Sistema de Administración de Transportes (SAT)

Bienvenido a la documentación del Sistema de Administración de Transportes (SAT).

Contenido:

Introducción

El Sistema de Administración de Transportes (SAT) es una aplicación diseñada para gestionar servicios de transporte, clientes, ventas y facturas. Este sistema permite a los usuarios realizar diversas operaciones como crear y actualizar servicios, gestionar clientes, realizar ventas y generar facturas.

Funcionalidades principales

- Gestión de servicios de transporte
- Administración de clientes
- Realización de ventas
- Generación de facturas

Requisitos del sistema

- Python 3.x
- SQLite3

Instalación

1. Clona el repositorio o descarga los archivos del proyecto.
2. Asegúrate de tener Python 3.x instalado en tu sistema.
3. No se requieren dependencias adicionales.

Uso básico

Para iniciar el sistema, ejecuta el script principal:

```
python main.py
```

Sigue las instrucciones en pantalla para navegar por los diferentes módulos y realizar las operaciones deseadas.

Índices y tablas

- Índice
- Índice de Módulos
- Página de Búsqueda

Funciones del Menú Principal

Este módulo contiene las funciones principales para la gestión del menú y la interacción con el usuario en el sistema SAT.

Funciones

Ejemplo de uso

```
# Inicializar objetos necesarios
conexion_db = ...  # Crear conexión a la base de datos
servicios = Servicios(conexion_db)
clientes = Clientes(conexion_db)
ventas = Ventas(conexion_db)
facturas = Facturas(conexion_db)

# Ejecutar el menú principal
generar_menu(conexion_db, servicios, ventas, clientes, facturas)
```

Nota: Este módulo maneja la interacción principal con el usuario y coordina las diferentes funcionalidades del sistema SAT.

Clase Servicios

class servicios.**Servicios**(*objeto_conexion*)

Bases: **object**

Esta clase representa un servicio con atributos como código, nombre, origen, destino, precio de venta, hora de salida, cantidad máxima de puestos y cantidad máxima de kilos.

actualizar_nombre_servicio(*objeto_conexion, codigo_servicio, nuevo_nombre*)

Actualiza el nombre de un servicio.

Args:

objeto_conexion: Conexión a la base de datos. nuevo_nombre: Nuevo nombre del servicio. codigo_servicio: Código del servicio a actualizar.

Returns:

bool: True si la actualización fue exitosa, False en caso contrario.

cantidad_max_kilos = *None*

cantidad_max_puestos = *None*

codigo_servicio = *None*

consultar_informacion_servicio(*objeto_conexion, codigo_servicio*)

Consulta la información de un servicio.

Args:

objeto_conexion: Conexión a la base de datos. codigo_servicio: Código del servicio a consultar.

Returns:

tuple: Tupla con los datos del servicio, o None si no se encuentra.

crear_nuevo_servicio(*objeto_conexion, mi_servicio*)

Inserta un nuevo servicio en la base de datos.

Args:

objeto_conexion: Conexión a la base de datos. mi_servicio: Tupla con los datos del servicio (código, nombre, etc.).

Returns:

bool: True si la inserción fue exitosa, False en caso contrario.

destino = *None*

hora_salida = *None*

nombre = *None*

origen = *None*

precio_venta = *None*

La clase `Servicios` representa un servicio de transporte en el sistema SAT.

Atributos

- `codigo_servicio` (int): Código único del servicio.
- `nombre` (str): Nombre del servicio.
- `origen` (str): Lugar de origen del servicio.
- `destino` (str): Lugar de destino del servicio.
- `precio_venta` (float): Precio de venta del servicio.
- `hora_salida` (datetime): Hora de salida del servicio.
- `cantidad_max_puestos` (int): Cantidad máxima de puestos disponibles.
- `cantidad_max_kilos` (int): Cantidad máxima de kilos permitidos.

Métodos

`Servicios.__init__(objeto_conexion)`
Constructor de la clase Servicio. Crea la tabla “servicios” en la base de datos si no existe.

Args:
objeto_conexion: Conexión a la base de datos.

`Servicios.crear_nuevo_servicio(objeto_conexion, mi_servicio)`
Inserta un nuevo servicio en la base de datos.

Args:
objeto_conexion: Conexión a la base de datos. mi_servicio: Tupla con los datos del servicio (código, nombre, etc.).

Returns:
bool: True si la inserción fue exitosa, False en caso contrario.

`Servicios.actualizar_nombre_servicio(objeto_conexion, codigo_servicio, nuevo_nombre)`
Actualiza el nombre de un servicio.

Args:
objeto_conexion: Conexión a la base de datos. nuevo_nombre: Nuevo nombre del servicio. codigo_servicio: Código del servicio a actualizar.

Returns:
bool: True si la actualización fue exitosa, False en caso contrario.

`Servicios.consultar_informacion_servicio(objeto_conexion, codigo_servicio)`
Consulta la información de un servicio.

Args:
objeto_conexion: Conexión a la base de datos. codigo_servicio: Código del servicio a consultar.

Returns:
tuple: Tupla con los datos del servicio, o None si no se encuentra.

Ejemplo de uso

```
# Crear una instancia de Servicios
servicios = Servicios(conexion_db)
```

```python
# Crear un nuevo servicio
nuevo_servicio = (1, "Ruta Express", "Ciudad A", "Ciudad B", 50.0, "2023-09-01 08:00:00", 5
servicios.crear_nuevo_servicio(conexion_db, nuevo_servicio)

# Actualizar el nombre de un servicio
servicios.actualizar_nombre_servicio(conexion_db, 1, "Ruta Rápida")

# Consultar información de un servicio
info_servicio = servicios.consultar_informacion_servicio(conexion_db, 1)
print(info_servicio)
```

Clase Clientes

class clientes.**Clientes**(*objeto_conexion*)

Bases: **object**

Esta clase representa un cliente con atributos como número de identificación, nombre, apellido, dirección, teléfono y correo electrónico.

actualizar_direccion_cliente(*objeto_conexion, no_identificacion_cliente, nueva_direccion*)

Actualiza la dirección de un cliente.

Args:

objeto_conexion: Conexión a la base de datos. no_identificacion_cliente: Número de identificación del cliente a actualizar. nueva_direccion: Nueva dirección del cliente.

Returns:

bool: True si la actualización fue exitosa, False en caso contrario.

apellido = *None*

consultar_informacion_cliente(*objeto_conexion, no_identificacion_cliente*)

Consulta la información de un cliente.

Args:

objeto_conexion: Conexión a la base de datos. no_identificacion_cliente: Número de identificación del cliente a consultar.

Returns:

tuple: Tupla con los datos del cliente, o None si no se encuentra.

correo_electronico = *None*

crear_nuevo_cliente(*objeto_conexion, mi_cliente*)

Inserta un nuevo cliente en la base de datos.

Args:

objeto_conexion: Conexión a la base de datos. mi_cliente: Tupla con los datos del cliente (número de identificación, nombre, etc.).

Returns:

bool: True si la inserción fue exitosa, False en caso contrario.

direccion = *None*

no_identificacion_cliente = *None*

nombre = *None*

telefono = *None*

La clase `Clientes` representa a un cliente en el sistema SAT.

Atributos

- `no_identificacion_cliente` (int): Número de identificación único del cliente.
- `nombre` (str): Nombre del cliente.
- `apellido` (str): Apellido del cliente.
- `direccion` (str): Dirección del cliente.
- `telefono` (str): Número de teléfono del cliente.
- `correo_electronico` (str): Correo electrónico del cliente.

Métodos

`Clientes.__init__(objeto_conexion)`

Constructor de la clase Clientes. Crea la tabla "clientes" en la base de datos si no existe.

Args:

objeto_conexion: Conexión a la base de datos.

`Clientes.crear_nuevo_cliente(objeto_conexion, mi_cliente)`

Inserta un nuevo cliente en la base de datos.

Args:

objeto_conexion: Conexión a la base de datos. mi_cliente: Tupla con los datos del cliente (número de identificación, nombre, etc.).

Returns:

bool: True si la inserción fue exitosa, False en caso contrario.

`Clientes.actualizar_direccion_cliente(objeto_conexion, no_identificacion_cliente, nueva_direccion)`

Actualiza la dirección de un cliente.

Args:

objeto_conexion: Conexión a la base de datos. no_identificacion_cliente: Número de identificación del cliente a actualizar. nueva_direccion: Nueva dirección del cliente.

Returns:

bool: True si la actualización fue exitosa, False en caso contrario.

`Clientes.consultar_informacion_cliente(objeto_conexion, no_identificacion_cliente)`

Consulta la información de un cliente.

Args:

objeto_conexion: Conexión a la base de datos. no_identificacion_cliente: Número de identificación del cliente a consultar.

Returns:

tuple: Tupla con los datos del cliente, o None si no se encuentra.

Ejemplo de uso

```
# Crear una instancia de Clientes
clientes = Clientes(conexion_db)

# Crear un nuevo cliente
```

```python
nuevo_cliente = (1001, "Juan", "Pérez", "Calle 123 #45-67", "1234567890", "juan@example.com'
clientes.crear_nuevo_cliente(conexion_db, nuevo_cliente)

# Actualizar la dirección de un cliente
clientes.actualizar_direccion_cliente(conexion_db, 1001, "Avenida 456 #78-90")

# Consultar información de un cliente
info_cliente = clientes.consultar_informacion_cliente(conexion_db, 1001)
print(info_cliente)
```

Clase Ventas

class ventas.**Ventas**(*objeto_conexion*)

Bases: **object**

Esta clase representa una venta con atributos como número de factura, número de identificación del cliente, código del servicio y cantidad vendida.

añadir_servicio_factura(*objeto_conexion, mi_venta*)

Registra la venta de un servicio en la base de datos.

generar_numero_factura(*objeto_conexion*)

Genera un nuevo número de factura.

quitar_servicio_factura(*objeto_conexion, no_factura*) ¶

Elimina un registro de venta de la base de datos.

verificar_disponibilidad(*objeto_conexion, mi_servicio, cantidad_vendida, carga_max*)

Verifica si hay disponibilidad para la venta.

La clase `Ventas` representa una venta en el sistema SAT.

Atributos

- `no_factura` (int): Número de factura de la venta.
- `no_identificacion_cliente` (int): Número de identificación del cliente.
- `codigo_servicio` (int): Código del servicio vendido.
- `cantidad_vendida` (int): Cantidad vendida del servicio.

Métodos

Ventas.**__init__**(*objeto_conexion*)

Constructor de la clase Ventas. Crea la tabla “ventas” en la base de datos si no existe.

Args:

objeto_conexion: Conexión a la base de datos.

Ventas.**generar_numero_factura**(*objeto_conexion*)

Genera un nuevo número de factura.

Ventas.**verificar_disponibilidad**(*objeto_conexion, mi_servicio, cantidad_vendida, carga_max*)

Verifica si hay disponibilidad para la venta.

Ventas.**añadir_servicio_factura**(*objeto_conexion, mi_venta*)

Registra la venta de un servicio en la base de datos.

Ventas.**quitar_servicio_factura**(*objeto_conexion, no_factura*)

Elimina un registro de venta de la base de datos.

Ejemplo de uso

```
# Crear una instancia de Ventas
ventas = Ventas(conexion_db)

# Generar un nuevo número de factura
nuevo_no_factura = ventas.generar_numero_factura(conexion_db)

# Verificar disponibilidad antes de la venta
if ventas.verificar_disponibilidad(conexion_db, servicio, cantidad, carga_max):
    # Añadir un servicio a la factura
    venta = (nuevo_no_factura, id_cliente, codigo_servicio, cantidad)
    ventas.añadir_servicio_factura(conexion_db, venta)

# Quitar un servicio de la factura
ventas.quitar_servicio_factura(conexion_db, nuevo_no_factura)
```

Nota: Esta clase maneja las operaciones relacionadas con las ventas, incluyendo la generación de números de factura, verificación de disponibilidad y registro de ventas en la base de datos.

Clase Facturas

class facturas.**Facturas**(*objetoConexion*)

Bases: **object**

imprimir_factura(*conexion, no_factura*)

La clase `Facturas` maneja la generación e impresión de facturas en el sistema SAT.

Atributos

- `objetoConexion` (objeto): Conexión a la base de datos.

Métodos

Facturas.**__init__**(*objetoConexion*)

Facturas.**imprimir_factura**(*conexion, no_factura*)

Ejemplo de uso

```
# Crear una instancia de Facturas
facturas = Facturas(conexion_db)

# Imprimir una factura
facturas.imprimir_factura(conexion_db, numero_factura)
```

Nota: Esta clase se encarga de generar y mostrar las facturas basándose en la información de ventas, clientes y servicios almacenada en la base de datos.