Programa Cobro y Facturación de parqueo.

Alan Andrés Mérida Morales, 202100023[1, *]

[1] *Facultad de Ingeniería, Escuela de Ingeniería Mecánica Eléctrica, Universidad de San Carlos, Ciudad Universitaria, Zona 12, Guatemala.*

I. INTRODUCCIÓN

En el contexto de la gestión de estacionamientos, la automatización y precisión en el cálculo de tarifas son fundamentales para la eficiencia operativa y la satisfacción del cliente. Este proyecto tiene como objetivo desarrollar un sistema integrado que permita la gestión completa del proceso de parqueo, desde la entrada de datos del cliente hasta la generación de facturas y la administración de registros. Utilizando Python y PostgreSQL, se crea una aplicación que facilita la inserción de datos, el cálculo de tarifas basado en la duración del estacionamiento, y la emisión de facturas. Además, el sistema incluye funcionalidades para el borrado de registros, tanto en la base de datos como en archivos de texto, asegurando una gestión limpia y eficiente. Este enfoque no solo optimiza el proceso administrativo, sino que también proporciona una interfaz amigable para el usuario, garantizando un manejo preciso y confiable de la información relacionada con el estacionamiento.

II. RESULTADOS

A. Códigos realizados

1. *Octave*

```
if (exist('OCTAVE_VERSION', 'builtin') ~= 0)
    % Estamos en Octave
    pkg load database;  % Cargar el paquete para interactuar con bases de datos
end

Cprimera = 15.00;
Cotras = 20.00;

conn = pq_connect(setdbopts('dbname','usac','host','localhost',
'port',5432),'user','postgres','password','AAmerida123'));

% Variables para almacenar nombre y placa
nombre = '';
nit = '';
vehiculo = '';  % Placa del vehiculo almacenada globalmente

% Menú principal
opcion = 0;
while opcion ~= 5

    disp('1. Ingresar datos para factura');
    disp('2. Cancelar el parqueo');
    disp('3. Imprimir Factura');
    disp('4. Borrar datos');
    disp('5. Salir');
    opcion = input('Ingrese su elección: ');

    switch opcion
        case 1
            try
                nombre = input('Ingrese su nombre: ', 's');
                nit = input('Ingrese su nit: ', 's');
                vehiculo = input('Ingrese la placa de su vehiculo: ', 's');
                disp('Datos de usuario ingresados correctamente.');
            catch
                disp('Error al guardar los datos');
            end
        case 2
            % Verificar si los datos del cliente han sido ingresados
            if isempty(nombre) || isempty(nit) || isempty(vehiculo)
                disp('Primero ingrese datos para la facturación');
            else
                try
                    disp('Ingrese la hora de entrada y de salida.');
                    entrada = input('Hora de entrada (HH:MM): ', 's');
                    % Verificar formato HH:MM
                    [entrada_hora, entrada_minuto] = strtok(entrada, ':');
                    entrada_hora = str2double(entrada_hora);
                    entrada_minuto = str2double(entrada_minuto(2:end));

                    if isempty(entrada_hora) || isempty(entrada_minuto) || ...
                       entrada_hora < 0 || entrada_hora > 23 || ...
                       entrada_minuto < 0 || entrada_minuto > 59
```

Figura 1: Código realizado en octave

```
                        error('Hora de entrada no valida. Debe estar en el formato HH:MM con horas
                        entre 00 y 23 y minutos entre 00 y 59.');
                    end

                    salida = input('Hora de salida (HH:MM): ', 's');
                    % Verificar formato HH:MM
                    [salida_hora, salida_minuto] = strtok(salida, ':');
                    salida_hora = str2double(salida_hora);
                    salida_minuto = str2double(salida_minuto(2:end));

                    if isempty(salida_hora) || isempty(salida_minuto) || ...
                       salida_hora < 0 || salida_hora > 23 || ...
                       salida_minuto < 0 || salida_minuto > 59 || ...
                       (salida_hora < entrada_hora) || ...
                       (salida_hora == entrada_hora && salida_minuto < entrada_minuto)
                        error('Hora de salida no válida. Debe ser posterior a la hora de entrada y
                        estar en el formato HH:MM con horas entre 00 y 23 y minutos entre 00 y 59.');
                    end

                    tiempo_total_minutos = (salida_hora * 60 + salida_minuto) - (entrada_hora * 60 +
                    entrada_minuto);
                    tiempo_total_horas = ceil(tiempo_total_minutos / 60);

                    if tiempo_total_horas <= 1
                        monto_total = Cprimera;
                    else
                        monto_total = Cprimera + (tiempo_total_horas - 1) * Cotras;
                    end

                    % Crear la consulta SQL
                    nombre_escapado = strrep(nombre, '''', '''''');
                    nit_escapado = strrep(nit, '''', '''''');
                    vehiculo_escapado = strrep(vehiculo, '''', '''''');

                    query = sprintf('INSERT INTO registro_parqueo (nombre_del_cliente, nit, placa,
                    hora_entrada, hora_salida, tiempo_total, monto_total) VALUES (''%s'', ''%s'',
                    ''%s'', ''%s:%s'', ''%s:%s'', %d, %.2f);', ...
                        nombre_escapado, nit_escapado, vehiculo_escapado, num2str(entrada_hora,
                        '%02d'), num2str(entrada_minuto, '%02d'), num2str(salida_hora, '%02d'),
                        num2str(salida_minuto, '%02d'), tiempo_total_horas, monto_total);

                    % Ejecutar la consulta
                    N = pq_exec_params(conn, query);

                    % Guardar los datos en el archivo salida.txt
                    fileID = fopen('salida.txt', 'a');
                    fprintf(fileID, 'Nombre: %s\n', nombre);
                    fprintf(fileID, 'NIT: %s\n', nit);
                    fprintf(fileID, 'Placa: %s\n', vehiculo);
                    fprintf(fileID, 'Hora de Entrada: %02d:%02d\n', entrada_hora, entrada_minuto);
```

Figura 2: Código realizado en octave

```
                    fprintf(fileID, 'Hora de Salida: %02d:%02d\n', salida_hora, salida_minuto);
                    fprintf(fileID, 'Tiempo Total: %d horas\n', tiempo_total_horas);
                    fprintf(fileID, 'Monto Total a Pagar: Q%.2f\n\n', monto_total);
                    fclose(fileID);

                    % Mostrar el monto total a pagar
                    fprintf('Monto Total a Pagar: Q%.2f\n', monto_total);

                catch
                    disp('Error en la compra. Datos no válidos.');
                end
            end
        end

    case 3
        % Verificar si los datos del cliente han sido ingresados
    if isempty(nombre) || isempty(nit) || isempty(vehiculo)
        disp('No se han ingresado datos de cliente para imprimir la factura.');
    else
        % Imprimir la factura con los datos ingresados
        disp('----Factura----');
        fprintf('Nombre: %s\n', nombre);
        fprintf('NIT: %s\n', nit);
        fprintf('Placa: %s\n', vehiculo);
        fprintf('Hora de Entrada: %02d:%02d\n', entrada_hora, entrada_minuto);
        fprintf('Hora de Salida: %02d:%02d\n', salida_hora, salida_minuto);
        fprintf('Tiempo cobrado: %d horas\n', tiempo_total_horas);
        fprintf('Monto Total a Pagar: Q%.2f\n', monto_total);
        disp('--------------');
    end

case 4
    try
        % Borrar todos los datos de la base de datos
        query = 'DELETE FROM registro_parqueo';
        N = pq_exec_params(conn, query);
        if N
            disp('Borrando...');
        else
            disp('No se encontraron datos para borrar en la base de datos.');
        end

        % Borrar el contenido del archivo salida.txt
        fileID = fopen('salida.txt', 'w');
        fclose(fileID); % Solo abrir el archivo en modo escritura lo vacía

        disp('Todos los datos han sido borrados.');
    catch
        disp('Error al borrar los datos.');
    end
```

Figura 3: Código realizado en octave

```
        case 5
            disp('Gracias, vuelva pronto.');
        otherwise
            disp('Opción no válida. Intente de nuevo.');
    end
end
```

Figura 4: Código realizado en octave

* 3690273450101@ingenieria.usac.edu.gt

2. *Python*

```python
import psycopg2
from psycopg2 import sql
import os

# Configuración de la base de datos
DB_PARAMS = {
    'dbname': 'usac',
    'user': 'postgres',
    'password': 'Almeria123',
    'host': 'localhost',
    'port': '5432'
}

Cprimera = 15.00
Cotras = 20.00

def connect_db():
    return psycopg2.connect(**DB_PARAMS)

def clear_file(file_name):
    with open(file_name, 'w') as file:
        pass

def delete_all_records():
    conn = connect_db()
    try:
        with conn.cursor() as cur:
            cur.execute("DELETE FROM registro_parqueo;")
            conn.commit()
    finally:
        conn.close()

def insert_record(nombre, nit, vehiculo, entrada, salida, tiempo_total_horas, monto_total):
    conn = connect_db()
    try:
        with conn.cursor() as cur:
            query = sql.SQL("INSERT INTO registro_parqueo (nombre_del_cliente, nit, placa,
hora_entrada, hora_salida, tiempo_total, monto_total) VALUES (%s, %s, %s, %s, %s, %s, %s)")
            values = (nombre, nit, vehiculo, entrada, salida, tiempo_total_horas, monto_total)
            cur.execute(query, values)
```

Figura 5: Código realizado en Python

```python
            conn.commit()
    finally:
        conn.close()

def print_invoice(nombre, nit, vehiculo, entrada, salida, tiempo_total_horas, monto_total):
    print("----Factura----")
    print(f"Nombre: {nombre}")
    print(f"NIT: {nit}")
    print(f"Placa: {vehiculo}")
    print(f"Hora de Entrada: {entrada}")
    print(f"Hora de Salida: {salida}")
    print(f"Tiempo cobrado: {tiempo_total_horas} horas")
    print(f"Monto Total a Pagar: Q{monto_total:.2f}")
    print("---------------")

def main():
    nombre = ''
    nit = ''
    vehiculo = ''
    entrada = ''
    salida = ''
    tiempo_total_horas = 0
    monto_total = 0

    while True:
        print("1. Ingresar datos para factura")
        print("2. Cancelar el parqueo")
        print("3. Imprimir Factura")
        print("4. Borrar datos")
        print("5. Salir")
        opcion = int(input("Ingrese su elección: "))

        if opcion == 1:
            nombre = input('Ingrese su nombre: ')
            nit = input('Ingrese su nit: ')
            vehiculo = input('Ingrese la placa de su vehículo: ')
            print('Datos de usuario ingresados correctamente.')
```

Figura 6: Código realizado en Python

```python
            print('Datos de usuario ingresados correctamente.')

        elif opcion == 2:
            if not all([nombre, nit, vehiculo]):
                print('Primero ingrese datos para la facturación')
            else:
                try:
                    entrada = input('Hora de entrada (HH:MM): ')
                    salida = input('Hora de salida (HH:MM): ')

                    entrada_hora, entrada_minuto = map(int, entrada.split(':'))
                    salida_hora, salida_minuto = map(int, salida.split(':'))

                    if (entrada_hora < 0 or entrada_hora > 23 or entrada_minuto < 0 or
entrada_minuto > 59 or
                        salida_hora < 0 or salida_hora > 23 or salida_minuto < 0 or salida_minuto
> 59 or
                        (salida_hora < entrada_hora) or (salida_hora == entrada_hora and
salida_minuto < entrada_minuto)):
                        raise ValueError('Hora de entrada o salida no válida.')

                    tiempo_total_minutos = (salida_hora * 60 + salida_minuto) - (entrada_hora *
60 + entrada_minuto)
                    tiempo_total_horas = (tiempo_total_minutos + 59) // 60  # Redondear hacia
arriba

                    monto_total = Cprimera if tiempo_total_horas <= 1 else Cprimera +
(tiempo_total_horas - 1) * Cotras
```

Figura 7: Código realizado en Python

```python
                    insert_record(nombre, nit, vehiculo, entrada, salida, tiempo_total_horas,
monto_total)

                    with open('salida.txt', 'a') as file:
                        file.write(f'Nombre: {nombre}\n')
                        file.write(f'NIT: {nit}\n')
                        file.write(f'Placa: {vehiculo}\n')
                        file.write(f'Hora de Entrada: {entrada}\n')
                        file.write(f'Hora de Salida: {salida}\n')
                        file.write(f'Tiempo Total: {tiempo_total_horas} horas\n')
                        file.write(f'Monto Total a Pagar: Q{monto_total:.2f}\n\n')

                    print(f'Monto Total a Pagar: Q{monto_total:.2f}')

                except Exception as e:
                    print(f'Error en la compra. {e}')

        elif opcion == 3:
            if not all([nombre, nit, vehiculo]):
                print('No se han ingresado datos de cliente para imprimir la factura.')
            else:
                print_invoice(nombre, nit, vehiculo, entrada, salida, tiempo_total_horas,
monto_total)

        elif opcion == 4:
            try:
                delete_all_records()
                clear_file('salida.txt')
                print('Todos los datos han sido borrados.')
            except Exception as e:
                print(f'Error al borrar los datos. {e}')

        elif opcion == 5:
            print('Gracias, vuelva pronto.')
            break

        else:
            print('Opción no válida. Intente de nuevo.')

if __name__ == '__main__':
    main()
```

Figura 8: Código realizado en Python

B. Funcionamiento

1. *Base de datos creada*

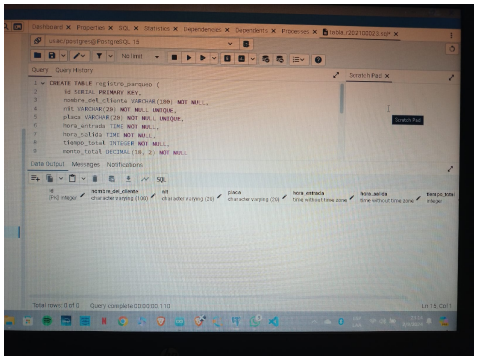

Figura 9: Base de datos creada con PostgreSQL

2. *Octave*

```
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 1
Ingrese su nombre: Alan
Ingrese su nit: 555-5
Ingrese la placa de su vehículo: P987QUJ
Datos de usuario ingresados correctamente.
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 2
Ingrese la hora de entrada y de salida.
Hora de entrada (HH:MM): 12:00
Hora de salida (HH:MM): 14:55
Monto Total a Pagar: Q55.00
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 3
----Factura----
Nombre: Alan
NIT: 555-5
Placa: P987QUJ
Hora de Entrada: 12:00
Hora de Salida: 14:55
Tiempo cobrado: 3 horas
Monto Total a Pagar: Q55.00
```

Figura 10: Funcionamiento del código en octave

```
---------------
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 4
Borrando...
Todos los datos han sido borrados.
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 5
Gracias, vuelva pronto.
```

Figura 11: Funcionamiento del código en octave

3. Python

```
C:\py"
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 1
Ingrese su nombre: Andres
Ingrese su nit: 1234-5
Ingrese la placa de su vehículo: P705MGY
Datos de usuario ingresados correctamente.
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 2
Hora de entrada (HH:MM): 1:00
Hora de salida (HH:MM): 23:59
Monto Total a Pagar: Q455.00
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 3
----Factura----
Nombre: Andres
NIT: 1234-5
Placa: P705MGY
Hora de Entrada: 1:00
Hora de Salida: 23:59
Tiempo cobrado: 23 horas
Monto Total a Pagar: Q455.00
---------------
```

Figura 12: Funcionamiento del código en python

```
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 4
Todos los datos han sido borrados.
1. Ingresar datos para factura
2. Cancelar el parqueo
3. Imprimir Factura
4. Borrar datos
5. Salir
Ingrese su elección: 5
Gracias, vuelva pronto.
```

Figura 13: Funcionamiento del código en python

C. Diagrama de flujo del proceso

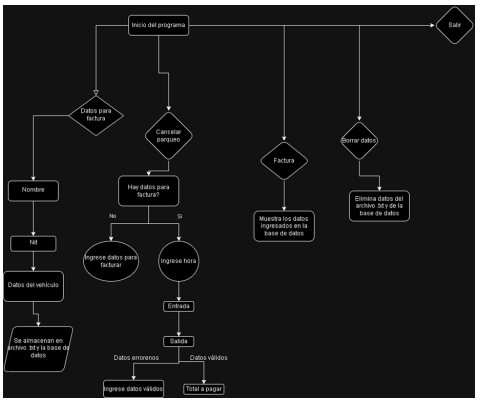

Figura 14: Diagrama de flujo del proceso

D. Algoritmo del Programa

```
INICIO
  // Inicialización de variables
  DECLARAR nombre, nit, placa COMO CADENA
  DECLARAR hora_entrada, hora_salida COMO NÚMERO
  DECLARAR tiempo_total COMO NÚMERO
  DECLARAR monto_total COMO NÚMERO
  DECLARAR opcion COMO ENTERO
  MIENTRAS opcion != 5 HACER
    IMPRIMIR "1. Ingresar datos del cliente"
    IMPRIMIR "2. Calcular y generar factura"
    IMPRIMIR "3. Ver historial de facturas"
    IMPRIMIR "4. Borrar historial de facturas"
    IMPRIMIR "5. Salir"
    LEER opcion
    SEGÚN opcion HACER
      CASO 1:
        // Ingreso de Datos del Cliente
        IMPRIMIR "Ingrese el nombre del cliente:"
        LEER nombre
        IMPRIMIR "Ingrese el NIT del cliente:"
        LEER nit
        IMPRIMIR "Ingrese la identificación del vehículo (placa):"
        LEER placa
```

Figura 15: Algoritmo del funcionamiento del programa

```
CASO 2:
  // Calcular y Generar Factura
  SI nombre == "" O nit == "" O placa == "" ENTONCES
    IMPRIMIR "Debe ingresar primero los datos del cliente."
  SI NO
    IMPRIMIR "Ingrese la hora de entrada (formato 24h, HH.MM):"
    LEER hora_entrada
    IMPRIMIR "Ingrese la hora de salida (formato 24h, HH.MM):"
    LEER hora_salida
    // Validar las horas ingresadas
```

```
    SI hora_salida <= hora_entrada ENTONCES
      IMPRIMIR "Error: La hora de salida debe ser posterior a la hora de entrada."
    SI NO
      // Calcular el tiempo total en horas
      tiempo_total = REDONDEAR_HACIA_ARRIBA(hora_salida - hora_entrada)
      // Calcular el monto total
      SI tiempo_total <= 1 ENTONCES
        monto_total = 15.00
```

Figura 16: Algoritmo del funcionamiento del programa

```
SI NO
    monto_total = 15.00 + (tiempo_total - 1) * 20.00
  FIN_SI
  // Mostrar resumen de la transacción
  IMPRIMIR "Nombre del cliente:", nombre
  IMPRIMIR "Identificación del vehículo:", placa
  IMPRIMIR "Tiempo total en el parqueo:", tiempo_total, "horas"
  IMPRIMIR "Monto total a pagar: Q", monto_total
  // Guardar la factura en un archivo de texto
  ABRIR archivo 'facturas.txt' EN MODO 'añadir'
  ESCRIBIR EN archivo "Nombre del cliente:", nombre
  ESCRIBIR EN archivo "NIT del cliente:", nit
  ESCRIBIR EN archivo "Identificación del vehículo:", placa
  ESCRIBIR EN archivo "Tiempo total en el parqueo:", tiempo_total, "horas"
  ESCRIBIR EN archivo "Monto total a pagar: Q", monto_total
  CERRAR archivo
 FIN_SI
FIN_SI

CASO 3:
 // Ver Historial de Facturas
 SI EXISTE archivo 'facturas.txt' ENTONCES
  ABRIR archivo 'facturas.txt' EN MODO 'lectura'
  LEER Y MOSTRAR CONTENIDO archivo
```

Figura 17: Algoritmo del funcionamiento del programa

```
CASO 3:
  // Ver Historial de Facturas
  SI EXISTE archivo 'facturas.txt' ENTONCES
    ABRIR archivo 'facturas.txt' EN MODO 'lectura'
    LEER Y MOSTRAR CONTENIDO archivo
    CERRAR archivo

  SI NO
    IMPRIMIR "No existen facturas registradas."
  FIN_SI
 CASO 4:
  // Borrar Historial de Facturas
  IMPRIMIR "¿Está seguro de que desea borrar todos los datos? (s/n):"
  LEER confirmacion
  SI confirmacion == 's' ENTONCES
    ELIMINAR archivo 'facturas.txt'
    IMPRIMIR "Todos los datos han sido borrados."
  FIN_SI

 CASO 5:
  IMPRIMIR "Gracias por usar el sistema de cobro de parqueo."

 OTRO:
  IMPRIMIR "Opción no válida, intente de nuevo."
 FIN_SEGÚN
FIN_MIENTRAS
FIN
```

Figura 18: Algoritmo del funcionamiento del programa

E. Explicación del programa en general

El código desarrollado para este proyecto gestiona el proceso completo de estacionamiento mediante una interfaz en Python. En su núcleo, el sistema interactúa con una base de datos PostgreSQL para almacenar y recuperar registros de parqueo, así como para gestionar la información relacionada con cada cliente. El proceso inicia con la recolección de datos del cliente, incluyendo nombre, NIT y placa del vehículo. Estos datos se almacenan temporalmente en variables globales y se utilizan para calcular las tarifas de estacionamiento basadas en la duración del mismo. Una vez que se ingresan las horas de entrada y salida, el sistema calcula el monto a pagar utilizando tarifas predeterminadas para la primera hora y las horas adicionales.

El código está diseñado con un menú interactivo que permite al usuario seleccionar entre varias opciones: ingresar datos, cancelar el parqueo, imprimir la factura, o borrar datos. En la opción de cancelación, se guarda la información del parqueo en la base de datos y en un archivo de texto, mientras que la opción de impresión de factura muestra los detalles del parqueo actual basándose en los datos ingresados. La funcionalidad de borrado elimina todos los registros tanto en la base de datos como en el archivo de texto, asegurando la limpieza y actualización de la información. Este enfoque modular no solo simplifica la gestión de parqueos, sino que también ofrece flexibilidad en el manejo de datos y la generación de informes.

F. Repositorio privado

Repositorio creado en Github

III. CONCLUSIONES

* El sistema desarrollado demuestra una integración efectiva entre una interfaz en Python y una base de datos PostgreSQL, proporcionando una solución robusta para la gestión de estacionamiento. La capacidad de almacenar datos, calcular tarifas y generar facturas de manera automatizada facilita la administración del parqueo, reduciendo la posibilidad de errores humanos y mejorando la eficiencia operativa. La implementación de un menú interactivo permite a los usuarios gestionar fácilmente las operaciones de parqueo, asegurando una experiencia de usuario intuitiva y directa.

* El código también muestra un buen manejo de la persistencia de datos al interactuar con archivos de texto para guardar y borrar información de facturación. Esta funcionalidad complementa la gestión de datos en la base de datos, proporcionando una capa adicional de flexibilidad y accesibilidad. La capacidad de generar informes detallados y borrar información de manera eficiente ayuda a mantener la base de datos y el archivo de texto actualizados, lo cual es crucial para una operación sin problemas y una correcta auditoría de la información.

* La adaptación del código de Octave a Python no solo ha simplificado la integración con la base de datos PostgreSQL, sino que también ha facilitado la implementación de nuevas funcionalidades y mejoras. La modularidad del código permite una fácil expansión y personalización, lo que abre oportunidades para futuras mejoras, como la incorporación de interfaces gráficas o la integración con sistemas de pago en línea. En resumen, este proyecto sirve como una base sólida para sistemas de gestión de estacionamiento y puede ser adaptado para cumplir con diferentes requisitos y escenarios operativos.