UNIWERSYTET ROLNICZY
Im. Hugona Kołłątaja w Krakowie
WYDZIAŁ INŻYNIERII PRODUKCJI I ENERGETYKI

PROJEKT ZALICZENIOWY:

Aplikacja bazodanowa w Pythonie

PRZEDMIOT:

Inżynieria Oprogramowania

PROWADZĄCY:

Dr. Krzysztof Molenda, prof. URK

AUTORZY:

Krzysztof Ostrowski

Maciej Hodurek

Adam Solarz

Inżynieria Mechatroniczna, rok: 2, gr D

Kraków 2024

1. Działanie aplikacji.

Nasza aplikacja została zaprojektowana, aby umożliwić użytkownikowi wygodne otwieranie, przeglądanie, edytowanie i zarządzanie bazami danych w formacie SQLite. Dzięki interfejsowi graficznemu opartemu na bibliotece Tkinter, nasza aplikacja jest łatwa w obsłudze i przyjazna dla użytkownika.

Funkcje aplikacji:

- **Otwieranie i przeglądanie bazy danych:** Użytkownik może łatwo otwierać istniejące bazy danych SQLite i przeglądać ich zawartość, w tym tabele i rekordy.
- **Dodawanie i usuwanie rekordów:** Aplikacja umożliwia dodawanie nowych rekordów do tabel oraz usuwanie istniejących rekordów z bazy danych.
- **Eksport i import danych:** Użytkownik może eksportować dane z bazy danych do plików CSV oraz importować dane z plików CSV do bazy danych. Ten proces jest prosty i intuicyjny, co pozwala użytkownikowi na łatwe przenoszenie danych między aplikacją a innymi narzędziami.
- **Filtrowanie i sortowanie rekordów:** Nasza aplikacja umożliwia użytkownikowi filtrowanie danych na podstawie określonych kryteriów oraz sortowanie rekordów według wybranych kolumn. To pozwala użytkownikowi na szybkie odnalezienie potrzebnych informacji i przeglądanie danych w sposób uporządkowany.

Przyjazny interfejs użytkownika:

Interfejs graficzny naszej aplikacji został starannie zaprojektowany, aby zapewnić użytkownikowi przyjazne i intuicyjne doświadczenie. Elementy interfejsu są klarowne i łatwe w nawigacji, a wszelkie działania, takie jak dodawanie, usuwanie, eksportowanie i importowanie danych, są łatwo dostępne i wykonywalne z poziomu głównego okna aplikacji.

2. Opis struktury bazy danych

Nasza baza danych składa się z czterech głównych tabel: Klienci, Samochody, Sprzedawcy i Sprzedaże. Poniżej znajduje się opis struktury każdej z tych tabel:

Tabela Klienci:

- KlientID: Unikalny identyfikator klienta (klucz główny)
- Imie: Imię klienta (tekst, nie może być puste)
- Nazwisko: Nazwisko klienta (tekst, nie może być puste)
- Email: Adres e-mail klienta (tekst, nie może być puste)
- Telefon: Numer telefonu klienta (tekst, nie może być puste)

Tabela Samochody:

- SamochodID: Unikalny identyfikator samochodu (klucz główny)
- Marka: Marka samochodu (tekst, nie może być puste)
- Model: Model samochodu (tekst, nie może być puste)
- Rok: Rok produkcji samochodu (liczba całkowita, nie może być puste)
- Cena: Cena samochodu (liczba zmiennoprzecinkowa, nie może być puste)

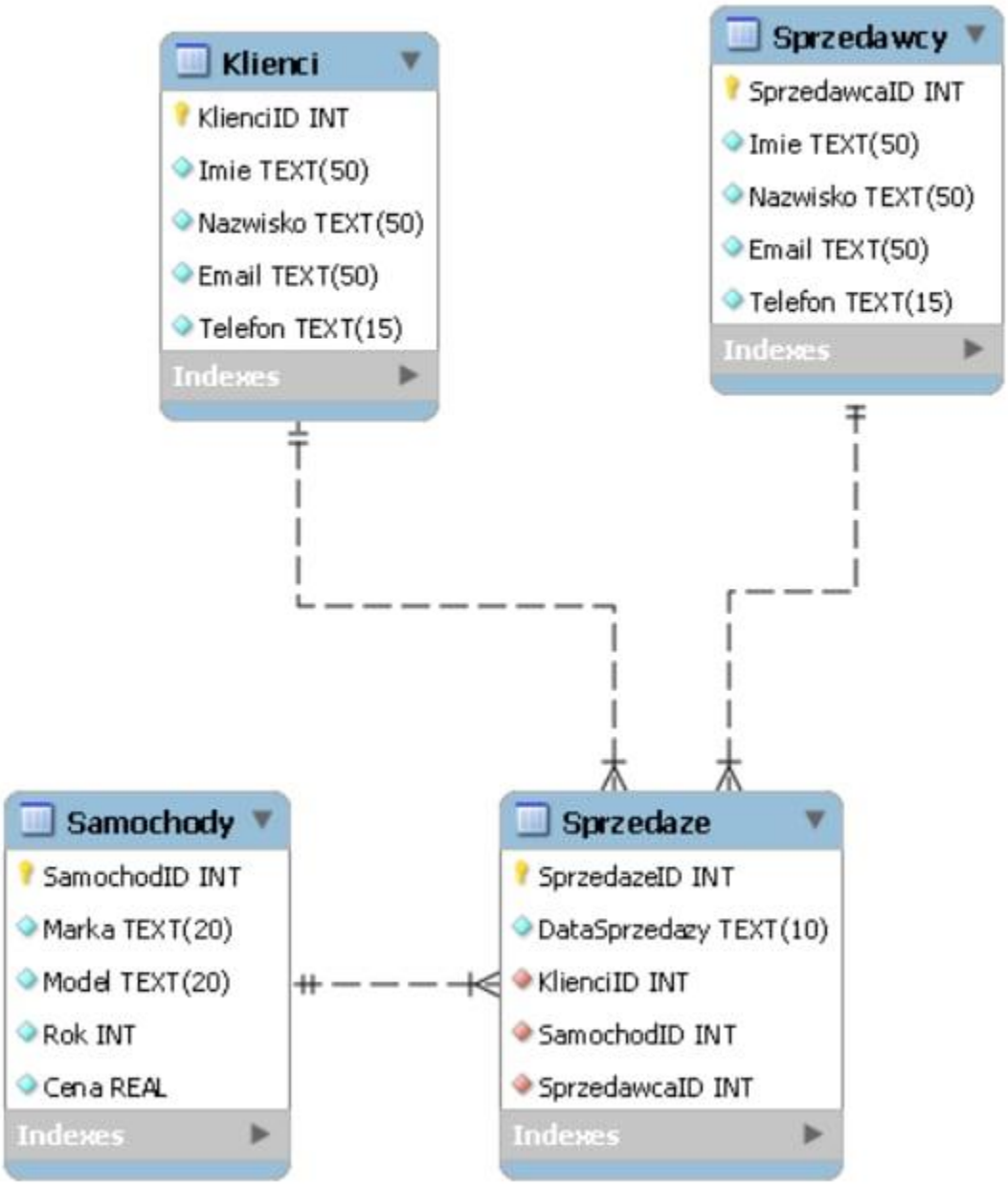
Tabela Sprzedawcy:

- SprzedawcaID: Unikalny identyfikator sprzedawcy (klucz główny)
- Imie: Imię sprzedawcy (tekst, nie może być puste)
- Nazwisko: Nazwisko sprzedawcy (tekst, nie może być puste)
- Email: Adres e-mail sprzedawcy (tekst, nie może być puste)
- Telefon: Numer telefonu sprzedawcy (tekst, nie może być puste)

Tabela Sprzedaże:

- SprzedazID: Unikalny identyfikator sprzedaży (klucz główny)
- KlientID: ID klienta (klucz obcy z tabeli Klienci)
- SamochodID: ID samochodu (klucz obcy z tabeli Samochody)
- SprzedawcaID: ID sprzedawcy (klucz obcy z tabeli Sprzedawcy)
- DataSprzedazy: Data sprzedaży samochodu (tekst, nie może być puste)

3. Diagram ERD



4. Kod SQL tworzący bazę danych.

```sql
-- Tworzenie tabeli Klienci
CREATE TABLE Klienci (
    KlientID INTEGER PRIMARY KEY,
    Imie TEXT NOT NULL,
    Nazwisko TEXT NOT NULL,
    Email TEXT NOT NULL,
    Telefon TEXT NOT NULL
);
CREATE TABLE Samochody (
    SamochodID INTEGER PRIMARY KEY,
    Marka TEXT NOT NULL,
    Model TEXT NOT NULL,
    Rok INTEGER NOT NULL,
    Cena REAL NOT NULL
);
CREATE TABLE Sprzedawcy (
    SprzedawcaID INTEGER PRIMARY KEY,
    Imie TEXT NOT NULL,
    Nazwisko TEXT NOT NULL,
    Email TEXT NOT NULL,
    Telefon TEXT NOT NULL
);
CREATE TABLE Sprzedaze (
    SprzedazID INTEGER PRIMARY KEY,
    KlientID INTEGER NOT NULL,
    SamochodID INTEGER NOT NULL,
    SprzedawcaID INTEGER NOT NULL,
    DataSprzedazy TEXT NOT NULL,
    FOREIGN KEY (KlientID) REFERENCES Klienci(KlientID),
    FOREIGN KEY (SamochodID) REFERENCES Samochody(SamochodID),
    FOREIGN KEY (SprzedawcaID) REFERENCES Sprzedawcy(SprzedawcaID)
);
```

5. Kod SQL wypełniający bazę danych przykładowymi danymi.

```sql
INSERT INTO Klienci (Imie, Nazwisko, Email, Telefon) VALUES
('Jan', 'Kowalski', 'jan.kowalski@interia.pl', '123456789'),
('Anna', 'Nowak', 'anna.nowak@interia.pl', '987654321'),
('Piotr', 'Wiśniewski', 'piotr.wisniewski@onet.pl', '555123456'),
('Maria', 'Wójcik', 'maria.wojcik@gmail.com', '444567890'),
('Krzysztof', 'Kowalczyk', 'krzysztof.kowalczyk@gmail.com', '333789012');
INSERT INTO Samochody (Marka, Model, Rok, Cena) VALUES
('Toyota', 'Camry', 2020, 24000),
('Honda', 'Civic', 2019, 20000),
('Ford', 'Mustang', 2021, 30000),
('Chevrolet', 'Malibu', 2018, 18000),
('Nissan', 'Altima', 2022, 25000);
INSERT INTO Sprzedawcy (Imie, Nazwisko, Email, Telefon) VALUES
('Alicja', 'Jankowska', 'alicja.jankowska@sklep.com', '123123123'),
('Bartłomiej', 'Wiśniewski', 'bartlomiej.wisniewski@sklep.com', '321321321'),
('Karolina', 'Kowalska', 'karolina.kowalska@sklep.com', '456456456'),
('Dawid', 'Nowak', 'dawid.nowak@sklep.com', '654654654'),
('Ewa', 'Kamińska', 'ewa.kaminska@sklep.com', '789789789');
INSERT INTO Sprzedaze (KlientID, SamochodID, SprzedawcaID, DataSprzedazy) VALUES
(1, 3, 2, '2023-01-15'),
(2, 1, 1, '2023-02-20'),
(3, 5, 3, '2023-03-25'),
(4, 2, 4, '2023-04-30'),
(5, 4, 5, '2023-05-05');
```

6. Kod źródłowy aplikacji, wraz z komentarzami.

Plik main.py:

```
import tkinter as tk
from gui import App
import database

if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
    root.mainloop()
```

Plik gui.py

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import sqlite3
import csv
from database import add_record, get_records, update_record, delete_record,
get_sorted_records, get_filtered_records, get_joined_records, get_column_names

class App:
    def __init__(self, root):   #Tworzenie okienka oraz konfigurowanie styli przyciskow,
tabel oraz pól.
        self.root = root
        self.root.title("Baza Danych Sklepu Samochodowego")
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.style = ttk.Style()
        self.style.theme_use('clam')
        self.style.configure("TFrame", background="#ececec")
        self.style.configure("TLabel", background="#ececec", font=('Arial', 12))
        self.style.configure("TButton", font=('Arial', 12), padding=5)
        self.style.configure("TEntry", font=('Arial', 12))
        self.style.configure("Treeview.Heading", font=('Arial', 12, 'bold'))

        self.db_path = filedialog.askopenfilename(title="Wybierz plik bazy danych",
filetypes=[("SQLite files", "*.sqlite *.db")]) #Wybór pliku bazy danych
        if not self.db_path:
            messagebox.showerror("Błąd", "Musisz wybrać plik bazy danych, aby
kontynuować.")
            root.destroy()
            return

        self.conn = sqlite3.connect(self.db_path)
        self.create_widgets()
        self.configure_grid()
        self.open_add_record_window()
```

```
    def on_closing(self):
        self.root.destroy()

    def create_widgets(self): #Tworzenie widgetow
        self.frame = ttk.Frame(self.root, padding="10")
        self.frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.left_frame = ttk.Frame(self.frame)
        self.left_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.right_frame = ttk.Frame(self.frame)
        self.right_frame.grid(row=0, column=1, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.table_label = ttk.Label(self.right_frame, text="Wybierz tabelę")
        self.table_label.grid(row=0, column=0, columnspan=2, pady=(0, 10))

        self.table_name = tk.StringVar()
        self.table_menu = ttk.OptionMenu(self.right_frame, self.table_name, '',
*self.get_tables(), command=self.load_records)
        self.table_menu.grid(row=1, column=0, columnspan=2, padx=5, pady=5, sticky="ew")

        self.records_label = ttk.Label(self.left_frame, text="Rekordy:")
        self.records_label.grid(row=0, column=0, pady=(0, 10))

        self.records_tree = ttk.Treeview(self.left_frame, show='headings')
        self.records_tree.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.records_tree.bind('<Double-1>', self.on_edit_record)

        self.add_button = ttk.Button(self.right_frame, text="Dodaj",
command=self.open_add_record_window)
        self.add_button.grid(row=2, column=0, columnspan=2, padx=5, pady=5, sticky="ew")

        self.delete_button = ttk.Button(self.right_frame, text="Usuń",
command=self.delete_record)
        self.delete_button.grid(row=3, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

        self.export_button = ttk.Button(self.right_frame, text="Eksportuj do CSV",
command=self.export_to_csv)
        self.export_button.grid(row=4, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

        self.import_button = ttk.Button(self.right_frame, text="Importuj z CSV",
command=self.import_from_csv)
        self.import_button.grid(row=5, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

        self.sort_column = tk.StringVar(value="rowid")
```

```
        self.sort_order = tk.StringVar(value="ASC")

        self.sort_column_menu = ttk.OptionMenu(self.right_frame, self.sort_column,
"rowid")
        self.sort_column_menu.grid(row=6, column=0, padx=5, pady=5, sticky="ew")

        self.sort_order_menu = ttk.OptionMenu(self.right_frame, self.sort_order, "ASC",
"ASC", "DESC")
        self.sort_order_menu.grid(row=6, column=1, padx=5, pady=5, sticky="ew")

        self.sort_button = ttk.Button(self.right_frame, text="Sortuj",
command=self.sort_records)
        self.sort_button.grid(row=7, column=0, columnspan=2, padx=5, pady=5, sticky="ew")

        self.filter_column = tk.StringVar(value="")
        self.filter_value = tk.StringVar()

        self.filter_column_menu = ttk.OptionMenu(self.right_frame, self.filter_column, "")
        self.filter_column_menu.grid(row=8, column=0, padx=5, pady=5, sticky="ew")

        self.filter_entry = ttk.Entry(self.right_frame, textvariable=self.filter_value)
        self.filter_entry.grid(row=8, column=1, padx=5, pady=5, sticky="ew")
        self.filter_entry.bind("<KeyRelease>", lambda event: self.filter_records())

        self.filter_button = ttk.Button(self.right_frame, text="Filtruj",
command=self.filter_records)
        self.filter_button.grid(row=9, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

        self.joined_records_button = ttk.Button(self.right_frame, text="Pokaż połączone
rekordy", command=self.show_joined_records)
        self.joined_records_button.grid(row=10, column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

    def open_add_record_window(self): #Okienko do dodawania rekordow
        table = self.table_name.get()
        if not table:
            messagebox.showerror("Powiadomienie", "Najpierw wybierz tabelę.")
            return
        columns = get_column_names(self.db_path, table)[1:]  # Skip the ID column

        add_window = tk.Toplevel(self.root)
        add_window.title("Dodaj rekord")
        add_window.configure(background="#ececec")

        entries = []
        for idx, column in enumerate(columns):
            label = ttk.Label(add_window, text=column)
            label.grid(row=idx, column=0, padx=5, pady=5, sticky="ew")
            entry = ttk.Entry(add_window)
```

```
            entry.grid(row=idx, column=1, padx=5, pady=5, sticky="ew")
            entries.append(entry)

        def add_record_to_db():
            data = [entry.get() for entry in entries]
            add_record(self.db_path, table, columns, data)
            self.load_records()
            add_window.destroy()

        save_button = ttk.Button(add_window, text="Dodaj", command=add_record_to_db)
        save_button.grid(row=len(columns), column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

    def load_records(self, *args): #Pobieranie rekordow z tabel
        table = self.table_name.get()
        if not table:
            return

        self.records_tree.delete(*self.records_tree.get_children())

        columns = get_column_names(self.db_path, table)
        self.records_tree["columns"] = columns
        for col in columns:
            self.records_tree.heading(col, text=col)
            self.records_tree.column(col, stretch=tk.YES)

        records = get_records(self.db_path, table)
        for record in records:
            self.records_tree.insert("", tk.END, values=record)

        self.sort_column_menu.set_menu(*columns)
        self.filter_column_menu.set_menu(*columns)

    def delete_record(self): #Usuwanie rekordow
        selected_item = self.records_tree.selection()[0]
        record_id = self.records_tree.item(selected_item)['values'][0]
        table = self.table_name.get()
        delete_record(self.db_path, table, record_id)
        self.load_records()

    def on_edit_record(self, event): #Edytowanie rekordow
        selected_item = self.records_tree.selection()[0]
        record_id = self.records_tree.item(selected_item)['values'][0]
        table = self.table_name.get()

        edit_window = tk.Toplevel(self.root)
        edit_window.title("Edytuj rekord")
        edit_window.configure(background="#ececec")

        columns = get_column_names(self.db_path, table)[1:]
```

```
        entries = []

        for idx, column in enumerate(columns):
            label = ttk.Label(edit_window, text=column)
            label.grid(row=idx, column=0, padx=5, pady=5, sticky="ew")
            entry = ttk.Entry(edit_window)
            entry.grid(row=idx, column=1, padx=5, pady=5, sticky="ew")
            entries.append(entry)

        record = get_records(self.db_path, table)
        record_data = [r for r in record if r[0] == record_id][0][1:]

        for entry, data in zip(entries, record_data):
            entry.insert(0, data)

        def save_edit(): #Zapisywanie edycji
            new_data = [entry.get() for entry in entries]
            update_record(self.db_path, table, record_id, columns, new_data)
            self.load_records()
            edit_window.destroy()

        save_button = ttk.Button(edit_window, text="Zapisz", command=save_edit)
        save_button.grid(row=len(columns), column=0, columnspan=2, padx=5, pady=5,
sticky="ew")

    def sort_records(self): #Sortowanie rekordow
        table = self.table_name.get()
        column = self.sort_column.get()
        order = self.sort_order.get()
        records = get_sorted_records(self.db_path, table, column, order)

        self.records_tree.delete(*self.records_tree.get_children())
        for record in records:
            self.records_tree.insert("", tk.END, values=record)

    def filter_records(self): #Filtrowanie rekordow
        table = self.table_name.get()
        column = self.filter_column.get()
        value = self.filter_value.get()
        records = get_filtered_records(self.db_path, table, column, value)

        self.records_tree.delete(*self.records_tree.get_children())
        for record in records:
            self.records_tree.insert("", tk.END, values=record)

    def show_joined_records(self): #Pokazuje polaczone rekordy
        records = get_joined_records(self.db_path)
        self.records_tree.delete(*self.records_tree.get_children())
        self.records_tree["columns"] = ("ID", "Imię klienta", "Nazwisko klienta", "Marka
samochodu", "Model samochodu", "Imię sprzedawcy", "Nazwisko sprzedawcy", "Data sprzedaży")
```

```
        for col in self.records_tree["columns"]:
            self.records_tree.heading(col, text=col)
            self.records_tree.column(col, stretch=tk.YES)

        for record in records:
            self.records_tree.insert("", tk.END, values=record)

    def export_to_csv(self): #Eksportowanie do CSV
        table = self.table_name.get()
        if not table:
            messagebox.showerror("Błąd", "Najpierw wybierz tabelę.")
            return

        file_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV
files", "*.csv")])
        if not file_path:
            return

        records = get_records(self.db_path, table)
        columns = get_column_names(self.db_path, table)

        with open(file_path, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(columns)
            writer.writerows(records)

    def import_from_csv(self): #Importowanie z CSV
        table = self.table_name.get()
        if not table:
            messagebox.showerror("Błąd", "Najpierw wybierz tabelę.")
            return

        file_path = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
        if not file_path:
            return

        with open(file_path, mode='r', newline='') as file:
            reader = csv.reader(file)
            columns = next(reader)
            for row in reader:
                add_record(self.db_path, table, columns[1:], row[1:])  # Skip ID column

        self.load_records()

    def get_tables(self): #Pokazywanie tabel
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
        tables = [row[0] for row in cursor.fetchall()]
        conn.close()
```

```
        return tables

    def configure_grid(self): #Konfiguracja siatki ułożenia poszczególnych przycisków,
rekordów itd...
        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)
        self.frame.columnconfigure(0, weight=1)
        self.frame.columnconfigure(1, weight=1)
        self.frame.rowconfigure(0, weight=1)
        self.left_frame.columnconfigure(0, weight=1)
        self.left_frame.rowconfigure(1, weight=1)
        self.right_frame.columnconfigure(0, weight=1)
        self.right_frame.columnconfigure(1, weight=1)

def main():
    root = tk.Tk()
    app = App(root)
    root.mainloop()

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

Plik database.py

```
import sqlite3

def connect_db(db_path): #Polaczenie z baza danych
    return sqlite3.connect(db_path)

def get_column_names(db_path, table): #Pobranie nazw kolumn
    conn = connect_db(db_path)
    cursor = conn.cursor()
    cursor.execute(f"PRAGMA table_info({table})")
    columns = [row[1] for row in cursor.fetchall()]
    conn.close()
    return columns

def add_record(db_path, table, columns, data): #DOdanie rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    column_names = ', '.join(columns)
    placeholders = ', '.join(['?' for _ in columns])
    cursor.execute(f"INSERT INTO {table} ({column_names}) VALUES ({placeholders})", data)
    conn.commit()
    conn.close()

def get_records(db_path, table): #Pobranie rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM {table}")
    records = cursor.fetchall()
```

```
    conn.close()
    return records

def update_record(db_path, table, record_id, columns, data): #Edytowanie rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    set_clause = ', '.join([f"{col} = ?" for col in columns])
    cursor.execute(f"UPDATE {table} SET {set_clause} WHERE rowid = ?", (*data, record_id))
    conn.commit()
    conn.close()

def delete_record(db_path, table, record_id): #Usuniecie rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    cursor.execute(f"DELETE FROM {table} WHERE rowid = ?", (record_id,))
    conn.commit()
    conn.close()

def get_sorted_records(db_path, table, column, order): #Sortowanie rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM {table} ORDER BY {column} {order}")
    records = cursor.fetchall()
    conn.close()
    return records

def get_filtered_records(db_path, table, filter_column, filter_value): #Filtrowanie
rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM {table} WHERE {filter_column} LIKE ?",
(f'%{filter_value}%',))
    records = cursor.fetchall()
    conn.close()
    return records

def get_joined_records(db_path): #Pokazanie polaczonych rekordow
    conn = connect_db(db_path)
    cursor = conn.cursor()
    cursor.execute('''SELECT Sprzedaze.rowid, Klienci.Imie, Klienci.Nazwisko,
Samochody.Marka, Samochody.Model, Sprzedawcy.Imie, Sprzedawcy.Nazwisko,
Sprzedaze.DataSprzedazy
                      FROM Sprzedaze
                      JOIN Klienci ON Sprzedaze.KlientID = Klienci.rowid
                      JOIN Samochody ON Sprzedaze.SamochodID = Samochody.rowid
                      JOIN Sprzedawcy ON Sprzedaze.SprzedawcaID = Sprzedawcy.rowid''')
    records = cursor.fetchall()
    conn.close()
    return records
```

7. Testy jednostkowe wybranych fragmentów kodu.

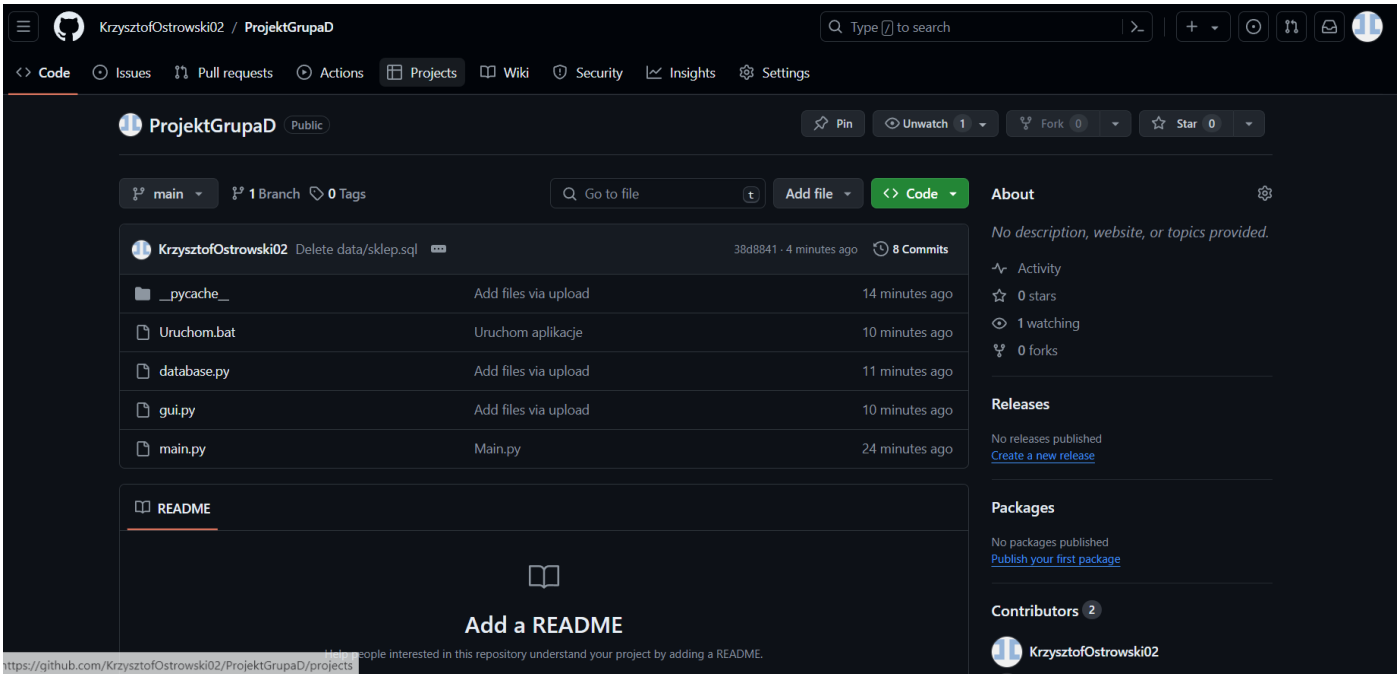
```
import unittest
import sqlite3
from database import connect_db, get_column_names, add_record, get_records, update_record,
delete_record
class TestDatabaseFunctions(unittest.TestCase):
    def setUp(self):
        # Utworzenie tymczasowej bazy danych w pamięci
        self.conn = connect_db()
        self.cursor = self.conn.cursor()
        self.cursor.execute("CREATE TABLE Test (ID INTEGER PRIMARY KEY, Name TEXT)")
    def tearDown(self):
        # Usunięcie tymczasowej bazy danych
        self.conn.close()

    def test_get_column_names(self):
        expected_columns = ["ID", "Name"]
        columns = get_column_names("Test")
        self.assertEqual(columns, expected_columns)

    def test_add_record(self):
        add_record("Test", ["Name"], ["Test Record"])
        self.cursor.execute("SELECT * FROM Test")
        records = self.cursor.fetchall()
        self.assertEqual(len(records), 1)

    def test_get_records(self):
        self.cursor.execute("INSERT INTO Test (Name) VALUES ('Record 1')")
        self.cursor.execute("INSERT INTO Test (Name) VALUES ('Record 2')")
        records = get_records("Test")
        self.assertEqual(len(records), 2)

    def test_update_record(self):
        self.cursor.execute("INSERT INTO Test (Name) VALUES ('Old Record')")
        self.conn.commit()
        update_record("Test", 1, ["Name"], ["New Record"])
        self.cursor.execute("SELECT Name FROM Test WHERE ID=1")
        record = self.cursor.fetchone()
        self.assertEqual(record[0], "New Record")

    def test_delete_record(self):
        self.cursor.execute("INSERT INTO Test (Name) VALUES ('Record to delete')")
        self.conn.commit()
        delete_record("Test", 1)
        self.cursor.execute("SELECT * FROM Test")
        records = self.cursor.fetchall()
        self.assertEqual(len(records), 0)
if __name__ == '__main__':
    unittest.main()
```

8. Instrukcja instalacji i uruchomienia aplikacji oraz jej użytkowania.

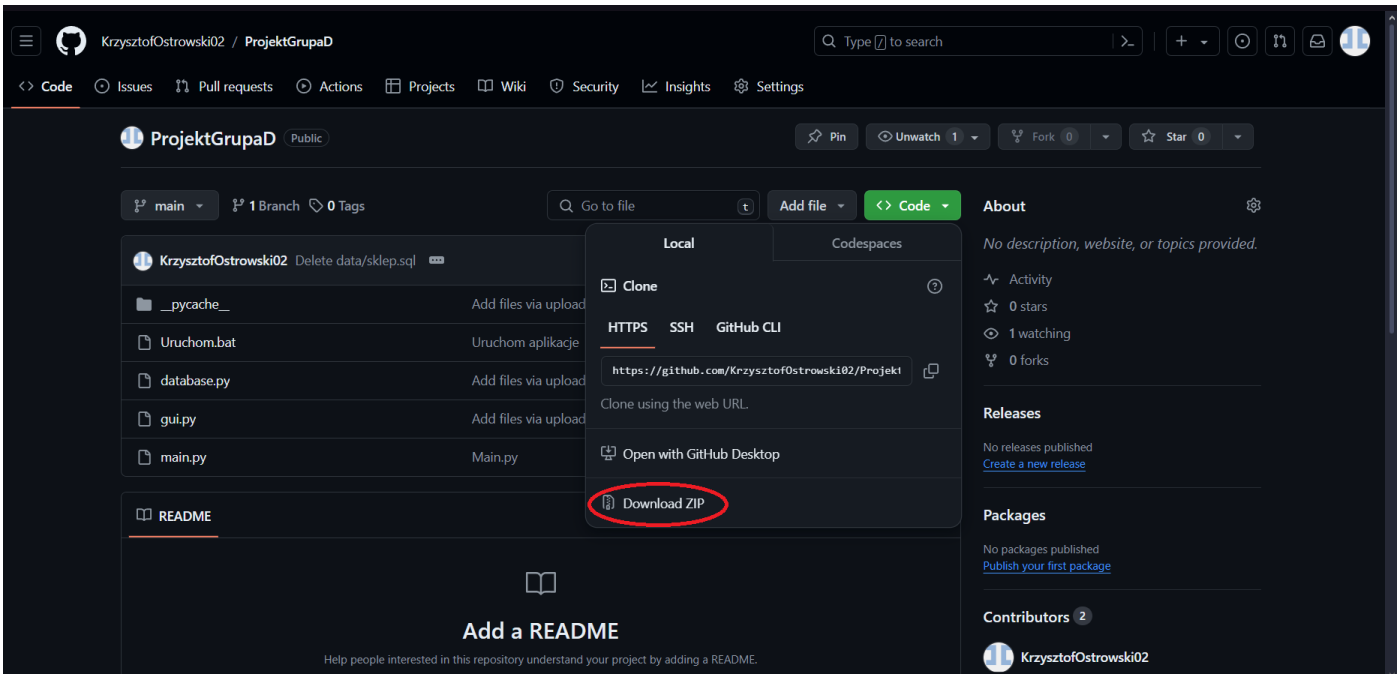
Instalacja aplikacji:

Metoda 1:

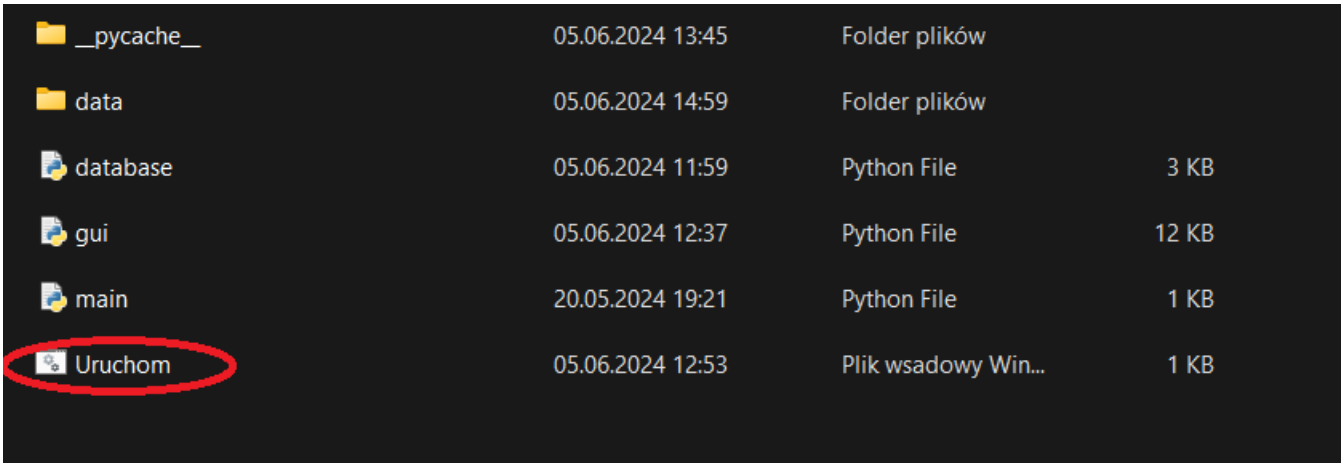
1. Należy zainstalować Python ze strony: https://www.python.org/downloads/
2. Wejść na stronę githuba projektu: https://github.com/KrzysztofOstrowski02/ProjektGrupaD



3. Należy rozwinąć zielony przycisk Code i przycisnąć Download ZIP



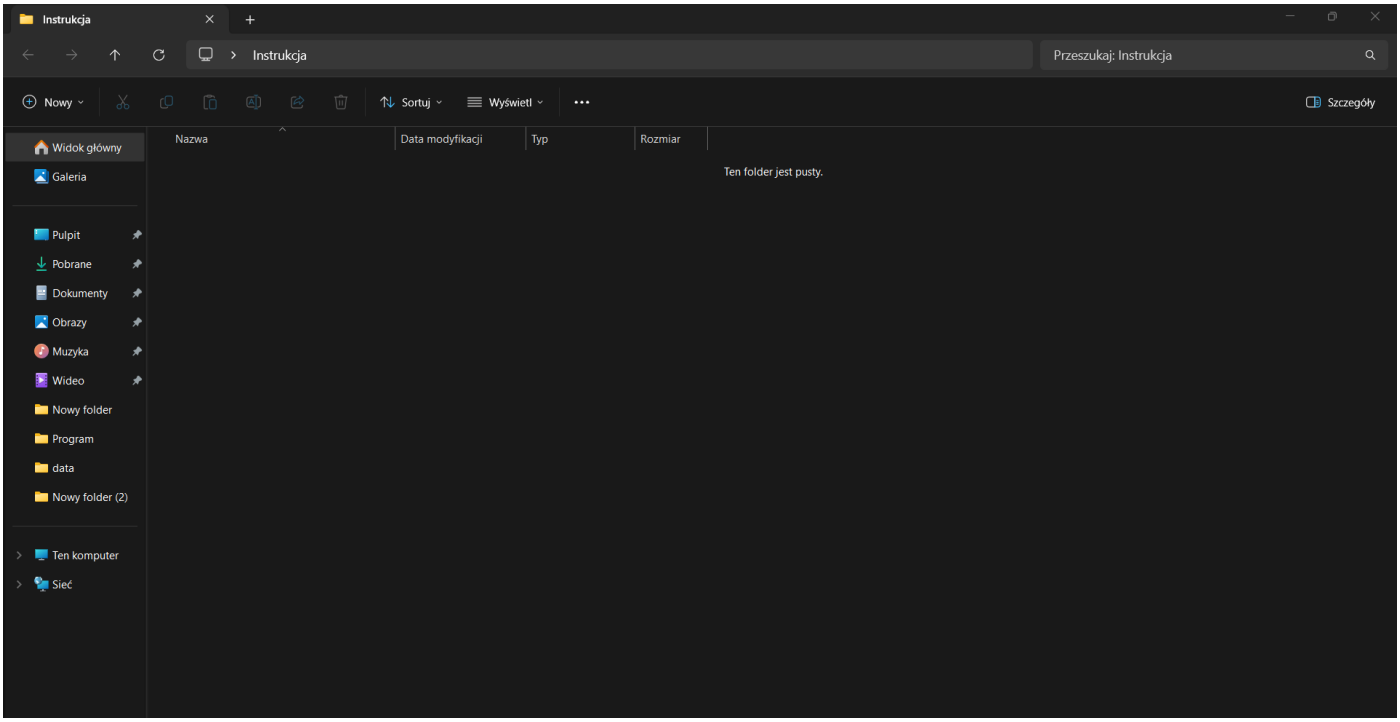
4. Po pobraniu pliku rozpakowaniu mamy pliki potrzebne do uruchomienia aplikacji. W tym celu uruchamiamy plik o nazwe ”Uruchom” o rozszerzeniu .bat



Metoda 2:

1. Należy zainstalować Python ze strony: https://www.python.org/downloads/ oraz GITa ze strony https://www.git-scm.com/downloads

2. Tworzymy folder, w którym ma być umiejscowiony program.



3. W miejscu, w którym jest ścieżka naszego folderu (Tam, gdzie na ostatnim screenshocie napisane jest" Instrukcja") wpisujemy cmd i zatwierdzamy enterem.

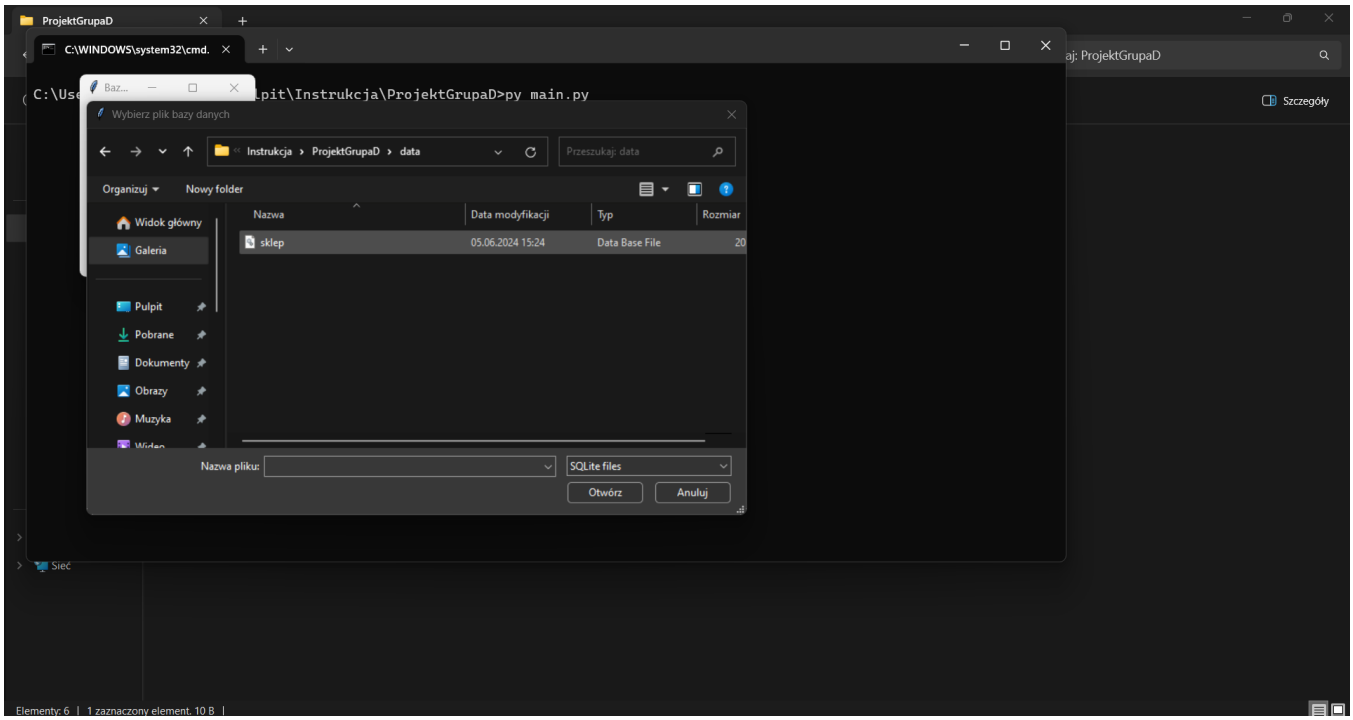
4. Do wiersza poleceń wpisujemy: git clone https:/github.com/KrzysztofOstrowski02/ProjektGrupaD i zatwierdzamy enterem.

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22631.3593]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\aquli\OneDrive\Pulpit\Instrukcja>git clone KrzysztofOstrowski02/ProjektGrupaD
fatal: repository 'KrzysztofOstrowski02/ProjektGrupaD' does not exist

C:\Users\aquli\OneDrive\Pulpit\Instrukcja>git clone https://github.com/KrzysztofOstrowski02/ProjektGrupaD.git
Cloning into 'ProjektGrupaD'...
remote: Enumerating objects: 29, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 29 (delta 4), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (29/29), 23.14 KiB | 816.00 KiB/s, done.
Resolving deltas: 100% (4/4), done.

C:\Users\aquli\OneDrive\Pulpit\Instrukcja>
```

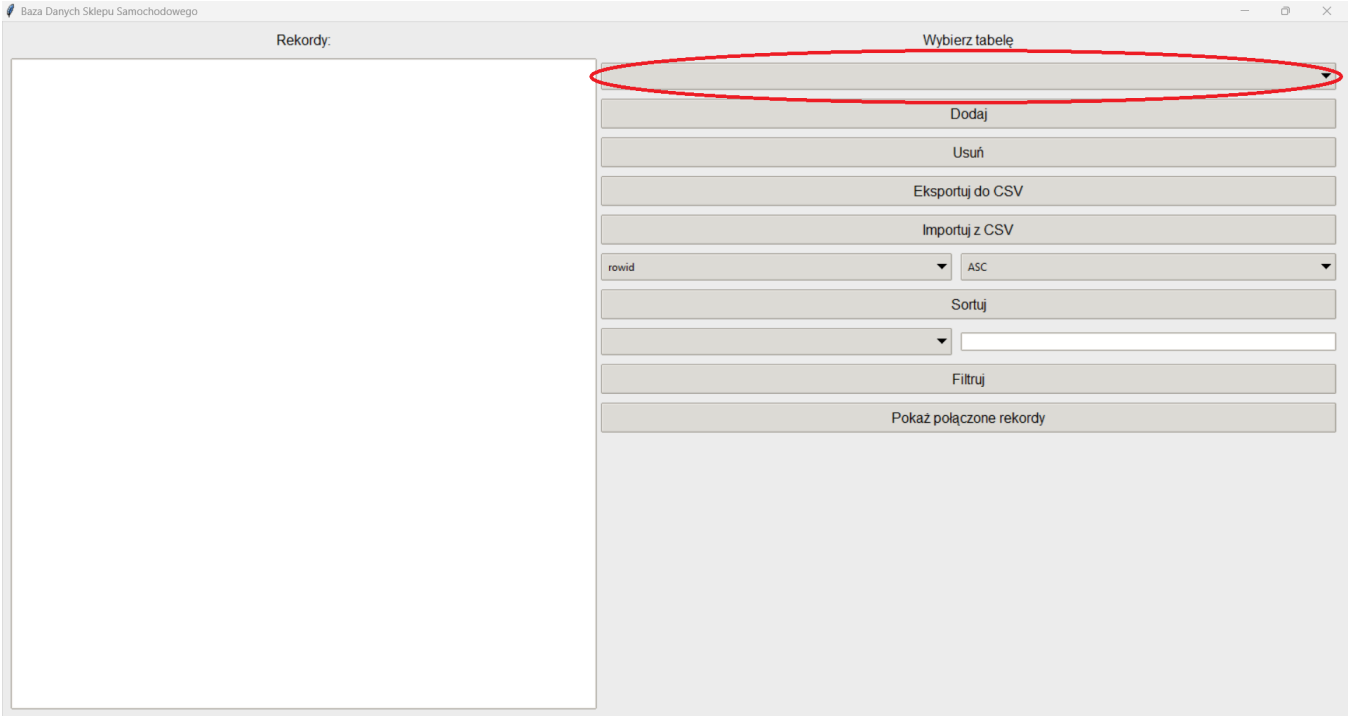
5. Po zakończeniu procesu należy wyłączyć terminal i otworzyć powstały plik ProjektGrupaD i jak w metodzie 1 otworzyć plik uruchom.

Instrukcja użytkowania aplikacji:

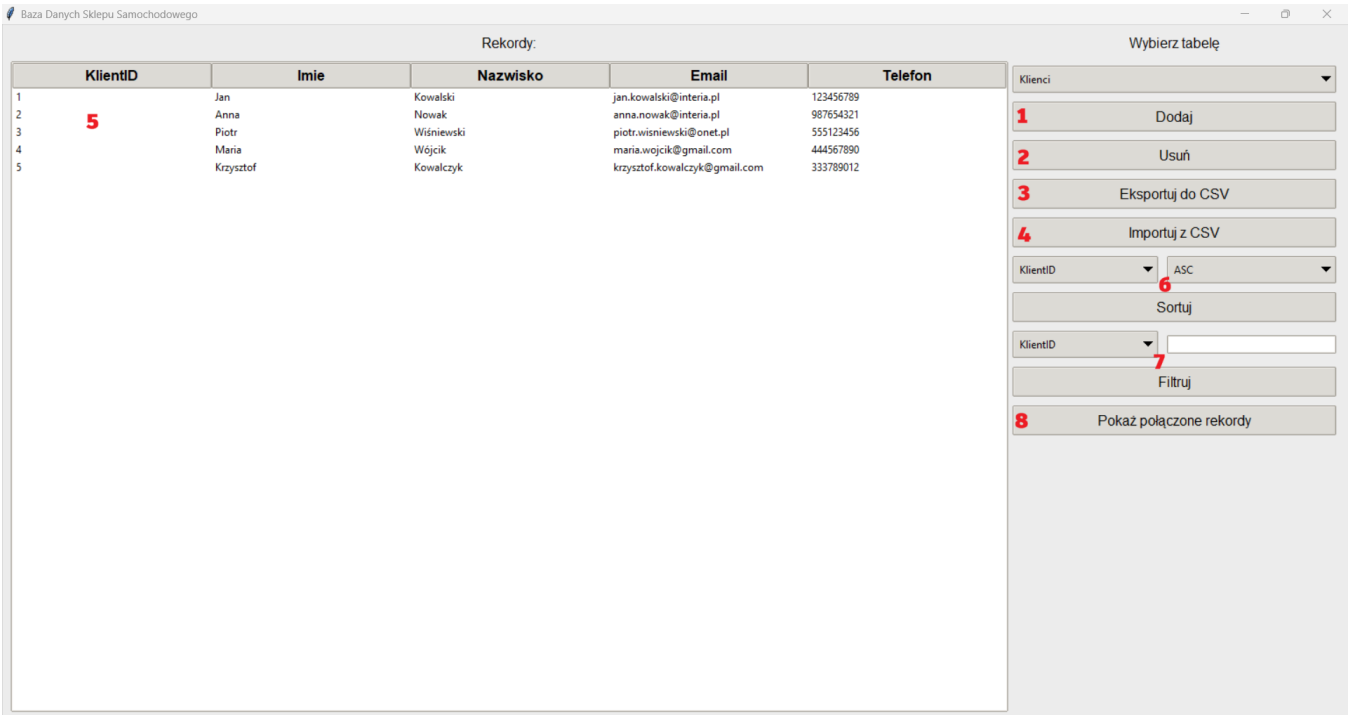
1. Po uruchomieniu aplikacji wyskoczy nam okienko z wyborem bazy danych. W tym celu otwieramy folder data i otwieramy plik sklep.db



2. Po uruchomieniu aplikacji należy wybrać tabele która zamierzamy zobaczyć lub edytować



3. Po wybraniu tabeli możemy w pełni edytować tabele klikając na wybrane podpisane przyciski.



1 – Przycisk służący do dodawania rekordów do naszej tabeli. Po wciśnięciu przycisku wyskoczy nam okienko służące do dodania rekordu.

2 – Przycisk służący do usuwania rekordów z naszej tabeli. By zadziałał należy wpierw zaznaczyć rekord, który chcemy usunąć.

3 – Przycisk służący do eksportu rekordów do pliku CSV. Po wciśnięciu należy wybrać w okienku, gdzie plik CSV ma zostać utworzony.

4 – Przycisk służący do importowania rekordów z pliku CSV. Po wciśnięciu należy wybrać plik CSV z którego rekordy mają zostać pobrane.

5 – By edytować rekord należy dwukrotnie go przycisnąć co spowoduje ukazanie się okienka edycji rekordu.

6 - Panel służący do sortowania rekordów. Należy wybrać kolumnę, po której ma się odbyć sortowanie oraz czy ma odbywać się ono malejąco i rosnąco.

7 – Panel służący do filtrowania rekordu. Należy wybrać kolumnę, którą chcemy przefiltrować i wpisać wartość lub słowo, według którego chcemy filtrować.
8 – Przycisk otwierający okno z tabelą posiadająca połączone rekordy z rekordami które są ze sobą połączone.

9. Wygenerowana dokumentacja kodu.

index
main c:\users\aquli\onedrive\pulpit\program\src\main.py

Modules

database tkinter

index
gui c:\users\aquli\onedrive\pulpit\program\src\gui.py

Modules

csv tkinter.messagebox tkinter
tkinter.filedialog sqlite3 tkinter.ttk

Classes

builtins.object

App

class **App**(builtins.object)
App(root)

Methods defined here:

__init__(self, root)
Initialize self. See help(type(self)) for accurate signature.

configure_grid(self)

create_widgets(self)

delete_record(self)

export_to_csv(self)

filter_records(self)

get_tables(self)

import_from_csv(self)

load_records(self, *args)

on_closing(self)

on_edit_record(self, event)

open_add_record_window(self)

show_joined_records(self)

sort_records(self)

Data descriptors defined here:

__dict__
dictionary for instance variables

__weakref__
list of weak references to the object

Functions

main()

index
database c:\users\aquli\onedrive\pulpit\program\src\database.py

Modules

sqlite3

Functions

add_record(db_path, table, columns, data)

connect_db(db_path)

delete_record(db_path, table, record_id)

get_column_names(db_path, table)

get_filtered_records(db_path, table, filter_column, filter_value)

get_joined_records(db_path)

get_records(db_path, table)

get_sorted_records(db_path, table, column, order)

update_record(db_path, table, record_id, columns, data)