

# Lab 02: GitHub API

Stalking z klasą

Jarosław Hryszko

`jaroslaw.hryszko@uj.edu.pl`

Instytut Informatyki i Matematyki Komputerowej UJ

Otwarte repozytoria kodu i pomiary oprogramowania

## Poprzednio w Git Archaeology...

- Kopaliśmy w historii gita **lokalnie** (`git log`, `git blame`)
- Dziś idziemy dalej — sięgamy po dane z **serwera GitHub**
- Issues, pull requesty, statystyki kontrybutorów, aktywność
- Wszystko dostępne programowo przez **REST API**

## Jak to działa?

- Zapytania HTTP GET do `https://api.github.com/...`
- Odpowiedzi w formacie JSON
- Autentykacja: Personal Access Token (PAT)
- Rate limit: 5000 req/h z tokenem, 60 bez

## Przykład

GET `/repos/psf/requests` — informacje o repozytorium  
GET `/repos/psf/requests/issues` — lista issues  
GET `/repos/psf/requests/pulls` — lista pull requestów

# Pułapki, o których warto wiedzieć

- ❶ **Paginacja** — API domyślnie zwraca 30 elementów. Chcesz więcej? `per_page=100` + nagłówek `Link` z URL następnej strony.
- ❷ **Rate limiting** — 5000/h brzmi dużo, ale zapytanie per issue per komentarz szybko zjada limit.
- ❸ **Issues vs PRs** — endpoint `/issues` zwraca **też pull requesty!** Trzeba filtrować po kluczu `pull_request`.
- ❹ **Daty** — format ISO 8601 z "Z" na końcu.

## ❶ Zadanie 1 — Rozgrzewka z API (30 min)

- Token, curl, pierwsze zapytania w Pythonie
- Sprawdzenie rate limitu

## ❷ Zadanie 2 — GitHub Profiler (60 min)

- Skrypt `github_profiler.py`
- Profil repo: metryki, issues, PRs, aktywność

## ❸ Zadanie 3 — Porównywarka (45 min, opcjonalne)

- Dwa repo obok siebie: które jest zdrowsze?

# Co oddajecie?

W branchu `lab02_nazwisko1_nazwisko2`:

- 1 `github_profiler.py` — działający skrypt
- 2 `report.txt` — output dla wybranego repo
- 3 (opcjonalnie) `compare_repos.py`

## Uwaga

Nie commitujcie tokena! Używajcie `os.environ["GITHUB_TOKEN"]`.

# Do roboty!

Instrukcja: README.md w repozytorium

*“Jeśli nie możesz tego zmierzyć, nie możesz tym zarządzać.”* — Peter Drucker (albo ktoś inny)