Licenciatura en Informática

Seminario de Práctica de Informática

Sistema de gestión y monitoreo de presupuestos

Docente/Tutor: Frias Hugo Fernando

Nombre Alumno: Benítez Micael Sebastián

Legajo: VINF09347

Fecha: 27/10/2024

Introducción:

El presente proyecto de desarrollo de software, está destinado a la empresa Placas y Maderas que se dedica a la venta de materiales e insumos para el hogar, tiene 10 años de antigüedad y hace 6 meses que expandió considerablemente su local comercial atrayendo más clientes, y con esto un crecimiento exponencial en el flujo de clientes y ventas.

Actualmente la empresa consta con un enorme local comercial ubicado en San Vicente Misiones, tiene 10 departamentos de ventas (Materiales de construcción, placas de maderas, pinturería, artística, artículos de piscina, bazar, jardinería, electricidad y ferretería).

En el contexto de la empresa, se ha identificado la necesidad de implementar un sistema de optimización y seguimiento de presupuestos. Este sistema se concibe como una herramienta fundamental para mejorar la eficiencia en la gestión de ventas y garantizar una atención más afectiva a los clientes

El principal desafío al que se enfrenta Placas y Maderas es la falta de un sistema centralizado que permita el seguimiento adecuado de los presupuestos generados para los clientes. Actualmente, el proceso de seguimiento se realiza de manera manual, lo que conlleva a la perdida de oportunidades de ventas, falta de seguimiento en el proceso de negociación y falta de visibilidad sobre el estado de los presupuestos.

Justificación:

La implementación de un sistema de optimización y seguimiento de presupuestos es fundamental para mejorar la competitividad y eficiencia de Placas y Maderas. Al automatizar y centralizar este proceso, se espera agilizar la generación de presupuestos, mejorar la comunicación con los clientes y facilitar el seguimiento de las negociaciones. Esto no solo conducirá a un aumento en las ventas y la satisfacción del cliente, sino que también optimizará los recursos internos de la empresa al reducir el tiempo y esfuerzo dedicado a la gestión manual de presupuestos.

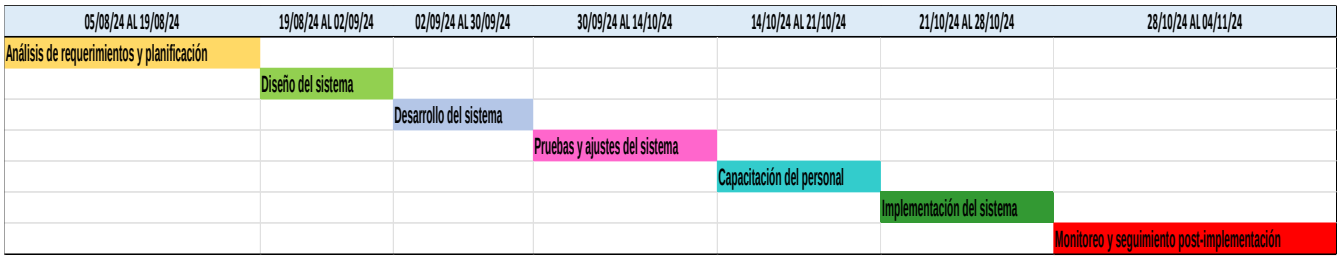
Objetivo general del proyecto:

Implementar un sistema de optimización y seguimiento de presupuestos en la empresa Placas y Maderas, con el fin de mejorar la eficiencia en la gestión de ventas, garantizar una atención más efectiva a los clientes, y fortalecer la competitividad en el mercado mediante la automatización y centralización del proceso de generación, seguimiento y negociación de presupuestos.

Diagrama de Gantt:

NOMBRE DE ACTIVIDAD	DURACION	FECHA DE INICIO	FECHA DE FINALIZACION
Análisis de requerimientos y planificación	2 SEMANAS	5/8/2024	19/8/2024
Diseño del sistema	2 SEMANAS	19/8/2024	2/9/2024
Desarrollo del sistema	4 SEMANAS	2/9/2024	30/9/2024
Pruebas y ajustes del sistema	2 SEMANAS	30/9/2024	14/10/2024
Capacitación del personal	1 SEMANA	14/10/2024	21/10/2024
Implementación del sistema	1 SEMANA	21/10/2024	28/10/2024
Monitoreo y seguimiento post-implementación	2 SEMANAS	28/10/2024	4/11/2024

Fuente: Elaboración propia.



Objetivos específicos del proyecto:

- **Automatización y Centralización del Proceso Presupuestario**: Desarrollar e implementar un sistema que permita la generación automática y centralizada de presupuestos, integrando todos los departamentos de Placas y Maderas para reducir el tiempo de respuesta y minimizar errores en la elaboración de presupuestos.
- **Mejora de la Atención al Cliente:** Establecer un sistema de seguimiento de presupuestos que permita a los empleados proporcionar información actualizada y precisa a los clientes en tiempo real, mejorando así la calidad del servicio y la satisfacción del cliente.
- **Optimización del Seguimiento y Análisis de Presupuestos:** Implementar herramientas de monitoreo y análisis que proporcionen a la dirección de Placas y Maderas una visión clara y detallada del estado de los presupuestos, facilitando la toma de decisiones estratégicas y mejorando la competitividad en el mercado mediante una gestión proactiva y eficiente de los recursos financieros.

Objetivo general del sistema:

Se desarrollará un sistema informático con una interfaz intuitiva que permitirá a los usuarios gestionar y monitorear los presupuestos de todos los departamentos de la empresa, incluyendo la carga de nuevos presupuestos, el listado de presupuestos existentes con todos los datos relevantes y consultas en tiempo real para obtener balances de los estados de los presupuestos. El sistema enviará notificaciones automáticas para alertar sobre presupuestos próximos a vencer, facilitando una gestión proactiva, y permitirá la impresión de listados para una mejor visualización y distribución de la información. Se garantizará la seguridad y confidencialidad de los datos mediante acceso controlado y cifrado de información sensible, y el diseño del sistema será escalable y flexible para adaptarse a futuras necesidades de la empresa.

Elicitación:

Para adquirir el conocimiento se realizaron distintas entrevistas. En primer lugar, se entrevistó al encargado de ventas, con el fin de recabar toda la información posible sobre el modelo de negocio que poseen y al que aspiran al concluir el proyecto. En segundo lugar, se entrevistó al personal de atención al cliente con el fin de conocer:

- El proceso de las emisiones de presupuestos.
- El proceso que realizan para el registro de presupuestos.
- El proceso de control y monitoreo.
- El proceso de postventa que realizan al emitir un presupuesto.
- Que software utilizan actualmente para el proceso.
- El hardware que poseen para dicho proceso.

Actividad del cliente:

Para el estudio de la actividad del cliente se plantearon las siguientes preguntas:

¿Qué modelo de negocio posee la empresa?

El modelo de negocio de Placas y Maderas es principalmente un modelo de venta minorista de materiales de construcción y productos relacionados. Este modelo se centra en la venta directa al público tanto en su local comercial como a través de mensajería instantánea o llamada telefónica. Además, el negocio cuenta con un sector de logística para la entrega de pedidos a domicilio, lo que se enfoca en la satisfacción y conveniencia del cliente.

¿Cómo podría integrarse este sistema de optimización y monitoreo de presupuestos al negocio?

Integrar este sistema de optimización y monitoreo de presupuestos al negocio de Placas y Maderas proporcionará una serie de beneficios que van desde la automatización y eficiencia hasta la mejora en la toma de decisiones. Esto contribuirá a mejorar el rendimiento general del negocio.

Conocimiento del negocio:

El sistema que se va a desarrollar debe permitir la gestión y monitoreo de los presupuestos que se emiten en el comercio.

Tomando como referencia un comercio de ventas de materiales de construcción, todos los días se emiten presupuestos, el cliente solicita el presupuesto de forma presencial o por otros medios de comunicación a un personal de atención al público, este almacena una copia del presupuesto en papel, el cual se le da un lapso de tres días para comunicar al cliente que su solicitud está en fecha de vencimiento y ese momento es cuando se debe llegar a una conclusión si se va concretar la negociación o se cancela la solicitud.

Este es un proceso en el cual no se lleva un registro ni control eficaz del proceso, se pierden solicitudes por falta de registro y se también no hay ningún medio o sistema que notifique la caducidad de los mismos.

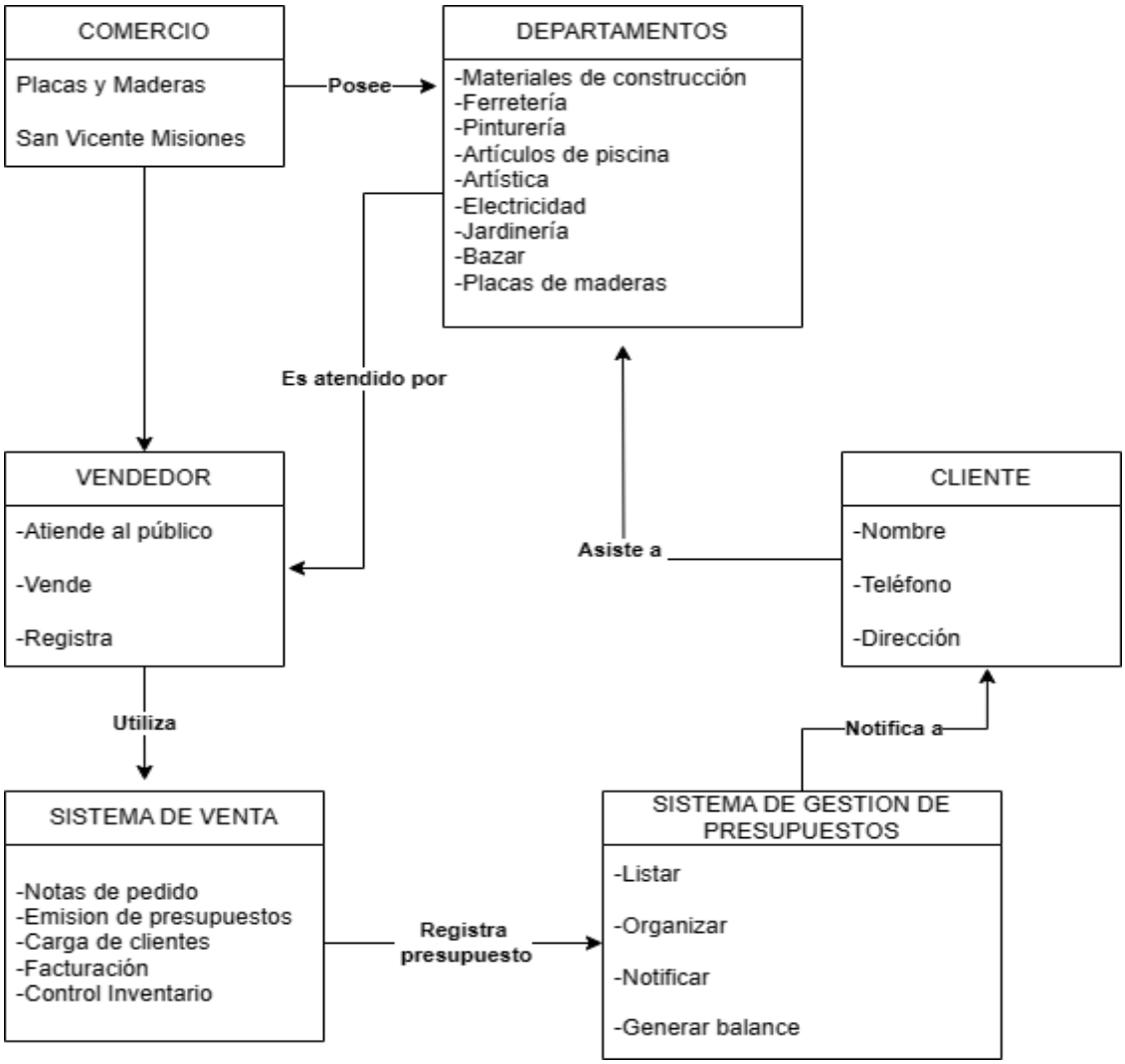
Para optimizar este proceso, se va a implementar en cada departamento de la empresa el sistema de gestión y monitoreo de presupuestos. Esto permitirá una gestión más eficiente y controlada de las solicitudes de presupuesto emitidas diariamente en el comercio de ventas de materiales de construcción. En lugar de depender únicamente de copias físicas de presupuestos almacenadas por el personal de atención al público, el sistema proporcionará una plataforma centralizada para registrar, monitorear y gestionar todas las solicitudes de presupuesto.

Con este sistema, cada vez que se emita un presupuesto, se registrará de manera digital en el sistema, lo que garantizará que ninguna solicitud se pierda o quede sin seguimiento. Además, el sistema automatizará el proceso de seguimiento al cliente al enviar notificaciones automáticas cuando un presupuesto esté por vencer, alertando al personal responsable para que tome las medidas necesarias.

Al contar con un registro centralizado de todos los presupuestos emitidos y un sistema de alertas para las fechas de vencimiento, el personal podrá realizar un seguimiento proactivo de las solicitudes pendientes, comunicarse con los clientes de manera oportuna y llegar a una

conclusión sobre la negociación en el plazo establecido. Esto mejorará la eficiencia operativa, reducirá la pérdida de oportunidades de negocio y garantizará una experiencia satisfactoria para los clientes al recibir una respuesta rápida y profesional a sus solicitudes de presupuesto.

Diagrama de dominio:



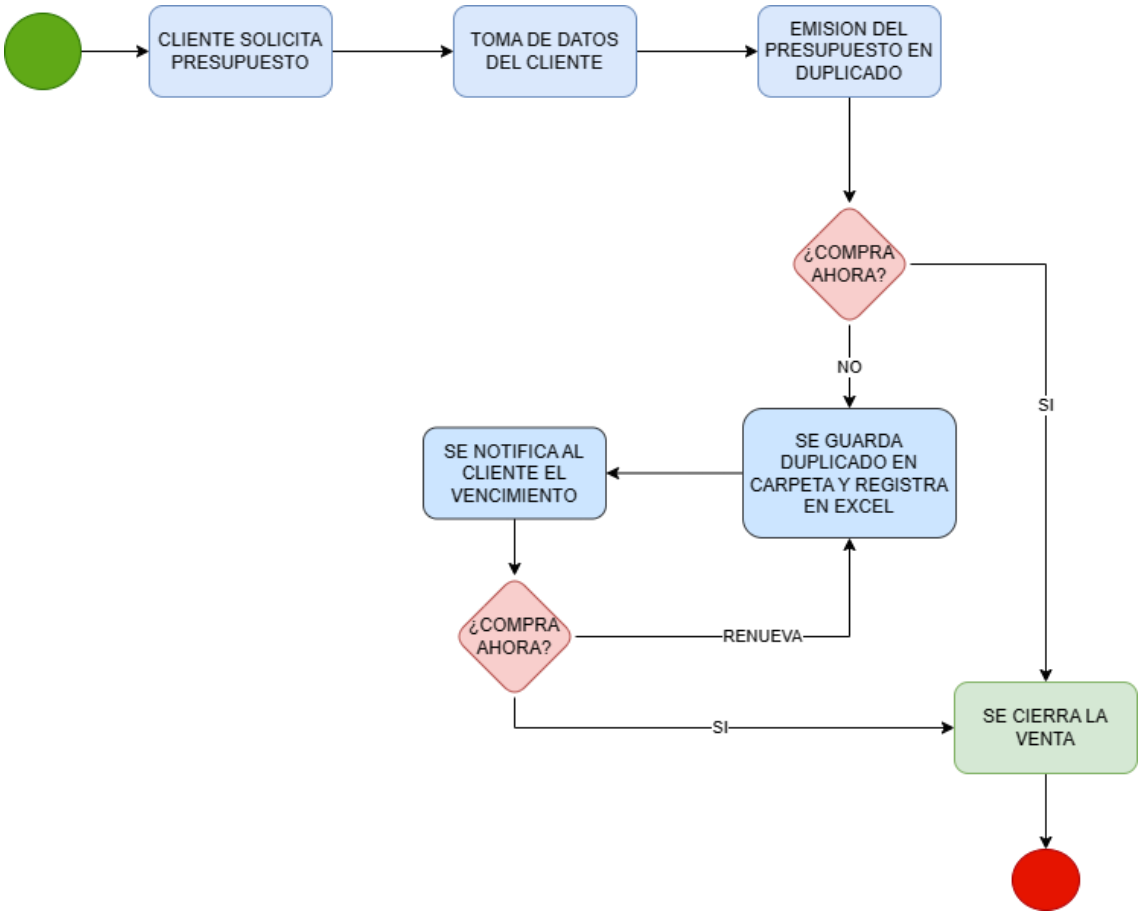
Fuente: Elaboración propia.

Proceso de negocio:

Proceso: Solicitud de presupuesto.

Roles: Personal de atención al público.

Pasos: El cliente solicita un presupuesto en algunos de los departamentos de la empresa, el vendedor toma los datos del cliente, emite el presupuesto desde el sistema de ventas y lo imprime en duplicado, si el cliente compra el presupuesto en el momento se hace cierre del presupuesto, sino un duplicado se registra en una carpeta y el otro le da al cliente, el personal de atención al publico tiene que crear el registro en un Excel del presupuesto solicitado y agregarle la fecha de vencimiento acorde al presupuesto, cuando es la fecha de vencimiento se le notifica al cliente para su cierre o renovación del mismo.



Fuente: Elaboración propia.

<u>Diagnóstico:</u>

Proceso: Solicitud de presupuestos.

Problema: Se ha identificado una problemática crucial que afecta la eficiencia y efectividad de las operaciones. El problema principal radica en la falta de un seguimiento constante en el control, registro, seguimiento y notificación de los presupuestos por parte del personal involucrado. Esta deficiencia conlleva a la no notificación oportuna de los presupuestos a los clientes, lo que resulta en la pérdida de ventas potenciales.

Causas: La raíz del problema se encuentra en el método actual de registro y seguimiento de los presupuestos, que se lleva a cabo mediante una planilla Excel. Este sistema no proporciona las herramientas necesarias para garantizar un seguimiento adecuado ni para notificar al personal sobre los plazos de vencimiento de los presupuestos. Como resultado, se produce una falta de control y visibilidad en el proceso, lo que dificulta la gestión eficiente de los presupuestos y la concreción de las ventas.

<u>Propuesta de solución:</u>

Para abordar esta situación de manera efectiva, es fundamental implementar un sistema de gestión y monitoreo de presupuestos robusto y automatizado que permita un seguimiento continuo y una notificación oportuna tanto al personal interno como a los clientes. Este sistema debería optimizar el registro de presupuestos, ofrecer funcionalidades de seguimiento de plazos y proporcionar alertas automáticas para asegurar que ningún presupuesto quede sin atención.

Con la implementación de estas medidas, se espera mejorar significativamente la eficiencia y la precisión en el proceso de solicitud de presupuestos, lo que a su vez contribuirá a aumentar la tasa de conversión de ventas y a fortalecer la relación con los clientes mediante una atención más diligente y oportuna.

Propuesta funcional:

- Alta de usuarios para acceso al sistema.
- Baja de usuarios según requerimientos de la empresa.
- Registro digital de presupuestos emitidos por departamento.
- Captura de datos del cliente.
- Asignación de fecha de emisión y vencimiento del presupuesto.
- Visualización de lista de presupuestos pendientes por departamento.
- Notificación automática de presupuestos próximos a vencer.
- Consulta de balance de estados de presupuestos en tiempo real.
- Interfaz de usuario intuitiva y fácil de usar.
- Seguridad y confidencialidad de los datos mediante control de acceso y cifrado.
- Diseño modular y escalable para adaptarse a futuras necesidades.
- Flexibilidad para integrar nuevas funcionalidades según requerimientos de la empresa.

Propuesta técnica:

El desarrollo del sistema de gestión y monitoreo de presupuestos para Placas y Maderas se llevará a cabo utilizando Java, un lenguaje de programación ampliamente reconocido por su robustez y amplia comunidad de desarrolladores. Java ofrece una amplia gama de bibliotecas y frameworks, como Spring, que facilitan el desarrollo de aplicaciones escalables y mantenibles.

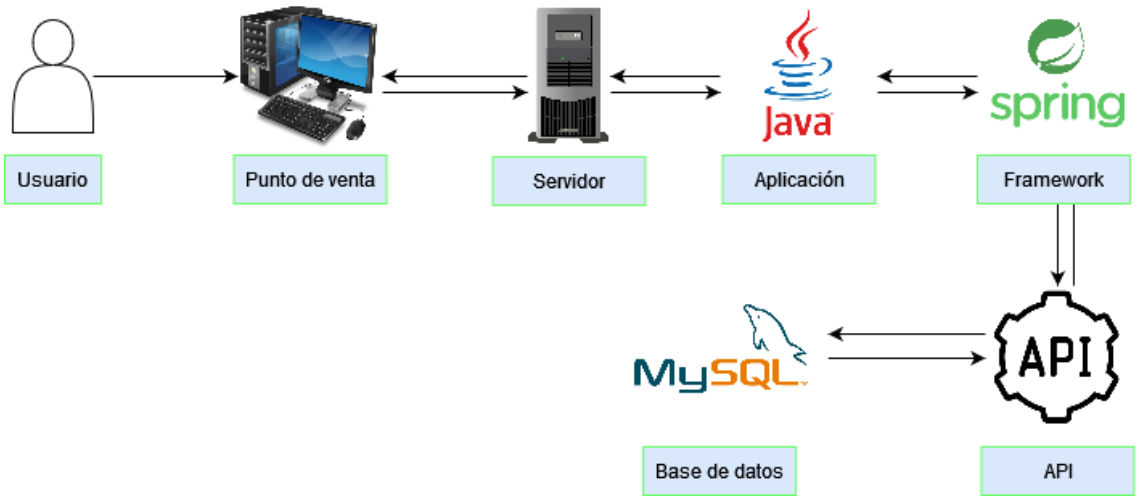
Para la gestión de la base de datos, se utilizará MySQL, una base de datos relacional que se distingue por su flexibilidad, rendimiento y confiabilidad en el almacenamiento de datos en tiempo real. La elección de MySQL proporciona una sólida base para el almacenamiento y recuperación eficiente de información crucial para el sistema.

Para interactuar con la base de datos desde Java, se empleará JDBC (Java Database Connectivity), una API estándar que permite a las aplicaciones Java conectarse y realizar operaciones de consulta y persistencia de datos en bases de datos relacionales. JDBC facilita la integración del sistema con la base de datos MySQL, asegurando una comunicación fluida y eficiente entre la aplicación y la fuente de datos.

La tecnología de comunicación a utilizar será la siguiente:

La comunicación dentro de la empresa se establecerá utilizando cableado Ethernet, una tecnología que garantiza una infraestructura de red confiable y eficiente. Esta elección es fundamental para asegurar la fiabilidad y el rendimiento necesarios en la gestión y monitoreo de los presupuestos.

La conexión entre el servidor y los puntos de ventas de los distintos departamentos está ubicado en la misma empresa, descartando un servidor en la nube que eleva el costo de mantenimiento. La empresa tiene el servidor y todos los puntos de ventas conectados a través del cableado ethernet de alta velocidad.



Fuente: Elaboración propia.

Requerimientos funcionales:

REQUERIMIENTO SISTEMA	DESCRIPCION
RFS01	El sistema debe permitir la gestión de usuarios.
RFS02	El sistema debe permitir la gestión de clientes.
RFS03	El sistema debe permitir el acceso mediante un usuario y contraseña.
RFS04	El sistema debe permitir a los usuarios cargar los presupuestos.
RFS05	El sistema debe permitir el envió de notificaciones a los clientes.
RFS06	El sistema debe registrar el vencimiento de los presupuestos.
RFS07	El sistema debe generar informes sobre estados de los presupuestos.

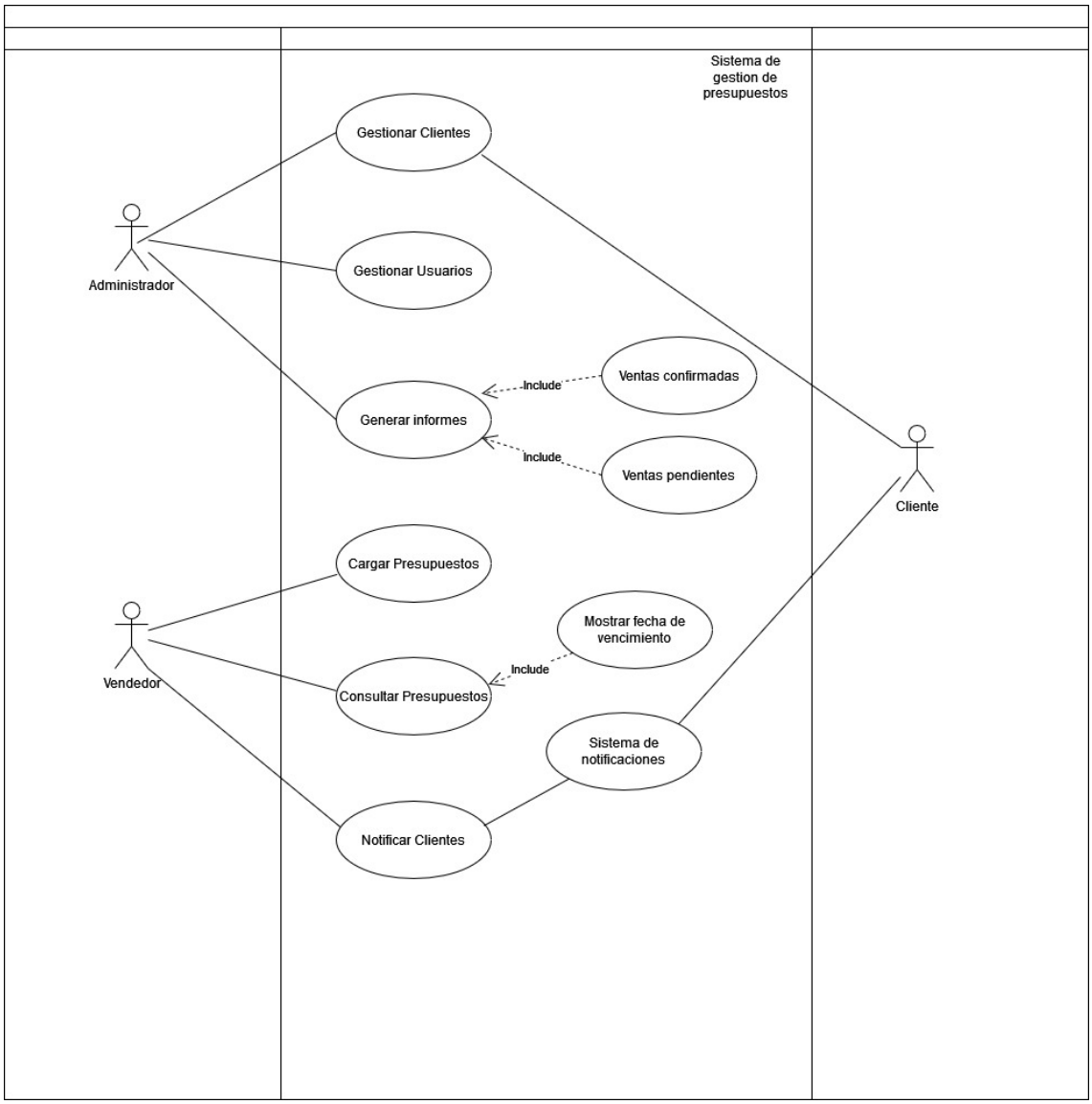
Fuente: Elaboración propia

Requerimientos no funcionales:

REQUERIMIENTO SISTEMA	DESCRIPCION
RNFS01	El sistema se ejecutará en aplicación.
RNFS02	El sistema dispondrá de una interfaz intuitiva y amigable.
RNFS03	El sistema va a estar desarrollado en Java.
RNFS04	El sistema debe ser escalable, es decir, debe poder agregarse alguna funcionalidad en caso de requerirse.
RNFS05	El sistema va a estar conectado a través de la intranet de la empresa

Fuente: Elaboración propia.

Diagrama de casos de uso:



Fuente: Elaboración propia.

Identificación de actores:

- **Administrador:** Es toda persona humana capaz de gestionar la carga de datos (cliente, usuario), y generar informes a partir de las ventas realizadas y pendientes.
- **Vendedor:** Es la persona humana que va a realizar la carga de presupuestos, consultarlos y notificar a sus clientes.
- **Cliente:** Es la persona que decide si va a realizar la compra o no.

Trazabilidad:

- CU001 Gestionar clientes
- CU002 Cargar presupuestos
- CU003 Notificar clientes

-Tabla de casos de uso

REQUERIMIENTO	CASO DE USO	ACTOR PRINCIPAL	PAQUETE DE ANALISIS	COMENTARIO
RFS02	CU001	Administrador	Gestión de cliente	Se realiza la carga de clientes tomando Nombre completo, domicilio, numero de teléfono.
RFS06	CU002	Vendedor	Consulta de presupuestos	Se realiza la consulta si el presupuesto esta próximo vencer o si necesita alguna modificación.
RFS05	CU003	Vendedor	Notificación a clientes	Se notifica al cliente si su presupuesto solicitado está en fecha de vencimiento, y si quiere renovarlo o cancelarlo.

Fuente: Elaboración propia.

-Descripción de casos de uso

Caso de uso	CU001 Gestión de cliente.	
Actores	Administrador.	
Referencias	RF02.	
Descripción	Permite al administrador agregar un cliente.	
Precondición	El administrador ha iniciado sesión al sistema. El administrador tiene los permisos necesarios para gestionar el cliente.	
Flujo principal	1	El administrador selecciona la opción "Agregar nuevo cliente" en la interfaz de "Clientes"
	2	El sistema muestra un formulario vacío para agregar los datos del nuevo cliente
	3	El administrador completa el formulario, insertando la siguiente información: * Nombre de cliente; *Apellido de cliente; *Domicilio; *Teléfono.
	4	El sistema valida la información ingresada por el administrador
	5	El administrador confirma la entrada de datos.
	6	El sistema agrega el nuevo cliente en la base de datos.
Postcondición	Se ha agregado un nuevo cliente al sistema con la información proporcionada por el administrador.	
Flujo alternativo	Validación de datos con falla. 1. El sistema muestra el mensaje "No se ingresaron los datos correctamente" 2. El sistema regresa al paso 2 para que el administrador corrija los datos	
Excepciones		No contempla.

Caso de uso	CU001 Gestión de cliente.	
Actores	Administrador.	
Referencias	RF02.	
Descripción	Permite al administrador modificar un cliente.	
Precondición	El administrador ha iniciado sesión al sistema. El administrador tiene los permisos necesarios para gestionar el cliente.	
Flujo principal	1	El administrador selecciona la opción "Modificar cliente" en la interfaz de "Clientes"
	2	El sistema muestra el formulario con la información del cliente.
	3	El administrador modifica el formulario con la siguiente información: * Nombre de cliente; *Apellido de cliente; *Domicilio; *Teléfono.
	4	El sistema valida la información ingresada por el administrador
	5	El administrador confirma la entrada de datos.
	6	El sistema modifica el cliente en la base de datos.
Postcondición	Se ha modificado el cliente en el sistema con la nueva información proporcionada por el administrador.	
Flujo alternativo	Validación de datos con falla. 1. El sistema muestra el mensaje "No se ingresaron los datos correctamente" 2. El sistema regresa al paso 2 para que el administrador corrija los datos	
Excepciones		No contempla.

Caso de uso	CU001 Gestión de cliente.	
Actores	Administrador.	
Referencias	RF02.	
Descripción	Permite al administrador eliminar un cliente.	
Precondición	El administrador ha iniciado sesión al sistema. El administrador tiene los permisos necesarios para gestionar el cliente.	
Flujo principal	1	El administrador selecciona la opción "Modificar cliente" en la interfaz de "Clientes"
	2	El sistema muestra el formulario con la información del cliente.
	3	El administrador da en la opción "Eliminar cliente" en la interfaz de "Modificar cliente"
	4	El sistema muestra un mensaje de dialogo de confirmación.
	5	El administrador confirma la opción en el cuadro de dialogo.
	6	El sistema elimina el cliente en la base de datos.
Postcondición	Se ha eliminado el cliente del sistema.	

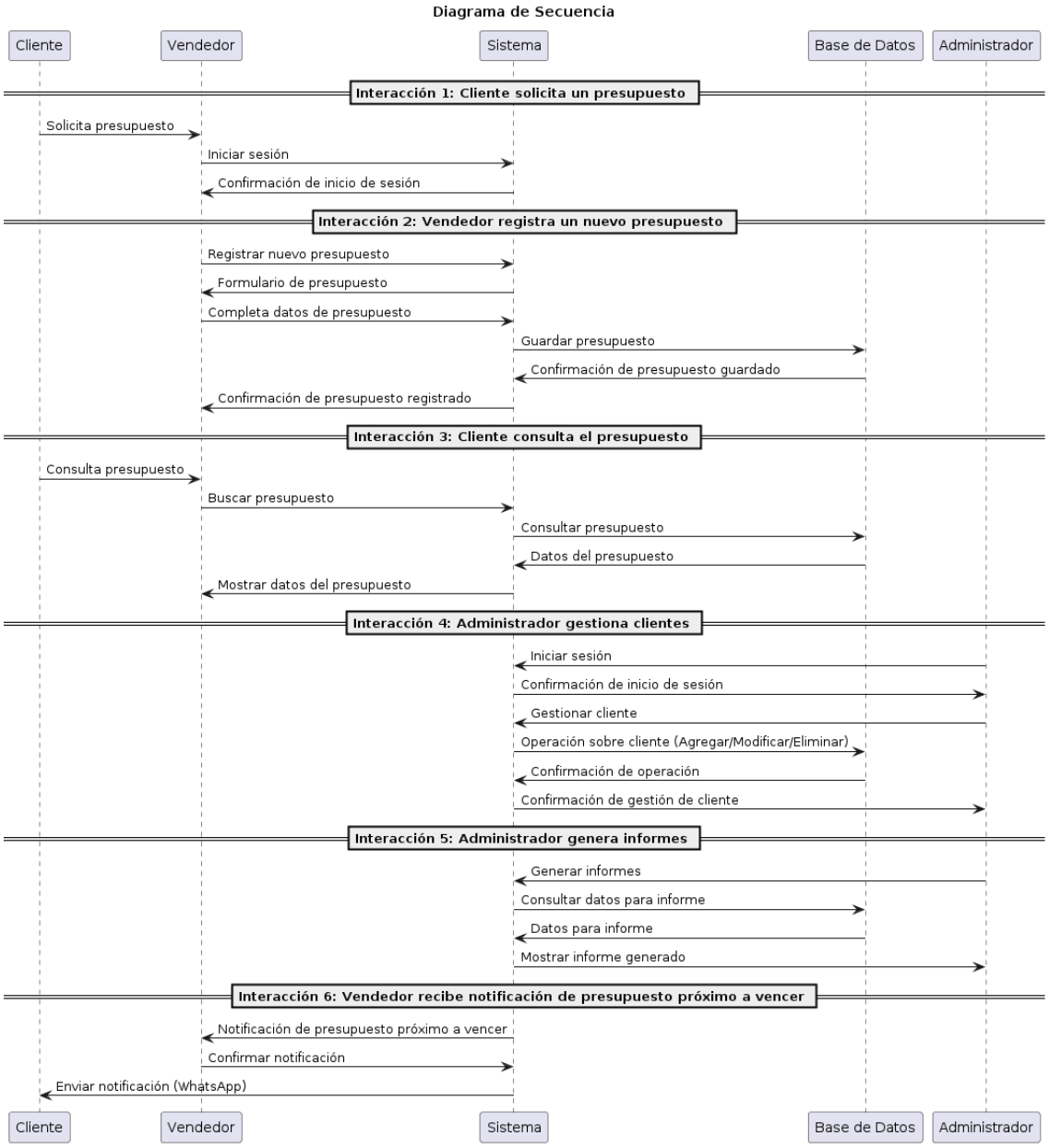
Flujo alternativo	No contempla	
Excepciones		No contempla.

Caso de uso	CU002 Cargar presupuestos.	
Actores	Vendedor.	
Referencias	RF03, RF04, RF06.	
Descripción	Permite al vendedor cargar los datos del presupuesto.	
Precondición	El vendedor ha iniciado sesión al sistema. El vendedor tiene los permisos necesarios para gestionar los presupuestos.	
Flujo principal	1	El vendedor selecciona la opción "Nuevo presupuesto" en la interfaz de "Gestión de presupuestos"
	2	El sistema muestra el formulario vacío para agregar los datos del nuevo presupuesto
	3	El vendedor completa el formulario con la siguiente información: *Cliente; *Numero de pedido; *Fecha de vencimiento.
	4	El sistema valida la información ingresada por el vendedor.
	5	El vendedor confirma la entrada de datos.
	6	El sistema agrega el nuevo presupuesto en la base de datos.
Postcondición	Se ha agregador el presupuesto al sistema con la información proporcionada por el vendedor.	
Flujo alternativo	Validación de datos con falla. 1. El sistema muestra el mensaje "No se ingresaron los datos correctamente" 2 El sistema regresa al paso 2 para que el vendedor corrija los datos.	
Excepciones		No contempla.

Caso de uso	CU003 Notificar clientes.	
Actores	Vendedor.	
Referencias	RF03, RF05.	
Descripción	Permite al vendedor notificar a los clientes el estado de sus presupuestos.	
Precondición	El vendedor ha iniciado sesión al sistema. El vendedor tiene los permisos necesarios para notificar al cliente.	
Flujo principal	1	El vendedor selecciona la opción "Lista presupuestos" en la interfaz de "Gestión de presupuestos"
	2	El sistema muestra la lista de presupuestos con los presupuestos próximos a vencer de color rojo.
	3	El vendedor selecciona el presupuesto próximo a vencer.
	4	El sistema muestra un cuadro de dialogo con el siguiente mensaje "Notificar al cliente"
	5	El vendedor confirma la opción.
	6	El sistema notifica al cliente a través de mensaje de texto vía WhatsApp.

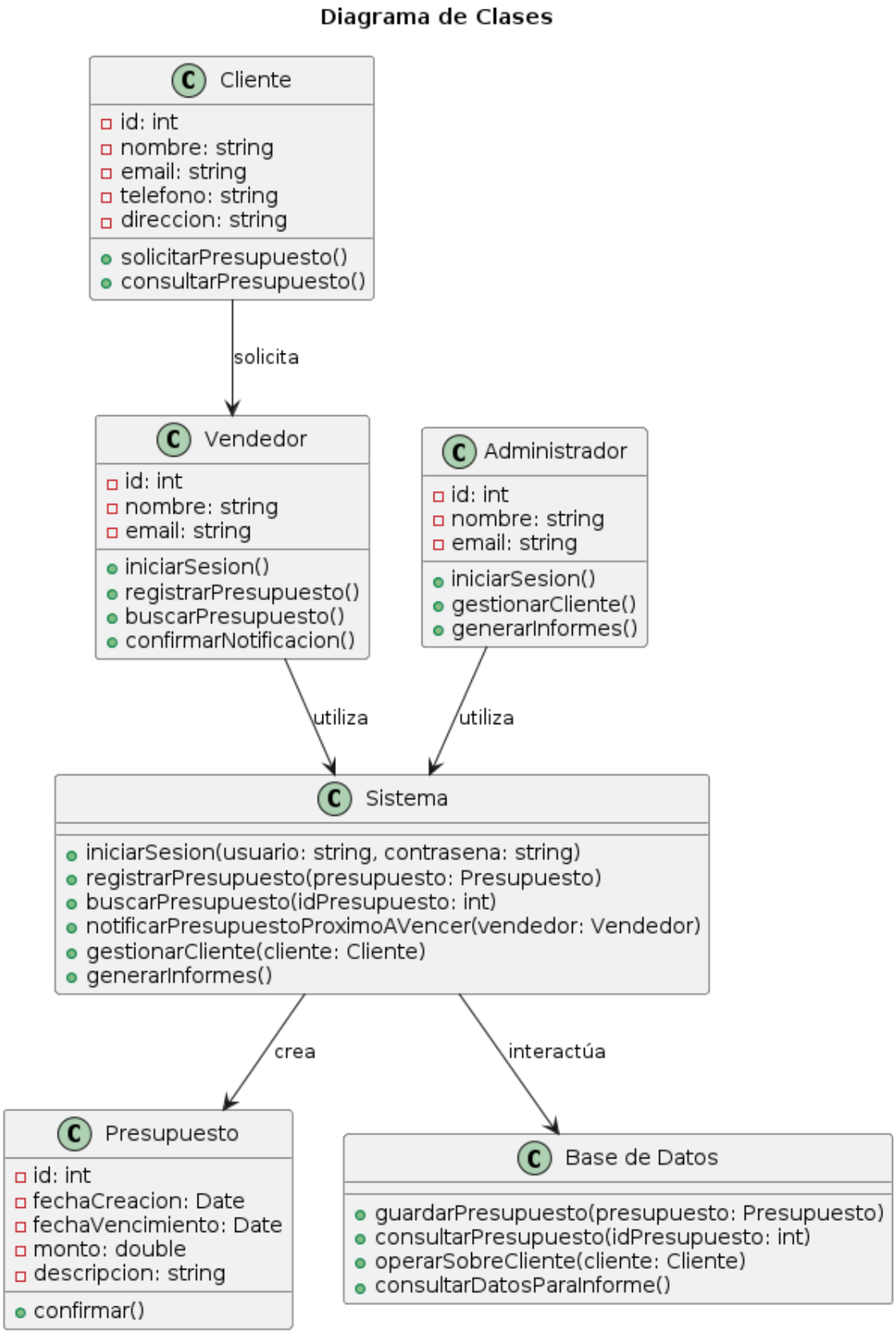
Postcondición	Se ha notificado al cliente sobre su presupuesto.	
Flujo alternativo	No contempla	
Excepciones		El cliente no puede recibir mensajes de WhatsApp.

Diagrama de secuencia:



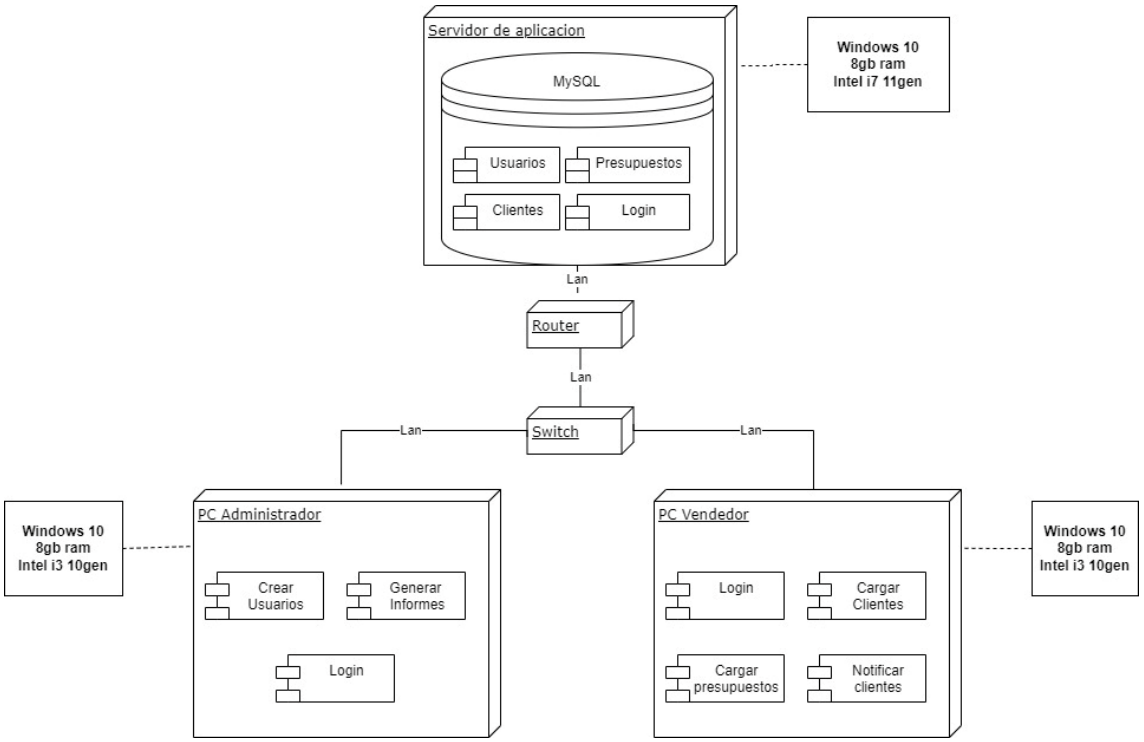
Fuente: Elaboración propia

Diagrama de clases:



Fuente: Elaboración propia.

Diagrama de despliegue:



Fuente: Elaboración propia.

Plan de pruebas:

Para esta instancia, se van a realizar las pruebas para los módulos de “Gestión de usuarios”, “Carga de presupuestos” y la “Generación de informes de estados de presupuestos” de la aplicación. El propósito de este plan es asegurar que cada módulo cumpla con los requisitos funcionales especificados y que el sistema opere de manera correcta y eficiente bajo las condiciones establecidas.

Caso de prueba 001:

Requisito funcional ID:	RFS001 Gestión de usuarios	**Fecha de prueba** 1/10/2024
Caso de prueba ID:	CP 001	
Descripción de caso de prueba:	Creación de usuario vendedor	
Prueba realizada por:	Benitez Micael Sebastian	
Precondiciones de la prueba:	El usuario debe poder abrir el módulo de gestión de usuarios en la aplicación	
Condiciones de la prueba:	El usuario debe poder ingresar los datos del nuevo vendedor y que estos se guarden en la base de datos.	
	Lista de pruebas de usuario	**Resultados esperados del sistema**
1	Inserción correcta de nombre de usuario	El usuario se ingresó correctamente
2	Inserción correcta de contraseña	La contraseña es valida
3	Confirmación de guardado en la	El mensaje de confirmación es

	base de datos	satisfactorio
Estado de la prueba	Finalizada, sin errores.	

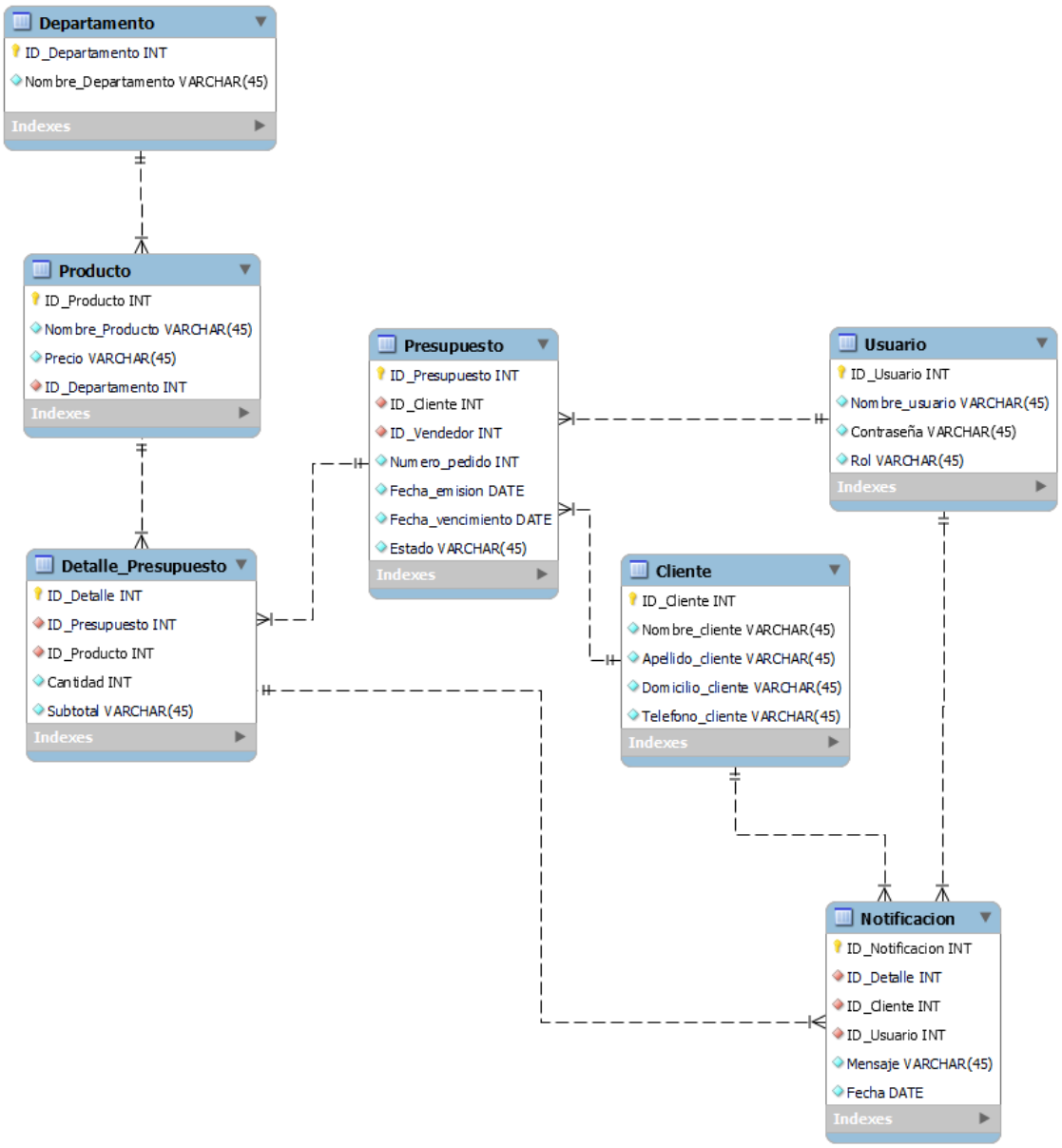
Caso de prueba 002:

Requisito funcional ID:	RFS04 Carga de presupuestos	**Fecha de prueba** 1/10/2024
Caso de prueba ID:	CP 002	
Descripción de caso de prueba:	Carga de presupuesto a la base de datos	
Prueba realizada por:	Benitez Micael Sebastian	
Precondiciones de la prueba:	El usuario debe poder abrir el módulo de presupuestos en la aplicación.	
Condiciones de la prueba:	El usuario debe poder ingresar los datos del presupuesto, del cliente y que estos se guarden en la base de datos.	
	Lista de pruebas de usuario	**Resultados esperados del sistema**
1	Inserción correcta de datos del presupuesto.	El usuario se ingresó correctamente
2	Inserción correcta de datos del cliente.	La contraseña es valida
3	Confirmación de guardado en la base de datos	El mensaje de confirmación es satisfactorio
Estado de la prueba	Finalizada, sin errores.	

Caso de prueba 003:

Requisito funcional ID:	RFS007 Generar informes de estados de presupuestos.	**Fecha de prueba** 1/10/2024
Caso de prueba ID:	CP 003	
Descripción de caso de prueba:	Generación de informes de estados de presupuestos.	
Prueba realizada por:	Benitez Micael Sebastian	
Precondiciones de la prueba:	El usuario debe poder abrir el módulo de estados de presupuestos.	
Condiciones de la prueba:	El usuario debe poder generar informes de estados de presupuestos y que este se muestre correctamente en la aplicación.	
	Lista de pruebas de usuario	**Resultados esperados del sistema**
1	Selección de parámetros para el informe.	Los parámetros son aceptados correctamente.
2	Generación del informe.	El informe se genera sin errores.
3	Visualización del informe.	El informe se muestra correctamente en la aplicación.
Estado de la prueba	Finalizada, sin errores.	

Diagrama entidad-relación:



Fuente: Elaboración propia.

Base de datos:

- **Creación de tablas:**

Tabla de Usuarios:

```
CREATE TABLE Usuario (
    ID_Usuario INT PRIMARY KEY,
    Nombre_Usuario VARCHAR(255) NOT NULL,
    Contraseña VARCHAR(255) NOT NULL,
    Rol ENUM('Administrador', 'Vendedor') NOT NULL
);
```

Tabla de Clientes:

```sql
CREATE TABLE Cliente (
    ID_Cliente INT AUTO_INCREMENT PRIMARY KEY,
    Nombre_Cliente VARCHAR(255) NOT NULL,
    Apellido_Cliente VARCHAR(255) NOT NULL,
    Domicilio VARCHAR(255),
    Telefono VARCHAR(20)
);
```

Tabla de Departamento:

```sql
CREATE TABLE Departamento (
    ID_Departamento INT AUTO_INCREMENT PRIMARY KEY,
    Nombre_Departamento VARCHAR(255) NOT NULL
);
```

Tabla de Producto:

```sql
CREATE TABLE Producto (
    ID_Producto INT AUTO_INCREMENT PRIMARY KEY,
    Nombre_Producto VARCHAR(255) NOT NULL,
    Precio DECIMAL(10, 2) NOT NULL,
    ID_Departamento INT,
    FOREIGN KEY (ID_Departamento) REFERENCES Departamento(ID_Departamento)
);
```

Tabla de Presupuesto:

```sql
CREATE TABLE Presupuesto (
    ID_Presupuesto INT AUTO_INCREMENT PRIMARY KEY,
    ID_Cliente INT,
    ID_Vendedor INT,
    Numero_Pedido INT,
    Fecha_Emision DATE NOT NULL,
    Fecha_Vencimiento DATE,
    Estado ENUM('Pendiente', 'Finalizado') NOT NULL,
    FOREIGN KEY (ID_Cliente) REFERENCES Cliente(ID_Cliente),
    FOREIGN KEY (ID_Vendedor) REFERENCES Usuario(ID_Usuario)
);
```

Tabla Detalle-Presupuesto:

```sql
CREATE TABLE Detalle_Presupuesto (
    ID_Detalle INT AUTO_INCREMENT PRIMARY KEY,
    ID_Presupuesto INT,
    ID_Producto INT,
    Cantidad INT NOT NULL,
    Subtotal DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (ID_Presupuesto) REFERENCES Presupuesto(ID_Presupuesto),
    FOREIGN KEY (ID_Producto) REFERENCES Producto(ID_Producto)
);
```

Tabla notificación:

```sql
CREATE TABLE Notificacion (
    ID_Notificacion INT AUTO_INCREMENT PRIMARY KEY,
    ID_Presupuesto INT,
    ID_Usuario INT,
    Mensaje TEXT NOT NULL,
    Fecha DATE NOT NULL,
    FOREIGN KEY (ID_Presupuesto) REFERENCES Presupuesto(ID_Presupuesto),
    FOREIGN KEY (ID_Usuario) REFERENCES Usuario(ID_Usuario)
);
```

- **Inserción de datos en las tablas:**

```sql
-- Insertar datos en la tabla Usuario
INSERT INTO Usuario (ID_Usuario, Nombre_Usuario, Contraseña, Rol) VALUES
(1, 'Matias', 'Gallardo123', 'Administrador'),
(2, 'Javier', 'Godoy123', 'Vendedor');

-- Insertar datos en la tabla Cliente
INSERT INTO Cliente (Nombre_Cliente, Apellido_Cliente, Domicilio, Telefono) VALUES
('Juan', 'Gallardo', 'Calle Florencio Sanchez 1223', '3755556477'),
('María', 'Gómez', 'Avenida Libertador 1516', '3755665544');

-- Insertar datos en la tabla Departamento
INSERT INTO Departamento (Nombre_Departamento) VALUES
('Ferreteria'),
('Pintureria');

-- Insertar datos en la tabla Producto
INSERT INTO Producto (Nombre_Producto, Precio, ID_Departamento) VALUES
('LLave de paso 1/2"', 1543.00, 1),
('Casablanca blanco 20Lts', 75850.99, 2);

-- Insertar datos en la tabla Presupuesto
INSERT INTO Presupuesto (ID_Cliente, ID_Vendedor, Numero_Pedido, Fecha_Emision, Fecha_Vencimiento, Estado) VALUES
(1, 2, 1001, '2023-06-01', '2023-06-10', 'Pendiente');

-- Insertar datos en la tabla Detalle_Presupuesto
INSERT INTO Detalle_Presupuesto (ID_Presupuesto, ID_Producto, Cantidad, Subtotal) VALUES
(1, 1, 1, 1543.00);

-- Insertar datos en la tabla Notificacion
INSERT INTO Notificacion (ID_Presupuesto, ID_Usuario, Mensaje, Fecha) VALUES
(1, 2, 'Su presupuesto está pendiente.', '2023-06-02');
```

- **Eliminación de registros:**

```sql
-- Borrar todos los registros de la tabla Notificacion
DELETE FROM Notificacion;

-- Borrar todos los registros de la tabla Detalle_Presupuesto
DELETE FROM Detalle_Presupuesto;

-- Borrar todos los registros de la tabla Presupuesto
DELETE FROM Presupuesto;

-- Borrar todos los registros de la tabla Producto
DELETE FROM Producto;

-- Borrar todos los registros de la tabla Departamento
DELETE FROM Departamento;

-- Borrar todos los registros de la tabla Cliente
DELETE FROM Cliente;

-- Borrar todos los registros de la tabla Usuario
DELETE FROM Usuario;
```

Desarrollo del sistema:

Para el desarrollo se van a implementar las clases de Usuario, Cliente, Presupuesto, Departamento, Producto, Notificacion, y se le agrega una clase “Menu” para la interacción del usuario con el sistema.

También se le conectara a la base de datos MySQL con las dos clases llamadas “DatabaseConnection” y “BaseDeDatos”.

Clase Usuario:

```java
public class Usuario {
    private int ID_Usuario;
    private String Nombre_Usuario;
    private String Contraseña;
    private String Rol;

    public Usuario(int ID_Usuario, String Nombre_Usuario, String Contraseña, String Rol) {
        this.ID_Usuario = ID_Usuario;
        this.Nombre_Usuario = Nombre_Usuario;
        this.Contraseña = Contraseña;
        this.Rol = Rol;
    }
```

En esta clase, se dan a conocer los atributos y constructores que va a contener el “Usuario” su ID, Nombre_Usuario, Contraseña y el Rol.

Para esta clase vamos a tener los siguientes métodos:

```
16      public int getID_Usuario() {
17          return ID_Usuario;
18      }
19
20      public void setID_Usuario(int ID_Usuario) {
21          this.ID_Usuario = ID_Usuario;
22      }
23
24      public String getNombre_Usuario() {
25          return Nombre_Usuario;
26      }
27
28      public void setNombre_Usuario(String Nombre_Usuario) {
29          this.Nombre_Usuario = Nombre_Usuario;
30      }
31
32      public String getContraseña() {
33          return Contraseña;
34      }
35
36      public void setContraseña(String Contraseña) {
37          this.Contraseña = Contraseña;
38      }
39
40      public String getRol() {
41          return Rol;
42      }
43
44      public void setRol(String Rol) {
45          this.Rol = Rol;
46      }
47  }
```

Clase Presupuestos:

En esta clase, se definen los atributos y el constructor para manejar la información de un presupuesto. Los atributos incluyen el “ID_Presupuesto”, “ID_Cliente”, “ID_Vendedor”, “Numero_Pedido”, “Fecha_Emision”, “Fecha_Vencimiento” y el “Estado” del presupuesto.

El constructor de la clase “Presupuesto” permite inicializar estos atributos cuando se crea una nueva instancia de presupuesto, estableciendo la información básica necesaria para gestionar cada presupuesto en el sistema.

```java
public class Presupuesto {
    private int ID_Presupuesto;
    private int ID_Cliente;
    private int ID_Vendedor;
    private int Numero_Pedido;
    private LocalDate Fecha_Emision;
    private LocalDate Fecha_Vencimiento;
    private String Estado;

    public Presupuesto(int ID_Presupuesto, int ID_Cliente, int ID_Vendedor, int Numero_Pedido,
                       LocalDate Fecha_Emision, LocalDate Fecha_Vencimiento, String Estado){
        this.ID_Presupuesto = ID_Presupuesto;
        this.ID_Cliente = ID_Cliente;
        this.ID_Vendedor = ID_Vendedor;
        this.Numero_Pedido = Numero_Pedido;
        this.Fecha_Emision = Fecha_Emision;
        this.Fecha_Vencimiento = Fecha_Vencimiento;
        this.Estado = Estado;
    }
```

Los métodos de la clase “Presupuesto” son:

```java
    public int getID_Presupuesto() {
        return ID_Presupuesto;
    }
```

```java
    public void setID_Presupuesto(int ID_Presupuesto) {
        this.ID_Presupuesto = ID_Presupuesto;
    }
```

```java
    public int getID_Cliente() {
        return ID_Cliente;
    }
```

```java
    public void setID_Cliente(int ID_Cliente) {
        this.ID_Cliente = ID_Cliente;
    }
```

```java
    public int getID_Vendedor() {
        return ID_Vendedor;
    }
```

```java
    public void setID_Vendedor(int ID_Vendedor) {
        this.ID_Vendedor = ID_Vendedor;
    }
```

```java
    public int getNumero_Pedido() {
        return Numero_Pedido;
    }
```

```
70      public LocalDate getFecha_Emision() {
71          return Fecha_Emision;
72      }
```

```
74      public void setFecha_Emision(LocalDate Fecha_Emision) {
75          this.Fecha_Emision = Fecha_Emision;
76      }
```

```
78      public LocalDate getFecha_Vencimiento() {
79          return Fecha_Vencimiento;
80      }
```

```
82      public void setFecha_Vencimiento(LocalDate Fecha_Vencimiento) {
83          this.Fecha_Vencimiento = Fecha_Vencimiento;
84      }
```

```
86      public String getEstado() {
87          return Estado;
88      }
```

```
90      public void setEstado(String Estado) {
91          this.Estado = Estado;
92      }
```

Método generarPresupuesto():

```
102     public static void generarPresupuesto(int ID_Cliente, int ID_Vendedor, int Numero_Pedido) {
103         LocalDate fechaEmision = LocalDate.now();
104         LocalDate fechaVencimiento = fechaEmision.plusDays(5);
105         String estado = "Pendiente";
106
107         String sql = "INSERT INTO presupuesto (ID_Cliente, ID_Vendedor, Numero_Pedido, Fecha_Emision, Fecha_Vencimiento, Estado) VALUES (?, ?, ?, ?, ?, ?)";
108
109         try (Connection conn = DatabaseConnection.getConnection();
110              PreparedStatement pstmt = conn.prepareStatement(sql)) {
111             pstmt.setInt(1, ID_Cliente);
112             pstmt.setInt(2, ID_Vendedor);
113             pstmt.setInt(3, Numero_Pedido);
114             pstmt.setDate(4, java.sql.Date.valueOf(fechaEmision));
115             pstmt.setDate(5, java.sql.Date.valueOf(fechaVencimiento));
116             pstmt.setString(6, estado);
117             pstmt.executeUpdate();
118         } catch (SQLException e) {
119             System.out.println(e.getMessage());
120         }
121     }
```

El método “generarPresupuesto” se encarga de crear e insertar un nuevo presupuesto en la base de datos. A continuación, se detalla su funcionamiento:

1. Parámetros de entrada:

- “ID_Cliente”: Identificador del cliente.

- “ID_Vendedor”: Identificador del vendedor.

- “Numero_Pedido”: Número del pedido relacionado con el presupuesto.

2. Lógica interna:

- Fecha de emisión: Se obtiene la fecha actual usando “LocalDate.now()”.

- Fecha de vencimiento: Se calcula sumando 5 días a la fecha de emisión (“plusDays(5)”).

- Estado inicial: Se asigna la cadena `"Pendiente"` para indicar que el presupuesto aún no ha sido procesado.

3. Consulta SQL:

- Se prepara una consulta de inserción (“INSERT”) que añadirá el presupuesto en la tabla “presupuesto”. Los valores se pasan mediante parámetros para evitar inyecciones SQL.

4. Conexión y ejecución:

- Se utiliza “DatabaseConnection.getConnection()” para obtener la conexión a la base de datos.

- Con un “PreparedStatement”, se asignan los valores recibidos y calculados a los respectivos campos de la consulta.

- Finalmente, se ejecuta la actualización con “pstmt.executeUpdate()”.

5. Manejo de excepciones:

- En caso de error al conectar o ejecutar la consulta, se captura una excepción “SQLException” y se imprime el mensaje del error en consola.

Este método es clave para automatizar la creación de presupuestos, garantizando que los datos se inserten correctamente en la base de datos y permitiendo un control del flujo de trabajo mediante el estado `"Pendiente"`.

Método listarPresupuestos():

```
123     public static List<Presupuesto> listarPresupuestos() {
124         List<Presupuesto> presupuestos = new ArrayList<>();
125         String sql = "SELECT * FROM presupuesto";
126 
127         try (Connection conn = DatabaseConnection.getConnection();
128              PreparedStatement pstmt = conn.prepareStatement(sql)) {
129             ResultSet rs = pstmt.executeQuery();
130             while (rs.next()) {
131                 Presupuesto presupuesto = new Presupuesto(
132                         rs.getInt("ID_Presupuesto"),
133                         rs.getInt("ID_Cliente"),
134                         rs.getInt("ID_Vendedor"),
135                         rs.getInt("Numero_Pedido"),
136                         rs.getDate("Fecha_Emision").toLocalDate(),
137                         rs.getDate("Fecha_Vencimiento").toLocalDate(),
138                         rs.getString("Estado")
139                 );
140                 presupuestos.add(presupuesto);
141             }
142         } catch (SQLException e) {
143             System.out.println(e.getMessage());
144         }
145         return presupuestos;
```

Este método recupera todos los presupuestos almacenados en la base de datos y los devuelve como una lista de objetos `Presupuesto`. A continuación, su funcionamiento paso a paso:

1. Preparación:

- Se crea una lista vacía llamada “presupuestos” para almacenar los resultados.

2. Consulta SQL:

- La consulta "SELECT FROM presupuesto" obtiene todos los registros de la tabla “presupuesto”.

3. Ejecución:

- Se establece la conexión con la base de datos y se ejecuta la consulta usando “PreparedStatement”.

- Los resultados se almacenan en un “ResultSet”.

4. Procesamiento de resultados:

- Por cada fila del “ResultSet”, se crea un objeto “Presupuesto” con los datos extraídos (ID, fechas, estado, etc.).

- Las fechas se convierten a “LocalDate” usando “toLocalDate()”.

- Cada objeto “Presupuesto” se añade a la lista.

5. Manejo de errores:

- Si ocurre un error durante la conexión o ejecución, se captura una “SQLException” y se imprime el mensaje en la consola.

6. Retorno:

- Se devuelve la lista con todos los presupuestos encontrados.

Este método permite que la aplicación acceda rápidamente a todos los presupuestos almacenados para ser mostrados o procesados.

Método modificarPresupuesto():

```
148     public static void modificarPresupuesto(int ID_Presupuesto, String nuevoEstado) {
149         String sql = "UPDATE presupuesto SET Estado = ? WHERE ID_Presupuesto = ?";
150 
151         try (Connection conn = DatabaseConnection.getConnection();
152              PreparedStatement pstmt = conn.prepareStatement(sql)) {
153             pstmt.setString(1, nuevoEstado);
154             pstmt.setInt(2, ID_Presupuesto);
155             pstmt.executeUpdate();
156         } catch (SQLException e) {
157             System.out.println(e.getMessage());
158         }
159     }
```

Este método actualiza el estado de un presupuesto en la base de datos según su ID. A continuación, su funcionamiento:

1. Consulta SQL:

- La sentencia UPDATE presupuesto SET Estado = ? WHERE ID_Presupuesto = ? se usa para modificar el estado del presupuesto correspondiente al ID dado.

2. Parámetros de entrada:

- ID_Presupuesto: Identificador del presupuesto que se va a modificar.

- nuevoEstado: El nuevo valor del estado (por ejemplo, "Aprobado", "Rechazado", "Pendiente").

3. Ejecución:

- Se establece la conexión con la base de datos mediante “DatabaseConnection.getConnection()”.

- Con un “PreparedStatement”, se asignan los valores a la consulta:

  - “pstmt.setString(1, nuevoEstado)”: Coloca el nuevo estado.

  - ”pstmt.setInt(2, ID_Presupuesto)”: Indica el presupuesto a modificar.

- Se ejecuta la actualización con “pstmt.executeUpdate()”.

Método eliminarPresupuesto():

```
161     public static void eliminarPresupuesto(int ID_Presupuesto) {
162         String sql = "DELETE FROM presupuesto WHERE ID_Presupuesto = ?";
163 
164         try (Connection conn = DatabaseConnection.getConnection();
165              PreparedStatement pstmt = conn.prepareStatement(sql)) {
166             pstmt.setInt(1, ID_Presupuesto);
167             pstmt.executeUpdate();
168         } catch (SQLException e) {
169             System.out.println(e.getMessage());
170         }
171     }
```

Este método elimina un presupuesto de la base de datos usando su ID como referencia. A continuación, se detalla su funcionamiento:

1. Consulta SQL:

- La sentencia “DELETE FROM presupuesto WHERE ID_Presupuesto = ?” elimina el registro del presupuesto cuyo ID coincide con el proporcionado.

2. Parámetro de entrada:

- “ID_Presupuesto”: Identificador del presupuesto que se desea eliminar.

3. Ejecución:

- Se establece la conexión con la base de datos mediante DatabaseConnection.getConnection().

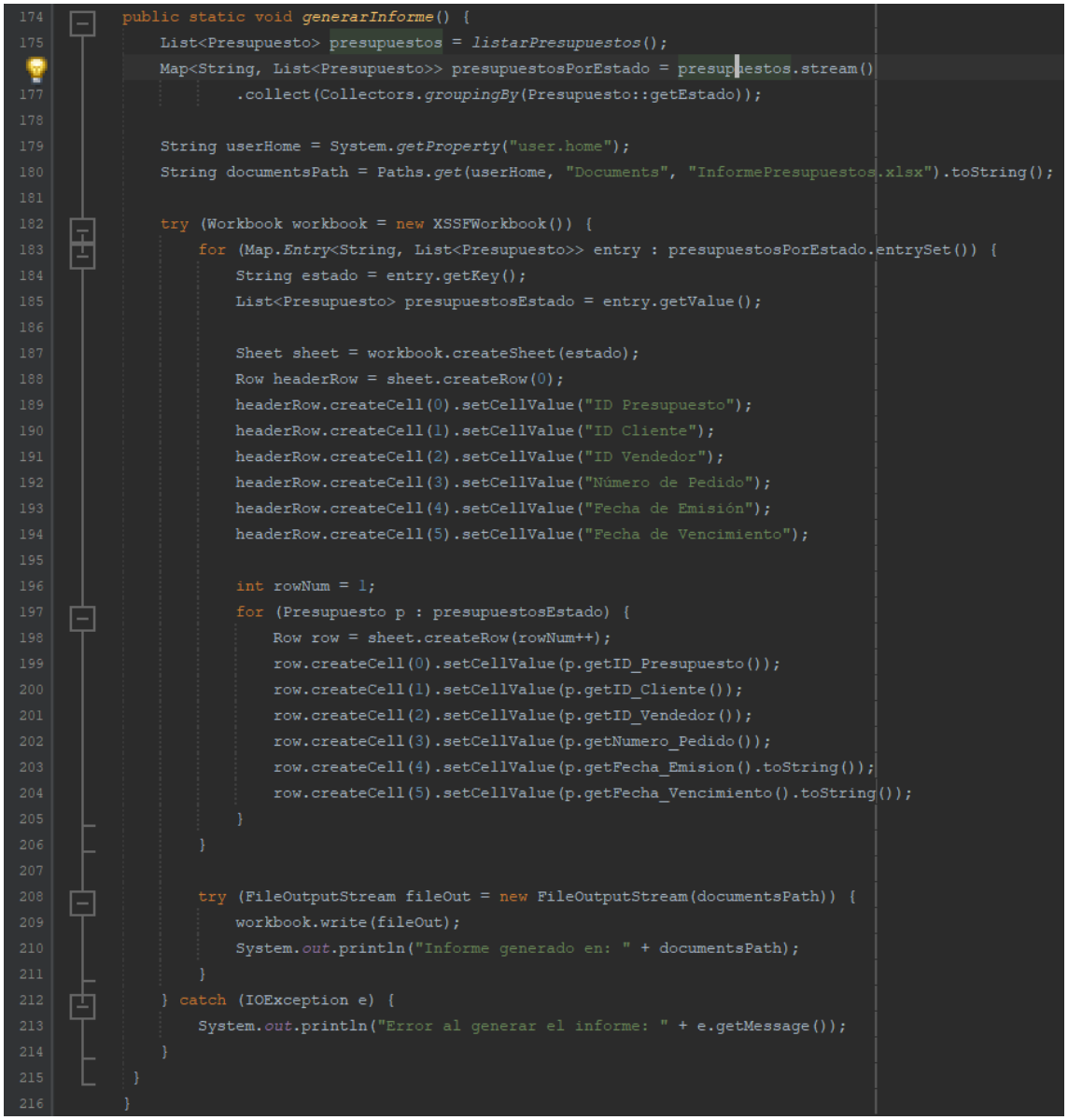
- Con un “PreparedStatement”, se asigna el ID a la consulta:

  - “pstmt.setInt(1, ID_Presupuesto)”: Define qué presupuesto eliminar.

- Se ejecuta la operación con “pstmt.executeUpdate()”, eliminando el registro correspondiente.

Este método es fundamental para permitir la eliminación de presupuestos de forma controlada, asegurando que solo se borren los registros específicos mediante su ID.

Método generarInforme():



```
public static void generarInforme() {
    List<Presupuesto> presupuestos = listarPresupuestos();
    Map<String, List<Presupuesto>> presupuestosPorEstado = presupuestos.stream()
            .collect(Collectors.groupingBy(Presupuesto::getEstado));

    String userHome = System.getProperty("user.home");
    String documentsPath = Paths.get(userHome, "Documents", "InformePresupuestos.xlsx").toString();

    try (Workbook workbook = new XSSFWorkbook()) {
        for (Map.Entry<String, List<Presupuesto>> entry : presupuestosPorEstado.entrySet()) {
            String estado = entry.getKey();
            List<Presupuesto> presupuestosEstado = entry.getValue();

            Sheet sheet = workbook.createSheet(estado);
            Row headerRow = sheet.createRow(0);
            headerRow.createCell(0).setCellValue("ID Presupuesto");
            headerRow.createCell(1).setCellValue("ID Cliente");
            headerRow.createCell(2).setCellValue("ID Vendedor");
            headerRow.createCell(3).setCellValue("Número de Pedido");
            headerRow.createCell(4).setCellValue("Fecha de Emisión");
            headerRow.createCell(5).setCellValue("Fecha de Vencimiento");

            int rowNum = 1;
            for (Presupuesto p : presupuestosEstado) {
                Row row = sheet.createRow(rowNum++);
                row.createCell(0).setCellValue(p.getID_Presupuesto());
                row.createCell(1).setCellValue(p.getID_Cliente());
                row.createCell(2).setCellValue(p.getID_Vendedor());
                row.createCell(3).setCellValue(p.getNumero_Pedido());
                row.createCell(4).setCellValue(p.getFecha_Emision().toString());
                row.createCell(5).setCellValue(p.getFecha_Vencimiento().toString());
            }
        }

        try (FileOutputStream fileOut = new FileOutputStream(documentsPath)) {
            workbook.write(fileOut);
            System.out.println("Informe generado en: " + documentsPath);
        }
    } catch (IOException e) {
        System.out.println("Error al generar el informe: " + e.getMessage());
    }
}
```

Este método genera un informe en formato Excel con los presupuestos almacenados en la base de datos, agrupándolos por su estado (por ejemplo, "Pendiente", "Aprobado", "Rechazado"). A continuación, su funcionamiento:

1. Obtención de presupuestos:

- Se llama al método “listarPresupuestos()” para obtener todos los presupuestos de la base de datos.

- Los presupuestos se agrupan por estado utilizando un “Map” con la función “Collectors.groupingBy()”.

2. Ruta de almacenamiento del informe:

- Se obtiene el directorio de Documentos del usuario mediante System.getProperty("user.home") y se define la ruta para guardar el archivo como “InformePresupuestos.xlsx”.

3. Creación del archivo Excel:

- Se crea un “Workbook” usando la clase “XSSFWorkbook” de Apache POI.

- Por cada estado encontrado, se genera una hoja (sheet) en el Excel con el nombre del estado (por ejemplo, "Pendiente", "Aprobado").

- En cada hoja, se agrega una fila de encabezado con los nombres de las columnas: "ID Presupuesto", "ID Cliente", "ID Vendedor", "Número de Pedido", "Fecha de Emisión" y "Fecha de Vencimiento".

4. Población de datos:

- Para cada presupuesto en la lista, se crea una nueva fila en la hoja correspondiente y se completan sus datos en las celdas: ID del presupuesto, ID del cliente, ID del vendedor, número de pedido y fechas.

5. Escritura y guardado del archivo:

- Se usa un “FileOutputStream” para escribir el contenido del workbook en el archivo Excel en la ruta definida.

6. Mensaje de confirmación:

- Si el proceso se completa correctamente, se imprime un mensaje en la consola indicando la ubicación del archivo generado.

Clase “Cliente”:

La clase Cliente representa a los clientes del sistema, almacenando su información personal y de contacto. Esta clase permite crear objetos que contienen los datos específicos de cada cliente.

```
public class Cliente {
    private int ID_Cliente;
    private String Nombre_Cliente;
    private String Apellido_Cliente;
    private String Domicilio;
    private String Telefono;

    public Cliente(int ID_Cliente, String Nombre_Cliente, String Apellido_Cliente, String Domicilio, String Telefono) {
        this.ID_Cliente = ID_Cliente;
        this.Nombre_Cliente = Nombre_Cliente;
        this.Apellido_Cliente = Apellido_Cliente;
        this.Domicilio = Domicilio;
        this.Telefono = Telefono;
    }
```

Parámetros de la clase “Cliente”:

```java
    public int getID_Cliente() {
        return ID_Cliente;
    }

    public void setID_Cliente(int ID_Cliente) {
        this.ID_Cliente = ID_Cliente;
    }

    public String getNombre_Cliente() {
        return Nombre_Cliente;
    }

    public void setNombre_Cliente(String Nombre_Cliente) {
        this.Nombre_Cliente = Nombre_Cliente;
    }

    public String getApellido_Cliente() {
        return Apellido_Cliente;
    }

    public void setApellido_Cliente(String Apellido_Cliente) {
        this.Apellido_Cliente = Apellido_Cliente;
    }

    public String getDomicilio() {
        return Domicilio;
    }

    public void setDomicilio(String Domicilio) {
        this.Domicilio = Domicilio;
    }

    public String getTelefono() {
        return Telefono;
    }

    public void setTelefono(String Telefono) {
        this.Telefono = Telefono;
    }
```

Método registrarCliente():

```java
public void registrarCliente() {
    String sql = "INSERT INTO cliente (ID_Cliente, Nombre_Cliente, Apellido_Cliente, Domicilio, Telefono) VALUES (?, ?, ?, ?, ?)";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, ID_Cliente);
        pstmt.setString(2, Nombre_Cliente);
        pstmt.setString(3, Apellido_Cliente);
        pstmt.setString(4, Domicilio);
        pstmt.setString(5, Telefono);
        pstmt.executeUpdate();
        System.out.println("Cliente registrado exitosamente.");
    } catch (SQLException e) {
        System.out.println("Error al registrar cliente: " + e.getMessage());
    }
}
```

Este método registra un nuevo cliente en la base de datos. A continuación, se detalla su funcionamiento:

1. Consulta SQL:
    - La sentencia “INSERT INTO cliente (ID_Cliente, Nombre_Cliente, Apellido_Cliente, Domicilio, Telefono) VALUES (?, ?, ?, ?, ?)” inserta un nuevo registro en la tabla cliente con los datos proporcionados de ID, nombre, apellido, domicilio y teléfono.
2. Parámetros de entrada:
    - ID_Cliente: Identificador único del cliente.
    - Nombre_Cliente: Nombre del cliente.
    - Apellido_Cliente: Apellido del cliente.
    - Domicilio: Dirección del cliente.
    - Telefono: Número de teléfono del cliente.
3. Ejecución:
    - Se establece una conexión con la base de datos utilizando DatabaseConnection.getConnection().
    - Con un PreparedStatement, se asignan los valores a la consulta:
        - pstmt.setInt(1, ID_Cliente): Define el ID del cliente.
        - pstmt.setString(2, Nombre_Cliente): Define el nombre del cliente.
        - pstmt.setString(3, Apellido_Cliente): Define el apellido del cliente.
        - pstmt.setString(4, Domicilio): Define el domicilio del cliente.
        - pstmt.setString(5, Telefono): Define el teléfono del cliente.
    - Se ejecuta la operación con pstmt.executeUpdate(), añadiendo el registro correspondiente a la tabla cliente.
    - Muestra un mensaje de éxito si la operación es exitosa.

Este método es esencial para el registro controlado de nuevos clientes, asegurando que los datos se inserten correctamente en la base de datos.

Método modificarCliente():

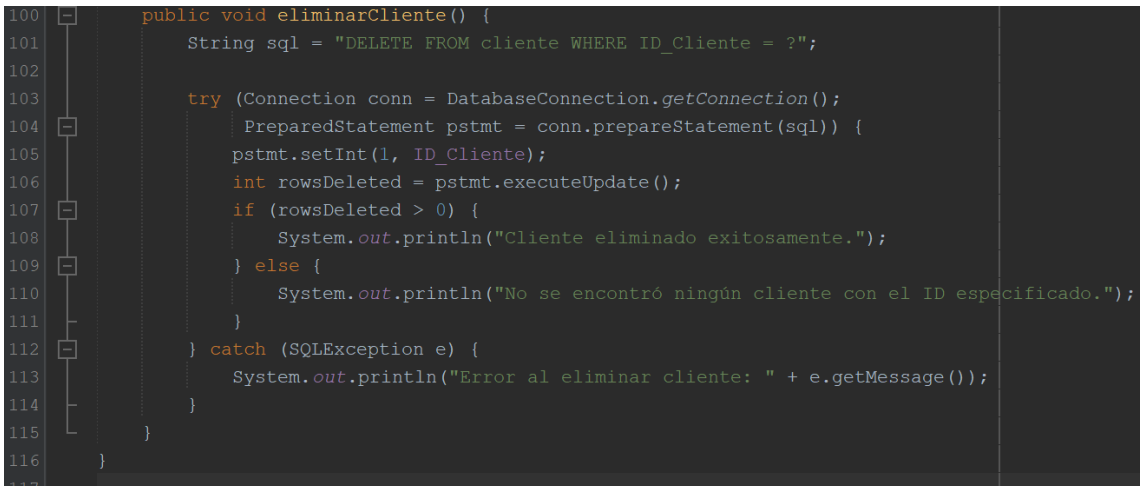
```java
public void modificarCliente() {
    String sql = "UPDATE cliente SET Nombre_Cliente = ?, Apellido_Cliente = ?, Domicilio = ?, Telefono = ? WHERE ID_Cliente = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, Nombre_Cliente);
        pstmt.setString(2, Apellido_Cliente);
        pstmt.setString(3, Domicilio);
        pstmt.setString(4, Telefono);
        pstmt.setInt(5, ID_Cliente);
        int rowsUpdated = pstmt.executeUpdate();
        if (rowsUpdated > 0) {
            System.out.println("Cliente modificado exitosamente.");
        } else {
            System.out.println("No se encontró ningún cliente con el ID especificado.");
        }
    } catch (SQLException e) {
        System.out.println("Error al modificar cliente: " + e.getMessage());
    }
}
```

Este método modifica los datos de un cliente existente en la base de datos, utilizando su ID como referencia. A continuación, se detalla su funcionamiento:

1. Consulta SQL:
    - La sentencia “UPDATE cliente SET Nombre_Cliente = ?, Apellido_Cliente = ?, Domicilio = ?, Telefono = ? WHERE ID_Cliente = ?” actualiza los campos Nombre_Cliente, Apellido_Cliente, Domicilio y Telefono del cliente cuyo ID_Cliente coincide con el proporcionado.
2. Parámetros de entrada:
    - Nombre_Cliente: Nuevo nombre del cliente.
    - Apellido_Cliente: Nuevo apellido del cliente.
    - Domicilio: Nuevo domicilio del cliente.
    - Telefono: Nuevo número de teléfono del cliente.
    - ID_Cliente: Identificador del cliente que se desea modificar.
3. Ejecución:
    - Se establece una conexión con la base de datos mediante DatabaseConnection.getConnection().
    - Con un PreparedStatement, se asignan los valores a la consulta:
        - pstmt.setString(1, Nombre_Cliente): Define el nuevo nombre.
        - pstmt.setString(2, Apellido_Cliente): Define el nuevo apellido.
        - pstmt.setString(3, Domicilio): Define el nuevo domicilio.
        - pstmt.setString(4, Telefono): Define el nuevo teléfono.
        - pstmt.setInt(5, ID_Cliente): Define el ID del cliente a modificar.

- Se ejecuta la operación con pstmt.executeUpdate() y se verifica si hubo cambios.
    - Si rowsUpdated es mayor a 0, indica que el cliente fue modificado exitosamente.
    - Si no, muestra un mensaje de que no se encontró ningún cliente con el ID especificado.

Método eliminarCliente():



```
100     public void eliminarCliente() {
101         String sql = "DELETE FROM cliente WHERE ID_Cliente = ?";
102
103         try (Connection conn = DatabaseConnection.getConnection();
104              PreparedStatement pstmt = conn.prepareStatement(sql)) {
105             pstmt.setInt(1, ID_Cliente);
106             int rowsDeleted = pstmt.executeUpdate();
107             if (rowsDeleted > 0) {
108                 System.out.println("Cliente eliminado exitosamente.");
109             } else {
110                 System.out.println("No se encontró ningún cliente con el ID especificado.");
111             }
112         } catch (SQLException e) {
113             System.out.println("Error al eliminar cliente: " + e.getMessage());
114         }
115     }
116 }
```

Este método elimina un cliente de la base de datos usando su ID como referencia. A continuación, se detalla su funcionamiento:

1. Consulta SQL:

- La sentencia “DELETE FROM cliente WHERE ID_Cliente = ?” elimina el registro del cliente cuyo “ID_Cliente” coincide con el proporcionado.

2. Parámetro de entrada:

- “ID_Cliente”: Identificador del cliente que se desea eliminar.

3. Ejecución:

- Se establece la conexión con la base de datos mediante “DatabaseConnection.getConnection()”.

- Con un “PreparedStatement”, se asigna el ID a la consulta:

  - “pstmt.setInt(1, ID_Cliente)”: Define qué cliente eliminar.

- Se ejecuta la operación con “pstmt.executeUpdate()” y se verifica si hubo eliminación.

  - Si “rowsDeleted” es mayor a 0, indica que el cliente fue eliminado exitosamente.

  - Si no, muestra un mensaje de que no se encontró ningún cliente con el ID especificado.

Clase Menú:

```java
public class Menu {
    public static void main(String[] args) {
        BaseDeDatos bd = new BaseDeDatos();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nSISTEMA DE GESTION DE PRESUPUESTOS");
            System.out.println("\nMenú Principal:");
            System.out.println("1. Registrar Usuario");
            System.out.println("2. Cliente");
            System.out.println("3. Gestionar Presupuestos");
            System.out.println("4. Salir");

            System.out.print("Seleccione una opción: ");
            String opcion = scanner.nextLine();

            switch (opcion) {
                case "1":
                    registrarUsuario(bd, scanner);
                    break;
                case "2":
                    menuCliente(bd, scanner);
                    break;
                case "3":
                    gestionarPresupuestos(scanner, bd);
                    break;
                case "4":
                    System.out.println("Saliendo del sistema.");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Opción no válida. Intente nuevamente.");
            }
        }
```

Este código implementa un menú principal para un sistema de gestión de presupuestos, permitiendo la interacción con el usuario mediante varias opciones. A continuación, se detalla su funcionamiento:

1. Inicialización de Objetos:

- Se crea una instancia de BaseDeDatos, que gestiona la conexión con la base de datos.
- Se utiliza un Scanner para capturar la entrada del usuario desde la consola.

2. Bucle Principal:

- Un bucle while (true) mantiene activo el menú hasta que el usuario elija salir.
- En cada iteración, se muestra el menú principal con las opciones:
  - 1. Registrar Usuario
  - 2. Cliente
  - 3. Gestionar Presupuestos
  - 4. Salir

3. Selección de Opciones:

- Se pide al usuario que ingrese una opción y se procesa con una estructura switch:

- Opción "1": Llama al método registrarUsuario, pasando bd y scanner como argumentos.

- Opción "2": Llama al método menuCliente, pasando bd y `scanner` para gestionar la información de los clientes.

- Opción "3": Llama al método gestionarPresupuestos, permitiendo la gestión de presupuestos.

- Opción "4": Muestra un mensaje de salida, cierra el scanner y termina la ejecución con “System.exit(0)”.

- Opción no válida: Muestra un mensaje de error si la opción ingresada no es reconocida.

Este menú es el punto de entrada del sistema y facilita la navegación entre las diferentes funciones del programa, ofreciendo una interfaz clara y sencilla para la interacción con el usuario.