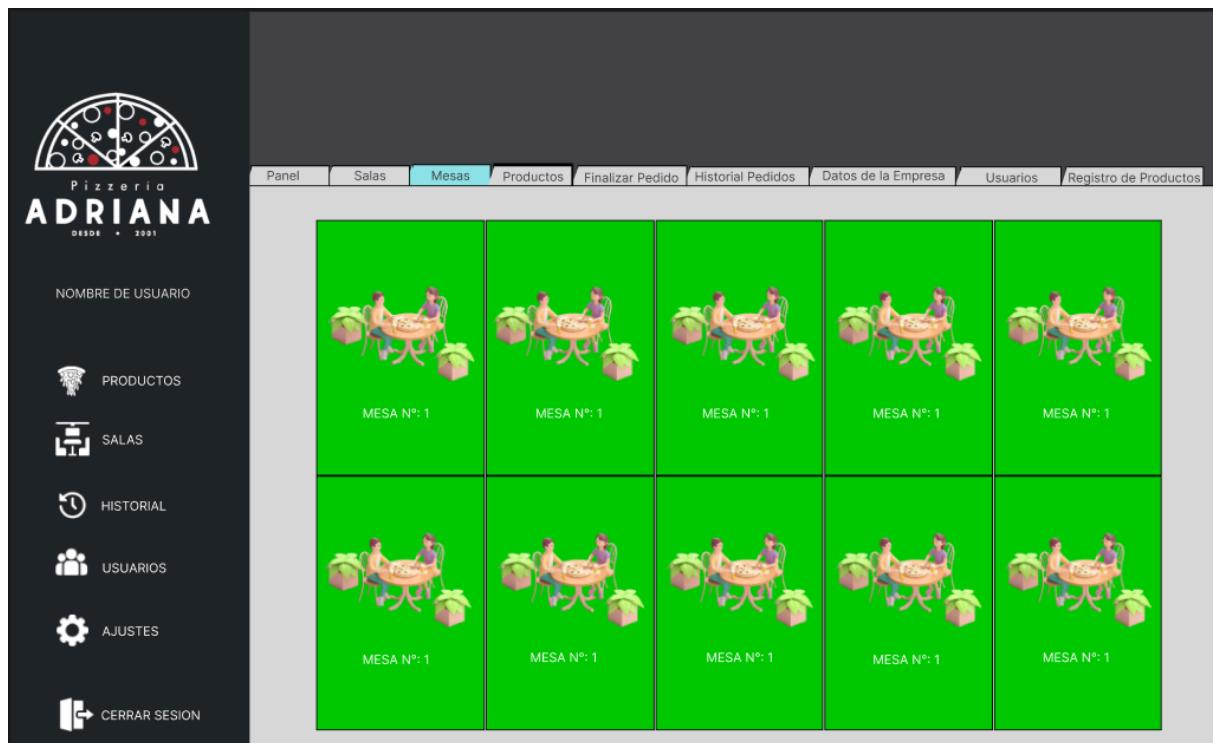
6.2.2.2 Interfaz de Salas

La interfaz de salas muestra las salas a escoger por el usuario



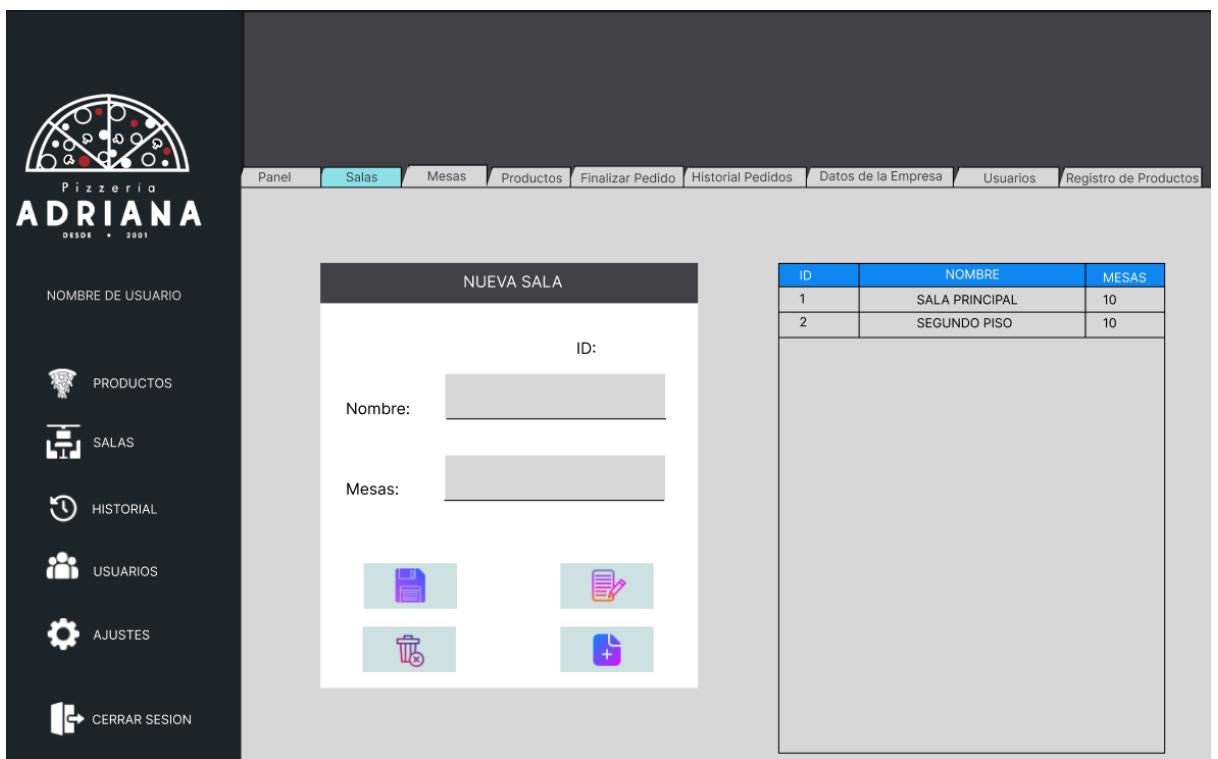
6.2.2.3. Interfaz de Mesas

Muestra las mesas a escoger por el usuario



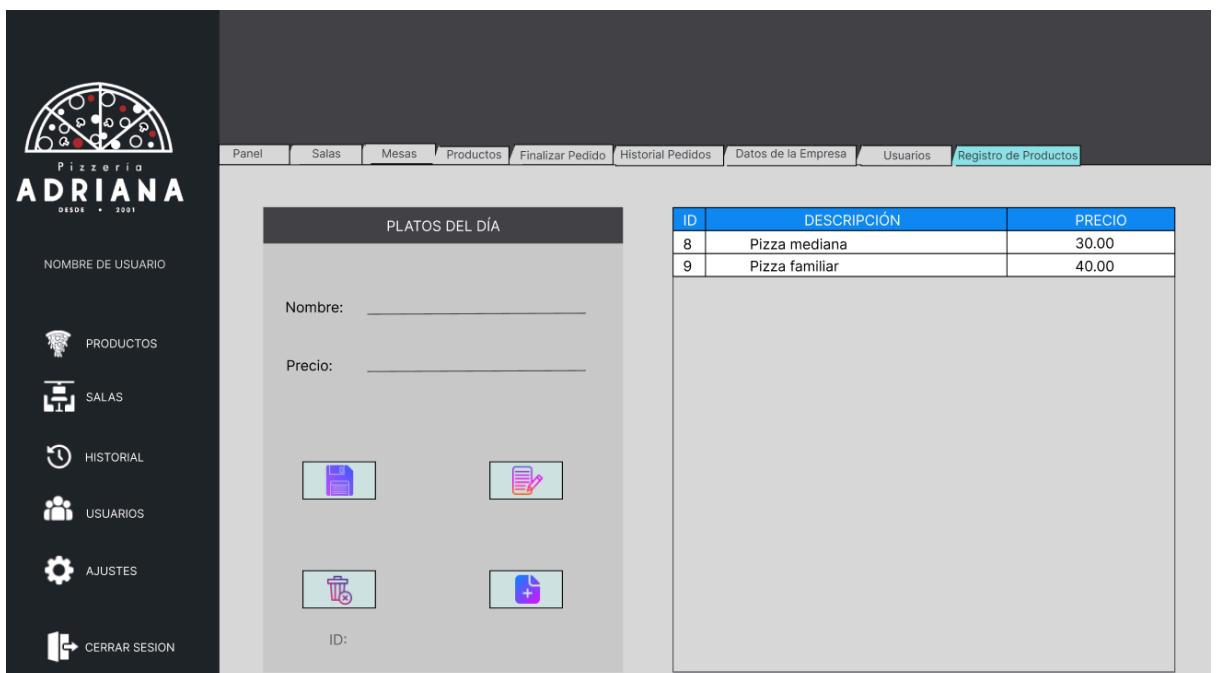
6.2.2.4. Interfaz de Registro de Salas y Mesas

El usuario podrá elegir la sala y el número de mesa.



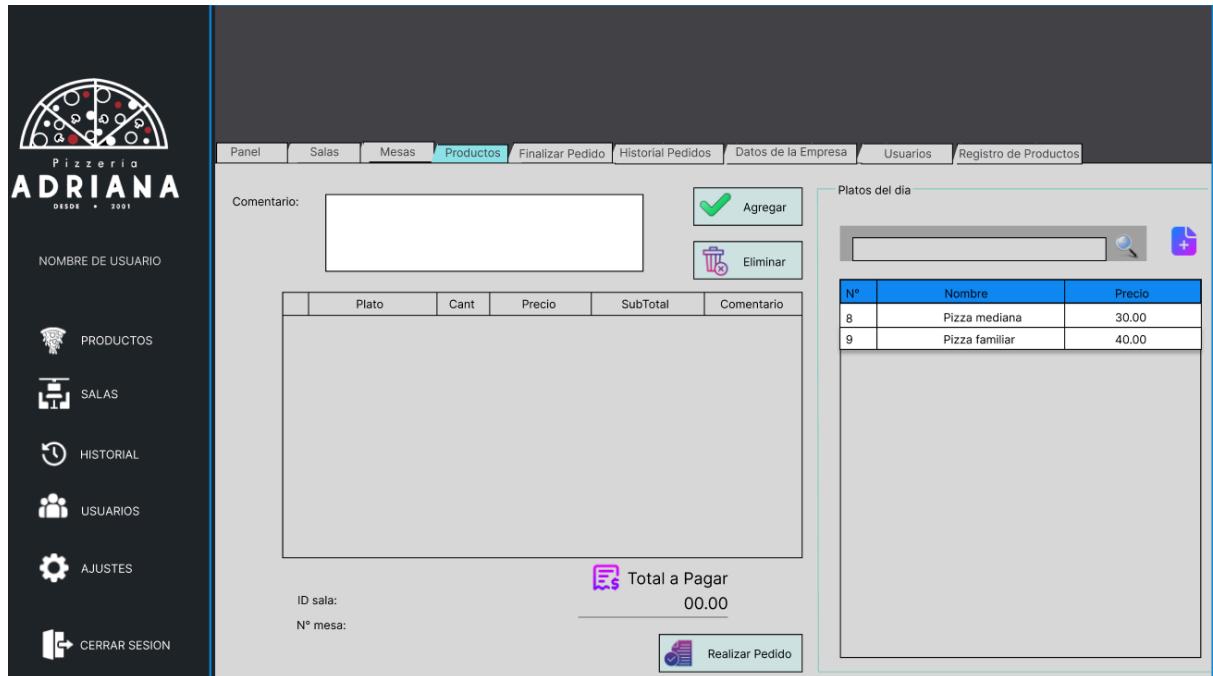
6.2.2.5 Interfaz de Registro de Productos

Permite registrar el nombre y precio de los productos a vender, además se puede guardar, editar, eliminar y agregar los productos.



6.2.2.6. Interfaz de Registro de Pedidos para el Cliente

Después de registrar los productos a vender, se registran los productos en la interfaz de pedidos del cliente, que permite agregar y eliminar comentarios adicionales a los pedidos realizados, también calcula el precio a pagar.



6.2.2.7. Interfaz de Historial de Pedidos

Después de haber registrado los pedidos del cliente, esta interfaz permite visualizar todo el historial de pedidos junto con el id, sala, nombre del mozo, N° de mesa, fecha, el precio total y el estado del pedido.

ID	Sala	Atendido	Nº Mesa	Fecha	Total	Estado
2	SEGUNDO PISO	ANGEL SIFUENTES	8	2024-08-21 15:47:54	30.00	FINALIZADO
3	SALA PRINCIPAL	ANGEL SIFUENTES	9	2024-08-21 15:47:54	28.00	FINALIZADO
4	SALA PRINCIPAL	ANGEL SIFUENTES	11	2024-08-21 15:47:54	20.00	PENDIENTE
1	SALA PRINCIPAL	ANGEL SIFUENTES	2	2024-08-21 15:47:54	78.00	FINALIZADO

6.2.2.8. Interfaz de Finalizar Pedido.

Al seleccionar un pedido en la interfaz de historial de pedidos, ingresamos a la interfaz de finalizar pedido, en la que nos muestra todos los detalles del pedido hecho por el cliente, también tiene la opción de Finalizar Pedido siempre y cuando se selecciona un pedido “PENDIENTE” y también hay la opción de imprimir el documento PDF de la cuenta.

Nº	Plato	Cantidad	Precio	SubTotal	Comentario
1	Pizza Mediana	1	30.00	30.00	
2	Pizza Familiar	1	40.00	40.00	

Fecha y Hora: 2024-08-21 15:47:25
 Sala: SEGUNDO PISO
 N° Mesa: 8

Total a Pagar: 30.00
 ID pedido:
 ID historial:

6.2.2.9. Interfaz de Datos de la Empresa.

En este campo podemos observar todos los datos referidos a la empresa, como: RUC, nombre, dirección, teléfono y el mensaje de agradecimiento. También hay la opción de modificar los datos.

NOMBRE DE USUARIO

PRODUCTOS

SALAS

HISTORIAL

USUARIOS

AJUSTES

CERRAR SESIÓN

RUC
65479877

Nombre
Pizzeria Adriana

Dirección
Abancay - Perú

Teléfono
954306632

Mensaje
Gracias por la compra

MODIFICAR

DATOS DE LA EMPRESA

Panel Salas Mesas Productos Finalizar Pedido Historial Pedidos Datos de la Empresa Usuarios Registro de Productos

6.2.2.10. Interfaz de Usuarios.

Esta interfaz permite registrar nuevos usuarios, requiriendo llenar los campos de correo electrónico, Password, nombre y rol del usuario. Estos datos se registran en una tabla, la cual se muestra al lado, con sus respectivos datos.

NOMBRE DE USUARIO

PRODUCTOS

SALAS

HISTORIAL

USUARIOS

AJUSTES

CERRAR SESIÓN

NUEVO USUARIO

Correo Electrónico
luis@gmail.com

Password

Nombre
Luis

Rol:
Administrador

REGISTRAR

ID	Nombre	Correo	Rol
3	Cristian	cristian@gmail.com	Administrador
5	Amilcar	amilcar@gmail.com	Asistente
6	Yuri	yuri@gmail.com	Asistente
7	Luis	luis@gmail.com	Administrador

Panel Salas Mesas Productos Finalizar Pedido Historial Pedidos Datos de la Empresa Usuarios Registro de Productos

6.3 Fase de Implementación:

UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC.



IMPLEMENTACIÓN DEL PROYECTO DE GESTIÓN DE PIZZERIA.



HARDWARE Y SOFTWARE USADOS

- Net Beans IDE 18
- MySQL 8.0 Command Line Client

LENGUAJE DE PROGRAMACIÓN

- Java
- MySQL

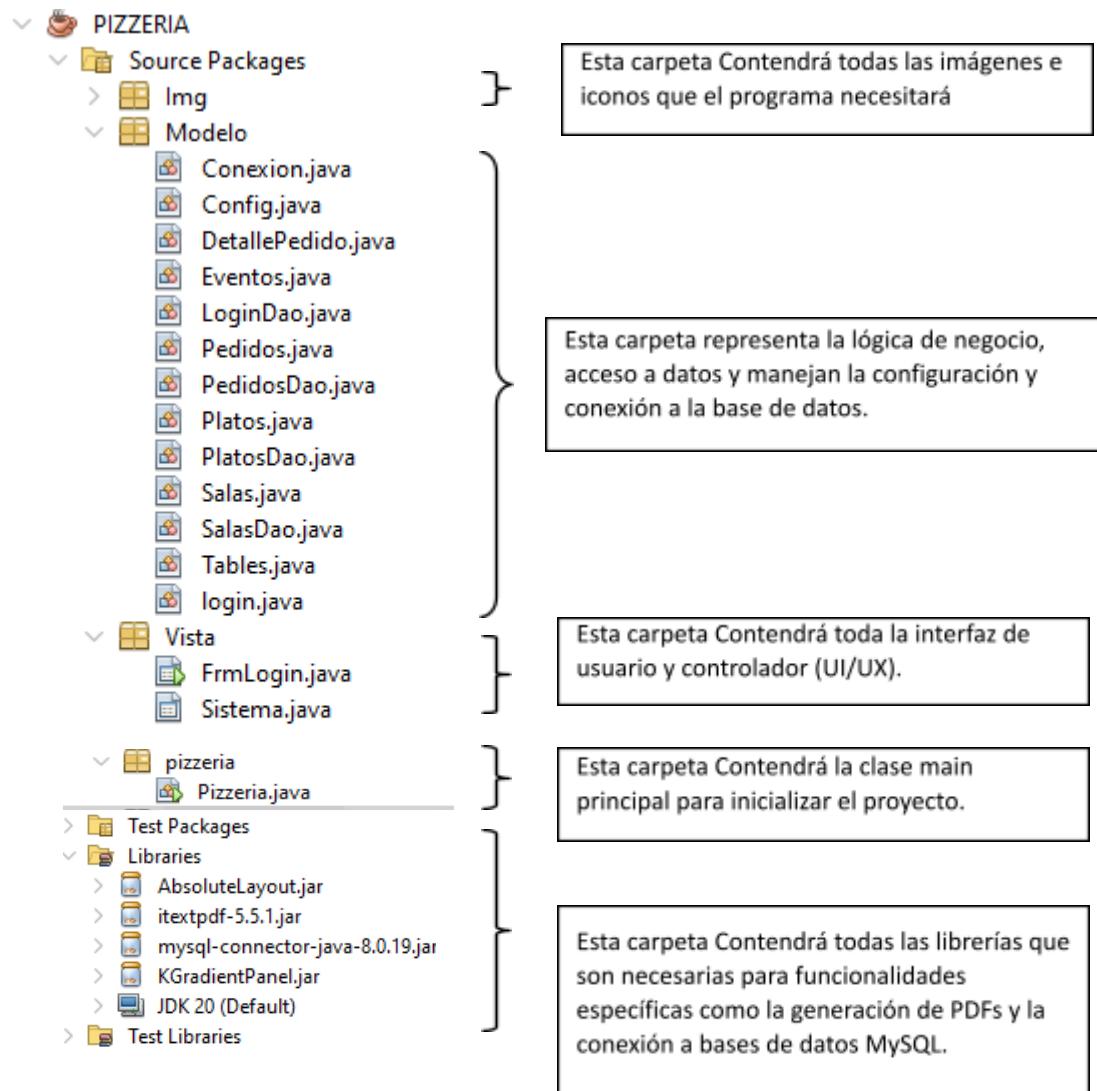
LIBRERIAS

- Absolute Layout
- ItextPdf 5.5.1
- MySQL connector 8.0.19
- KGradient Panel
- JDK 20

REFERENCIAS

- <https://github.com/k33ptoo/KGradientPanel>
- <https://www.youtube.com/watch?v=Wmrnt6CJA30>
 - <https://dev.mysql.com/downloads/connector/j/>
- <https://www.oracle.com/pe/java/technologies/downloads/#java20>

6.3.1 Desarrollo de la estructura de directorios del proyecto



6.3.2 Desarrollo De La Clase Pizzeria(Main/Principal):

```
package pizzeria;

import Vista.FrmLogin;

public class Pizzeria {

    public static void main(String[] args) {
        FrmLogin iniciar = new FrmLogin();
        iniciar.setVisible(b: true);
    }

}
```

6.3.3 Desarrollo Del Modelo Del Proyecto

- Clase Conexión : Facilitara la conexión con la base de datos MySQL

```
package Modelo;

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

public class Conexion {

    Connection con;

    // Establece y devuelve una conexión a la base de datos
    public Connection getConnection() {
        try {
            String myBD = "jdbc:mysql://localhost:3306/restaurante"; // URL de la base de datos
            con = DriverManager.getConnection(url:myBD, user:"root", password:"");
            return con;
        } catch (SQLException e) {
            System.out.println(e.toString()); // Imprime el error en caso de excepción
        }
        return null; // Devuelve null si ocurre un error
    }
}
```

- Clase config: Es una clase de un modelo de datos que encapsula información de configuración, proporcionando métodos para acceder y modificar cada atributo.

```

public class Config {
    private int id;
    private String ruc;
    private String nombre;
    private String telefono;
    private String direccion;
    private String mensaje;
    // Constructor por defecto
    public Config(){
    }
    // Constructor con todos los atributos
    public Config(int id, String ruc, String nombre, String telefono, String direccion, String mensaje) {
        this.id = id;
        this.ruc = ruc;
        this.nombre = nombre;
        this.telefono = telefono;
        this.direccion = direccion;
        this.mensaje = mensaje;
    }
}

```

- Clase Detalle pedido: Modela los detalles de un pedido, incluyendo atributos como ID, nombre del producto, precio, cantidad, comentario y ID del pedido. Proporciona métodos para acceder y modificar estos atributos.

```

public class DetallePedido {
    private int id;
    private String nombre;
    private double precio;
    private int cantidad;
    private String comentario;
    private int id_pedido;

    // Constructor por defecto
    public DetallePedido() {}

    // Constructor con todos los atributos
    public DetallePedido(int id, String nombre, double precio, int cantidad, String comentario, int id_pedido) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
        this.comentario = comentario;
        this.id_pedido = id_pedido;
    }

    // Métodos getter y setter para cada atributo
    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 double getPrecio() { return precio; }
    public void setPrecio(double precio) { this.precio = precio; }

    public int getCantidad() { return cantidad; }
    public void setCantidad(int cantidad) { this.cantidad = cantidad; }

    public String getComentario() { return comentario; }
    public void setComentario(String comentario) { this.comentario = comentario; }

    public int getId_pedido() { return id_pedido; }
    public void setId_pedido(int id_pedido) { this.id_pedido = id_pedido; }
}

```

- Clase Eventos: Maneja eventos de teclado para campos de texto. Tiene métodos para permitir solo letras, solo números, o números con un punto decimal, restringiendo los caracteres que se pueden ingresar.

```

public class Eventos {

    // Permite solo letras (mayúsculas y minúsculas), espacio y retroceso
    public void textKeyPress(KeyEvent evt) {
        char car = evt.getKeyChar();
        if ((car < 'a' || car > 'z') && (car < 'A' || car > 'Z')
            && (car != (char) KeyEvent.VK_BACK_SPACE) && (car != (char) KeyEvent.VK_SPACE)) {
            evt.consume();
        }
    }

    // Permite solo números y retroceso
    public void numberKeyPress(KeyEvent evt) {
        char car = evt.getKeyChar();
        if ((car < '0' || car > '9') && (car != (char) KeyEvent.VK_BACK_SPACE)) {
            evt.consume();
        }
    }

    // Permite solo números y un punto decimal (para valores decimales), además de retroceso
    public void numberDecimalKeyPress(KeyEvent evt, JTextField textField) {
        char car = evt.getKeyChar();
        if ((car < '0' || car > '9') && textField.getText().contains(": ".concat(".")) && (car != (char) KeyEvent.VK_BACK_SPACE)) {
            evt.consume();
        } else if ((car < '0' || car > '9') && (car != '.') && (car != (char) KeyEvent.VK_BACK_SPACE)) {
            evt.consume();
        }
    }
}

```

- Clase LoginDao: Maneja operaciones de base de datos relacionadas con usuarios y configuración. Permite verificar el login de usuarios, registrar nuevos usuarios, listar todos los usuarios, modificar datos de configuración y obtener los datos de configuración de la empresa.

```

public class LoginDao {
    Connection con;
    PreparedStatement ps;
    ResultSet rs;
    Conexion cn = new Conexion();

    public login log(String correo, String pass){
        login l = new login();
        String sql = "SELECT * FROM usuarios WHERE correo = ? AND pass = ?";
        try {
            con = cn.getConnection();
            ps = con.prepareStatement(string: sql);
            ps.setString(i: 1, string: correo);
            ps.setString(i: 2, string: pass);
            rs= ps.executeQuery();
            if (rs.next()) {
                l.setId(id: rs.getInt(string: "id"));
                l.setNombre(nombre: rs.getString(string: "nombre"));
                l.setCorreo(correo: rs.getString(string: "correo"));
                l.setPass(pass: rs.getString(string: "pass"));
                l.setRol(rol: rs.getString(string: "rol"));

            }
        } catch (SQLException e) {
            System.out.println(: e.toString());
        }
        return l;
    }
}

```

- Clase Pedidos: Modela la información de un pedido en un sistema, incluyendo atributos como ID, sala, número de mesa, fecha, total, sala, usuario y estado. Proporciona métodos para acceder y modificar estos atributos.

```

public class Pedidos {
    private int id;
    private int id_sala;
    private int num_mesa;
    private String fecha;
    private double total;
    private String sala;
    private String usuario;
    private String estado;

    // Constructor por defecto
    public Pedidos() {}

    // Constructor con todos los atributos
    public Pedidos(int id, int id_sala, int num_mesa, String fecha, double total, String sala, String usuario, String estado) {
        this.id = id;
        this.id_sala = id_sala;
        this.num_mesa = num_mesa;
        this.fecha = fecha;
        this.total = total;
        this.sala = sala;
        this.usuario = usuario;
        this.estado = estado;
    }

    // Métodos getter y setter para cada atributo
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public int getId_sala() { return id_sala; }
    public void setId_sala(int id_sala) { this.id_sala = id_sala; }

    public int getNum_mesa() { return num_mesa; }
    public void setNum_mesa(int num_mesa) { this.num_mesa = num_mesa; }

    public String getFecha() { return fecha; }
    public void setFecha(String fecha) { this.fecha = fecha; }

    public double getTotal() { return total; }
    public void setTotal(double total) { this.total = total; }

    public String getSala() { return sala; }
    public void setSala(String sala) { this.sala = sala; }

    public String getUsuario() { return usuario; }
    public void setUsuario(String usuario) { this.usuario = usuario; }

    public String getEstado() { return estado; }
    public void setEstado(String estado) { this.estado = estado; }
}

```

- Clase PedidosDao: Gestiona operaciones de base de datos para pedidos, incluyendo la creación, actualización y consulta de datos. Permite registrar nuevos pedidos y sus detalles, verificar el estado de un pedido, y generar informes en PDF con el detalle del pedido y la información del usuario. Utiliza JDBC para interactuar con la base de datos y la biblioteca iText para crear reportes PDF. Además, se encarga de manejar excepciones y cerrar recursos de manera adecuada.

```

public class PedidosDao {
    Connection con;
    Conexion cn = new Conexion();
    PreparedStatement ps;
    ResultSet rs;
    int r;

    public int IdPedido() {
        int id = 0;
        String sql = "SELECT MAX(id) FROM pedidos";
        try {
            con = cn.getConnection();
            ps = con.prepareStatement(string: sql);
            rs = ps.executeQuery();
            if (rs.next()) {
                id = rs.getInt(i: 1);
            }
        } catch (SQLException e) {
            System.out.println(: e.toString());
        }
        return id;
    }

    public int verificarEstado(int mesa, int id_sala){
        int id_pedido = 0;
        String sql = "SELECT id FROM pedidos WHERE num_mesa=? AND id_sala=? AND estado = ?";
        try {
            con = cn.getConnection();
            ps = con.prepareStatement(string: sql);
            ps.setInt(i: 1, ii: mesa);
            ps.setInt(i: 2, ii: id_sala);
        }
    }
}

```

```

        ps.setString(i: 3, string: "PENDIENTE");
        rs = ps.executeQuery();
        if(rs.next()){
            id_pedido = rs.getInt(string: "id");
        }
    } catch (SQLException e) {
        System.out.println(: e.toString());
    }
    return id_pedido;
}

public int RegistrarPedido(Pedidos ped){
    String sql = "INSERT INTO pedidos (id_sala, num_mesa, total, usuario) VALUES (?, ?, ?, ?)";
    try {
        con = cn.getConnection();
        ps = con.prepareStatement(string: sql);
        ps.setInt(i: 1, ii: ped.getId_sala());
        ps.setInt(i: 2, ii: ped.getNum_mesa());
        ps.setDouble(i: 3, d: ped.getTotal());
        ps.setString(i: 4, string: ped.getUsuario());
        ps.execute();
    } catch (SQLException e) {
        System.out.println(: e.toString());
    }finally{
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println(: e.toString());
        }
    }
    return r;
}
public int RegistrarDetalle(DetallePedido det){
    String sql = "INSERT INTO detalle_pedidos (nombre, precio, cantidad, comentario, id_pedido) VALUES (?, ?, ?, ?, ?)";
    try {
        con = cn.getConnection();
        ps = con.prepareStatement(string: sql);
        ps.setString(i: 1, string: det.getNombre());
        ps.setDouble(i: 2, d: det.getPrecio());
        ps.setInt(i: 3, ii: det.getCantidad());
        ps.setString(i: 4, string: det.getComentario());
        ps.setInt(i: 5, ii: det.getId_pedido());
        ps.execute();
    } catch (SQLException e) {
        System.out.println(: e.toString());
    }
    return r;
}

public List verPedidoDetalle(int id_pedido){
    List<DetallePedido> Lista = new ArrayList();
    String sql = "SELECT d.* FROM pedidos p INNER JOIN detalle_pedidos d ON p.id = d.id_pedido WHERE p.id = ?";
    try {
        con = cn.getConnection();
        ps = con.prepareStatement(string: sql);
        ps.setInt(i: 1, ii: id_pedido);
        rs = ps.executeQuery();
        while (rs.next()) {
            DetallePedido det = new DetallePedido();
            det.setId(id: rs.getInt(string: "id"));
            det.setNombre(nombre: rs.getString(string: "nombre"));
            det.setPrecio(precio: rs.getDouble(string: "precio"));
            det.setCantidad(cantidad: rs.getInt(string: "cantidad")));
            Lista.add(det);
        }
    } catch (SQLException e) {
        System.out.println(: e.toString());
    }
    return Lista;
}

```

```

        det.setComentario(comentario: rs.getString(string: "comentario"));
        Lista.add(e: det);
    }
} catch (SQLException e) {
    System.out.println(string: e.toString());
}
return Lista;
}

public Pedidos verPedido(int id_pedido){
    Pedidos ped = new Pedidos();
    String sql = "SELECT p.*, s.nombre FROM pedidos p INNER JOIN salas s ON p.id_sala = s.id WHERE p.id = ?";
    try {
        con = cn.getConnection();
        ps = con.prepareStatement(string: sql);
        ps.setInt(i: 1, ii: id_pedido);
        rs = ps.executeQuery();
        if (rs.next()) {

            ped.setId(id: rs.getInt(string: "id"));
            ped.setFecha(fecha: rs.getString(string: "fecha"));
            ped.setSala(sala: rs.getString(string: "nombre"));
            ped.setNum_mesa(num_mesa: rs.getInt(string: "num_mesa"));
            ped.setTotal(total: rs.getDouble(string: "total"));
        }
    } catch (SQLException e) {
        System.out.println(string: e.toString());
    }
    return ped;
}
public List finalizarPedido(int id_pedido){
    List<DetallePedido> Lista = new ArrayList();
    String sql = "SELECT d.* FROM pedidos p INNER JOIN detalle_pedidos d ON p.id = d.id_pedido WHERE p.id = ?";
    try {
        con = cn.getConnection();
        ps = con.prepareStatement(string: sql);
        ps.setInt(i: 1, ii: id_pedido);
        rs = ps.executeQuery();
        while (rs.next()) {
            DetallePedido det = new DetallePedido();
            det.setId(id: rs.getInt(string: "id"));
            det.setNombre(nombre: rs.getString(string: "nombre"));
            det.setPrecio(precio: rs.getDouble(string: "precio"));
            det.setCantidad(cantidad: rs.getInt(string: "cantidad"));
            det.setComentario(comentario: rs.getString(string: "comentario"));
            Lista.add(e: det);
        }
    } catch (SQLException e) {
        System.out.println(string: e.toString());
    }
    return Lista;
}

public void pdfPedido(int id_pedido) {
    String fechaPedido = null, usuario = null, total = null, sala = null, num_mesa = null;
    try {
        FileOutputStream archivo;
        String url = FileSystemView.getFileSystemView().getDefau

```

```

PdfWriter.getInstance(document: doc, os: archivo);
doc.open();

// Titulo del documento
Paragraph titulo = new Paragraph(string: "Reporte de Pedido",
    new Font(family: Font.FontFamily.TIMES_ROMAN, size: 18, style: Font.BOLD, color: BaseColor.BLACK));
titulo.setAlignment(alignment: Element.ALIGN_CENTER);
doc.add(element: titulo);
doc.add(element: Chunk.NEWLINE);

// Cargar la imagen del logo
Image img = Image.getInstance(url: getClass().getResource(name: "/Img/logo_pdf.png"));
img.scaleToFit(fitWidth: 100, fitHeight: 100);

// Información del pedido
String informacion = "SELECT p.*, s.nombre FROM pedidos p INNER JOIN salas s ON p.id_sala = s.id WHERE p.id = ?";
try {
    ps = con.prepareStatement(string: informacion);
    ps.setInt(i: 1, ii: id_pedido);
    rs = ps.executeQuery();
    if (rs.next()) {
        num_mesa = rs.getString(string: "num_mesa");
        sala = rs.getString(string: "nombre");
        fechaPedido = rs.getString(string: "fecha");
        usuario = rs.getString(string: "usuario");
        total = rs.getString(string: "total");
    }
} catch (SQLException e) {
    System.out.println(e.toString());
}

// Encabezado con logo e información de la empresa
PdfPTable encabezado = new PdfPTable(numColumns: 4);
encabezado.setWidthPercentage(widthPercentage: 100);
encabezado.getDefaultCell().setBorder(border: Rectangle.NO_BORDER);
encabezado.setWidths(new float[]{20f, 20f, 60f, 60f});

encabezado.addCell(image: img);
encabezado.addCell(text: "");

// Información de la empresa
String config = "SELECT * FROM config";
String mensaje = "";
try {
    ps = con.prepareStatement(string: config);
    rs = ps.executeQuery();
    if (rs.next()) {
        mensaje = rs.getString(string: "mensaje");
        encabezado.addCell("RUC: " + rs.getString(string: "ruc")
            + "\nNombre: " + rs.getString(string: "nombre")
            + "\nTeléfono: " + rs.getString(string: "telefono")
            + "\nDirección: " + rs.getString(string: "direccion"));
    }
} catch (SQLException e) {
    System.out.println(e.toString());
}

// Información del pedido y usuario
Paragraph info = new Paragraph();
Font negrita = new Font(family: Font.FontFamily.TIMES_ROMAN, size: 12, style: Font.BOLD, color: BaseColor.BLACK);
info.add("Atendido por: " + usuario
    + "\nNº Pedido: " + id_pedido
    + "\nFecha: " + fechaPedido

```

```

        ps.setString(i: 1, string: "FINALIZADO");
        ps.setInt(i: 2, int: id_pedido);
        ps.execute();
        return true;
    } catch (SQLException e) {
        System.out.println(: e.toString());
        return false;
    }
}

public List listarPedidos() {
    List<Pedidos> Lista = new ArrayList();
    String sql = "SELECT p.*, s.nombre FROM pedidos p INNER "
        + "JOIN salas s ON p.id_sala = s.id ORDER BY p.fecha DESC";
    try {
        con = cn.getConnection();
        ps = con.prepareStatement(string: sql);
        rs = ps.executeQuery();
        while (rs.next()) {
            Pedidos ped = new Pedidos();
            ped.setId(id: rs.getInt(string: "id"));
            ped.setSala(sala: rs.getString(string: "nombre"));
            ped.setNum_mesa(num_mesa: rs.getInt(string: "num_mesa"));
            ped.setFecha(fecha: rs.getString(string: "fecha"));
            ped.setTotal(total: rs.getDouble(string: "total"));
            ped.setUsuario(usuario: rs.getString(string: "usuario"));
            ped.setEstado(estado: rs.getString(string: "estado"));
            Lista.add(e: ped);
        }
    } catch (SQLException e) {
        System.out.println(: e.toString());
    }
    return Lista;
}
}

```

- Clase Platos: Representa un modelo para los platos en un sistema. Define atributos para el identificador, nombre, precio y fecha de un plato. Incluye métodos para obtener y establecer estos valores, así como dos constructores: uno vacío y otro con parámetros para inicializar todos los atributos.

```

public class Platos {
    private int id; // Identificador único del plato
    private String nombre; // Nombre del plato
    private double precio; // Precio del plato
    private String fecha; // Fecha de actualización del plato

    // Constructor vacío
    public Platos() {
    }

    // Constructor con parámetros para inicializar todos los atributos
    public Platos(int id, String nombre, double precio, String fecha) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.fecha = fecha;
    }

    // Getter y Setter para el identificador del plato
    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;
    }

    // Getter y Setter para el precio del plato
    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }

    // Getter y Setter para la fecha de actualización del plato
    public String getFecha() {
        return fecha;
    }

    public void setFecha(String fecha) {
        this.fecha = fecha;
    }
}

```

- Clase PlatosDao: Registrar: Inserta un nuevo plato en la base de datos.
Listar: Recupera una lista de platos basándose en un nombre parcial y una fecha. Eliminar: Elimina un plato de la base de datos usando su id. Modificar: Actualiza la información de un plato existente en la base de datos.

```
public class PlatosDao {

    Connection con;
    Conexion cn = new Conexion();
    PreparedStatement ps;
    ResultSet rs;

    public boolean Registrar(Platos pla) {
        String sql = "INSERT INTO platos (nombre, precio, fecha) VALUES (?, ?, ?)";
        try {
            con = cn.getConnection();
            ps = con.prepareStatement(string: sql);
            ps.setString(i: 1, string: pla.getNombre());
            ps.setDouble(i: 2, d: pla.getPrecio());
            ps.setString(i: 3, string: pla.getFecha());
            ps.execute();
            return true;
        } catch (SQLException e) {
            System.out.println(: e.toString());
            return false;
        }finally {
            try {
                con.close();
            } catch (SQLException ex) {
                System.out.println(: ex.toString());
            }
        }
    }

    public List Listar(String valor, String fecha) {
        List<Platos> Lista = new ArrayList();
        String sql = "SELECT * FROM platos WHERE fecha = ?";
```

```

String consulta = "SELECT * FROM platos WHERE nombre LIKE '%"+valor+"%' AND fecha = ?";
try {
    con = cn.getConnection();
    if(valor.equalsIgnoreCase(anotherString: ""))
        ps = con.prepareStatement(string: sql);
    else{
        ps = con.prepareStatement(string: consulta);
    }
    ps.setString(i: 1, string: fecha);
    rs = ps.executeQuery();
    while (rs.next()) {
        Platos pl = new Platos();
        pl.setId(id: rs.getInt(string: "id"));
        pl.setNombre(nombre: rs.getString(string: "nombre"));
        pl.setPrecio(precio: rs.getDouble(string: "precio"));
        Lista.add(e: pl);
    }
} catch (SQLException e) {
    System.out.println(: e.toString());
}
return Lista;
}

public boolean Eliminar(int id) {
String sql = "DELETE FROM platos WHERE id = ?";
try {
    ps = con.prepareStatement(string: sql);
    ps.setInt(i: 1, il: id);
    ps.execute();
    return true;
} catch (SQLException e) {
    return false;
} finally {
    try {
        con.close();
    } catch (SQLException ex) {
        System.out.println(: ex.toString());
    }
}
}

public boolean Modificar(Platos pla) {
String sql = "UPDATE platos SET nombre=?, precio=? WHERE id=?";
try {
    ps = con.prepareStatement(string: sql);
    ps.setString(i: 1, string: pla.getNombre());
    ps.setDouble(i: 2, d: pla.getPrecio());
    ps.setInt(i: 3, il: pla.getId());
    ps.execute();
}

```

```

        return true;
    } catch (SQLException e) {
        System.out.println(":" + e.toString());
        return false;
    } finally {
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println(":" + e.toString());
        }
    }
}

```

- Clase Salas: Sirve para representar una sala dentro del sistema, proporcionando un modelo para gestionar su identificador, nombre y número de mesas.

```

// Método para establecer el nombre de la sala
public void setNombre(String nombre) {
    this.nombre = nombre;
}

// Método para obtener el número de mesas en la sala
public int getMesas() {
    return mesas;
}

// Método para establecer el número de mesas en la sala
public void setMesas(int mesas) {
    this.mesas = mesas;
}

```

- Clase SalasDao: Proporciona métodos para manejar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la tabla salas de la base de datos.

```

public class SalasDao {
    Connection con;
    Conexion cn = new Conexion();
    PreparedStatement ps;
    ResultSet rs;
    public boolean RegistrarSala(Salas sl){
        String sql = "INSERT INTO salas(nombre, mesas) VALUES (?,?)";
        try {
            con = cn.getConnection();
            ps = con.prepareStatement(string: sql);
            ps.setString(i: 1, string: sl.getNombre());
            ps.setInt(i: 2, il: sl.getMesas());
            ps.execute();
            return true;
        } catch (SQLException e) {
            System.out.println(: e.toString());
            return false;
        }finally{
            try {
                con.close();
            } catch (SQLException e) {
                System.out.println(: e.toString());
            }
        }
    }

    public List Listar(){
        List<Salas> Lista = new ArrayList();
        String sql = "SELECT * FROM salas";
        try {
            System.out.println(: e.toString());
        }
    }

    public boolean Modificar(Salas sl){
        String sql = "UPDATE salas SET nombre=?, mesas=? WHERE id=?";
        try {
            con = cn.getConnection();
            ps = con.prepareStatement(string: sql);
            ps.setString(i: 1, string: sl.getNombre());
            ps.setInt(i: 2, il: sl.getMesas());
            ps.setInt(i: 3, il: sl.getId());
            ps.execute();
            return true;
        }
    }
}

```

```

        } catch (SQLException e) {
            System.out.println("e.toString());
            return false;
        }finally{
            try {
                con.close();
            } catch (SQLException e) {
                System.out.println("e.toString());
            }
        }
    }
}

```

- Clase Tables: extiende DefaultTableCellRenderer para personalizar la apariencia de las celdas en una tabla (JTable).

```

public class Tables extends DefaultTableCellRenderer{

    @Override
    public Component getTableCellRendererComponent(
        JTable jTable, Object o, boolean bln, boolean blnl, int row, int col) {
        super.getTableCellRendererComponent(jTable, value: o, isSelected: bln, hasFocus: blnl, row, column: col);
        switch (jTable.getValueAt(row, column: 6).toString()) {
            case "PENDIENTE" -> {
                setBackground(new Color(r: 255,g: 51,b: 51));
                setForeground(c: Color.white);
            }
            case "FINALIZADO" -> {
                setBackground(new Color(r: 1,g: 202,b: 2));
                setForeground(c: Color.white);
            }
            default -> {
                setBackground(c: Color.white);
                setForeground(c: Color.black);
            }
        }
        return this;
    }
}

```

- Clase login: Representa un modelo de datos para la información de inicio de sesión de un usuario, incluyendo atributos como ID, nombre, correo electrónico, contraseña y rol.

```
public class login {
    private int id;
    private String nombre;
    private String correo;
    private String pass;
    private String rol;

    public login() {
    }

    public login(int id, String nombre,
                String correo, String pass, String rol) {
        this.id = id;
        this.nombre = nombre;
        this.correo = correo;
        this.pass = pass;
        this.rol = rol;
    }

    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 getCorreo() {
    return correo;
}

public void setCorreo(String correo) {
    this.correo = correo;
}

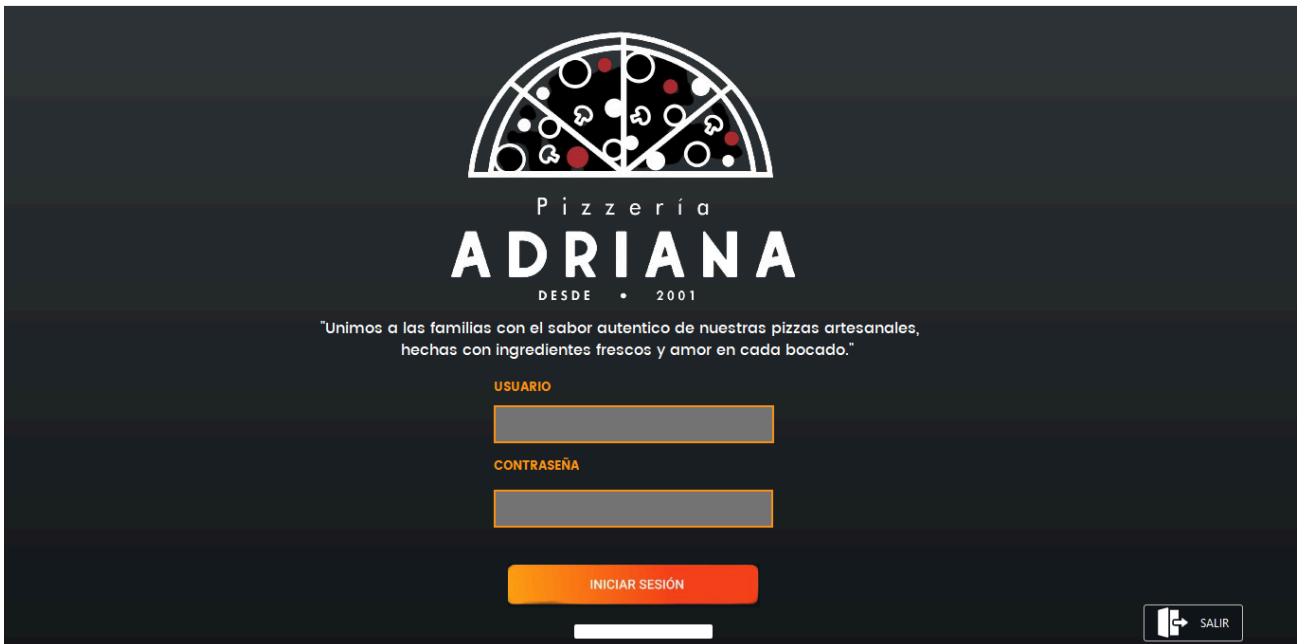
public String getPass() {
    return pass;
}

public void setPass(String pass) {
    this.pass = pass;
}

public String getRol() {
    return rol;
}

public void setRol(String rol) {
    this.rol = rol;
}
```

6.3.4 Desarrollo De La Vista, Interfaz De Usuario (Ui/Ux) De Inicio De Sesión



Código del JFrame Form de Inicio de sesión

```
public class FrmLogin extends javax.swing.JFrame {
    // Instancia de la clase `login`, que representa al usuario que intenta iniciar sesión.
    login lg = new login();
    // Instancia de la clase `LoginDao`, que gestiona las operaciones de acceso a datos para el login.
    LoginDao login = new LoginDao();
    // Timer para controlar el progreso de la barra durante la validación de las credenciales.
    private Timer tiempo;
    // Contador para el progreso de la barra.
    int contador;
    // Tiempo en segundos para el progreso de la barra.
    int segundos = 30;

    public FrmLogin() {
        initComponents();
        this.setLocationRelativeTo(null); // Centra la ventana en la pantalla.
        //this.setExtendedState(FrmLogin.MAXIMIZED_BOTH); //comenzar al maximo
        // Configuración de inicio: establece un correo y contraseña predeterminados.
        txtCorreo.setText("cristian@gmail.com");
        txtPass.setText("cristianonelover");
        // Oculta la barra de progreso al inicio.
        barra.setVisible(false);
    }

    // Clase interna para manejar el progreso de la barra cuando se validan las credenciales.
    public class BarraProgreso implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent ae) {
            contador++; // Incrementa el contador.
            barra.setValue((int) contador); // Actualiza el valor de la barra de progreso.
            // Si el progreso llega al 100%, detiene el Timer y abre el sistema principal.
            if (contador == 100) {
                tiempo.stop();
                if (barra.getValue() == 100) {
                    Sistema sis = new Sistema(priv:lg);
                    sis.setVisible(true); // Muestra la ventana del sistema.
                    dispose(); // Cierra la ventana de login.
                }
            }
        }
    }
}
```

```

// Método para validar las credenciales ingresadas por el usuario.
public void validar(){
    String correo = txtCorreo.getText(); // Obtiene el correo ingresado.
    String pass = String.valueOf(data:txtPass.getPassword()); // Obtiene la contraseña ingresada.

    // Verifica que los campos de correo y contraseña no estén vacíos.
    if (!"".equals(anObject:correo) || !"".equals(anObject:pass)) {

        // Llama al método log() de LoginDao para validar las credenciales.
        lg = login.log(correo, pass);

        // Si las credenciales son correctas, muestra la barra de progreso.
        if (lg.getCorreo() != null && lg.getPass() != null) {
            barra.setVisible(aFlag: true); // Muestra la barra de progreso.
            contador = -1; // Reinicia el contador.
            barra.setValue(n: 0); // Reinicia la barra de progreso.
            barra.setStringPainted(b: true); // Muestra el valor en la barra.
            tiempo = new Timer(delay: segundos, new BarraProgreso()); // Configura el Timer para la barra.
            tiempo.start(); // Inicia el Timer.
        } else {
            // Muestra un mensaje de error si las credenciales son incorrectas.
            JOptionPane.showMessageDialog(parentComponent:null, message:"Correo o la Contraseña incorrecta");
        }
    }
}

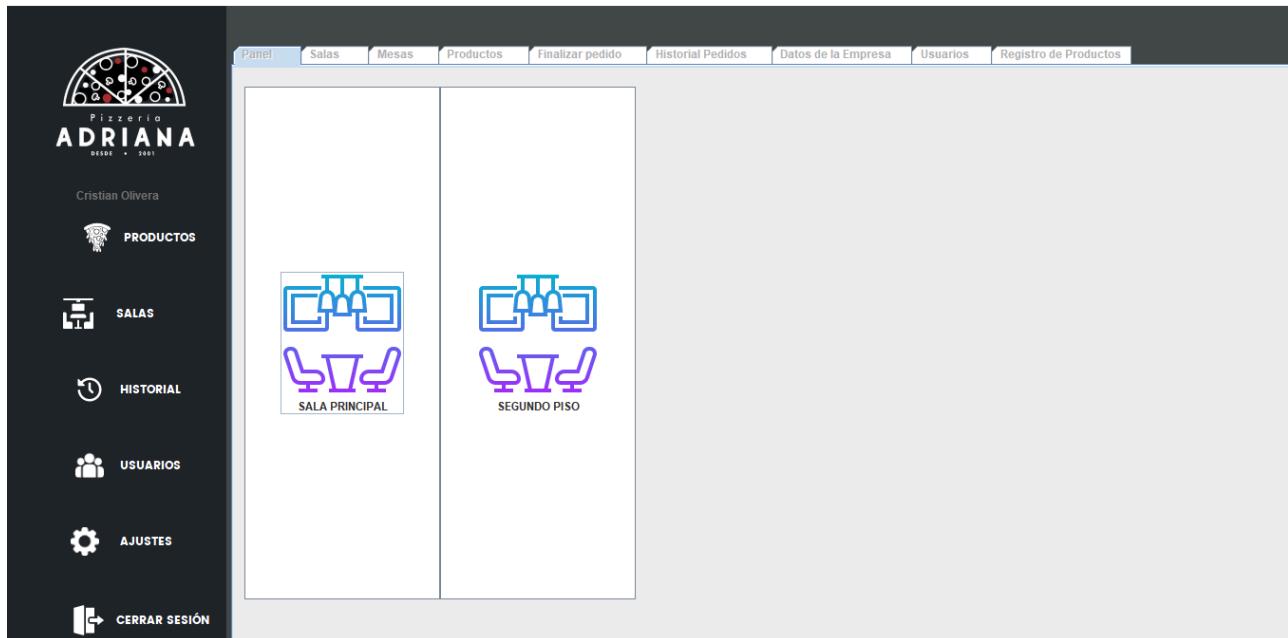
private void btnInicioSesionActionPerformed(java.awt.event.ActionEvent evt) {
    validar();
}

private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(status: 0); //salir del sistema
}

```

6.3.5 Desarrollo De La Vista Y Controlador Del Sistema

- Interfaz principal (PANEL): Al ingresar al sistema se le mostrará por predeterminado la siguiente interfaz.



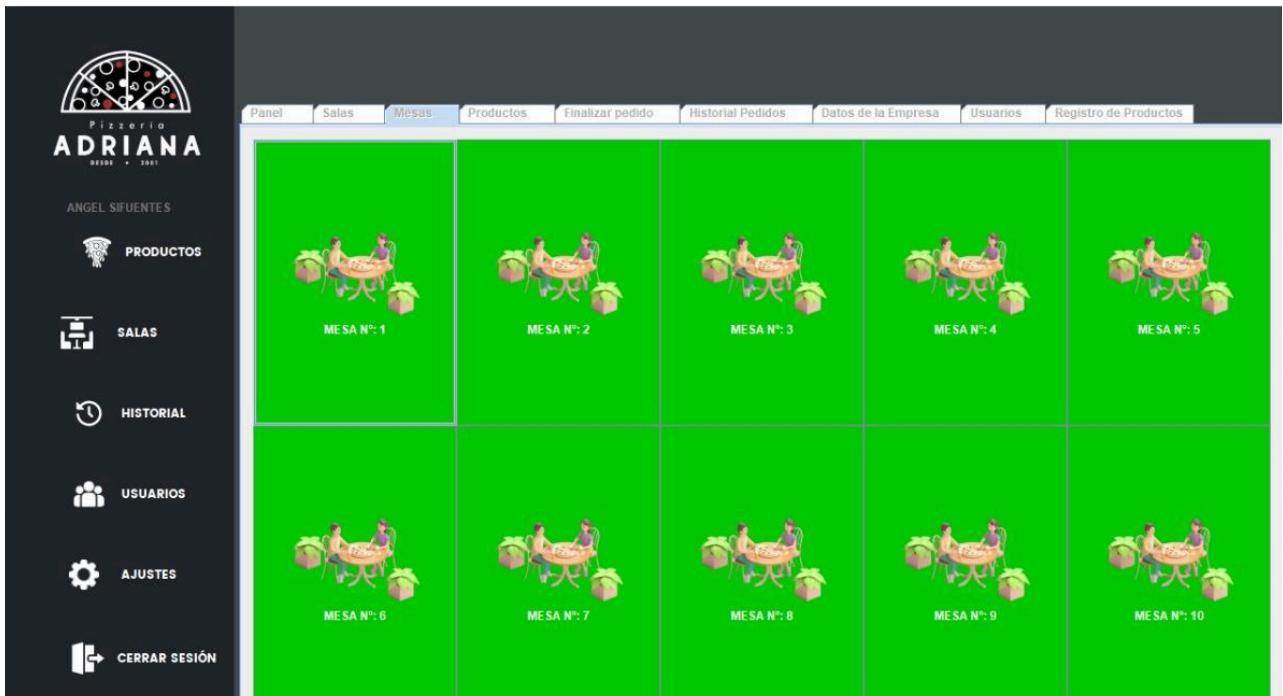
- Interfaz de Salas: Se mostrará una interfaz donde el usuario pueda elegir guardar, editar, borrar, agregar, una nueva sala (cuarto/piso) ya la vez el usuario elegirá el número de mesas que contendrá esta sala. Se le mostrará en una tablet.

The screenshot shows a user interface for creating a new room. On the left, there's a sidebar with a logo for "Pizzería ADRIANA" and navigation links: Administrador, PRODUCTOS, SALAS, HISTORIAL, USUARIOS, AJUSTES, and CERRAR SESIÓN. The main area has a header with tabs: Panel, Salas, Mesas, Productos, Finalizar pedido, Historial Pedidos, Datos de la Empresa, Usuarios, and Registro de Productos. A sub-header "Nueva Sala" is displayed above a form. The form contains fields for "Nombre:" (Name) and "Mesas:" (Tables), along with four icons representing different actions: Guardar (Save), Editar (Edit), Eliminar (Delete), and Nuevo (New).

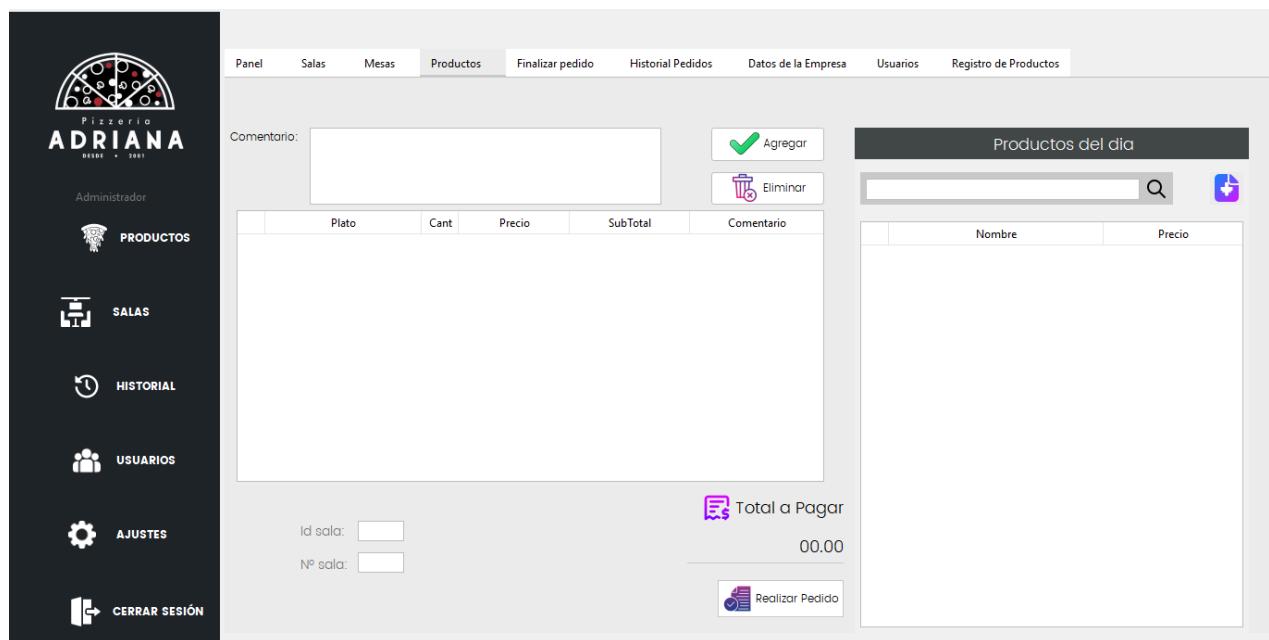
- Interfaz de registro de productos: donde el usuario puede elegir productos como pizzas, bebidas, etc.

The screenshot shows a user interface for registering products. It features a sidebar with the same navigation as the previous interface. The main area has a header with tabs: Panel, Salas, Mesas, Productos, Finalizar pedido, Historial Pedidos, Datos de la Empresa, Usuarios, and Registro de Productos. A sub-header "Productos del Día" is shown above a form. The form includes fields for "Nombre:" (Name), "Precio:" (Price), and an "Id:" field. Below the form are four icons for actions: Guardar (Save), Editar (Edit), Eliminar (Delete), and Nuevo (New). To the right is a table titled "Registro de Productos" with columns: ID, DESCRIPCIÓN (Description), and PRECIO (Price). The table is currently empty.

- Interfaz de Mesas: donde el usuario pueda elegir la mesa en el cual se le atenderá al cliente.



- Interfaz de selección de productos para el cliente: El usuario elegirá los productos para el cliente específico y podrá realizar el pedido.



- Interfaz de Historial de pedidos: El historial de pedidos mostrará los pedidos de los cuales el usuario ha registrado.

Id	Sala	Atendido	N° Mesa	Fecha	Total	Estado

- Finalizar pedido: El usuario podrá finalizar el pedido que se registró como también podrá generar un pdf con los pedidos ingresados y cerrar la mesa.

Plato	Cantidad	Precio	SubTotal	Comentario

Fecha y Hora:

Sala:

N° Mesa:

Id pedido:

Id historial:

Total a Pagar
00.00

- Datos de la Empresa: El usuario podrá modificar los datos de la empresa(el software).

Pizzería ADRIANA DESDE 1981

Panel Salas Mesas Productos Finalizar pedido Historial Pedidos **Datos de la Empresa** Usuarios Registro de Productos

DATOS DE LA EMPRESA

RUC	Nombre
_____	_____
Dirección	Teléfono
_____	_____
Mensaje	
Id config: <input type="text"/>	
Modificar	

- Usuarios (Trabajadores): Se podrá registrar usuarios como empleados y administradores.

Pizzería ADRIANA DESDE 1981

Panel Salas Mesas Productos Finalizar pedido Historial Pedidos **Datos de la Empresa** **Usuarios** Registro de Productos

Nuevo Usuario

Id	Nombre	Correo	Rol
_____	_____	_____	_____

Correo Electrónico

Password

Nombre:

Rol:
Administrador

Registrar

Código Completo de la vista del Sistema:

```
jTabbedPane.setForeground(Color.BLACK);
//personalizar el fondo de los lblBtn's como un boton
btnPlatos.setOpaque(isOpaque: true);
btnPlatos.setBackground(new Color(z: 33, g: 36, b: 41));

btnSala.setOpaque(isOpaque: true);
btnSala.setBackground(new Color(z: 33, g: 36, b: 41));

btnVentas.setOpaque(isOpaque: true);
btnVentas.setBackground(new Color(z: 33, g: 36, b: 41));

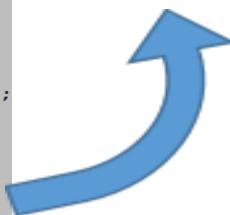
btnUsuarios.setOpaque(isOpaque: true);
btnUsuarios.setBackground(new Color(z: 33, g: 36, b: 41));

lblBtnAjustes.setOpaque(isOpaque: true);
lblBtnAjustes.setBackground(new Color(z: 33, g: 36, b: 41));

lblBtnCerrarSesion.setOpaque(isOpaque: true);
lblBtnCerrarSesion.setBackground(new Color(z: 33, g: 36, b: 41));

txtIdHistorialPedido.setVisible(aFlag: false);
txtIdConfig.setVisible(aFlag: false);
if (priv.getRol().equals(anObject: "Asistente")) {
    btnSala.setEnabled(enabled: false);
    btnConfig.setEnabled(enabled: false);
    LabelVendedor.setText(text: priv.getNombre());
} else {
    LabelVendedor.setText(text: priv.getNombre());
}
```

```
txtIdConfig.setVisible(aFlag: false);
txtIdHistorialPedido.setVisible(aFlag: false);
txtIdPedido.setVisible(aFlag: false);
txtIdPlato.setVisible(aFlag: false);
txtIdSala.setVisible(aFlag: false);
txtTempIdSala.setVisible(aFlag: false);
txtTempNumMesa.setVisible(aFlag: false);
jTabbedPane.setEnabled(enabled: false);
panelSalas();
```



```
private void btnActualizarConfigActionPerformed(java.awt.event.ActionEvent evt) {
    if (!"".equals(anObject: txtRucConfig.getText()) || !"".equals(anObject: txtNombreConfig.getText())
        || !"".equals(anObject: txtTelefonoConfig.getText()) || !"".equals(anObject: txtDireccionConfig.getText())) {
        conf.setRuc(ruc: txtRucConfig.getText());
        conf.setNombre(nombre: txtNombreConfig.getText());
        conf.setTelefono(telefono: txtTelefonoConfig.getText());
        conf.setDireccion(direccion: txtDireccionConfig.getText());
        conf.setMensaje(mensaje: txtMensaje.getText());
        conf.setId(id: Integer.parseInt(: txtIdConfig.getText()));
        lgDao.ModificarDatos(conf);
        JOptionPane.showMessageDialog(parentComponent: null, message: "Datos de la empresa modificado");
        //ListarConfig();
    } else {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Los campos estan vacios");
    }
}

private void btnPdfPedidoActionPerformed(java.awt.event.ActionEvent evt) {
    if (txtIdHistorialPedido.getText().equals(anObject: "")) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "Selecciona una fila");
    } else {
        pedDao.pdfPedido(id_pedido: Integer.parseInt(: txtIdHistorialPedido.getText()));
        txtIdHistorialPedido.setText(z: "");
    }
}
```

```

private void TablePedidosMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int fila = TablePedidos.rowAtPoint(point: evt.getPoint());
    int id_pedido = Integer.parseInt(: TablePedidos.getValueAt(row:fila, column: 0).toString());
    LimpiarTable();
    verPedido(id_pedido);
    verPedidoDetalle(id_pedido);
    jTabbedPane.setSelectedIndex(index: 4);
    btnFinalizar.setEnabled(b: false);
    txtIdHistorialPedido.setText(""+id_pedido);
}

private void tableSalaMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int fila = tableSala.rowAtPoint(point: evt.getPoint());
    txtIdSala.setText(: tableSala.getValueAt(row:fila, column: 0).toString());
    txtNombreSala.setText(: tableSala.getValueAt(row:fila, column: 1).toString());
    txtMesas.setText(: tableSala.getValueAt(row:fila, column: 2).toString());
}

private void txtIdConfigActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void btnRegistrarSalaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (txtNombreSala.getText().equals(anObject: "") || txtMesas.getText().equals(anObject: "")) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Los campos estan vacios");
    } else {
        sl.setNombre(nombre: txtNombreSala.getText());

        sl.setMesas(mesas: Integer.parseInt(: txtMesas.getText()));
        slDao.RegistrarSala(sl);
        JOptionPane.showMessageDialog(parentComponent:null, message:"Sala Registrada");
        LimpiarSala();
        LimpiarTable();
        ListarSalas();
    }
}

private void btnActualizarSalaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if ("".equals(anObject: txtIdSala.getText())) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Seleccione una fila");
    } else {
        if (!"".equals(anObject: txtNombreSala.getText())) {
            sl.setNombre(nombre: txtNombreSala.getText());
            sl.setId(id: Integer.parseInt(: txtIdSala.getText()));
            slDao.Modificar(sl);
            JOptionPane.showMessageDialog(parentComponent:null, message:"Sala Modificado");
            LimpiarSala();
            LimpiarTable();
            ListarSalas();
        }
    }
}

```

```

private void btnNuevoSalaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    LimpiarSala();
}

private void btnEliminarSalaActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (!"".equals(anObject: txtIdSala.getText())) {
        int pregunta = JOptionPane.showConfirmDialog(parentComponent: null, message:"Esta seguro de eliminar");
        if (pregunta == 0) {
            int id = Integer.parseInt(: txtIdSala.getText());
            s1Dao.Eliminar(id);
            LimpiarSala();
            LimpiarTable();
            ListarSalas();
        }
    } else {
        JOptionPane.showMessageDialog(parentComponent: null, message:"Seleccione una fila");
    }
}

private void btnRegistrarUsuarioActionPerformed(java.awt.event.ActionEvent evt) {
    if (txtNombre.getText().equals(anObject: "") || txtCorreo.getText().equals(anObject: ""))
        || txtPass.getPassword().equals(obj: ""))
        JOptionPane.showMessageDialog(parentComponent: null, message:"Todo los campos son requeridos");
    } else {
        login lg = new login();
        String correo = txtCorreo.getText();
        String pass = String.valueOf(data: txtPass.getPassword());
        String nom = txtNombre.getText();
        String rol = cbxRol.getSelectedItem().toString();
        lg.setNombre(nombre: nom);
        lg.setCorreo(correo);
        lg.setPass(pass);
        lg.setRol(rol);
        lgDao.Registrar(reg:lg);
        JOptionPane.showMessageDialog(parentComponent: null, message:"Usuario Registrado");
    }
}

private void txtBuscarPlatoKeyReleased(java.awt.event.KeyEvent evt) {
    LimpiarTable();
    ListarPlatos(tabla: tblTemPlatos);
}

private void btnAddPlatoActionPerformed(java.awt.event.ActionEvent evt) {
    if (tblTemPlatos.getSelectedRow() >= 0) {
        int id = Integer.parseInt(: tblTemPlatos.getValueAt(row:tblTemPlatos.getSelectedRow(), column: 0).toString());
        String descripcion = tblTemPlatos.getValueAt(row:tblTemPlatos.getSelectedRow(), column: 1).toString();
        double precio = Double.parseDouble(: tblTemPlatos.getValueAt(row:tblTemPlatos.getSelectedRow(), column: 2).toString());
        double total = 1 * precio;
        item = item + 1;
        tmp = (DefaultTableModel) tableMenu.getModel();
        for (int i = 0; i < tableMenu.getRowCount(); i++) {
            if (tableMenu.getValueAt(row:i, column: 0).equals(obj:id)) {
                int cantActual = Integer.parseInt(: tableMenu.getValueAt(row:i, column: 2).toString());
                int nuevoCantidad = cantActual + 1;
                double nuevoSub = precio * nuevoCantidad;
                tmp.setValueAt aValue: nuevoCantidad, row:i, column: 2);
                tmp.setValueAt aValue: nuevoSub, row:1, column: 4);
                TotalPagar(tabla: tableMenu, label: totalMenu);
                return;
            }
        }
    }
}

```

```

        ArrayList lista = new ArrayList();
        lista.add(item);
        lista.add(id);
        lista.add(descripcion);
        lista.add(l);
        lista.add(precio);
        lista.add(total);
        Object[] O = new Object[6];
        O[0] = lista.get(1);
        O[1] = lista.get(2);
        O[2] = lista.get(3);
        O[3] = lista.get(4);
        O[4] = lista.get(5);
        O[5] = "";
        tmp.addRow(rowData:O);
        tableMenu.setModel(dataModel:tmp);
        TotalPagar(tabla:tableMenu, label:totalMenu);
    } else {
        JOptionPane.showMessageDialog(parentComponent:null, message:"SELECCIONA UNA FILA");
    }
}

private void btnGenerarPedidoActionPerformed(java.awt.event.ActionEvent evt) {
    if (tableMenu.getRowCount() > 0) {
        RegistrarPedido();
        detallePedido();
        LimpiearTableMenu();
        JOptionPane.showMessageDialog(parentComponent:null, message:"PEDIDO REGISTRADO");
        jTabbedPane.setSelectedIndex(index: 0);
    } else {
        JOptionPane.showMessageDialog(parentComponent:null, message:"NO HAY PRODUCTO EN LA PEDIDO");
    }
}

private void btnEditarPlatoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if ("".equals(anObject:txtIdPlato.getText())) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Seleccione una fila");
    } else {
        if (!"".equals(anObject:txtNombrePlato.getText()) || !"".equals(anObject:txtPrecioPlato.getText())) {
            pla.setNombre(nombre: txtNombrePlato.getText());
            pla.setPrecio(precio: Double.parseDouble(txtPrecioPlato.getText()));
            pla.setId(id: Integer.parseInt(txtIdPlato.getText()));
            if (plaDao.Modificar(pla)) {
                JOptionPane.showMessageDialog(parentComponent:null, message:"Plato Modificado");
                LimpiearTable();
                ListarPlatos(tabla:TablePlatos);
                LimpiearPlatos();
            }
        }
    }
}

private void btnEliminarPlatoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if ("".equals(anObject:txtIdPlato.getText())) {
        int pregunta = JOptionPane.showConfirmDialog(parentComponent:null, message:"Esta seguro de eliminar");
        if (pregunta == 0) {
            int id = Integer.parseInt(txtIdPlato.getText());
            plaDao.Eliminar(id);
            LimpiearTable();
            LimpiearPlatos();
            ListarPlatos(tabla:TablePlatos);
        }
    }
}

```

```

private void btnFinalizarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int pregunta = JOptionPane.showConfirmDialog(parentComponent:null, message:"Esta seguro de finalizar");
    if (pregunta == 0) {
        if (pedDao.actualizarEstado(id_pedido: Integer.parseInt(: txtIdPedido.getText()))) {
            pedDao.pdfPedido(id_pedido: Integer.parseInt(: txtIdPedido.getText()));
        }
    }
}

private void txtPrecioPlatoKeyTyped(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    event.numberDecimalKeyPress(evt, textField: txtPrecioPlato);
}

private void btnGuardarPlatoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (!"".equals(anObject: txtNombrePlato.getText()) || !"".equals(anObject: txtPrecioPlato.getText())) {
        pla.setNombre(nombre: txtNombrePlato.getText());
        pla.setPrecio(precio: Double.parseDouble(: txtPrecioPlato.getText()));
        pla.setFecha(fecha: fechaFormato);
        if (plaDao.Registrar(pla)) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Plato Registrado");
            LimpiarTable();
            ListarPlatos(tabla: TablePlatos);
            LimpiarPlatos();
        }
    } else {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Los campos estan vacios");
    }
}

private void btnNuevoPlatoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    LimpiarPlatos();
}

private void TablePlatosMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int fila = TablePlatos.rowAtPoint(point: evt.getPoint());
    txtIdPlato.setText(: TablePlatos.getValueAt(row:fila, column: 0).toString());
    txtNombrePlato.setText(: TablePlatos.getValueAt(row:fila, column: 1).toString());
    txtPrecioPlato.setText(: TablePlatos.getValueAt(row:fila, column: 2).toString());
}

private void btnPlatosMouseClicked(java.awt.event.MouseEvent evt) {
    btnPlatos.setBackground(new Color(: 109,g: 109,b: 109)); //cambiar el fondo al darle click
    btnPlatos.setForeground(new Color(: 254,g: 147,b: 21));//cambia el color de las letras a amarillo
    //los demas se cambian a blancos

    btnSala.setForeground(fg: Color.WHITE);
    btnVentas.setForeground(fg: Color.WHITE);
    btnUsuarios.setForeground(fg: Color.WHITE);
    lblBtnAjustes.setForeground(fg: Color.WHITE);
    lblBtnCerrarSesion.setForeground(fg: Color.WHITE);

    jTabbedPane.setSelectedIndex(index: 8);
}

```

```
        LimpiarTable();
        ListarPlatos(tabla: TablePlatos);
    }

    private void btnPlatosMouseEntered(java.awt.event.MouseEvent evt) {
        btnPlatos.setBackground(new Color(r: 66,g: 71,b: 74)); //cambia el fondo del label
    }

    private void btnPlatosMouseExited(java.awt.event.MouseEvent evt) {
        btnPlatos.setBackground(new Color(r: 33,g: 36,b: 41)); // se devuelde al color inicial al sa
    }

    private void btnSalaMouseClicked(java.awt.event.MouseEvent evt) {
        btnSala.setBackground(new Color(r: 109,g: 109,b: 109));
        btnSala.setForeground(new Color(r: 254,g: 147,b: 21));

        btnPlatos.setForeground(fg: Color.WHITE);
        btnVentas.setForeground(fg: Color.WHITE);
        btnUsuarios.setForeground(fg: Color.WHITE);
        lblBtnAjustes.setForeground(fg: Color.WHITE);
        lblBtnCerrarSession.setForeground(fg: Color.WHITE);

        LimpiarTable();
        ListarSalas();
        jTabbedPane.setSelectedIndex(index: 1);
    }

}
```

```

private void btnSalaMouseEntered(java.awt.event.MouseEvent evt) {
    btnSala.setBackground(new Color(z: 66,g: 71,b: 74));
}

private void btnSalaMouseExited(java.awt.event.MouseEvent evt) {
    btnSala.setBackground(new Color(z: 33,g: 36,b: 41));
}

private void lblBtnAjustesMouseClicked(java.awt.event.MouseEvent evt) {
    lblBtnAjustes.setBackground(new Color(z: 109,g: 109,b: 109));
    lblBtnAjustes.setForeground(new Color(z: 254,g: 147,b: 21));

    btnPlatos.setForeground(fg: Color.WHITE);
    btnSala.setForeground(fg: Color.WHITE);
    btnVentas.setForeground(fg: Color.WHITE);
    btnUsuarios.setForeground(fg: Color.WHITE);
    lblBtnCerrarSesion.setForeground(fg: Color.WHITE);

    jTabbedPane.setSelectedIndex(index: 6);
    ListarConfig();
}

private void lblBtnAjustesMouseEntered(java.awt.event.MouseEvent evt) {
    lblBtnAjustes.setBackground(new Color(z: 66,g: 71,b: 74));
}

private void lblBtnAjustesMouseExited(java.awt.event.MouseEvent evt) {
    lblBtnAjustes.setBackground(new Color(z: 33,g: 36,b: 41));
}

private void btnVentasMouseClicked(java.awt.event.MouseEvent evt) {
    btnVentas.setBackground(new Color(z: 109,g: 109,b: 109));
    btnVentas.setForeground(new Color(z: 254,g: 147,b: 21));

    btnPlatos.setForeground(fg: Color.WHITE);
    btnSala.setForeground(fg: Color.WHITE);
    btnUsuarios.setForeground(fg: Color.WHITE);
    lblBtnAjustes.setForeground(fg: Color.WHITE);
    lblBtnCerrarSesion.setForeground(fg: Color.WHITE);

    LimpiaTable();
    ListarPedidos();
    jTabbedPane.setSelectedIndex(index: 5);
}

private void btnVentasMouseEntered(java.awt.event.MouseEvent evt) {
    btnVentas.setBackground(new Color(z: 66,g: 71,b: 74));
}

private void btnVentasMouseExited(java.awt.event.MouseEvent evt) {
    btnVentas.setBackground(new Color(z: 33,g: 36,b: 41));
}

private void btnUsuariosMouseClicked(java.awt.event.MouseEvent evt) {
    btnUsuarios.setBackground(new Color(z: 109,g: 109,b: 109));
    btnUsuarios.setForeground(new Color(z: 254,g: 147,b: 21));

    btnPlatos.setForeground(fg: Color.WHITE);
    btnSala.setForeground(fg: Color.WHITE);
}

btnVentas.setForeground(fg: Color.WHITE);
lblBtnAjustes.setForeground(fg: Color.WHITE);
lblBtnCerrarSesion.setForeground(fg: Color.WHITE);

LimpiaTable();
ListarUsuarios();
jTabbedPane.setSelectedIndex(index: 7);

private void btnUsuariosMouseEntered(java.awt.event.MouseEvent evt) {
    btnUsuarios.setBackground(new Color(z: 66,g: 71,b: 74));
}

private void btnUsuariosMouseExited(java.awt.event.MouseEvent evt) {
    btnUsuarios.setBackground(new Color(z: 33,g: 36,b: 41));
}

private void btnConfigMouseClicked(java.awt.event.MouseEvent evt) {
    jTabbedPane.setSelectedIndex(index: 0);
    ListarConfig();
}

private void lblBtnCerrarSessionMouseClicked(java.awt.event.MouseEvent evt) {
    lblBtnCerrarSesion.setBackground(new Color(z: 109,g: 109,b: 109));
    lblBtnCerrarSesion.setForeground(new Color(z: 254,g: 147,b: 21));

    btnPlatos.setForeground(fg: Color.WHITE);
    btnSala.setForeground(fg: Color.WHITE);
    btnVentas.setForeground(fg: Color.WHITE);
    btnUsuarios.setForeground(fg: Color.WHITE);
    lblBtnAjustes.setForeground(fg: Color.WHITE);

    int r=JOptionPane.showConfirmDialog(parentComponent:null,
ssage:"Desea cerrar sesion",title: "Cerrar sesion?",mType: JOptionPane.YES_NO_OPTION,messageType: JOptionPane.QUESTION_MESSAGE);
if(r==JOptionPane.YES_OPTION){
    dispose();
    FrmLogin login= new FrmLogin();
    login.setVisible(b: true);
}
}

private void lblBtnCerrarSessionMouseEntered(java.awt.event.MouseEvent evt) {
    lblBtnCerrarSesion.setBackground(new Color(z: 66,g: 71,b: 74));
}

private void lblBtnCerrarSessionMouseExited(java.awt.event.MouseEvent evt) {
    lblBtnCerrarSesion.setBackground(new Color(z: 33,g: 36,b: 41));
}

```

```

private void TotalPagar(JTable tabla, JLabel label) {
    Totalpagar = 0.00;
    int numFila = tabla.getRowCount();
    for (int i = 0; i < numFila; i++) {
        double cal = Double.parseDouble(tabla.getValueAt(i, columnIndex));
        Totalpagar += cal;
    }
    label.setText(String.format(format: "%,.2f", args: Totalpagar));
}

private void LimpiarTableMenu() {
    tmp = (DefaultTableModel) tableMenu.getModel();
    int fila = tableMenu.getRowCount();
    for (int i = 0; i < fila; i++) {
        tmp.removeRow(row: 0);
    }
}

public void ListarConfig() {
    conf = lgDao.datosEmpresa();
    txtIdConfig.setText(" " + conf.getId());
    txtRucConfig.setText(" " + conf.getRuc());
    txtNombreConfig.setText(" " + conf.getNombre());
    txtTelefonoConfig.setText(" " + conf.getTelefono());
    txtDireccionConfig.setText(" " + conf.getDireccion());
    txtMensaje.setText(" " + conf.getMensaje());
}

private void ListarPedidos() {
    Tables color = new Tables();
    List<Pedidos> Listar = pedDao.listarPedidos();
    modelo = (DefaultTableModel) TablePedidos.getModel();
    Object[] ob = new Object[7];
    for (int i = 0; i < Listar.size(); i++) {
        ob[0] = Listar.get(index: i).getId();
        ob[1] = Listar.get(index: i).getSala();
        ob[2] = Listar.get(index: i).getUsuario();
        ob[3] = Listar.get(index: i).getNum_mesa();
        ob[4] = Listar.get(index: i).getFecha();
        ob[5] = Listar.get(index: i).getTotal();
        ob[6] = Listar.get(index: i).getEstado();
        modelo.addRow(rowData: ob);
    }
    colorHeader(tabla: TablePedidos);
    TablePedidos.setDefaultRenderer(columnClass: Object.class,
                                    renderer: color);
}

public void LimpiarTable() {
    for (int i = 0; i < modelo.getRowCount(); i++) {
        modelo.removeRow(row: i);
        i = i - 1;
    }
}

private void ListarUsuarios() {
    List<login> Listar = lgDao.ListarUsuarios();
    modelo = (DefaultTableModel) TableUsuarios.getModel();
    Object[] ob = new Object[4];
    for (int i = 0; i < Listar.size(); i++) {
}

```

```

        ob[0] = Listar.get(index: i).getId();
        ob[1] = Listar.get(index: i).getNombre();
        ob[2] = Listar.get(index: i).getCorreo();
        ob[3] = Listar.get(index: i).getRol();
        modelo.addRow(rowData: ob);
    }
    colorHeader(tabla: TableUsuarios);
}

private void ListarSalas() {
    List<Salas> Listar = slDao.Listar();
    modelo = (DefaultTableModel) tableSala.getModel();
    Object[] ob = new Object[3];
    for (int i = 0; i < Listar.size(); i++) {
        ob[0] = Listar.get(index: i).getId();
        ob[1] = Listar.get(index: i).getNombre();
        ob[2] = Listar.get(index: i).getMesas();
        modelo.addRow(rowData: ob);
    }
    colorHeader(tabla: tableSala);
}

private void colorHeader(JTable tabla) {
    tabla.setModel(dataModel: modelo);
    JTableHeader header = tabla.getTableHeader();
    header.setOpaque(isOpaque: false);
    header.setBackground(new Color(r: 0, g: 110, b: 255));
    header.setForeground(fg: Color.white);

    private void LimpiaSala() {
        txtIdSala.setText(" ");
        txtNombreSala.setText(" ");
        txtMesas.setText(" ");
    }

    private void LimpiaPlatos() {
        txtIdPlato.setText(" ");
        txtNombrePlato.setText(" ");
        txtPrecioPlato.setText(" ");
    }

    private void panelSalas() {
        List<Salas> Listar = slDao.Listar();
        for (int i = 0; i < Listar.size(); i++) {
            int id = Listar.get(index: i).getId();
            int cantidad = Listar.get(index: i).getMesas();
            JButton boton = new JButton(
                text: Listar.get(index: i).getNombre(),
                location: getClass().getResource(name: "/Img/salaofi.png"));
            boton.setCursor(new Cursor(type: Cursor.HAND_CURSOR));
            boton.setHorizontalTextPosition(textPosition: JButton.CENTER);
            boton.setVerticalTextPosition(textPosition: JButton.BOTTOM);
            boton.setBackground(bg: Color.WHITE);
            PanelSalas.add(comp: boton);
            boton.addActionListener((ActionEvent e) -> {
                LimpiarTable();
                PanelMesas.removeAll();
                panelMesas(id_sala: id, cant: cantidad);
                jTabbedPane.setSelectedIndex(index: 2);
            });
        }
    }
}

```

```

//crear mesas
private void panelMesas(int id_sala, int cant) {
    for (int i = 1; i <= cant; i++) {
        int num_mesa = i;
        //verificar estado
        JButton boton = new JButton("MESA N°: " + i, new ImageIcon(
            location: getClass().getResource(name: "/Img/inicio_mesas.png")));
        boton.setHorizontalTextPosition(textPosition: JButton.CENTER);
        boton.setVerticalTextPosition(textPosition: JButton.BOTTOM);
        int verificar = pedDao.verificarEstado(mesa: num_mesa, id_sala);
        if (verificar > 0) {
            boton.setBackground(bg: Color.WHITE);
        } else {
            boton.setBackground(new Color(z: 1, g: 202, b: 2));
        }
        boton.setForeground(fg: Color.WHITE);
        boton.setFocusable(focusable: false);
        boton.setCursor(cursor: Cursor(type: Cursor.HAND_CURSOR));
        PanelMesas.add(comp: boton);
        boton.addActionListener((ActionEvent e) -> {
            if (verificar > 0) {
                LimpiarTable();
                verPedido(id_pedido: verificar);
                verPedidoDetalle(id_pedido: verificar);
                btnFinalizar.setEnabled(b: true);
                btnPdfPedido.setEnabled(b: false);
                jTabbedPane.setSelectedIndex(index: 4);
            } else {
                LimpiarTable();
                ListarPlatos(tabla: tblTempPlatos);
                jTabbedPane.setSelectedIndex(index: 3);
                txtTempIdSala.setText("+" + id_sala);
                txtTempNumMesa.setText("+" + num_mesa);
            }
        });
    }
}

// platos
private void ListarPlatos(JTable tabla) {
    List<Platos> Listar = pl Dao.Listar(valor: txtBuscarPlato.getText(), fecha: fechaFormato);
    modelo = (DefaultTableModel) tabla.getModel();
    Object[] ob = new Object[3];
    for (int i = 0; i < Listar.size(); i++) {
        ob[0] = Listar.get(index: i).getId();
        ob[1] = Listar.get(index: i).getNombre();
        ob[2] = Listar.get(index: i).getPrecio();
        modelo.addRow(rowData: ob);
    }
    colorHeader(tabla);
}

//registrar pedido
private void RegistrarPedido() {
    int id_sala = Integer.parseInt(z: txtTempIdSala.getText());
    int num_mesa = Integer.parseInt(z: txtTempNumMesa.getText());
    double monto = Totalpagar;
    ped.setId_sala(id_sala);
    ped.setNum_mesa(num_mesa);
    ped.setTotal(total: monto);
}

```

```

        ped.setUsuario(usuario:LabelVendedor.getText());
        pedDao.RegistrarPedido(ped);
    }

    private void detallePedido() {
        int id = pedDao.IdPedido();
        for (int i = 0; i < tableMenu.getRowCount(); i++) {
            String nombre = tableMenu.getValueAt(row:1, column: 1).toString();
            int cant = Integer.parseInt(tableMenu.getValueAt(row:i, column: 2).toString());
            double precio = Double.parseDouble(tableMenu.getValueAt(row:i, column: 3).toString());
            detPedido.setNombre(nombre);
            detPedido.setCantidad(cantidad:cant);
            detPedido.setPrecio(precio);
            detPedido.setComentario(comentario: tableMenu.getValueAt(row:i, column: 5).toString());
            detPedido.setId_pedido(id_pedido: id);
            pedDao.RegistrarDetalle(det:detPedido);
        }
    }

    public void verPedidoDetalle(int id_pedido) {
        List<DetallePedido> Listar = pedDao.verPedidoDetalle(id_pedido);
        modelo = (DefaultTableModel) tableFinalizar.getModel();
        Object[] ob = new Object[6];
        for (int i = 0; i < Listar.size(); i++) {
            ob[0] = Listar.get(index: i).getId();
            ob[1] = Listar.get(index: i).getNombre();
            ob[2] = Listar.get(index: i).getCantidad();
            ob[3] = Listar.get(index: i).getPrecio();
            ob[4] = Listar.get(index: i).getCantidad() * Listar.get(index: i).getPrecio();
            ob[5] = Listar.get(index: i).getComentario();
            modelo.addRow(rowData:ob);
        }
        colorHeader(tabla: tableFinalizar);
    }

    public void verPedido(int id_pedido) {
        ped = pedDao.verPedido(id_pedido);
        totalFinalizar.setText(""+ ped.getTotal());
        txtFechaHora.setText(""+ ped.getFecha());
        txtSalaFinalizar.setText(""+ ped.getSala());
        txtNumMesaFinalizar.setText(""+ ped.getNum_mesa());
        txtIdPedido.setText(""+ ped.getId());
    }

}

```

VII. CRONOGRAMA DEL PROYECTO

Cronograma de actividades	Fecha de inicio	Fecha de finalización	Asignado	Estado	07.06.2024	14.06.2024	21.06.2024	28.06.2024	05.07.2024	12.07.2024	19.07.2024	26.07.2024	02.08.2024	09.08.2024	16.08.2024	23.08.2024	30.08.2024	06.09.2024	13.09.2024
REQUISITOS	07.06.2024	21.06.2024	Todos	Terminado															
Recolección de requisitos con los mosos y cajero	10.06.2024	14.06.2024	Todos	Terminado															
Analisis de los requisitos y documentacion	16.06.2024	21.06.2024	Todos	Terminado															
DISEÑO	28.06.2024	05.07.2024	Todos	Terminado															
<i>Diseño de la arquitectura del sistema</i>	29.06.2024	01.07.2024	Todos	Terminado															
<i>Diseño de la base de datos</i>	02.07.2024	02.07.2024	Todos	Terminado															
<i>Diseño de la interfaz de usuario</i>	02.07.2024	02.07.2024	Todos	Terminado															
Revision y aprobacion del diseño	03.07.2024	05.07.2024	Todos	Terminado															
IMPLEMENTACIÓN	12.07.2024	19.07.2024	Todos	Terminado															
<i>desarrollo del modulo de gestion de pedidos</i>	12.07.2024	13.07.2024	Todos	Terminado															
<i>Desarrollo del modulo de gestion de cobros</i>	13.07.2024	14.07.2024	Todos	Terminado															
<i>integracion de modulos</i>	15.07.2024	19.07.2024	Todos	Terminado															
PRUEBAS			Todos																
Pruebas unitarias			Todos																
Pruebas de integracion			Todos																
pruebas del sistema			Todos																
Pruebas de aceptacion			Todos																
DESPLIEGUE			Todos																
<i>Intalacion y configuracion del sistema en el entorno de produccion</i>			Todos																
capacitacion			Todos																
Mantenimiento			Todos																
Resolucion de problemas y ajustes post-despliegue			Todos																
FIN																			

VIII. CONCLUSIÓN

El proyecto de desarrollo de un software para la gestión de pedidos de una pizzería tiene como fin automatizar y optimizar procesos clave, como la toma de pedidos y el reporte de venta por cada mesa , con el objetivo de mejorar la eficiencia operativa y reducir errores. Este sistema busca agilizar las tareas del personal, permitiendo una atención más rápida y precisa a los clientes, lo que a su vez aumenta la satisfacción y reduce los tiempos de espera.

Uno de los principales beneficios es la especialización de módulos para cada rol, como mozos y cajeros, optimizando sus funciones y minimizando errores. Además, el sistema facilita el control en tiempo real de las operaciones y ajusta procesos según sea necesario, mejorando así la toma de decisiones.

Finalmente, este proyecto contribuye significativamente a la eficiencia y rentabilidad de la pizzería, proporcionando una herramienta tecnológica que no solo facilita las operaciones diarias, sino que también mejora la experiencia tanto de los empleados como de los clientes. Su implementación es un paso crucial hacia la modernización y sostenibilidad del negocio a largo plazo.

IX. ANEXOS

Anexo N° 1: 07/06/2024



Anexo N° 2: 07/06/2024



Anexo N° 3: 29/06/2024



Anexo N° 4: 29/07/2024



Anexo N° 5: 29/07/2024



Anexo N° 6: 29/07/2024



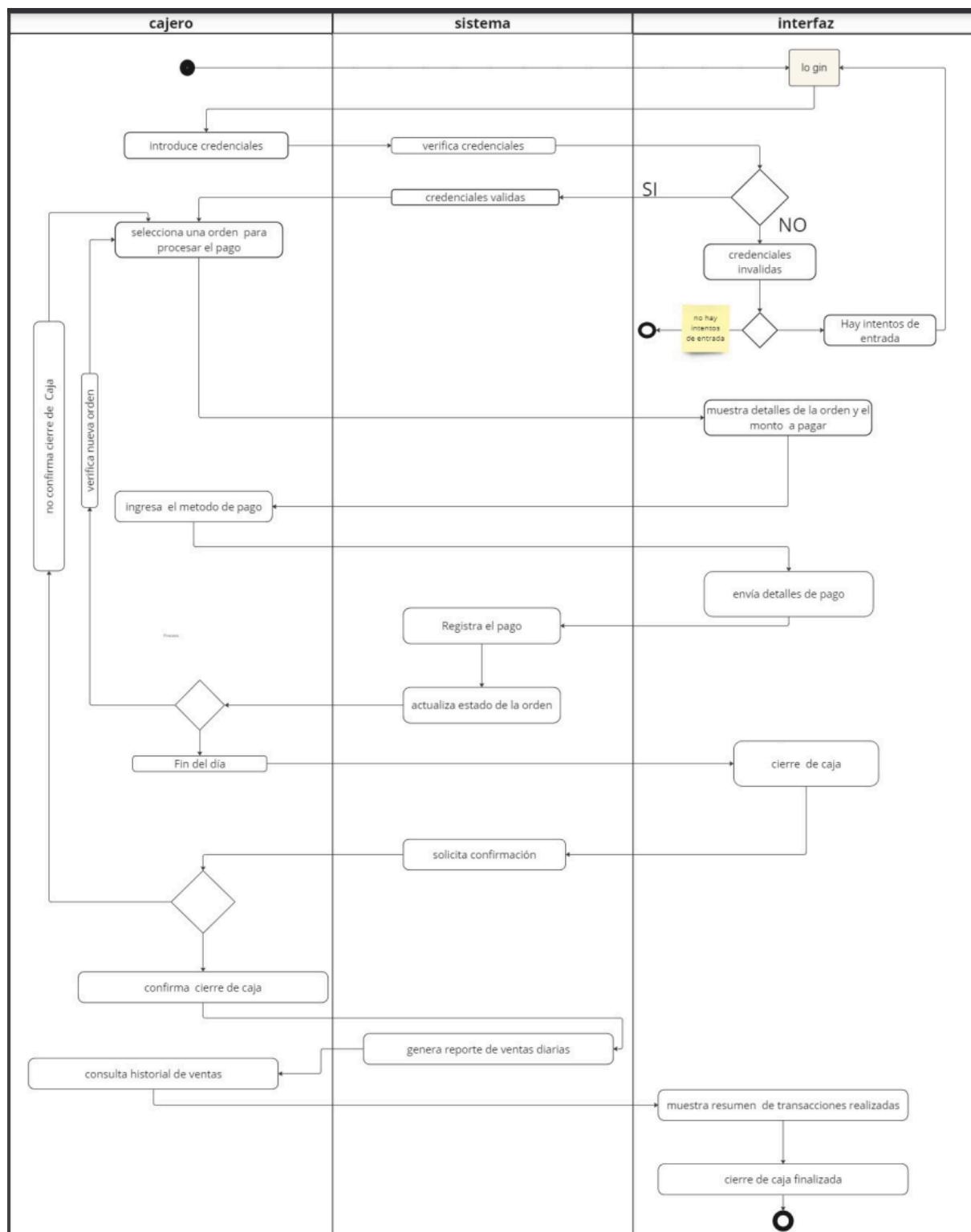
Anexo N° 7: 20/08/2024



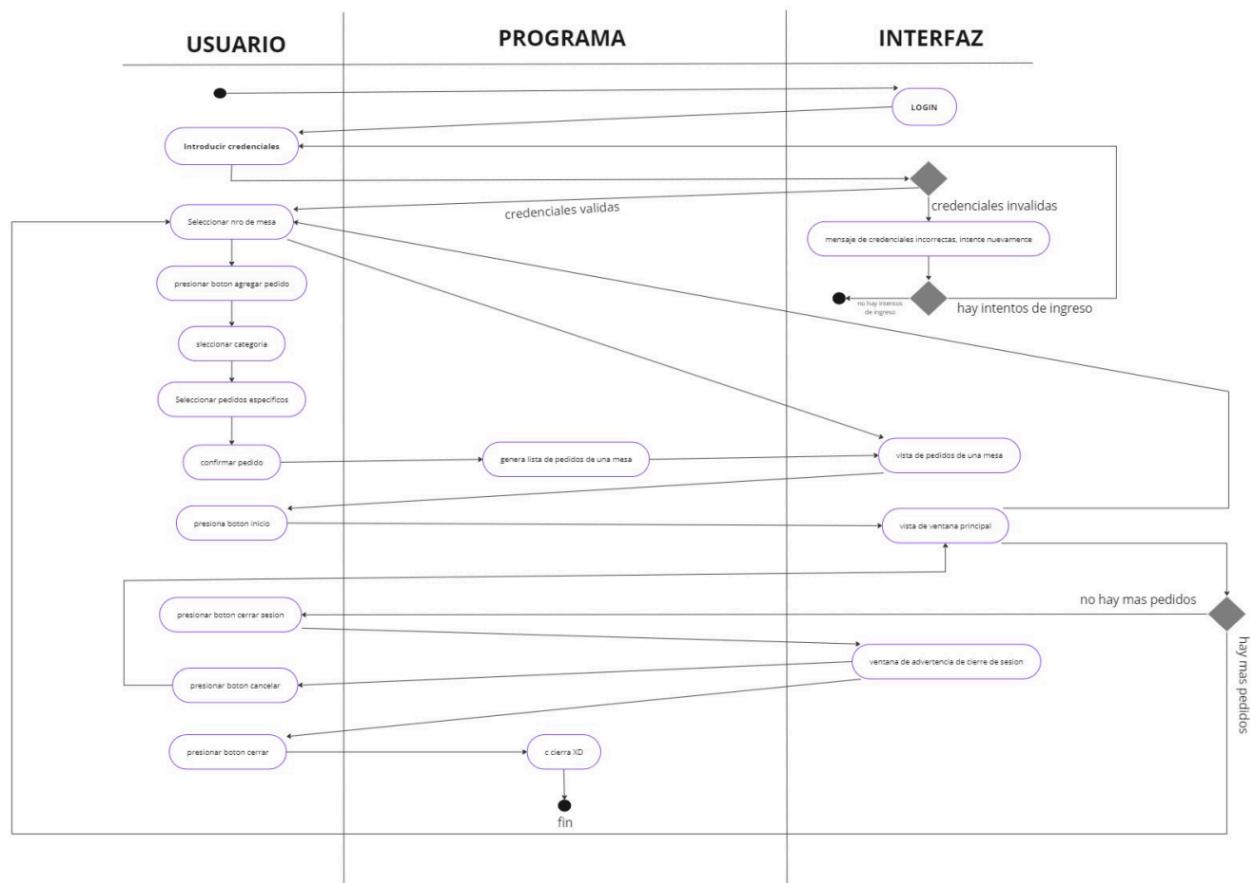
Anexo N° 8: 22/08/2024 (*Pizzeria Adriana*)



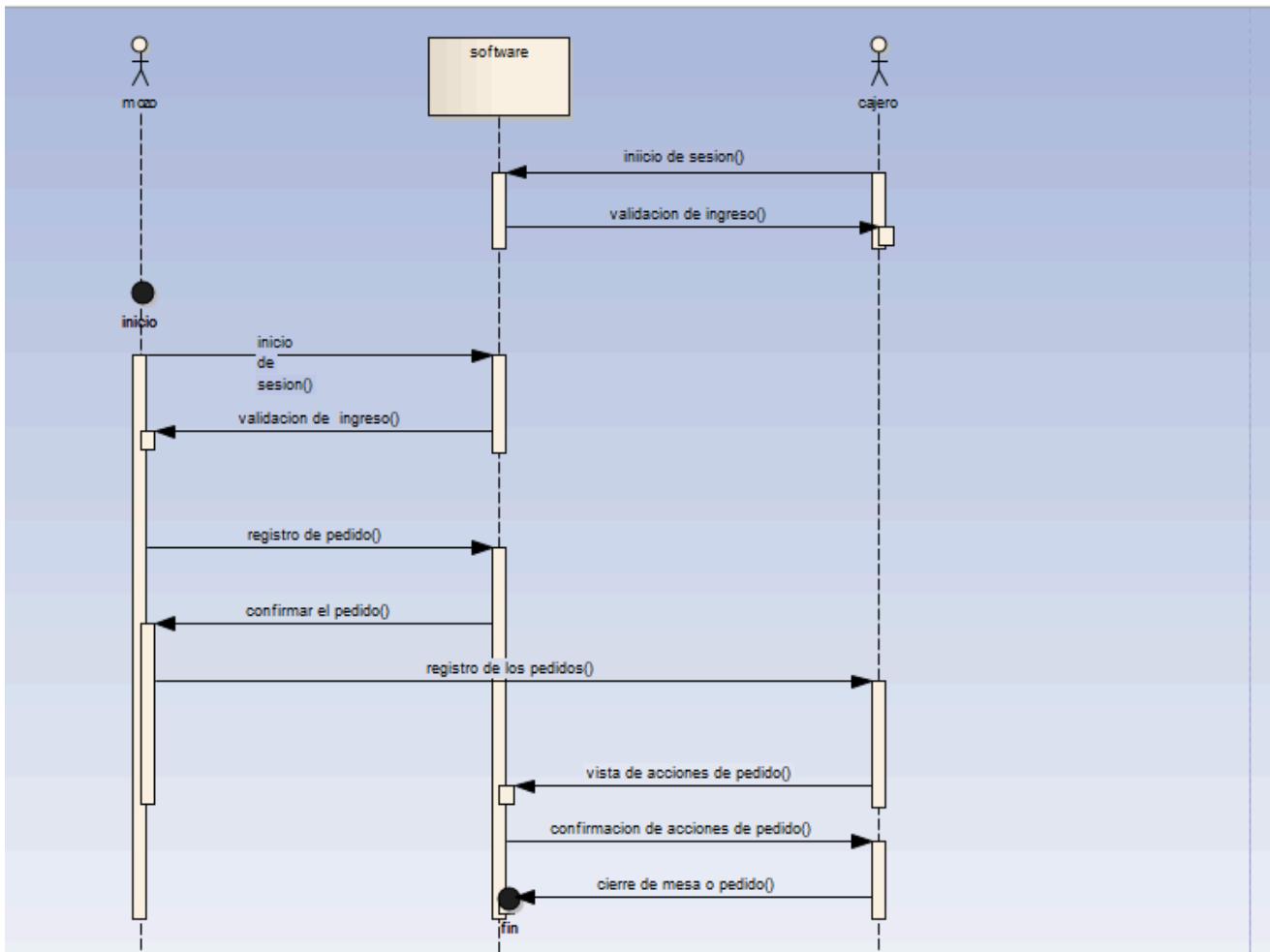
Anexo N° 9: *Diagrama de Secuencia*



Anexo N° 10: *Diagrama de Secuencia N°2*



Anexo N° 11: Diagrama de Secuencia (Architec).



X. REFERENCIAS BIBLIOGRÁFICAS

- Becerra Real , MY, Navarro Avalos, JS, Sedano Cruz, JF. (22 jul 2017).IEEE 830 1998: *Software Requirements Specification (Especificación de requisitos de software)*.slideshare a scribd company.
<https://es.slideshare.net/slideshow/software-requirements-specification-especificacion-de-requisitos-de-software/78149904>
- Tomas Sepulveda. (17 may 2000). *Flujo Caja Economico Sapag, Proyectos de Inversión*. YouTube. <https://youtu.be/argyZjuKlak?si=HnhVrtNpQDJmP-Ih>
- Robert Ormeño. (15 mar 2000). [Enterprise Architect] *Modelo de Casos de Uso de Negocio*. YouTube. https://youtu.be/Aglz_F8QRB8?si=1C3oJv8fwY9tOLkP
- Robert Ormeño. (29 jun 2018). [Enterprise Architect] MCUN - Caso: *Calidad Educativa*. YouTube. <https://youtu.be/dginITE4aRg?si=bRw2VJyzx2FIRHyh>
- Robert Ormeño. (29 dic 2018). [Enterprise Architect] Laboratorio 1 - *Análisis y Diseño de Sistemas*. YouTube.
<https://youtu.be/dginITE4aRg?si=bRw2VJyzx2FIRHyh>
- Amilcar, Cristian O, Cristian A, Luis JP, Tito C. (2024). *Dashboard del software/Gestión de Restaurantes [Diseño de Figma]*. Figma.
<https://www.figma.com/design/Ut1l9FlVm1clLieQ49H5Bb/Dashboard-del-software%2FGestion-de-Restaurantes?node-id=0-1&t=2eBYujFjLCkRouth-0>