Projekt zaliczeniowy – Języki skryptowe (Python)

Tytuł projektu: Budżet osobisty z analizą wydatków.

Autorzy: Bartłomiej Konopka, Jakub Kowalik
Grupa: 2ID12B
Data oddania: 17 czerwca 2025

1. Opis projektu

Cel projektu

Celem projektu było stworzenie funkcjonalnej aplikacji służącej do zarządzania osobistym budżetem, z obsługą wielu użytkowników, możliwością dodawania i analizowania transakcji oraz graficznym przedstawieniem salda i wydatków.

Funkcje aplikacji

- Rejestracja i logowanie użytkownika
- Dodawanie, edytowanie i usuwanie transakcji
- Obliczanie salda miesięcznego i sum wg kategorii
- Zapisywanie i odczytywanie danych z plików CSV
- Tworzenie raportów tekstowych
- Wizualizacja danych za pomocą wykresów w GUI

Zakres funkcjonalny

- Obsługa danych użytkowników (haszowanie haseł)
- Praca na plikach lokalnych (CSV, TXT)
- Wykorzystanie bibliotek graficznych (tkinter, matplotlib)
- Zastosowanie testów jednostkowych (unittest)

2. Struktura projektu

Foldery i pliki

- /data/ – dane CSV użytkowników
- /tests/ – testy jednostkowe

- gui.py – główny interfejs użytkownika
- auth.py – rejestracja i logowanie
- budget.py – operacje na budżecie
- logic.py – definicja transakcji
- charts.py – rysowanie wykresów
- utils.py – zapis raportów
- README.md, raport.pdf, raport.txt – dokumentacja i wygenerowane raporty

Omówienie klas

- Transaction – pojedyncza transakcja
- Budget – lista transakcji i metody analityczne
- BudgetApp – główna klasa GUI
- auth.* – rejestracja i logowanie użytkownika
- data.* – zapis i odczyt CSV
- utils.* – generowanie raportów tekstowych

3. Technologie i biblioteki

- **Python 3.11**
- tkinter – interfejs graficzny
- csv, os, hashlib – operacje systemowe i plikowe
- datetime – obsługa dat
- matplotlib – wykresy słupkowe i poziome
- unittest – testowanie
- sha256 – bezpieczne haszowanie haseł

4. Sposób działania programu

Instrukcja uruchomienia

1. Instalacja zależności:

pip install matplotlib

2. Uruchomienie:

python gui.py

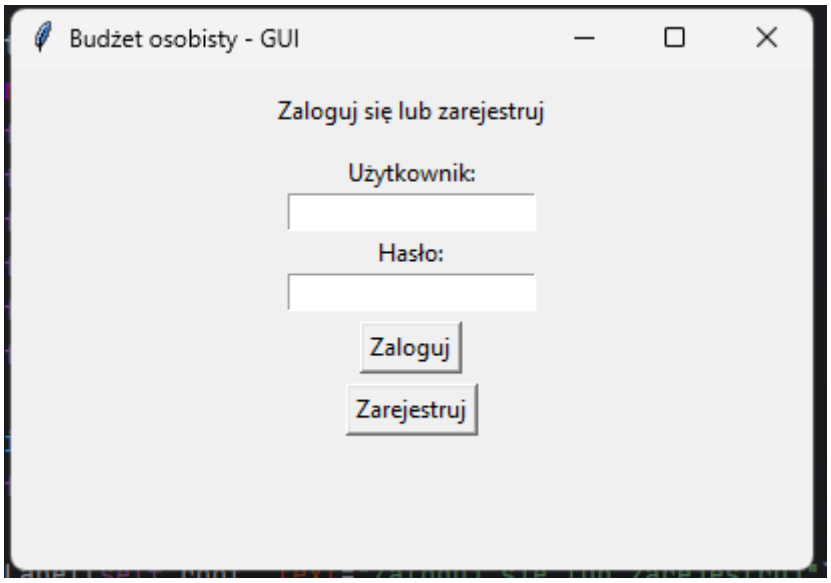
Dane wejściowe

- Kwota, kategoria, data i opcjonalny opis

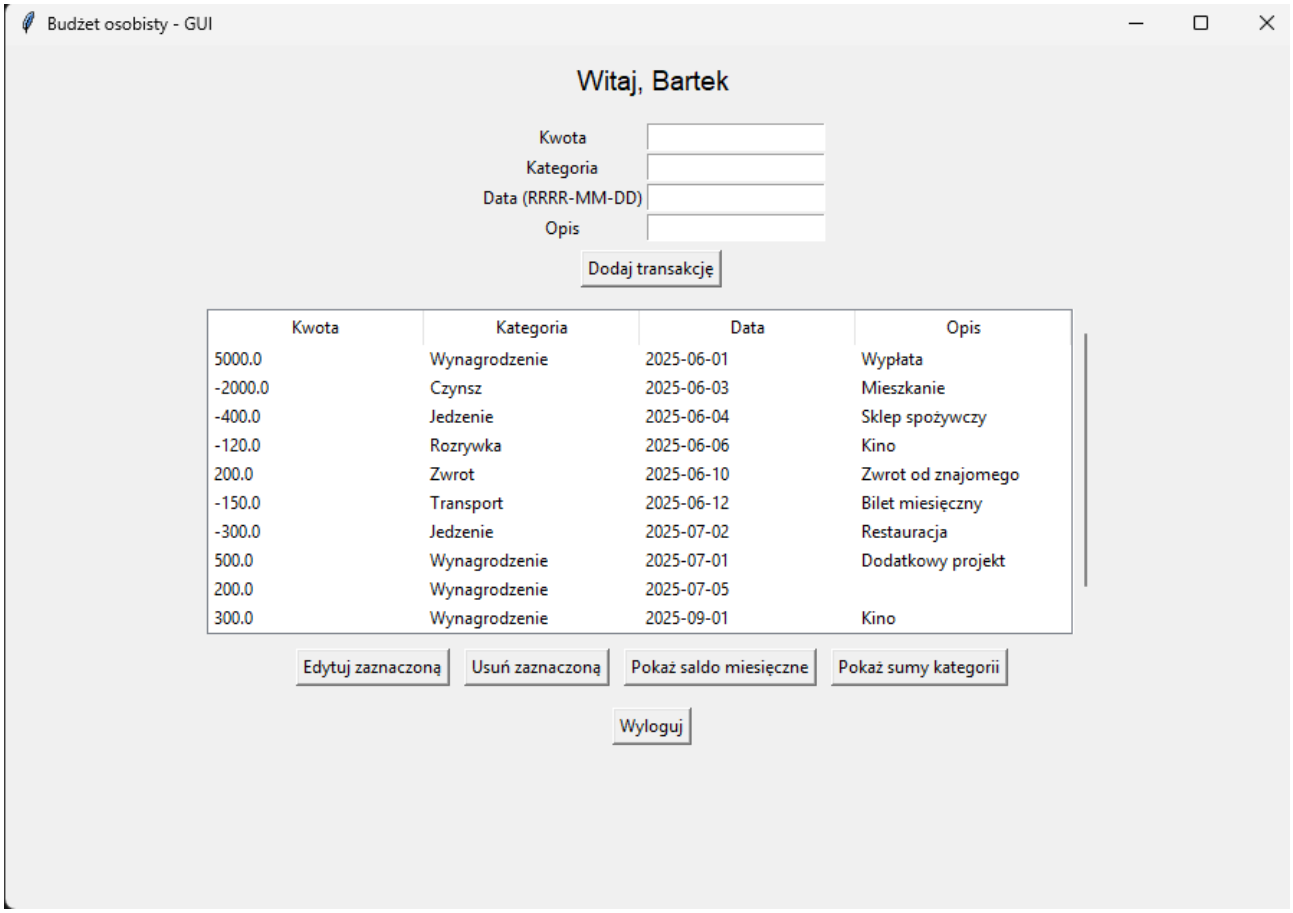
Dane wyjściowe

- Pliki *.csv z transakcjami
- Plik raport.txt z podsumowaniem
- Dynamiczne wykresy

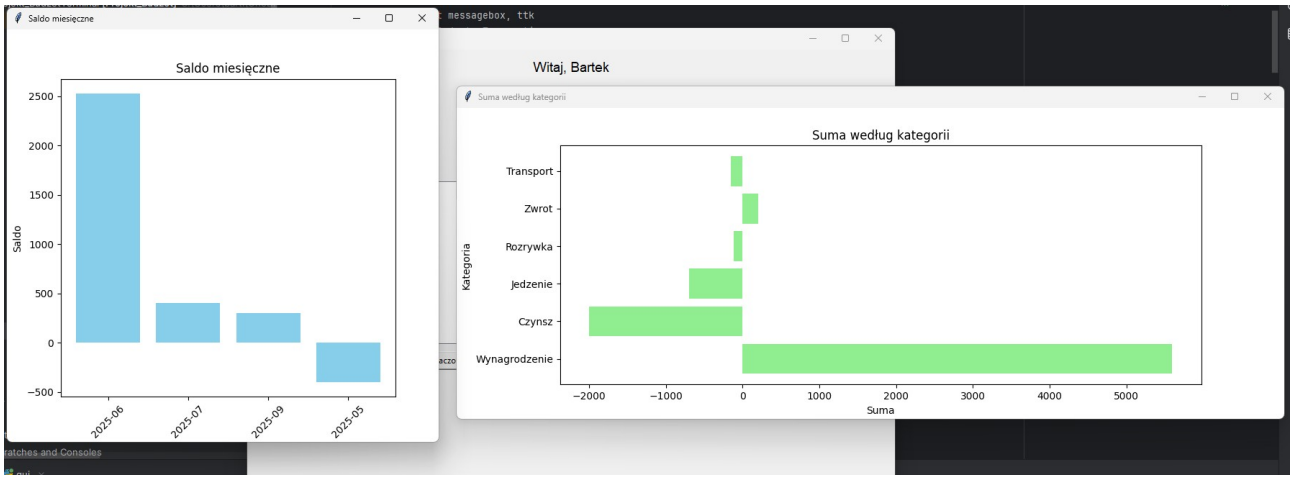
Zrzuty ekranu



(Rys.1) Logowanie



(Rys.2) Okno aplikacji z transakcjami



(Rys.3) Wygenerowane wykresy z aplikacji dla transakcji użytkownika

5. Przykłady kodu

Fragment funkcji funkcyjnej

```
def zapisz_raport_do_pliku(budget, filename="raport.txt"):
    with open(filename, mode="w", encoding="utf-8") as file:
        for month, total in budget.get_monthly_balance().items():
            file.write(f"{month}: {total:.2f} zł\n")
```

Opis: Funkcja zapisuje raport miesięczny do pliku tekstowego, korzystając z metod klasy Budget.

Fragment klasy – konstruktor i metoda dodająca

```
class Transaction:
    def __init__(self, amount, category, date_str, description=""):
        self.amount = float(amount)
        self.category = category
        self.date = self._parse_date(date_str)
        self.description = description
```

Opis: Konstruktor klasy transakcji przetwarza dane wejściowe, w tym parsuje datę i rzutuje kwotę.

Obsługa wyjątków – format daty

```
def _parse_date(self, date_str):
    try:
        return datetime.strptime(date_str, "%Y-%m-%d").date()
    except ValueError:
        raise ValueError("Nieprawidłowy format daty. Oczekiwany: RRRR-MM-DD")
```

Opis: Walidacja formatu daty w klasie Transaction. W przypadku błędu użytkownik dostaje jasny komunikat.

Obsługa GUI – logowanie użytkownika

```
def login(self):
    username = self.username_entry.get()
    password = self.password_entry.get()
    user = login_user(username, password)
    if user:
```

```
        self.user = user
        self.budget.transactions = load_transactions_from_csv(f"data/{user}_transactions.csv")
        self.main_screen()
    else:
        messagebox.showerror("Błąd", "Niepoprawne dane logowania")
```

Opis: Logika logowania zintegrowana z graficznym interfejsem użytkownika – po pomyślnym logowaniu dane są wczytywane z pliku użytkownika.

Przykład interakcji z plikiem CSV

```
def save_transactions_to_csv(transactions, filename):
    with open(filename, mode="w", newline='', encoding="utf-8") as file:
        writer = csv.writer(file)
        writer.writerow(["amount", "category", "date", "description"])
        for t in transactions:
            writer.writerow([t.amount, t.category, t.date.isoformat(), t.description])
```

Opis: Zapis listy transakcji do pliku CSV – każda transakcja jest konwertowana do odpowiedniego formatu.

Wyświetlenie wykresu w GUI (tkinter + matplotlib)

```
def show_plot(self, fig, title):
    window = tk.Toplevel(self.root)
    window.title(title)
    canvas = FigureCanvasTkAgg(fig, master=window)
    canvas.draw()
    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
```

Opis: Integracja matplotlib z tkinter – wykres wyświetlany w osobnym oknie wewnątrz aplikacji.

6. Testowanie

Opis testów

Testy jednostkowe zostały zrealizowane w osobnym folderze tests/, podzielone zgodnie z modułami:

- test_auth.py
- test_budget.py
- test_data.py
- test_gui.py
- test_logic.py
- test_utils.py

Przypadki brzegowe

- Nieprawidłowe formaty dat
- Pusta kategoria/kwota
- Edycja nieistniejącej transakcji
- Rejestracja z już istniejącą nazwą

7. Wnioski

Projekt pozwolił na praktyczne utrwalenie umiejętności związanych z programowaniem w Pythonie, w tym: obsługi plików CSV, pracy z GUI w tkinter, stosowania bibliotek zewnętrznych (matplotlib) oraz organizacji kodu w sposób modułowy. Zrealizowano pełne pokrycie funkcjonalności oraz testy jednostkowe. Aplikacja może być w przyszłości rozwijana o np. eksport do PDF, filtrowanie danych w GUI lub synchronizację w chmurze.