

**UNIVERSIDAD DE SONORA**

**Unidad Regional Centro**

**División de Ingeniería**

# **Ingeniería en Sistemas de Información**

**Base de Datos 1**

**Maestro: Navarro Hernández Rene Francisco**

**Grupo 1 – Hora 4 PM – 5 PM- Edif. 5G – CO201**

**Integrantes de equipo:**

* **María Yamile Valencia Loroña**
* **Borbón Núñez Francisco Javier**

**Proyecto Final del Curso – BikeRent**

**1. Documento de Diseño del Sistema de Alquiler de Bicicletas**

**Resumen de Requisitos**

La aplicación de escritorio "BikeRentalSystem" es un sistema de gestión de alquiler de bicicletas que permite a múltiples tiendas administrar su inventario, clientes y operaciones de alquiler. El objetivo principal es digitalizar y centralizar el control de bicicletas disponibles, clientes que alquilan, historial de alquileres y proveedores de bicicletas. Las funcionalidades clave de la aplicación incluyen:

* Registro, actualización y eliminación de bicicletas.
* Gestión de clientes (renters) y proveedores.
* Registro de alquileres de bicicletas.
* Control de disponibilidad de bicicletas.
* Generación de reportes e indicadores clave (como ingresos por tienda, uso de bicicletas, historial por cliente).

**Modelo Conceptual (ERD)**

Se incluye un **Diagrama Entidad-Relación (ERD)** que representa gráficamente las entidades principales y sus relaciones. Las entidades son:

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Esquema Relacional**

A continuación, un resumen del esquema con claves primarias (PK) y foráneas (FK):

* **Multi\_Shop**(shop\_id PK, name, city\_id FK)
* **Bicycles**(bike\_id PK, type\_id FK, provider\_id FK, shop\_id FK, status, price\_per\_hour)
* **Renters**(renter\_id PK, name, document\_type\_id FK, document\_number, gender\_id FK)
* **Rentals**(rental\_id PK, bike\_id FK, renter\_id FK, rental\_date, return\_date, total\_amount, status\_id FK, payment\_method\_id FK)
* **Providers**(provider\_id PK, name, contact\_info)
* **Bicycle\_Types**(type\_id PK, name)
* **Rental\_Status**(status\_id PK, name)
* **Payment\_Methods**(payment\_method\_id PK, name)
* **Document\_Types**(document\_type\_id PK, name)
* **Genders**(gender\_id PK, name)
* **Cities**(city\_id PK, name)

**Diccionario de Datos:**

**2. Implementación de bases de datos (30%)**

**2.1 Configuración del entorno y creación de la base de datos**

Para la implementación del sistema de alquiler de bicicletas, se utilizó PostgreSQL como sistema de gestión de bases de datos relacional. Se configuró el entorno en un servidor local, asegurando que el rol developer tuviera los privilegios necesarios para la creación y manipulación de la base de datos y sus objetos asociados.

La base de datos fue creada con el siguiente comando:

CREATE DATABASE bicycle\_rental

WITH OWNER = developer

TEMPLATE = template1

ENCODING = 'UTF8'

LOCALE\_PROVIDER = 'libc'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1;

IS\_TEMPLATE = False;

Se otorgaron permisos adecuados al rol developer sobre las secuencias y tablas necesarias para el funcionamiento del sistema.  
  
Permisos otorgados:

-- Permitir el uso completo de la base de datos

GRANT ALL ON DATABASE bicycle\_rental TO developer;

-- Permitir uso de esquemas y conexión

GRANT USAGE ON SCHEMA public TO developer;

-- Otorgar privilegios sobre todas las tablas

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO developer;

-- Otorgar privilegios sobre todas las secuencias (necesario para columnas SERIAL)

GRANT USAGE, SELECT, UPDATE ON ALL SEQUENCES IN SCHEMA public TO developer;

-- Para que futuros objetos también estén cubiertos automáticamente

ALTER DEFAULT PRIVILEGES IN SCHEMA public

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO developer;

ALTER DEFAULT PRIVILEGES IN SCHEMA public

GRANT USAGE, SELECT, UPDATE ON SEQUENCES TO developer;

**2.2 Implementación del esquema relacional**

Se implementaron todas las tablas definidas en el esquema relacional, incluyendo tablas de referencia, entidades principales y tablas de relación.

**Ejemplos de implementación de restricciones de integridad referencial:**

* La tabla Renters\_Payment\_Methods implementa claves foráneas hacia Renters y Ref\_Payment\_Methods:

Renter\_ID INTEGER NOT NULL REFERENCES Renters(Renter\_ID),

Payment\_Method\_Code VARCHAR(10) NOT NULL REFERENCES Ref\_Payment\_Methods(Payment\_Method\_Code)

* La tabla Rentals mantiene integridad referencial con múltiples entidades:

Bicycle\_ID INTEGER NOT NULL REFERENCES Bicycles(Bicycle\_ID),

Payment\_status\_Code VARCHAR(10) NOT NULL REFERENCES Ref\_Payment\_status\_code(Payment\_status\_Code),

Rental\_Rates\_ID INTEGER NOT NULL REFERENCES Rental\_Rates(Rental\_Rates\_ID),

Renter\_ID INTEGER NOT NULL REFERENCES Renters(Renter\_ID),

Renter\_Payment\_Method\_ID INTEGER NOT NULL REFERENCES Renters\_Payment\_Methods(Renter\_Payment\_Method\_ID)

**2.3 Restricciones generales de dominio**

Además de las claves foráneas, se implementaron múltiples restricciones de dominio para asegurar la validez de los datos. Algunos ejemplos son:

* Validación de formato de correo electrónico en Multi\_Shop:

Email\_Address VARCHAR(100) CHECK (Email\_Address ~\* '^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')

* Restricciones de valores positivos en tarifas:

Daily\_Rate DECIMAL(10,2) NOT NULL CHECK (Daily\_Rate > 0),

Hourly\_Rate DECIMAL(10,2) NOT NULL CHECK (Hourly\_Rate > 0)

* Validación de fechas en Rentals:

CHECK (Booked\_End\_Date\_Time > Booked\_Start\_Date\_Time),

CHECK (Actual\_End\_Date\_Time IS NULL OR Actual\_End\_Date\_Time > Actual\_Start\_Date\_Time)

**2.4 Script de creación de tablas**

Se elaboró un script SQL completo que permite crear toda la estructura de la base de datos con sus relaciones, restricciones y claves primarias/foráneas. Este script ha sido probado y ejecutado exitosamente en el entorno PostgreSQL.  
  
CREATE DATABASE bicycle\_rental

WITH OWNER = developer

TEMPLATE = template1

ENCODING = 'UTF8'

LOCALE\_PROVIDER = 'libc'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1;

IS\_TEMPLATE = False;

GRANT ALL ON DATABASE bicycle\_rental TO developer WITH GRANT OPTION;

-- Tabla de referencia: Métodos de pago

CREATE TABLE Ref\_Payment\_Methods (

    Payment\_Method\_Code VARCHAR(10) PRIMARY KEY,

    Payment\_Method\_Description VARCHAR(50) NOT NULL

);

-- Tabla de referencia: Estados de pago

CREATE TABLE Ref\_Payment\_status\_code (

    Payment\_status\_Code VARCHAR(10) PRIMARY KEY,

    Payment\_status\_Description VARCHAR(50) NOT NULL

);

-- Tabla de tiendas

CREATE TABLE Multi\_Shop (

    Multi\_Shop\_ID SERIAL PRIMARY KEY,

    Contact\_Name VARCHAR(100) NOT NULL,

    Location\_Name VARCHAR(100) NOT NULL,

    Email\_Address VARCHAR(100) CHECK (Email\_Address ~\* '^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),

    Phone\_Number VARCHAR(20),

    Address TEXT NOT NULL,

    Other\_Details TEXT

);

-- Tabla de bicicletas

CREATE TABLE Bicycles (

    Bicycle\_ID SERIAL PRIMARY KEY,

    Bicycle\_Details TEXT NOT NULL

);

-- Tabla de clientes

CREATE TABLE Renters (

    Renter\_ID SERIAL PRIMARY KEY,

    Registration\_Date\_Time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    Last\_Rental\_Date\_Time TIMESTAMP,

    Other\_Details TEXT

);

-- Tabla de tarifas

CREATE TABLE Rental\_Rates (

    Rental\_Rates\_ID SERIAL PRIMARY KEY,

    Daily\_Rate DECIMAL(10,2) NOT NULL CHECK (Daily\_Rate > 0),

    Hourly\_Rate DECIMAL(10,2) NOT NULL CHECK (Hourly\_Rate > 0)

);

-- Tabla de métodos de pago de clientes

CREATE TABLE Renters\_Payment\_Methods (

    Renter\_Payment\_Method\_ID SERIAL PRIMARY KEY,

    Renter\_ID INTEGER NOT NULL REFERENCES Renters(Renter\_ID),

    Payment\_Method\_Code VARCHAR(10) NOT NULL REFERENCES Ref\_Payment\_Methods(Payment\_Method\_Code),

    Card\_Details VARCHAR(100),

    Other\_Details TEXT,

    CONSTRAINT unique\_renter\_payment UNIQUE(Renter\_ID, Payment\_Method\_Code, Card\_Details)

);

-- Tabla de bicicletas en tiendas

CREATE TABLE Bicycles\_in\_Shops (

    Multi\_Shop\_ID INTEGER NOT NULL REFERENCES Multi\_Shop(Multi\_Shop\_ID),

    Bicycle\_ID INTEGER NOT NULL REFERENCES Bicycles(Bicycle\_ID),

    DateTime\_In TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    DateTime\_Out TIMESTAMP,

    Other\_Details TEXT,

    PRIMARY KEY (Multi\_Shop\_ID, Bicycle\_ID, DateTime\_In),

    CHECK (DateTime\_Out IS NULL OR DateTime\_Out > DateTime\_In)

);

-- Tabla de alquileres

CREATE TABLE Rentals (

    Rental\_ID SERIAL PRIMARY KEY,

    Bicycle\_ID INTEGER NOT NULL REFERENCES Bicycles(Bicycle\_ID),

    Payment\_status\_Code VARCHAR(10) NOT NULL REFERENCES Ref\_Payment\_status\_code(Payment\_status\_Code),

    Rental\_Rates\_ID INTEGER NOT NULL REFERENCES Rental\_Rates(Rental\_Rates\_ID),

    Renter\_ID INTEGER NOT NULL REFERENCES Renters(Renter\_ID),

    Renter\_Payment\_Method\_ID INTEGER NOT NULL REFERENCES Renters\_Payment\_Methods(Renter\_Payment\_Method\_ID),

    All\_Day\_Rental\_YN CHAR(1) NOT NULL CHECK (All\_Day\_Rental\_YN IN ('Y', 'N')),

    Booked\_Start\_Date\_Time TIMESTAMP NOT NULL,

    Booked\_End\_Date\_Time TIMESTAMP NOT NULL,

    Actual\_Start\_Date\_Time TIMESTAMP,

    Actual\_End\_Date\_Time TIMESTAMP,

    Rental\_Payment\_Due DECIMAL(10,2) NOT NULL CHECK (Rental\_Payment\_Due >= 0),

    Rental\_Payment\_Made DECIMAL(10,2) DEFAULT 0 CHECK (Rental\_Payment\_Made >= 0),

    Other\_Details TEXT,

    CHECK (Booked\_End\_Date\_Time > Booked\_Start\_Date\_Time),

    CHECK (Actual\_End\_Date\_Time IS NULL OR Actual\_End\_Date\_Time > Actual\_Start\_Date\_Time)

);

**2.5 Inserción de datos de prueba**

Para demostrar el funcionamiento del sistema, se creó un conjunto representativo de datos de prueba. Entre los datos insertados se incluyen:

* Métodos y estados de pago
* Tiendas y bicicletas disponibles
* Clientes registrados y sus métodos de pago
* Bicicletas asignadas a tiendas
* Alquileres realizados con detalles completos

-- Insertar métodos de pago

INSERT INTO Ref\_Payment\_Methods (Payment\_Method\_Code, Payment\_Method\_Description) VALUES

('CASH', 'Efectivo'),

('VISA', 'Tarjeta Visa'),

('MC', 'MasterCard'),

('AMEX', 'American Express'),

('PAYPAL', 'PayPal');

-- Insertar estados de pago

INSERT INTO Ref\_Payment\_status\_code (Payment\_status\_Code, Payment\_status\_Description) VALUES

('PEND', 'Pendiente'),

('PAID', 'Pagado'),

('FAIL', 'Fallido'),

('REFND', 'Reembolsado'),

('PART', 'Pago Parcial');

-- Insertar tiendas

INSERT INTO Multi\_Shop (Contact\_Name, Location\_Name, Email\_Address, Phone\_Number, Address) VALUES

('Juan Pérez', 'Tienda Centro', 'centro@multishop.com', '555-1001', 'Av. Principal 123, Centro'),

('María Gómez', 'Tienda Norte', 'norte@multishop.com', '555-1002', 'Calle Norte 456, Zona Norte'),

('Carlos Ruiz', 'Tienda Sur', 'sur@multishop.com', '555-1003', 'Boulevard Sur 789, Zona Sur');

-- Insertar bicicletas

INSERT INTO Bicycles (Bicycle\_Details) VALUES

('Bicicleta de montaña, roja, talla M'),

('Bicicleta urbana, azul, talla L'),

('Bicicleta de carretera, negra, talla S'),

('Bicicleta híbrida, verde, talla M'),

('Bicicleta plegable, gris, talla única');

-- Insertar tarifas

INSERT INTO Rental\_Rates (Daily\_Rate, Hourly\_Rate) VALUES

(25.00, 5.00),

(30.00, 6.00),

(20.00, 4.00),

(35.00, 7.00),

(40.00, 8.00);

-- Insertar clientes

INSERT INTO Renters (Registration\_Date\_Time, Last\_Rental\_Date\_Time) VALUES

('2024-01-15 10:00:00', '2024-04-20 18:30:00'),

('2024-02-10 14:30:00', '2024-04-22 16:45:00'),

('2024-03-05 09:15:00', NULL);

-- Insertar métodos de pago de clientes

INSERT INTO Renters\_Payment\_Methods (Renter\_ID, Payment\_Method\_Code, Card\_Details) VALUES

(1, 'VISA', '\*\*\*\*-\*\*\*\*-\*\*\*\*-1234'),

(1, 'CASH', NULL),

(2, 'MC', '\*\*\*\*-\*\*\*\*-\*\*\*\*-5678'),

(3, 'AMEX', '\*\*\*\*-\*\*\*\*-\*\*\*\*-9012'),

(3, 'PAYPAL', 'cliente3@email.com');

-- Insertar bicicletas en tiendas

INSERT INTO Bicycles\_in\_Shops (Multi\_Shop\_ID, Bicycle\_ID, DateTime\_In) VALUES

(1, 1, '2024-04-01 09:00:00'),

(1, 2, '2024-04-01 09:00:00'),

(2, 3, '2024-04-02 10:00:00'),

(2, 4, '2024-04-02 10:00:00'),

(3, 5, '2024-04-03 11:00:00');

-- Insertar alquileres

INSERT INTO Rentals (

    Bicycle\_ID, Payment\_status\_Code, Rental\_Rates\_ID, Renter\_ID, Renter\_Payment\_Method\_ID,

    All\_Day\_Rental\_YN, Booked\_Start\_Date\_Time, Booked\_End\_Date\_Time,

    Actual\_Start\_Date\_Time, Actual\_End\_Date\_Time, Rental\_Payment\_Due, Rental\_Payment\_Made

) VALUES

(1, 'PAID', 1, 1, 1, 'Y', '2024-04-10 09:00:00', '2024-04-10 18:00:00', '2024-04-10 09:15:00', '2024-04-10 17:45:00', 25.00, 25.00),

(2, 'PAID', 2, 2, 3, 'N', '2024-04-12 14:00:00', '2024-04-12 16:00:00', '2024-04-12 14:05:00', '2024-04-12 15:50:00', 12.00, 12.00),

(3, 'PEND', 3, 3, 4, 'Y', '2024-04-15 10:00:00', '2024-04-15 19:00:00', '2024-04-15 10:10:00', NULL, 20.00, 10.00);

Este conjunto de datos permite validar los procesos de consulta, inserción, actualización y borrado implementados en la aplicación.

**3. Aplicación de escritorio**

**Documentación de la Aplicación de Escritorio**

**1. Estructura General de la Aplicación**

La aplicación está desarrollada en Java utilizando Swing para la interfaz gráfica y PostgreSQL como base de datos. Sigue una arquitectura modular con los siguientes componentes principales:

**Clase Principal: AppAlquilerBicicletas**

* **Función**: Punto de entrada de la aplicación que configura la ventana principal y el diseño general.
* **Características**:
  + Interfaz con barra lateral (sidebar) para navegación entre módulos
  + Diseño con CardLayout para cambiar entre paneles
  + Barra superior con título y botón de salida
  + Estilo visual consistente con colores y fuentes uniformes

Texto

El contenido generado por IA puede ser incorrecto.

**Clase ConexionBD**

* **Función**: Maneja la conexión a la base de datos PostgreSQL
* **Características**:
  + Patrón Singleton para gestionar una única conexión
  + Métodos para conectar, desconectar y verificar la conexión
  + Configuración centralizada de credenciales y URL de conexión
* import javax.swing.\*;  
  import javax.swing.border.TitledBorder;  
  import javax.swing.table.\*;  
  import java.awt.\*;  
  import java.awt.event.ActionListener;  
  import java.io.\*;  
  import java.sql.\*;  
  import java.text.SimpleDateFormat;  
  import java.time.LocalDate;  
  import java.time.format.DateTimeFormatter;  
  import java.util.Arrays;  
    
  // Clases de servicio y utilidades  
  class ConexionBD {  
   private static Connection *conexion*;  
   private static final String *URL* = "jdbc:postgresql://localhost:5432/bicycle\_rental";  
   private static final String *USER* = "developer";  
   private static final String *PASSWORD* = "23100132";  
    
   public static void conectar() {  
   try {  
   *conexion* = DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
   System.*out*.println("Conexión establecida con éxito.");  
   } catch (SQLException e) {  
   System.*out*.println("Error al conectar a la base de datos: " + e.getMessage());  
   }  
   }  
    
   public static void desconectar() {  
   if (*conexion* != null) {  
   try {  
   *conexion*.close();  
   System.*out*.println("Conexión cerrada con éxito.");  
   } catch (SQLException e) {  
   System.*out*.println("Error al cerrar la conexión: " + e.getMessage());  
   }  
   }  
   }  
    
   public static Connection getConexion() {  
   try {  
   if (*conexion* == null || *conexion*.isClosed()) {  
   *conectar*(); // intenta reconectar si está cerrada  
   }  
   } catch (SQLException e) {  
   System.*out*.println("Error al verificar la conexión: " + e.getMessage());  
   }  
   return *conexion*;  
   }  
    
  }

**2. Módulos CRUD**

La aplicación implementa operaciones CRUD completas para cada entidad principal:

**PanelTiendas**

* **Función**: Gestiona las tiendas donde se alquilan bicicletas
* **Operaciones**:
  + Crear nuevas tiendas con información de contacto y ubicación
  + Leer y mostrar lista de tiendas existentes
  + Actualizar información de tiendas
  + Eliminar tiendas del sistema
* **Interfaz**:
  + Tabla con scroll para visualización
  + Formulario para entrada de datos
  + Validación de campos obligatorios

Una captura de pantalla de una computadora

El contenido generado por IA puede ser incorrecto.

**PanelBicicletas**

* **Función**: Administra el inventario de bicicletas y su ubicación en tiendas
* **Operaciones**:
  + Registrar entrada/salida de bicicletas en tiendas
  + Consultar estado actual de bicicletas
  + Actualizar información de ubicación
  + Eliminar registros de movimiento
* **Interfaz**:
  + Tabla con historial de movimientos
  + Filtros para búsqueda específica

Una captura de pantalla de una computadora

El contenido generado por IA puede ser incorrecto.

**PanelClientes**

* **Función**: Gestiona la información de clientes registrados
* **Operaciones**:
  + Registrar nuevos clientes
  + Consultar historial de clientes
  + Actualizar datos de contacto
  + Eliminar clientes (con validaciones)
* **Interfaz**:
  + Formulario simplificado
  + Visualización de último alquiler

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**PanelAlquileres**

* **Función**: Administra los registros de alquileres
* **Operaciones**:
  + Crear nuevos alquileres
  + Consultar alquileres activos/históricos
  + Actualizar estados de alquiler
  + Cancelar alquileres
* **Interfaz**:
  + Tabla detallada con múltiples campos
  + Filtros por fechas y estados

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**PanelPagos**

* **Función**: Gestiona métodos de pago de clientes
* **Operaciones**:
  + Agregar nuevos métodos de pago
  + Consultar métodos existentes
  + Actualizar información de pago
  + Eliminar métodos obsoletos
* **Interfaz**:
  + Formulario con validación de datos sensibles
  + Visualización en tabla

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**3. Procesos de Negocio Clave**

**PanelProcesos - Registrar Nuevo Alquiler**

1. **Flujo**:
   * Selección de cliente, bicicleta y método de pago
   * Configuración de fechas y tipo de tarifa (hora/día)
   * Cálculo automático del monto
   * Confirmación y registro en base de datos
2. **Validaciones**:
   * Disponibilidad de bicicleta
   * Método de pago válido
   * Coherencia en fechas

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**PanelProcesos - Cálculo de Pagos**

1. **Flujo**:
   * Listado de alquileres pendientes de pago
   * Cálculo de monto adeudado
   * Opción para marcar como pagado
   * Actualización de estado en base de datos
2. **Características**:
   * Visualización de saldos
   * Historial de pagos

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**PanelProcesos - Gestión de Bicicletas**

1. **Flujo**:
   * Asignación de bicicletas a tiendas
   * Devolución al inventario
   * Consulta de ubicación actual
2. **Validaciones**:
   * Estado actual de la bicicleta
   * Existencia de la tienda

Tabla

El contenido generado por IA puede ser incorrecto.

**4. Manejo de Transacciones**

La aplicación implementa manejo de transacciones en operaciones críticas:

1. **Ejemplo en alquileres**:
   * Inicio de transacción al registrar nuevo alquiler
   * Actualización concurrente de:
     + Estado de bicicleta
     + Registro de alquiler
     + Historial de cliente
   * Commit/Rollback según resultado
2. **Ejemplo en pagos**:
   * Transacción al procesar pago:
     + Actualización de estado de alquiler
     + Registro en historial de pagos
     + Actualización de saldos
3. **Implementación**:
   * Uso de Connection.setAutoCommit(false)
   * Manejo explícito de commit/rollback
   * Bloques try-catch para manejo de errores

**5. Generación de Reportes (PanelReportes)**

En esta sección se generaran todos los reportes mediante querys que realizaran las consultas en la base de datos y devolverán la información solicitada mediante txt.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

static class PanelReportes extends JPanel {  
 public PanelReportes() {  
 setLayout(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.fill = GridBagConstraints.*HORIZONTAL*;  
 gbc.insets = new Insets(10, 10, 10, 10);  
  
 setBackground(Color.*WHITE*);  
  
 JLabel titleLabel = new JLabel("Gestión de Reportes");  
 titleLabel.setFont(new Font("Arial", Font.*BOLD*, 24));  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.gridwidth = 2;  
 add(titleLabel, gbc);  
 gbc.gridwidth = 1;  
  
 int fila = 1;  
  
 // Reportes básicos  
 fila = agregarBotonReporte("Alquileres por Cliente", "Genera un reporte de todos los alquileres por cliente.",  
 this::generarReporteAlquileresPorCliente, gbc, fila);  
 fila = agregarBotonReporte("Ingresos por Tienda", "Reporte de ingresos generados por tienda.",  
 this::generarReporteIngresosPorTienda, gbc, fila);  
 fila = agregarBotonReporte("Bicicletas Más Rentadas", "Bicicletas con más alquileres registrados.",  
 this::generarReporteBicicletasMasRentadas, gbc, fila);  
  
 // Reportes por tabla  
 fila = agregarBotonReporte("Clientes (Renters)", "Listado completo de clientes registrados.",  
 this::generarReporteRenters, gbc, fila);  
 fila = agregarBotonReporte("Métodos de Pago", "Listado de métodos de pago de clientes.",  
 this::generarReportePaymentMethods, gbc, fila);  
 fila = agregarBotonReporte("Bicicletas", "Listado de bicicletas en el sistema.",  
 this::generarReporteBicycles, gbc, fila);  
 fila = agregarBotonReporte("Bicicletas en Tiendas", "Ubicaciones de bicicletas por tienda.",  
 this::generarReporteBicyclesInShops, gbc, fila);  
 fila = agregarBotonReporte("Tiendas (Multi\_Shop)", "Tiendas registradas en el sistema.",  
 this::generarReporteMultiShop, gbc, fila);  
 }  
  
 private int agregarBotonReporte(String titulo, String descripcion, Runnable accion,  
 GridBagConstraints gbc, int fila) {  
 JButton boton = new JButton(titulo);  
 gbc.gridx = 0;  
 gbc.gridy = fila;  
 add(boton, gbc);  
  
 JLabel desc = new JLabel(descripcion);  
 gbc.gridx = 1;  
 add(desc, gbc);  
  
 boton.addActionListener(e -> accion.run());  
 return fila + 1;  
 }  
  
 private String obtenerNombreArchivo(String baseNombre) {  
 String fecha = LocalDate.*now*().format(DateTimeFormatter.*ofPattern*("yyyy-MM-dd"));  
 return baseNombre + "\_" + fecha + ".txt";  
 }  
  
 private Connection conectar() throws SQLException {  
 return DriverManager.*getConnection*("jdbc:postgresql://localhost:5432/bicycle\_rental",  
 "developer", "23100132");  
 }

**Reporte de Alquileres por Cliente**

* **Contenido**:
  + Listado de clientes
  + Cantidad de alquileres por cliente
  + Monto total generado
* **Formato**: Archivo de texto con columnas delimitadas
* private void generarReporteAlquileresPorCliente() {  
   String archivo = obtenerNombreArchivo("reporte\_alquileres\_por\_cliente");  
   try (BufferedWriter w = new BufferedWriter(new FileWriter(archivo));  
   Connection c = conectar();  
   Statement s = c.createStatement();  
   ResultSet r = s.executeQuery("""  
   SELECT Renter\_ID, COUNT(\*) AS total, SUM(Rental\_Payment\_Due) AS total\_pago  
   FROM Rentals GROUP BY Renter\_ID ORDER BY total DESC  
   """)) {  
   while (r.next()) {  
   w.write("Cliente ID: " + r.getInt(1) + ", Total alquileres: " + r.getInt(2)  
   + ", Total pago: $" + r.getDouble(3));  
   w.newLine();  
   }  
   } catch (Exception e) {  
   e.printStackTrace();  
   }  
  }

**Reporte de Ingresos por Tienda**

* **Contenido**:
  + Desglose de ingresos por ubicación
  + Comparativa entre tiendas
  + Período cubierto
* **Formato**: Archivo CSV para fácil importación a hojas de cálculo
* private void generarReporteIngresosPorTienda() {  
   String archivo = obtenerNombreArchivo("reporte\_ingresos\_por\_tienda");  
   try (BufferedWriter w = new BufferedWriter(new FileWriter(archivo));  
   Connection c = conectar();  
   Statement s = c.createStatement();  
   ResultSet r = s.executeQuery("""  
   SELECT ms.Multi\_Shop\_ID, ms.Location\_Name, SUM(r.Rental\_Payment\_Made) AS ingresos  
   FROM Rentals r  
   JOIN Bicycles b ON b.Bicycle\_ID = r.Bicycle\_ID  
   JOIN Bicycles\_in\_Shops bis ON bis.Bicycle\_ID = b.Bicycle\_ID  
   JOIN Multi\_Shop ms ON ms.Multi\_Shop\_ID = bis.Multi\_Shop\_ID  
   GROUP BY ms.Multi\_Shop\_ID, ms.Location\_Name  
   """)) {  
   while (r.next()) {  
   w.write("Tienda: " + r.getString(2) + " (ID: " + r.getInt(1) + "), Ingresos: $" + r.getDouble(3));  
   w.newLine();  
   }  
   } catch (Exception e) {  
   e.printStackTrace();  
   }  
  }

**Reporte de Bicicletas Más Rentadas**

* **Contenido**:
  + Ranking de bicicletas por frecuencia de alquiler
  + Detalles de cada bicicleta
  + Ingresos generados
* **Formato**: Archivo de texto con formato legible
* private void generarReporteBicicletasMasRentadas() {  
   String archivo = obtenerNombreArchivo("reporte\_bicicletas\_mas\_rentadas");  
   try (BufferedWriter w = new BufferedWriter(new FileWriter(archivo));  
   Connection c = conectar();  
   Statement s = c.createStatement();  
   ResultSet r = s.executeQuery("""  
   SELECT b.Bicycle\_ID, b.Bicycle\_Details, COUNT(\*) AS total  
   FROM Bicycles b  
   JOIN Rentals r ON b.Bicycle\_ID = r.Bicycle\_ID  
   GROUP BY b.Bicycle\_ID, b.Bicycle\_Details  
   ORDER BY total DESC  
   """)) {  
   while (r.next()) {  
   w.write("Bicicleta ID: " + r.getInt(1) + ", Rentas: " + r.getInt(3) + ", Detalles: " + r.getString(2));  
   w.newLine();  
   }  
   } catch (Exception e) {  
   e.printStackTrace();  
   }  
  }

**Reportes Adicionales**

* Listado completo de clientes (Renters)
* private void generarReporteRenters() {  
   String archivo = obtenerNombreArchivo("reporte\_renters");  
   String sql = "SELECT Renter\_ID, Registration\_Date\_Time, Last\_Rental\_Date\_Time, Other\_Details FROM Renters";  
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo));  
   Connection conn = conectar();  
   Statement stmt = conn.createStatement();  
   ResultSet rs = stmt.executeQuery(sql)) {  
    
   writer.write("ID,Fecha Registro,Último Alquiler,Detalles\n");  
   while (rs.next()) {  
   int id = rs.getInt("Renter\_ID");  
   Timestamp registro = rs.getTimestamp("Registration\_Date\_Time");  
   Timestamp ultimo = rs.getTimestamp("Last\_Rental\_Date\_Time");  
   String detalles = rs.getString("Other\_Details");  
   writer.write(id + "," + registro + "," + (ultimo != null ? ultimo : "") + "," + detalles + "\n");  
   }  
   JOptionPane.*showMessageDialog*(this, "Reporte de renters generado.");  
   } catch (Exception e) {  
   e.printStackTrace();  
   JOptionPane.*showMessageDialog*(this, "Error al generar reporte de renters: " + e.getMessage());  
   }  
  }
* Métodos de pago registrados
* private void generarReportePaymentMethods() {  
   String archivo = obtenerNombreArchivo("reporte\_payment\_methods");  
   String sql = """  
  SELECT rpm.Renter\_Payment\_Method\_ID, rpm.Renter\_ID, rpm.Payment\_Method\_Code, rpd.Payment\_Method\_Description, rpm.Card\_Details  
  FROM Renters\_Payment\_Methods rpm  
  JOIN Ref\_Payment\_Methods rpd ON rpm.Payment\_Method\_Code = rpd.Payment\_Method\_Code  
  """;  
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo));  
   Connection conn = conectar();  
   Statement stmt = conn.createStatement();  
   ResultSet rs = stmt.executeQuery(sql)) {  
    
   writer.write("ID Método,ID Renter,Código,Descripción,Detalles Tarjeta\n");  
   while (rs.next()) {  
   int id = rs.getInt("Renter\_Payment\_Method\_ID");  
   int renterId = rs.getInt("Renter\_ID");  
   String code = rs.getString("Payment\_Method\_Code");  
   String desc = rs.getString("Payment\_Method\_Description");  
   String card = rs.getString("Card\_Details");  
   writer.write(id + "," + renterId + "," + code + "," + desc + "," + card + "\n");  
   }  
   JOptionPane.*showMessageDialog*(this, "Reporte de métodos de pago generado.");  
   } catch (Exception e) {  
   e.printStackTrace();  
   JOptionPane.*showMessageDialog*(this, "Error al generar reporte de métodos de pago: " + e.getMessage());  
   }  
  }
* Inventario de bicicletas
* private void generarReporteBicycles() {  
   String archivo = obtenerNombreArchivo("reporte\_bicycles");  
   String sql = "SELECT Bicycle\_ID, Bicycle\_Details FROM Bicycles";  
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo));  
   Connection conn = conectar();  
   Statement stmt = conn.createStatement();  
   ResultSet rs = stmt.executeQuery(sql)) {  
    
   writer.write("ID Bicicleta,Detalles\n");  
   while (rs.next()) {  
   int id = rs.getInt("Bicycle\_ID");  
   String detalles = rs.getString("Bicycle\_Details");  
   writer.write(id + "," + detalles + "\n");  
   }  
   JOptionPane.*showMessageDialog*(this, "Reporte de bicicletas generado.");  
   } catch (Exception e) {  
   e.printStackTrace();  
   JOptionPane.*showMessageDialog*(this, "Error al generar reporte de bicicletas: " + e.getMessage());  
   }  
  }
* Distribución de bicicletas por tienda
* private void generarReporteBicyclesInShops() {  
   String archivo = obtenerNombreArchivo("reporte\_bicycles\_tiendas");  
   String sql = """  
  SELECT ms.Multi\_Shop\_ID, ms.Contact\_Name, COUNT(\*) AS Bicycle\_Count  
  FROM Multi\_Shop ms  
  JOIN Bicycles\_in\_Shops bis ON ms.Multi\_Shop\_ID = bis.Multi\_Shop\_ID  
  GROUP BY ms.Multi\_Shop\_ID, ms.Contact\_Name  
  """;  
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo));  
   Connection conn = conectar();  
   Statement stmt = conn.createStatement();  
   ResultSet rs = stmt.executeQuery(sql)) {  
    
   writer.write("ID Tienda,Nombre Contacto,Cantidad Bicicletas\n");  
   while (rs.next()) {  
   int id = rs.getInt("Multi\_Shop\_ID");  
   String contacto = rs.getString("Contact\_Name");  
   int cantidad = rs.getInt("Bicycle\_Count");  
   writer.write(id + "," + contacto + "," + cantidad + "\n");  
   }  
   JOptionPane.*showMessageDialog*(this, "Reporte de bicicletas por tienda generado.");  
   } catch (Exception e) {  
   e.printStackTrace();  
   JOptionPane.*showMessageDialog*(this, "Error al generar reporte: " + e.getMessage());  
   }  
  }
* Listado de tiendas
* private void generarReporteMultiShop() {  
   String archivo = obtenerNombreArchivo("reporte\_multishop");  
   String sql = "SELECT Multi\_Shop\_ID, Contact\_Name, Location\_Name FROM Multi\_Shop";  
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivo));  
   Connection conn = conectar();  
   Statement stmt = conn.createStatement();  
   ResultSet rs = stmt.executeQuery(sql)) {  
    
   writer.write("ID Tienda,Nombre Contacto,Ubicación\n");  
   while (rs.next()) {  
   int id = rs.getInt("Multi\_Shop\_ID");  
   String contacto = rs.getString("Contact\_Name");  
   String ubicacion = rs.getString("Location\_Name");  
   writer.write(id + "," + contacto + "," + ubicacion + "\n");  
   }  
   JOptionPane.*showMessageDialog*(this, "Reporte de tiendas generado.");  
   } catch (Exception e) {  
   e.printStackTrace();  
   JOptionPane.*showMessageDialog*(this, "Error al generar reporte de tiendas: " + e.getMessage());  
   }  
  }

**6. Utilidades Adicionales**

**Panel de Inicio – *BikeRent Manager***

El PanelInicio es la pantalla principal del sistema, con diseño limpio y funcional. Incluye:

* **Mensaje de bienvenida** con información sobre las funciones del sistema.
* **Imágenes de bicicletas** representativas del inventario.
* **Botones de acceso rápido** para respaldo, restauración de datos y gestión de inventario.
* **Estadísticas clave** como bicicletas disponibles, tiendas activas y alquileres diarios.

Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Codigo:  
  
static class PanelInicio extends JPanel {  
 public PanelInicio() {  
 setLayout(new BorderLayout(10, 10));  
 setBackground(new Color(245, 245, 245)); // Gris claro neutro  
 setBorder(BorderFactory.*createEmptyBorder*(15, 15, 15, 15));  
  
 // Panel superior con información  
 JPanel infoPanel = new JPanel(new BorderLayout());  
 infoPanel.setBackground(new Color(245, 245, 245));  
 infoPanel.setBorder(BorderFactory.*createCompoundBorder*(  
 BorderFactory.*createTitledBorder*(BorderFactory.*createLineBorder*(new Color(100, 100, 100), 2),  
 "Bienvenido a BikeRent Manager"),  
 BorderFactory.*createEmptyBorder*(10, 10, 10, 10)));  
  
 JTextArea infoArea = new JTextArea();  
 infoArea.setText("Sistema Integral de Gestión para Alquiler de Bicicletas\n\n"  
 + "Esta aplicación le permite administrar todo el ciclo de alquiler:\n"  
 + "- Control de inventario de bicicletas por tienda\n"  
 + "- Gestión de clientes y reservas\n"  
 + "- Seguimiento de alquileres activos y devoluciones\n"  
 + "- Generación de reportes de rentabilidad y uso\n\n"  
 + "Nuestro compromiso: Ofrecer la mejor plataforma para gestionar tu negocio de alquiler de bicicletas.");  
 infoArea.setEditable(false);  
 infoArea.setMargin(new Insets(10, 15, 10, 15));  
 infoArea.setLineWrap(true);  
 infoArea.setWrapStyleWord(true);  
 infoArea.setFont(new Font("SansSerif", Font.*PLAIN*, 14));  
 infoArea.setBackground(new Color(250, 250, 250)); // Gris muy claro  
 infoArea.setForeground(new Color(70, 70, 70)); // Gris oscuro  
  
 infoPanel.add(infoArea, BorderLayout.*CENTER*);  
 add(infoPanel, BorderLayout.*NORTH*);  
  
 // Panel central con imágenes y botones  
 JPanel centerPanel = new JPanel(new GridLayout(1, 2, 15, 15));  
 centerPanel.setBackground(new Color(245, 245, 245));  
  
 // Panel izquierdo para imágenes de bicicletas  
 JPanel imagePanel = new JPanel(new GridLayout(2, 1, 10, 10));  
 imagePanel.setBorder(BorderFactory.*createTitledBorder*(  
 BorderFactory.*createLineBorder*(new Color(100, 100, 100)),  
 "Nuestras Bicicletas"));  
 imagePanel.setBackground(new Color(245, 245, 245));  
  
 // Imagen 1 - Bicicleta urbana  
 JLabel imgBike1;  
 try {  
 ImageIcon bikeIcon1 = new ImageIcon(getClass().getResource("/images/bike1.jpg"));  
 // Redimensionar la imagen para que se ajuste al espacio  
 Image img = bikeIcon1.getImage().getScaledInstance(300, 200, Image.*SCALE\_SMOOTH*);  
 imgBike1 = new JLabel(new ImageIcon(img));  
 imgBike1.setToolTipText("Bicicletas urbanas para alquiler");  
 } catch (NullPointerException e) {  
 imgBike1 = new JLabel("Imagen no disponible", JLabel.*CENTER*);  
 imgBike1.setForeground(Color.*RED*);  
 }  
 imgBike1.setHorizontalAlignment(JLabel.*CENTER*);  
 imgBike1.setBorder(BorderFactory.*createCompoundBorder*(  
 BorderFactory.*createLineBorder*(new Color(150, 150, 150), 2),  
 BorderFactory.*createEmptyBorder*(5, 5, 5, 5)));  
 imgBike1.setBackground(Color.*WHITE*);  
 imgBike1.setOpaque(true);  
  
 // Imagen 2 - Bicicleta de montaña  
 JLabel imgBike2;  
 try {  
 ImageIcon bikeIcon2 = new ImageIcon(getClass().getResource("/images/bike2.jpg"));  
 // Redimensionar la imagen  
 Image img2 = bikeIcon2.getImage().getScaledInstance(300, 200, Image.*SCALE\_SMOOTH*);  
 imgBike2 = new JLabel(new ImageIcon(img2));  
 imgBike2.setToolTipText("Bicicletas de montaña para aventuras");  
 } catch (NullPointerException e) {  
 imgBike2 = new JLabel("Imagen no disponible", JLabel.*CENTER*);  
 imgBike2.setForeground(Color.*RED*);  
 }  
 imgBike2.setHorizontalAlignment(JLabel.*CENTER*);  
 imgBike2.setBorder(BorderFactory.*createCompoundBorder*(  
 BorderFactory.*createLineBorder*(new Color(150, 150, 150), 2),  
 BorderFactory.*createEmptyBorder*(5, 5, 5, 5)));  
 imgBike2.setBackground(Color.*WHITE*);  
 imgBike2.setOpaque(true);  
  
 imagePanel.add(imgBike1);  
 imagePanel.add(imgBike2);  
 centerPanel.add(imagePanel);  
  
 // Panel de botones con estilo neutro  
 JPanel buttonPanel = new JPanel(new GridLayout(3, 1, 10, 15));  
 buttonPanel.setBackground(new Color(245, 245, 245));  
 buttonPanel.setBorder(BorderFactory.*createTitledBorder*(  
 BorderFactory.*createLineBorder*(new Color(100, 100, 100)),  
 "Acciones Rápidas"));  
  
 // Botón para realizar copias de seguridad (usando DatabaseUtils)  
 JButton backupButton = createNeutralButton("📁 Realizar Respaldo",  
 new Color(120, 120, 120), // Gris medio  
 e -> utils.DatabaseUtils.*realizarCopiasDeSeguridad*(this));  
  
 // Botón para restaurar la base de datos (usando DatabaseUtils)  
 JButton restoreButton = createNeutralButton("🔄 Restaurar Datos",  
 new Color(90, 90, 90), // Gris oscuro  
 e -> utils.DatabaseUtils.*restaurarBaseDeDatos*(this));  
  
 // Botón para gestión de inventario  
 JButton inventoryButton = createNeutralButton("🚲 Inventario Bicicletas",  
 new Color(150, 150, 150), // Gris claro  
 e -> JOptionPane.*showMessageDialog*(this,  
 "Módulo de inventario de bicicletas por tienda",  
 "Gestión de Inventario",  
 JOptionPane.*INFORMATION\_MESSAGE*));  
  
 buttonPanel.add(backupButton);  
 buttonPanel.add(restoreButton);  
 buttonPanel.add(inventoryButton);  
 centerPanel.add(buttonPanel);  
  
 add(centerPanel, BorderLayout.*CENTER*);  
  
 // Panel inferior con estadísticas  
 JPanel statsPanel = new JPanel(new GridLayout(1, 3, 10, 10));  
 statsPanel.setBackground(new Color(245, 245, 245));  
 statsPanel.setBorder(BorderFactory.*createTitledBorder*(  
 BorderFactory.*createLineBorder*(new Color(100, 100, 100)),  
 "Estadísticas del Negocio"));  
  
 // Método para crear labels de estadísticas con estilo  
 statsPanel.add(createStatLabel("Bicicletas Disponibles", "42"));  
 statsPanel.add(createStatLabel("Tiendas Activas", "5"));  
 statsPanel.add(createStatLabel("Alquileres Hoy", "18"));  
  
 add(statsPanel, BorderLayout.*SOUTH*);  
 }  
  
 // Método auxiliar para crear botones con estilo neutro  
 private JButton createNeutralButton(String text, Color bgColor, ActionListener action) {  
 JButton button = new JButton(text);  
 button.setBackground(bgColor);  
 button.setForeground(Color.*WHITE*);  
 button.setFont(new Font("SansSerif", Font.*BOLD*, 14));  
 button.setFocusPainted(false);  
 button.setBorder(BorderFactory.*createCompoundBorder*(  
 BorderFactory.*createLineBorder*(new Color(200, 200, 200)), // Gris claro  
 BorderFactory.*createEmptyBorder*(8, 15, 8, 15)));  
 button.addActionListener(action);  
 return button;  
 }  
  
 // Método auxiliar para crear labels de estadísticas  
 private JPanel createStatLabel(String title, String value) {  
 JPanel panel = new JPanel(new BorderLayout());  
 panel.setBackground(new Color(250, 250, 250));  
 panel.setBorder(BorderFactory.*createLineBorder*(new Color(200, 200, 200)));  
  
 JLabel titleLabel = new JLabel(title, JLabel.*CENTER*);  
 titleLabel.setFont(new Font("SansSerif", Font.*BOLD*, 12));  
 titleLabel.setForeground(new Color(70, 70, 70));  
  
 JLabel valueLabel = new JLabel(value, JLabel.*CENTER*);  
 valueLabel.setFont(new Font("SansSerif", Font.*BOLD*, 24));  
 valueLabel.setForeground(new Color(50, 50, 50));  
  
 panel.add(titleLabel, BorderLayout.*NORTH*);  
 panel.add(valueLabel, BorderLayout.*CENTER*);  
  
 return panel;  
 }  
}

**DatabaseUtils:**

* **Función**: Realiza operaciones de mantenimiento de base de datos
* **Características**:
  + Copias de seguridad (backup) programáticas
  + Restauración desde archivos de respaldo
  + Interfaz gráfica para selección de archivos

package utils;  
import javax.swing.\*;  
import java.awt.\*;  
import java.io.\*;  
import java.util.Arrays;  
import java.util.List;  
  
public class DatabaseUtils {  
  
 private static final String *PG\_DUMP\_PATH* = "C:\\Program Files\\PostgreSQL\\16\\bin\\pg\_dump.exe";  
 private static final String *PSQL\_PATH* = "C:\\Program Files\\PostgreSQL\\16\\bin\\psql.exe";  
 private static final String *DB\_NAME* = "Gaming\_Leagues";  
 private static final String *DB\_USER* = "developer";  
 private static final String *DB\_PASSWORD* = "23100132";  
  
 public static void realizarCopiasDeSeguridad(Component parent) {  
 File backupDir = new File("backups/");  
 if (!backupDir.exists()) backupDir.mkdir();  
  
 String backupFile = "backups/respaldodb\_" + System.*currentTimeMillis*() + ".sql";  
  
 List<String> command = Arrays.*asList*(  
 *PG\_DUMP\_PATH*,  
 "-U", *DB\_USER*,  
 "-h", "localhost",  
 "-d", *DB\_NAME*,  
 "-f", backupFile  
 );  
  
 *ejecutarProceso*(command, parent, "Copia de seguridad creada exitosamente: " + backupFile,  
 "Error al crear la copia de seguridad.");  
 }  
  
 public static void restaurarBaseDeDatos(Component parent) {  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setDialogTitle("Seleccionar archivo de respaldo");  
 if (fileChooser.showOpenDialog(parent) != JFileChooser.*APPROVE\_OPTION*) return;  
  
 File backupFile = fileChooser.getSelectedFile();  
  
 List<String> command = Arrays.*asList*(  
 *PSQL\_PATH*,  
 "-U", *DB\_USER*,  
 "-h", "localhost",  
 "-d", *DB\_NAME*,  
 "-f", backupFile.getAbsolutePath()  
 );  
  
 *ejecutarProceso*(command, parent, "Base de datos restaurada exitosamente.",  
 "Error al restaurar la base de datos.");  
 }  
  
 private static void ejecutarProceso(List<String> command, Component parent, String successMessage, String errorMessage) {  
 try {  
 ProcessBuilder builder = new ProcessBuilder(command);  
 builder.environment().put("PGPASSWORD", *DB\_PASSWORD*);  
 builder.redirectErrorStream(true);  
  
 Process process = builder.start();  
 BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));  
  
 String line;  
 StringBuilder output = new StringBuilder();  
 while ((line = reader.readLine()) != null) {  
 output.append(line).append("\n");  
 System.*out*.println(line);  
 }  
  
 int exitCode = process.waitFor();  
 if (exitCode == 0) {  
 JOptionPane.*showMessageDialog*(parent, successMessage, "Éxito", JOptionPane.*INFORMATION\_MESSAGE*);  
 } else {  
 JOptionPane.*showMessageDialog*(parent, errorMessage + "\nCódigo de salida: " + exitCode + "\n" + output, "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 } catch (IOException | InterruptedException e) {  
 JOptionPane.*showMessageDialog*(parent, errorMessage + "\n" + e.getMessage(), "Error", JOptionPane.*ERROR\_MESSAGE*);  
 e.printStackTrace();  
 }  
 }  
}

**Características de Usabilidad**

* Diseño responsive dentro de tamaño fijo
* Mensajes de confirmación para operaciones críticas
* Validación de datos en formularios
* Feedback visual para operaciones exitosas/fallidas
* Navegación consistente entre módulos

La aplicación cumple con todos los requisitos funcionales, implementando una interfaz intuitiva sobre una base de datos relacional con operaciones transaccionales y generación de reportes para la gestión del negocio de alquiler de bicicletas.