FUNCIONAMENTO DO SISTEMA DE GESTÃO VEICULAR - SisGEVEC

Síntese do funcionamento do sistema e objetivo

O SisGEVEC tem o objetivo final atender o usuário a fazer gestão interna de atendimento veicular de reboques. O sistema SisGEVEC apresenta janelas de login e registros dos produtos que neste caso são os veículos.

Para entrar no **SisGEVEC**, o usuário faz o acesso inicial pela janela de Login e autenticação, registrando usuário e senha que abrirá uma tela principal de **"Registrar Atendimentos".**

Nessa janela de "**Registrar o atendimentos"** temos as caixas de preenchimento do Funcionário, Empresa, Placa do Veículo, Origem, Destino, Quilometragem e a Descrição do Atendimento. Temos também a aba de "**Cadastrar Veículos**", aonde faço o cadastro do veículo informando sua marca, modelo e placa. Temos outra aba de "**Cadastro de Empresas**", que solicitará o Nome, Telefone e Endereço da empresa.

a aba de **"Lista de Atendimentos"** temos o resultado do cadastro feito pelo o usuário. Nessa lista temos as seguintes informações em formato de tabela: ID, data, placa do véiculo, empresa, funcionário, origem, destino e descrição do serviço proposto, há também a quilometragem e o preço cobrado.

Síntese do funcionamento do código

O código foi desenvolvido em linguagem de programação **Python** e foi usado a tecnologia **RAD** (desenvolvimento rápido de aplicações),usando as bibliotecas **Tkinter**, responsável pela interface gráfica do código, **Sqlite3** que é o Banco de Dados interno do Python. Criada as tabelas de usuários, empresas, veículos, placas e funcionários. Há um relacionamento direto entre atendimentos e funcionários, atendimentos e empresas, atendimentos e veículos. Importado também datetime, e o messagebox que mostra avisos de erros e exceções.

No campo de desenvolvimento do código temos:

1.Banco de Dados: nesse campo temos a criação do banco "atendimento_veicular.db" e as tabelas. Importamos o Sqlite3, abrimos a conexão pelo "conn.cursor", o qual fará a execução "cursor.execute" na criação dessas tabelas.

2.Login: nesse campo temos a autenticação do usuário e senha. Foi aplicada o uso de erro e exceções, caso o usuário erre ou esqueça a senha, com uma janela de resposta pelo elemento "messagebox"

3.Cadastros: nesse campo, por meio da função def cadastrar_empresa e def cadastrar_veiculo podemos cadastrar as empresas prestadoras e os veículos. Coletar o que foi registrado. Também aplicado o elemento "messagebox" para mostrar o erro ou sucesso no cadastro, informando placa já cadastrada ou cadastro com sucesso.Foi usado o elemento "entry e o método get"para receber a entrada do cadastro.

4.Registrar Atendimento: nesse campo fazemos os registros de data, placa, funcionário, descrição, quilometragem, origem e destino. Todos registros foram usado o método get para coletar as informações de entrada. Aplicado a condicional " if , else" para daí gerar o resultado no messagebox, caso não tenha preenchido todos os dados, como também se a placa ainda não estiver sido cadastrada, gerará a mensagem "placa não encontrada". A quilometragem segue com uma condicional "if , else", usando o elemento "replace" e gerar um erro de "quilometragem inválida" no messagebox. O preço do serviço será o resultado da equação (preço=150 +(5*quilometragem).

No final, a conexão será encerrada pelo método "conn.close" e o código será executado pelo "conn.commit"

5.Exclusões e Atualizações: nesse campo ocorrerá as exclusões e atualizações de empresas, veículos, atendimento e placas. Para excluir foi utilizado o elemento "tree.selection" em conjunto com o "messagebox.askyesno" que abrirá uma caixa para responder a confirmação de sim ou não para a exclusão. Enquanto na atualização foi usado o "fetchall" que mostra todas as linhas e o "tree.get_children" que retornará os itens do treeView que serão apagados e atualizados.

6.Interface Principal: aqui temos todas as formatações das janelas , botões , títulos, e toda responsividade da interface gráfica de um modo que ficou bem apresentável. Foi aplicado os elementos "combo" e o método "put" e outras diversas funcionalidades que renderizaram o código.

SEGUE O CÓDIGO ABAIXO:

```
import sqlite3
from tkinter import *
from tkinter import ttk, messagebox
import datetime

# ---------------------------
# BANCO DE DADOS
# ---------------------------
def criar_banco():
    conn = sqlite3.connect('atendimento_veicular.db')
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS usuarios (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            email TEXT UNIQUE NOT NULL,
            senha TEXT NOT NULL
        )
    ''')

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS empresas (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nome TEXT NOT NULL,
            telefone TEXT,
            endereco TEXT
        )
```

```
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS veiculos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        marca TEXT NOT NULL,
        modelo TEXT NOT NULL
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS placas (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        numero TEXT NOT NULL UNIQUE,
        veiculo_id INTEGER NOT NULL,
        FOREIGN KEY (veiculo_id) REFERENCES veiculos(id)
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS funcionarios (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nome TEXT NOT NULL
    )
''')

cursor.execute('''
    CREATE TABLE IF NOT EXISTS atendimentos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
            data TEXT NOT NULL,
            veiculo_id INTEGER,
            empresa_id INTEGER,
            funcionario_id INTEGER,
            descricao TEXT,
            quilometragem REAL,
            preco REAL,
            origem TEXT,
            destino TEXT,
            FOREIGN KEY (veiculo_id) REFERENCES veiculos(id),
            FOREIGN KEY (empresa_id) REFERENCES empresas(id),
            FOREIGN KEY (funcionario_id) REFERENCES funcionarios(id)
        )
    ''')

    # Inserções iniciais
    cursor.execute("SELECT COUNT(*) FROM usuarios")
    if cursor.fetchone()[0] == 0:
        cursor.execute("INSERT INTO usuarios (email, senha) VALUES (?, ?)",
("admin@admin.com", "1234"))

    cursor.execute("SELECT COUNT(*) FROM funcionarios")
    if cursor.fetchone()[0] == 0:
        funcionarios = [("João",), ("Maria",), ("Carlos",), ("Ana",)]
        cursor.executemany("INSERT INTO funcionarios (nome) VALUES (?)", funcionarios)

    conn.commit()
    conn.close()

# ----------------------------
```

```
# LOGIN
# ----------------------------
def verificar_login():
    email = entry_email.get()
    senha = entry_senha.get()

    conn = sqlite3.connect('atendimento_veicular.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM usuarios WHERE email=? AND senha=?", (email,
senha))
    usuario = cursor.fetchone()
    conn.close()

    if usuario:
        login_window.destroy()
        abrir_janela_principal()
    else:
        messagebox.showerror("Erro", "E-mail ou senha incorretos.")

def esqueci_senha():
    messagebox.showinfo("Recuperação de Senha", "Entre em contato com o
administrador para redefinir sua senha.")

# ----------------------------
# CADASTROS
# ----------------------------
def cadastrar_empresa():
    nome = entry_nome_empresa.get()
    telefone = entry_telefone_empresa.get()
    endereco = entry_endereco_empresa.get()
```

```
    if nome.strip():
        conn = sqlite3.connect('atendimento_veicular.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO empresas (nome, telefone, endereco) VALUES (?, ?, ?)", (nome, telefone, endereco))
        conn.commit()
        conn.close()
        messagebox.showinfo("Sucesso", "Empresa cadastrada com sucesso.")
        entry_nome_empresa.delete(0, END)
        entry_telefone_empresa.delete(0, END)
        entry_endereco_empresa.delete(0, END)
        atualizar_combo_empresas()
    else:
        messagebox.showwarning("Erro", "Nome da empresa é obrigatório.")

def cadastrar_veiculo():
    marca = entry_marca.get()
    modelo = entry_modelo.get()
    placa = entry_placa.get()

    if marca and modelo and placa:
        conn = sqlite3.connect('atendimento_veicular.db')
        cursor = conn.cursor()
        try:
            cursor.execute("INSERT INTO veiculos (marca, modelo) VALUES (?, ?)", (marca, modelo))
            veiculo_id = cursor.lastrowid
            cursor.execute("INSERT INTO placas (numero, veiculo_id) VALUES (?, ?)", (placa, veiculo_id))
```

```
            conn.commit()
            messagebox.showinfo("Sucesso", "Veículo e placa cadastrados com sucesso.")
            entry_marca.delete(0, END)
            entry_modelo.delete(0, END)
            entry_placa.delete(0, END)
            atualizar_combo_placas()
        except sqlite3.IntegrityError:
            messagebox.showwarning("Erro", "Placa já cadastrada.")
        conn.close()
    else:
        messagebox.showwarning("Erro", "Preencha todos os campos.")


# ----------------------------
# REGISTRAR ATENDIMENTO
# ----------------------------
def registrar_atendimento():
    data = datetime.datetime.now().strftime("%Y-%m-%d")
    placa = combo_placa.get()
    empresa = combo_empresa.get()
    funcionario = combo_funcionario.get()
    descricao = entry_descricao.get()
    quilometragem = entry_quilometragem.get()
    origem = entry_origem.get()
    destino = entry_destino.get()

    if not quilometragem.replace('.', '', 1).isdigit():
        messagebox.showwarning("Erro", "Informe uma quilometragem válida (somente
números).")
        return
```

```
    if not origem or not destino:
        messagebox.showwarning("Erro", "Preencha os campos de Origem e Destino.")
        return

    quilometragem = float(quilometragem)
    preco = 150 + (5 * quilometragem)

    if placa and empresa and funcionario:
        conn = sqlite3.connect('atendimento_veicular.db')
        cursor = conn.cursor()

        cursor.execute("SELECT veiculo_id FROM placas WHERE numero = ?", (placa,))
        veiculo_data = cursor.fetchone()
        if not veiculo_data:
            messagebox.showwarning("Erro", "Placa não encontrada.")
            conn.close()
            return

        veiculo_id = veiculo_data[0]
        cursor.execute("SELECT id FROM empresas WHERE nome = ?", (empresa,))
        empresa_id = cursor.fetchone()[0]
        cursor.execute("SELECT id FROM funcionarios WHERE nome = ?", (funcionario,))
        funcionario_id = cursor.fetchone()[0]

        cursor.execute("""
            INSERT INTO atendimentos (data, veiculo_id, empresa_id, funcionario_id,
descricao,
                                      quilometragem, preco, origem, destino)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
```

```
        """, (data, veiculo_id, empresa_id, funcionario_id, descricao, quilometragem, preco, origem, destino))
        conn.commit()
        conn.close()

        messagebox.showinfo("Sucesso", f"Atendimento registrado!\nPreço calculado: R${preco:.2f}")
        entry_descricao.delete(0, END)
        entry_quilometragem.delete(0, END)
        entry_origem.delete(0, END)
        entry_destino.delete(0, END)
        atualizar_lista_atendimentos()
    else:
        messagebox.showwarning("Erro", "Todos os campos são obrigatórios.")

# ----------------------------
# EXCLUSÕES E ATUALIZAÇÕES
# ----------------------------
def excluir_empresa():
    nome = combo_empresa.get()
    if not nome:
        messagebox.showwarning("Aviso", "Selecione uma empresa.")
        return
    if messagebox.askyesno("Confirmação", f"Deseja excluir a empresa '{nome}'?"):
        conn = sqlite3.connect('atendimento_veicular.db')
        cursor = conn.cursor()
        cursor.execute("DELETE FROM empresas WHERE nome = ?", (nome,))
        conn.commit()
        conn.close()
        atualizar_combo_empresas()
```

```
        messagebox.showinfo("Sucesso", "Empresa excluída.")


def excluir_veiculo():
    placa = combo_placa.get()
    if not placa:
        messagebox.showwarning("Aviso", "Selecione uma placa.")
        return
    if messagebox.askyesno("Confirmação", f"Deseja excluir o veículo com placa
'{placa}'?"):
        conn = sqlite3.connect('atendimento_veicular.db')
        cursor = conn.cursor()
        cursor.execute("SELECT veiculo_id FROM placas WHERE numero = ?", (placa,))
        veiculo = cursor.fetchone()
        if veiculo:
            veiculo_id = veiculo[0]
            cursor.execute("DELETE FROM placas WHERE veiculo_id = ?", (veiculo_id,))
            cursor.execute("DELETE FROM veiculos WHERE id = ?", (veiculo_id,))
            conn.commit()
        conn.close()
        atualizar_combo_placas()
        messagebox.showinfo("Sucesso", "Veículo excluído.")


def excluir_atendimento():
    item = tree.selection()
    if not item:
        messagebox.showwarning("Aviso", "Selecione um atendimento.")
        return
    id_atendimento = tree.item(item[0])['values'][0]
    if messagebox.askyesno("Confirmação", "Excluir este atendimento?"):
        conn = sqlite3.connect('atendimento_veicular.db')
```

```
        cursor = conn.cursor()
        cursor.execute("DELETE FROM atendimentos WHERE id = ?", (id_atendimento,))
        conn.commit()
        conn.close()
        atualizar_lista_atendimentos()
        messagebox.showinfo("Sucesso", "Atendimento excluído.")

def atualizar_combo_empresas():
    conn = sqlite3.connect('atendimento_veicular.db')
    cursor = conn.cursor()
    cursor.execute("SELECT nome FROM empresas ORDER BY nome")
    empresas = [e[0] for e in cursor.fetchall()]
    combo_empresa['values'] = empresas
    conn.close()

def atualizar_combo_placas():
    conn = sqlite3.connect('atendimento_veicular.db')
    cursor = conn.cursor()
    cursor.execute("SELECT numero FROM placas ORDER BY numero")
    placas = [p[0] for p in cursor.fetchall()]
    combo_placa['values'] = placas
    conn.close()

def atualizar_lista_atendimentos():
    for item in tree.get_children():
        tree.delete(item)

    conn = sqlite3.connect('atendimento_veicular.db')
    cursor = conn.cursor()
```

```
    try:
        cursor.execute("""
            SELECT
                a.id, a.data, p.numero, v.modelo, e.nome, f.nome, a.origem, a.destino,
                a.descricao, IFNULL(a.quilometragem, 0),
                printf('R$ %.2f', IFNULL(a.preco, 0))
            FROM atendimentos a
            JOIN veiculos v ON a.veiculo_id = v.id
            JOIN placas p ON v.id = p.veiculo_id
            JOIN empresas e ON a.empresa_id = e.id
            JOIN funcionarios f ON a.funcionario_id = f.id
            ORDER BY a.data DESC
        """)
        for row in cursor.fetchall():
            tree.insert('', END, values=row)
    except Exception as e:
        print("Erro ao carregar lista:", e)
    conn.close()

# ----------------------------
# INTERFACE PRINCIPAL
# ----------------------------
def abrir_janela_principal():
    global combo_funcionario, combo_empresa, combo_placa
    global entry_descricao, entry_quilometragem, entry_origem, entry_destino, tree
    global entry_nome_empresa, entry_telefone_empresa, entry_endereco_empresa
    global entry_marca, entry_modelo, entry_placa

    root = Tk()
```

```
root.title("SISTEMA DE GESTÃO VEICULAR - SisGEVEC")
root.geometry("1200x750")

# --- TÍTULO EM NEGRITO ---
Label(root, text="SISTEMA DE GESTÃO VEICULAR - SisGEVEC",
    font=("Arial", 16, "bold")).pack(pady=10)

abas = ttk.Notebook(root)
aba_empresa = Frame(abas)
aba_veiculo = Frame(abas)
aba_atendimento = Frame(abas)
aba_lista = Frame(abas)

abas.add(aba_atendimento, text="Registrar Atendimento")
abas.add(aba_veiculo, text="Cadastro de Veículos")
abas.add(aba_empresa, text="Cadastro de Empresas")
abas.add(aba_lista, text="Lista de Atendimentos")
abas.pack(fill='both', expand=True)

# --- ÍCONES SIMBÓLICOS (gerados via código) ---
icone_func = PhotoImage(width=20, height=20)
icone_func.put(("blue",), to=(5,5,15,15))
icone_emp = PhotoImage(width=20, height=20)
icone_emp.put(("green",), to=(5,5,15,15))
icone_placa = PhotoImage(width=20, height=20)
icone_placa.put(("gray",), to=(5,5,15,15))
icone_origem = PhotoImage(width=20, height=20)
icone_origem.put(("orange",), to=(5,5,15,15))
icone_destino = PhotoImage(width=20, height=20)
```

```
    icone_destino.put(("red",), to=(5,5,15,15))
    icone_km = PhotoImage(width=20, height=20)
    icone_km.put(("purple",), to=(5,5,15,15))

    # --- Aba Atendimento (com ícones) ---
    frame = Frame(aba_atendimento)
    frame.pack(pady=10)

    # Funcionário
    Label(frame, image=icone_func, text=" Funcionário:", compound="left").grid(row=0, column=0, sticky="w")
    combo_funcionario = ttk.Combobox(frame, state="readonly", width=50)
    combo_funcionario.grid(row=0, column=1, padx=5, pady=5)

    # Empresa
    Label(frame, image=icone_emp, text=" Empresa:", compound="left").grid(row=1, column=0, sticky="w")
    combo_empresa = ttk.Combobox(frame, state="readonly", width=50)
    combo_empresa.grid(row=1, column=1, padx=5, pady=5)
    Button(frame, text="Excluir Empresa", fg="red", command=excluir_empresa).grid(row=1, column=2, padx=5)

    # Placa
    Label(frame, image=icone_placa, text=" Placa do Veículo:", compound="left").grid(row=2, column=0, sticky="w")
    combo_placa = ttk.Combobox(frame, state="readonly", width=50)
    combo_placa.grid(row=2, column=1, padx=5, pady=5)
    Button(frame, text="Excluir Veículo", fg="red", command=excluir_veiculo).grid(row=2, column=2, padx=5)

    # Origem
```

```
    Label(frame, image=icone_origem, text=" Origem:", compound="left").grid(row=3, column=0, sticky="w")
    entry_origem = Entry(frame, width=50)
    entry_origem.grid(row=3, column=1, padx=5, pady=5)

    # Destino
    Label(frame, image=icone_destino, text=" Destino:", compound="left").grid(row=4, column=0, sticky="w")
    entry_destino = Entry(frame, width=50)
    entry_destino.grid(row=4, column=1, padx=5, pady=5)

    # Quilometragem
    Label(frame, image=icone_km, text=" Quilometragem:", compound="left").grid(row=5, column=0, sticky="w")
    entry_quilometragem = Entry(frame, width=20)
    entry_quilometragem.grid(row=5, column=1, padx=5, pady=5)

    # Descrição
    Label(frame, text="Descrição do Atendimento:").grid(row=6, column=0, sticky="w")
    entry_descricao = Entry(frame, width=60)
    entry_descricao.grid(row=6, column=1, padx=5, pady=5)

    Button(frame, text="Registrar Atendimento", command=registrar_atendimento).grid(row=7, column=1, pady=15)

    # --- Aba Empresa ---
    Label(aba_empresa, text="Nome:").pack()
    entry_nome_empresa = Entry(aba_empresa, width=50)
    entry_nome_empresa.pack()
    Label(aba_empresa, text="Telefone:").pack()
    entry_telefone_empresa = Entry(aba_empresa, width=50)
```

```
    entry_telefone_empresa.pack()
    Label(aba_empresa, text="Endereço:").pack()
    entry_endereco_empresa = Entry(aba_empresa, width=50)
    entry_endereco_empresa.pack()
    Button(aba_empresa, text="Cadastrar Empresa",
command=cadastrar_empresa).pack(pady=10)

    # --- Aba Veículo ---
    Label(aba_veiculo, text="Marca:").pack()
    entry_marca = Entry(aba_veiculo, width=40)
    entry_marca.pack()
    Label(aba_veiculo, text="Modelo:").pack()
    entry_modelo = Entry(aba_veiculo, width=40)
    entry_modelo.pack()
    Label(aba_veiculo, text="Placa:").pack()
    entry_placa = Entry(aba_veiculo, width=40)
    entry_placa.pack()
    Button(aba_veiculo, text="Cadastrar Veículo",
command=cadastrar_veiculo).pack(pady=10)

    # --- Aba Lista ---
    cols = ("ID", "Data", "Placa", "Veículo", "Empresa", "Funcionário", "Origem",
"Destino", "Descrição", "Km", "Preço")
    tree = ttk.Treeview(aba_lista, columns=cols, show='headings')
    for col in cols:
        tree.heading(col, text=col)
        tree.column(col, width=110)
    tree.pack(fill='both', expand=True)
    Button(aba_lista, text="Excluir Atendimento Selecionado",
command=excluir_atendimento).pack(pady=10)
```

```
    # Inicializações
    conn = sqlite3.connect('atendimento_veicular.db')
    cursor = conn.cursor()
    cursor.execute("SELECT nome FROM funcionarios")
    funcionarios = [f[0] for f in cursor.fetchall()]
    conn.close()
    combo_funcionario['values'] = funcionarios

    atualizar_combo_empresas()
    atualizar_combo_placas()
    atualizar_lista_atendimentos()

    # Manter referências dos ícones
    frame.icon_images = [icone_func, icone_emp, icone_placa, icone_origem,
icone_destino, icone_km]

    root.mainloop()

# ----------------------------
# LOGIN
# ----------------------------
criar_banco()
login_window = Tk()
login_window.title("Login - SisGEVEC")
login_window.geometry("400x300")

Label(login_window, text="E-mail:").pack(pady=5)
entry_email = Entry(login_window, width=40)
entry_email.pack()
Label(login_window, text="Senha:").pack(pady=5)
```

```
entry_senha = Entry(login_window, width=40, show="*")

entry_senha.pack()

Button(login_window, text="Login", width=20,
command=verificar_login).pack(pady=10)

Button(login_window, text="Esqueci a Senha", command=esqueci_senha,
fg="blue").pack()


login_window.mainloop()
```