TiDA 04[amb] – Dokumentacja testów i automatyczne generowanie raportów w Markdown

---

Cele zajęć

Po zakończeniu tego bloku będziesz potrafił:

- automatycznie tworzyć i formatować raporty z testów w języku **Markdown**,
- dodawać do raportu informacje o środowisku testowym,
- generować statystyki, wykresy i zapisywać je do raportu,
- konwertować pliki `.md` do innych formatów: **HTML**, **PDF**, **DOCX**,
- tworzyć klasy do raportowania testów i automatyzować proces dokumentacji.

Struktura profesjonalnego raportu testowego

Sekcja	Zawartość
Cel testu	Opis testowanego elementu systemu oraz jego funkcji
Środowisko testowe	System operacyjny, procesor, wersja Pythona, zależności
Parametry testu	Dane wejściowe, zakres testów, wersje modułów
Scenariusze testowe	Lista przypadków testowych (Test Case ID, opis, oczekiwany wynik)
Wyniki	Dane z wykonanych testów – sukcesy, błędy, czasy, statystyki
Wnioski i rekomendacje	Co należy poprawić, jakie są ograniczenia testu

---

Automatyczne pobieranie danych o środowisku

```python
import platform
import datetime
import psutil  # biblioteka do monitorowania systemu

def system_info():
    info = {
        "System": platform.system(),
        "Wersja": platform.version(),
        "Procesor": platform.processor(),
        "Architektura": platform.architecture()[0],
        "Python": platform.python_version(),
        "RAM (GB)": round(psutil.virtual_memory().total / (1024**3), 2),
        "Data testu": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    }
    return info
```

Markdown w praktyce

Przykład z kodem i tabelą

```
### Wyniki testu sortowania

| Algorytm | Rozmiar | Czas (s) | Złożoność |
|----------|---------|----------|-----------|
| Bubble Sort | 1000 | 0.326 | O(n²) |
| Quick Sort | 1000 | 0.021 | O(n log n) |

```python
def quicksort(a):
    if len(a) <= 1:
        return a
    pivot = a[len(a)//2]
    left = [x for x in a if x < pivot]
    middle = [x for x in a if x == pivot]
    right = [x for x in a if x > pivot]
    return quicksort(left) + middle + quicksort(right)
```
```

Klasa MarkdownReporter

```python
import platform
import datetime
import os

class MarkdownReporter:
    def __init__(self, report_name="raport_testow"):
        self.filename = f"{report_name}_{datetime.date.today()}.md"
        self.lines = []

    def add_header(self, text, level=1):
        self.lines.append("#" * level + f" {text}\n")

    def add_line(self, text=""):
        self.lines.append(text + "\n")

    def add_table(self, headers, rows):
        header_line = "| " + " | ".join(headers) + " |"
        separator = "| " + " | ".join(["-" * len(h) for h in headers]) + " |"
        self.lines.append(header_line)
        self.lines.append(separator)
        for row in rows:
            self.lines.append("| " + " | ".join(str(x) for x in row) + " |")
        self.lines.append("\n")

    def save(self):
        with open(self.filename, "w", encoding="utf-8") as f:
            f.write("\n".join(self.lines))
        print(f"✅ Raport zapisano jako: {self.filename}")
```

```python
# Przykład użycia
report = MarkdownReporter("raport_wydajnosci")
report.add_header("Raport testów sortowania", 1)
report.add_line(f"Data: {datetime.datetime.now()}\n")
report.add_header("Wyniki", 2)
report.add_table(["Rozmiar", "Czas (s)"], [[100, 0.01], [1000, 0.33], [5000, 2.85]])
report.add_header("Wnioski", 2)
report.add_line("Wyniki potwierdzają złożoność O(n²) dla Bubble Sort.")
report.save()
```

📈 Dodawanie wykresu i osadzanie w raporcie

```python
import matplotlib.pyplot as plt

sizes = [100, 1000, 5000, 10000]
times = [0.01, 0.33, 2.85, 11.24]

plt.plot(sizes, times, marker='o')
plt.title("Zależność czasu sortowania od rozmiaru tablicy")
plt.xlabel("Rozmiar tablicy")
plt.ylabel("Czas [s]")
plt.grid(True)
plt.savefig("czas_sortowania.png")

# Dodaj obrazek do raportu
report.add_line("![Wykres wyników](czas_sortowania.png)")
report.save()
```

🌐 Konwersja raportu na inne formaty

```python
import pypandoc

pypandoc.convert_text(
    open("raport_wydajnosci_2025-10-27.md").read(),
    'pdf',
    format='md',
    outputfile='raport_wydajnosci.pdf',
    extra_args=['--standalone']
)
```

> 🔧 Upewnij się, że masz zainstalowany **Pandoc** i **LaTeX** (dla PDF).

🎓 Zadania (do oddania w formie sprawozdania)

1. Rozszerz klasę `MarkdownReporter` o:
    - automatyczne dodawanie sekcji środowiska (`platform`, `psutil`),
    - możliwość dodania wykresów i zrzutów ekranu,
    - zapisywanie kilku raportów w jednym katalogu z datą i godziną.
2. Stwórz test porównawczy dwóch algorytmów (np. Bubble Sort vs. Merge Sort) i wygeneruj raport porównujący ich wydajność.
3. Dodaj statystyki do raportu: średni czas testów, odchylenie standardowe, min/max.
4. Utwórz automatyczny skrypt `run_tests_and_report.py`, który:
    - uruchamia testy,
    - zapisuje wyniki do `.csv`,
    - generuje raport `.md` z tabelą i wykresem.
5. (Bonus) Dodaj opcję eksportu raportu do **HTML** z użyciem `pypandoc`.