

## **Projekt zaliczeniowy – Języki skryptowe (Python)**

**Tytuł projektu: Budżet osobisty z analizą wydatków.**

**Autorzy: Bartłomiej Konopka, Jakub Kowalik**

**Grupa: 2ID12B**

**Data oddania: 19 czerwca 2025**

**Link do projektu:** <https://github.com/BartlomiejKonopka03/BudzetOsobisty.git>

### **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 JSON
- Tworzenie raportów tekstowych
- Wizualizacja danych za pomocą wykresów w GUI

#### **Zakres funkcjonalny**

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

## 2. Struktura projektu

Foldery i pliki

- /data/ – dane JSON użytkowników
- /tests/ – testy jednostkowe
- gui.py – główny interfejs użytkownika
- auth.py – rejestracja i logowanie
- data.py – zapis i odczyt JSON
- logic.py – definicja transakcji
- charts.py – rysowanie wykresów
- utils.py – zapis raportów
- README.md, raport.pdf, raport.txt – dokumentacja i raporty

Omówienie klas

- Transaction – pojedyncza transakcja
- IncomeTransaction – dziedziczy po Transaction, służy do oznaczania przychodów
- Budget – lista transakcji i metody analityczne
- BudgetApp – główna klasa GUI
- auth.\* – rejestracja i logowanie użytkownika
- data.\* – zapis i odczyt JSON
- utils.\* – generowanie raportów tekstowych

## 3. Technologie i biblioteki

- Python 3.11
- tkinter – interfejs graficzny
- json, 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
```

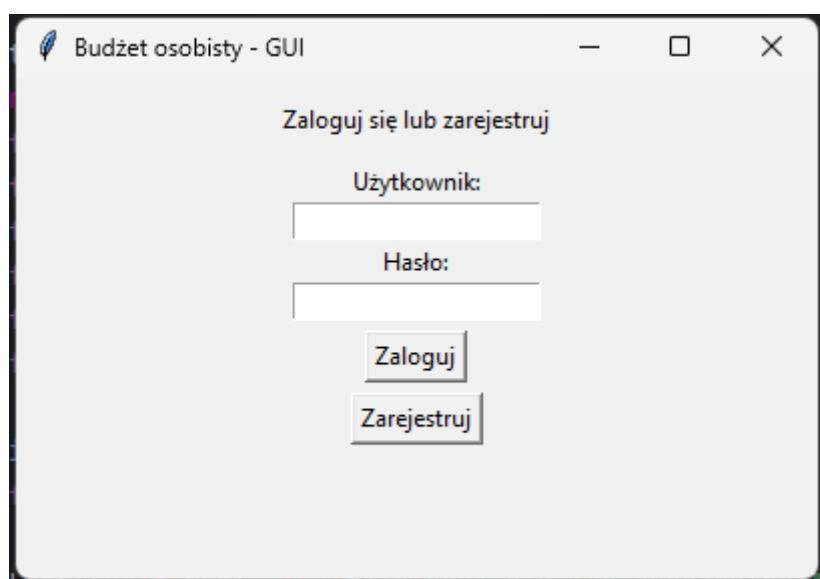
Dane wejściowe

- Kwota, kategoria, data i opcjonalny opis

Dane wyjściowe

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

## Zrzuty ekranu



(Rys. 1) Logowanie

Budżet osobisty - GUI

Witaj, Bartek

Kwota:

Kategoria:

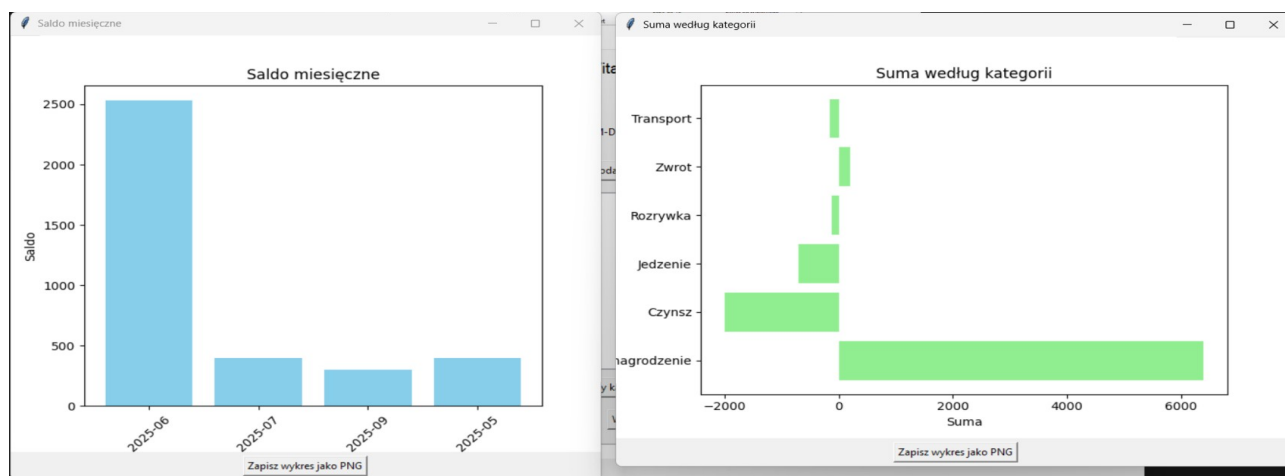
Data (RRRR-MM-DD):

Opis:

| Kwota   | Kategoria     | Data       | Opis               |
|---------|---------------|------------|--------------------|
| 5000.0  | Wynagrodzenie | 2025-06-01 | Wypłata            |
| -2000.0 | Czynsz        | 2025-06-03 | Mieszkanie         |
| -400.0  | Jedzenie      | 2025-06-04 | Sklep spożywczy    |
| -120.0  | Rozrywka      | 2025-06-06 | Kino               |
| 200.0   | Zwrot         | 2025-06-10 | Zwrot od znajomego |
| -150.0  | Transport     | 2025-06-12 | Bilet miesięczny   |
| -300.0  | Jedzenie      | 2025-07-02 | Restauracja        |
| 500.0   | Wynagrodzenie | 2025-07-01 | Dodatkowy projekt  |
| 200.0   | Wynagrodzenie | 2025-07-05 |                    |
| 300.0   | Wynagrodzenie | 2025-09-01 | Kino               |

Wyszukaj:

(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:

```
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:

```
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.

Wyświetlenie wykresu w GUI:

```
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.

## 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 JSON, 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.