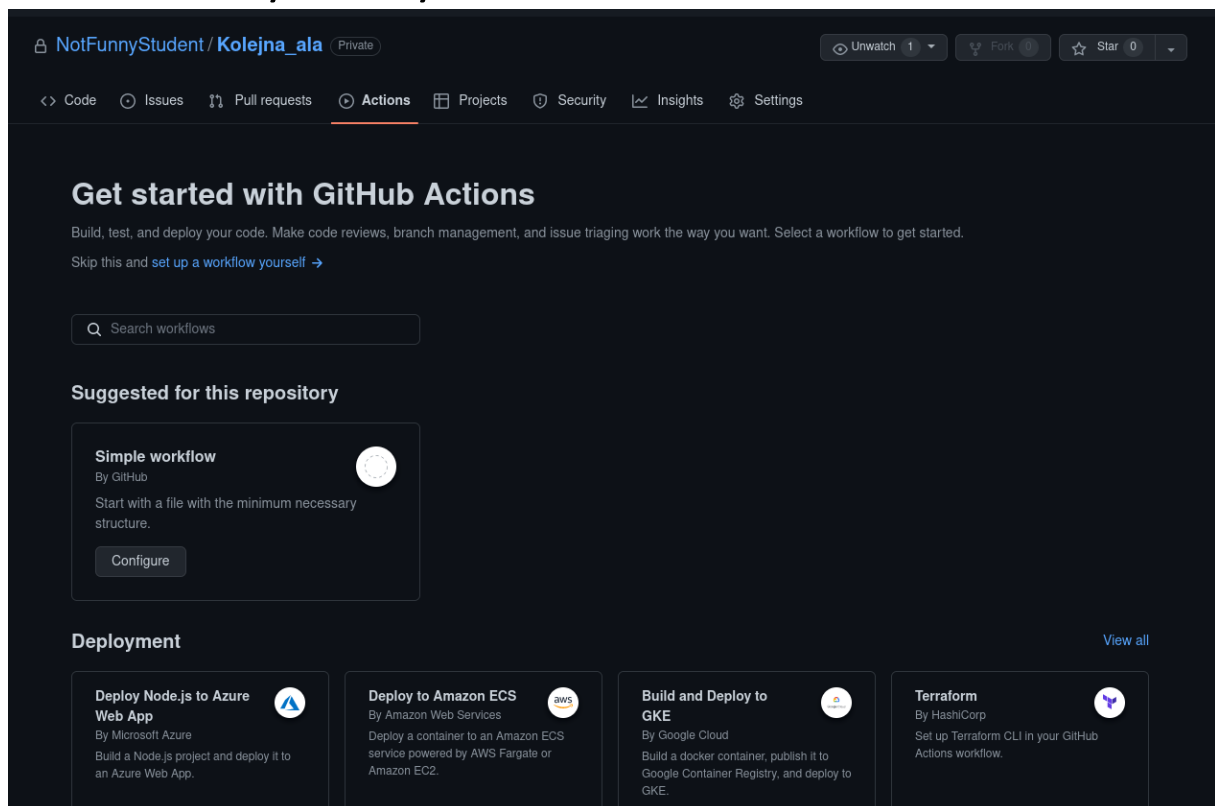
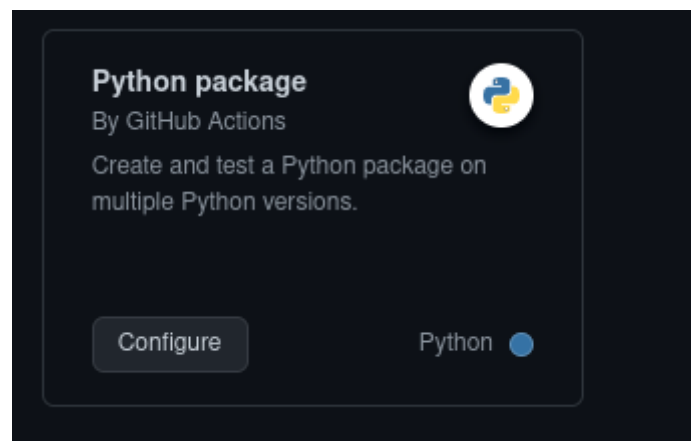


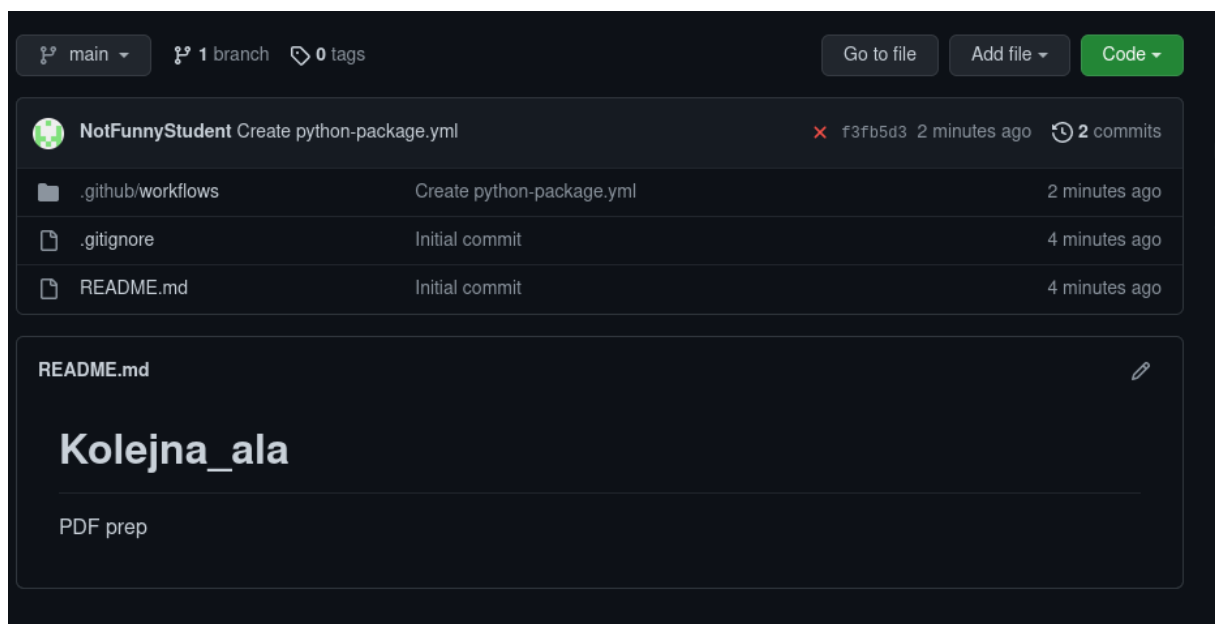
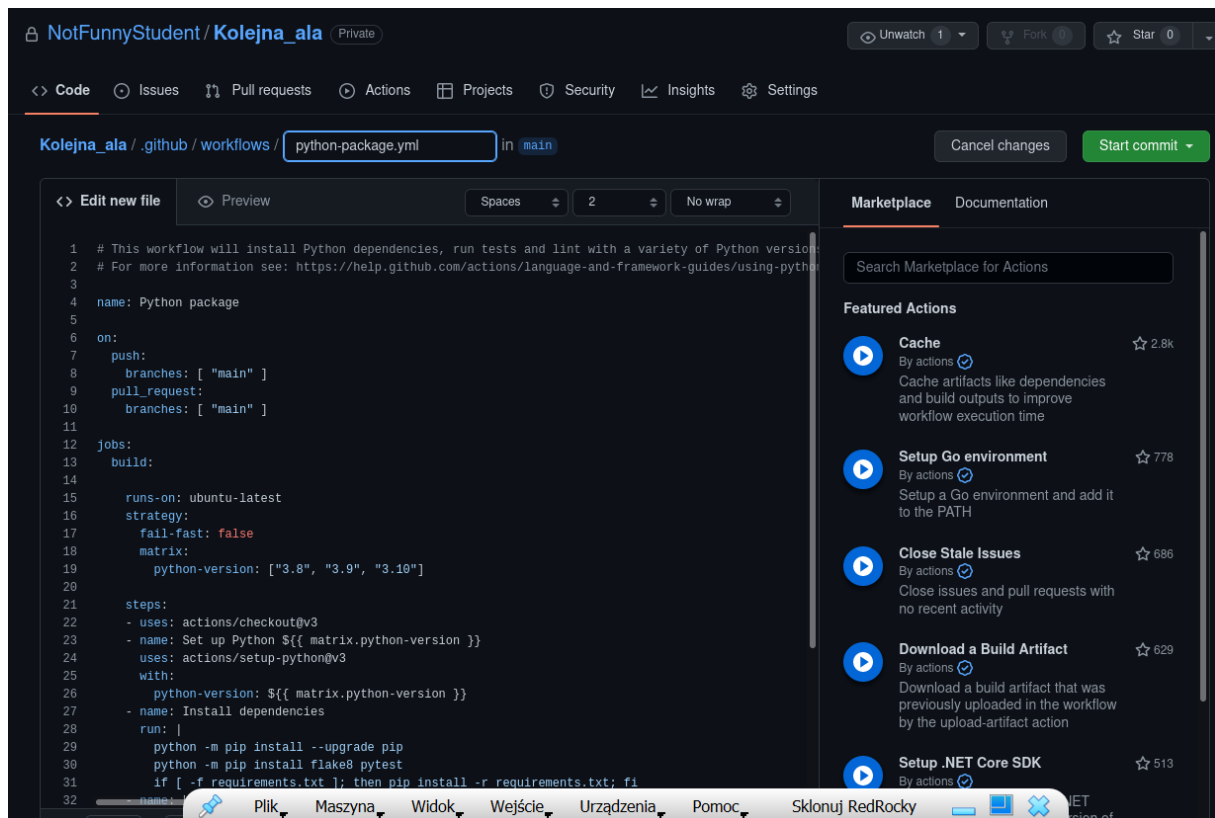
# Ustawianie automatycznego testowania kodu z wykorzystaniem GitHub Actions



Szukamy takiego cuda

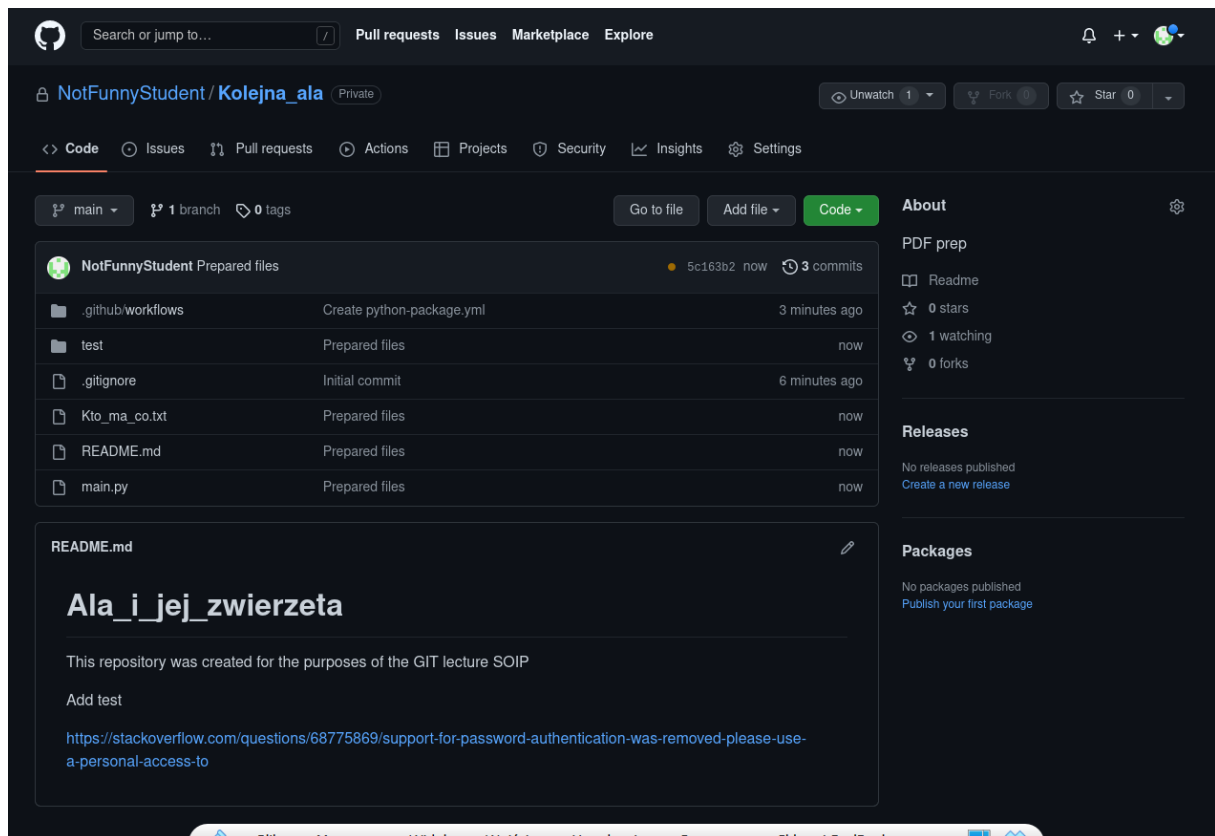


Pytest'a można też zainstalować z łapki, tak samo jak i stworzyć całkowicie własną konfigurację, ale powiedzmy, że na spotkaniu zajmiemy się tylko podstawami.



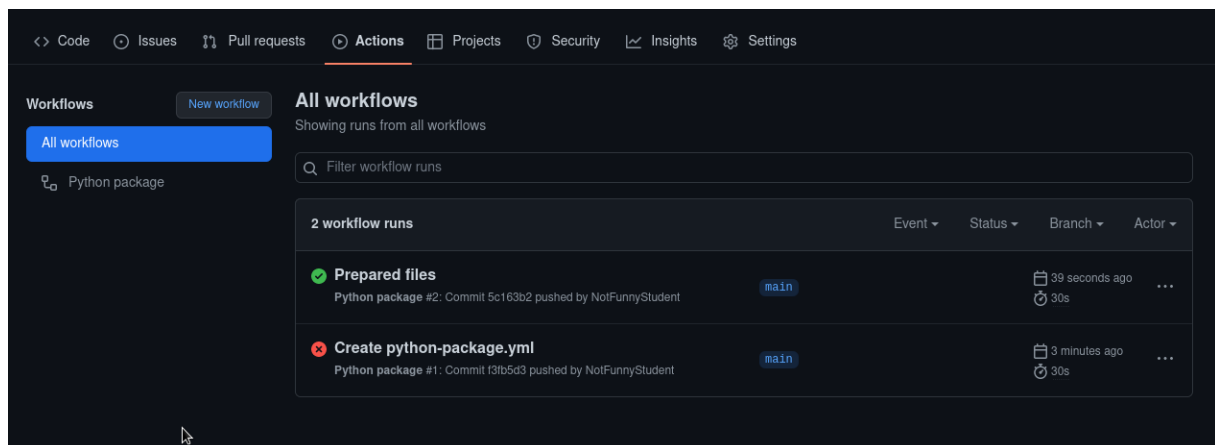
Zostaliśmy z takim repozytorium i co dalej?

Robimy commita naszymi plikami



I jak widzicie czerwony x nam gdzieś zaginął, jakaś żółta kropka się pojawiła obok hasha

Skoczmy sobie do zakładki Actions



The screenshot shows the GitHub Actions interface for a repository named 'NotFunnyStudent / Kolejna\_ala'. The workflow is titled 'Prepared files Python package #2'. The 'build (3.8)' job is selected, showing a list of steps: 'Set up job', 'Run actions/checkout@v3', 'Set up Python 3.8', 'Install dependencies', 'Lint with flake8', and 'Test with pytest'. The 'Test with pytest' step is expanded, showing the command 'Run pytest' and the output of the test session, which indicates that the test passed successfully. The bottom of the image shows a Windows taskbar with various application icons.

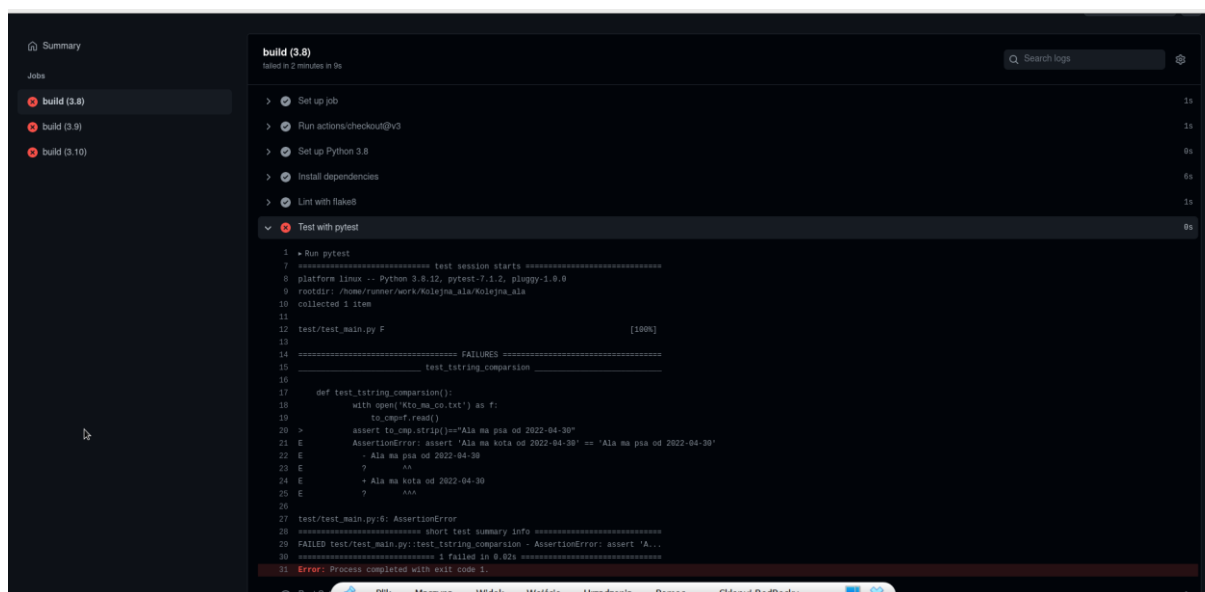
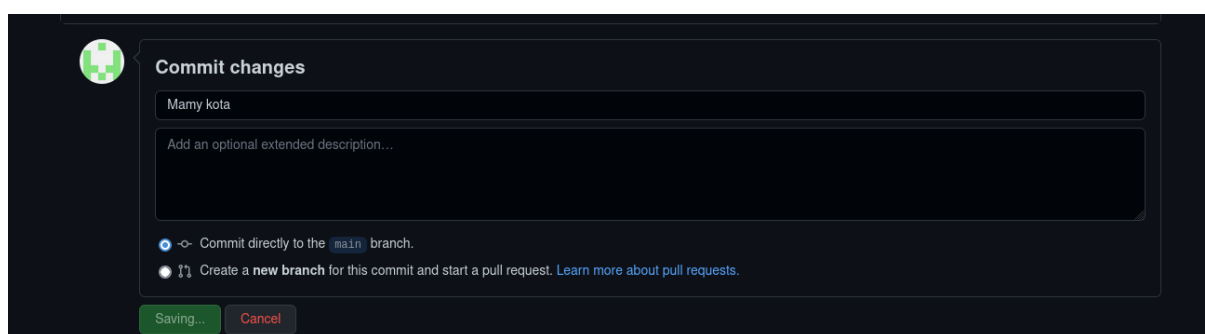
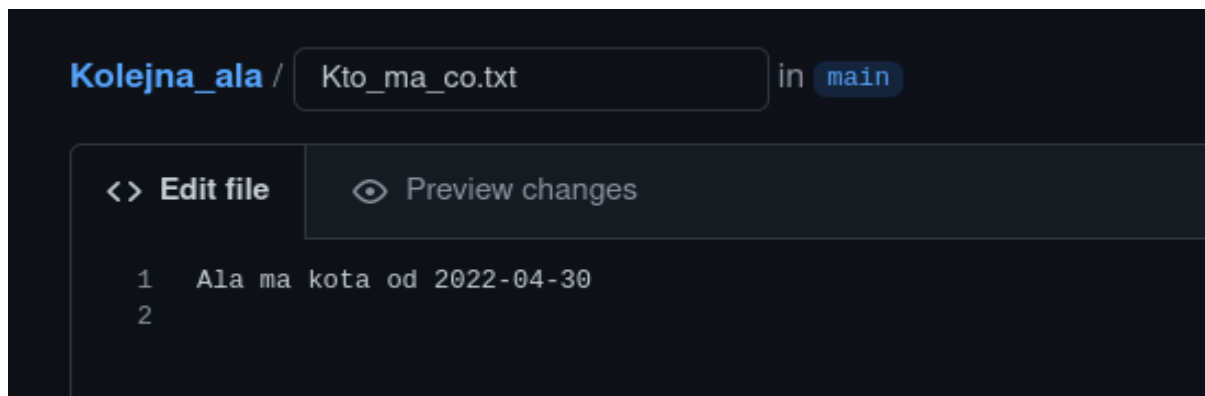
I co się okazuje, że coś nam się potestowało, a przynajmniej nie rzuciło błędów.

No, ale teraz tak warto się zastanowić, czym jest ten nasz test

The screenshot shows the GitHub code editor for the file 'test\_main.py' in the 'Kolejna\_ala' repository. The file is 6 lines long (5 lines of code) and 173 bytes in size. The code is as follows:

```
1 import pytest
2
3 def test_tstring_comparision():
4     with open('Kto_ma_co.txt') as f:
5         to_cmp=f.read()
6         assert to_cmp.strip()=="Ała ma psa od 2022-04-30"
```

I co się stanie jeśli spróbujemy wpisać do pliku inny string i zasymulujemy zdarzenie on\_push?



Jak widzimy w raporcie program nam się wysypał. Także otwieramy szampana, nasz kod ma pokrycie równe 100% i jesteśmy bezpieczni.

Zwróćcie uwagę na to, że raport mówi nam co dokładnie poszło nie tak

```
Test with pytest

1 ▶ Run pytest
7 ===== test session starts =====
8 platform linux -- Python 3.8.12, pytest-7.1.2, pluggy-1.0.0
9 rootdir: /home/runner/work/Kolejna_ala/Kolejna_ala
10 collected 1 item
11
12 test/test_main.py F [100%]
13
14 ===== FAILURES =====
15 _____ test_tstring_comparsion _____
16
17     def test_tstring_comparsion():
18         with open('Kto_ma_co.txt') as f:
19             to_cmp=f.read()
20 >     assert to_cmp.strip()=="Ala ma psa od 2022-04-30"
21 E     AssertionError: assert 'Ala ma kota od 2022-04-30' == 'Ala ma psa od 2022-04-30'
22 E         - Ala ma psa od 2022-04-30
23 E           ?      ^^
24 E         + Ala ma kota od 2022-04-30
25 E           ?      ^^^
26
27 test/test_main.py:6: AssertionError
28 ===== short test summary info =====
29 FAILED test/test_main.py::test_tstring_comparsion - AssertionError: assert 'A...
30 ===== 1 failed in 0.02s =====
31 Error: Process completed with exit code 1.
```

W funkcji takiej to a takiej, to i to mi się nie zgadza. Co prawda sami również możemy trochę modyfikować jakie dane zwraca nam raport, ale przyznacie, że jest to już potężne ułatwienie.

\*Test failed w domyślnym ustawieniu śle nam na maila powiadomienie, że oj coś poszło nie tak. Z jednej strony fajnie, z drugiej szybko można sobie zaśmiec pocztę jeśli tego nie zauważymy.

Co powinniśmy zapamiętać:

- Te nazwy jak i struktura to nie jest całkowity przypadek, aby to działało poprawnie jednak musimy współpracować z wytycznymi dokumentacji
- Testy jednostkowe powinny testować mały fragment - przeważnie jedną funkcję.
- Testy w naszej przestrzeni zwracają tylko dwa stany True/False
- Testy powinny być szybkie i niezależne