

Sprawozdanie

Narzędzia do automatyzacji budowy oprogramowania

Kowal Maciej
100400

January 2025

Spis treści

1	Repozytorium GitHub	3
2	Komendy	3
3	Konfiguracja lokalnego repozytorium	3
3.1	Schemat	4
3.2	tokeny	4
3.3	Branch	4
3.4	Konflikt	4
4	Fork	5
4.1	Flask	5
4.2	Format .json	6
4.3	Markdown	6
4.4	Błędy	6
4.5	Działanie	7
4.6	Testowanie	8
5	Makefile	9
5.1	Dlaczego ważne jest single point of entry?	10
6	CircleCI	10
6.1	YAML	10
6.2	Setup	10
7	Docker	12
7.1	Dockerfile	12
8	Screenshoty	13
8.1	CircleCI	13
8.2	Dockerhub	13
8.3	Commity Github	13

Spis rysunków

1	Enter Caption	3
2	Schemat pull request	4
3	Opcja utworzenia tokenu	4
4	Konflikt gałęzi	5
5	Komentarze po merge	5
6	Przycisk Fork	5
7	Zmiana imienia	6
8	Zmiana Ensure	7
9	Markdown bug	7
10	Aplikacja przed zmianami stałych	8

11	Aplikacja po zmianach stałych	8
12	Zwrócony format .json(mgs jeszcze nieporawione)	8
13	Test nr 1	8
14	Test nr2	9
15	Makefile	9
16	Przykładowy setup na CircleCi	10
17	Udane testy projektu	10
18	Udane testy projektu	11
19	Nieudane testy	11
20	Logi nieudanego testu	11
21	Lista kontenerów dockera	12
22	Docker build	12
23	Docker nie uruchamia aplikacji	12
24	CircleCI screenshot	13
25	Github screenshot	13

1 Repozytorium GitHub

Nazwa użytkownika: MaciejK11

Link do repozytorium github z pierwszych zajęć:

https://github.com/MaciejK11/nauka_gita_mk_wsb.git

Link do forka(żeby było wiadomo który jest mój):

https://github.com/MaciejK11/2025_L_II_NWh_INFI_IAMm.git

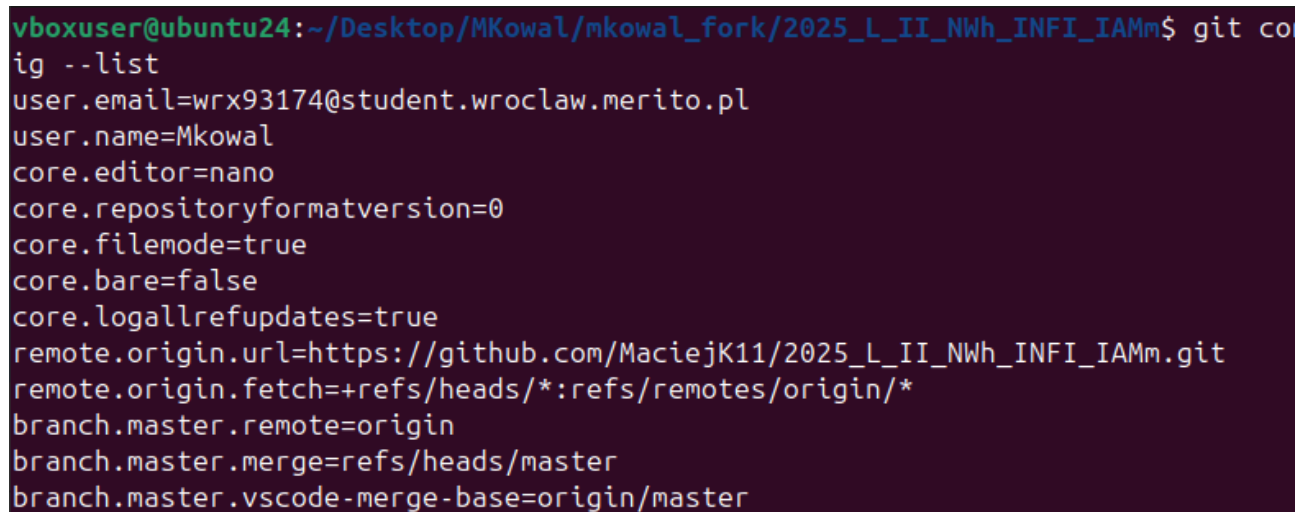
2 Komendy

Lista podstawowych komend wykorzystanych w ramach laboratorium

1. **git init** Tworzy nowe lokalne repozytorium Git w bieżącym katalogu, tworzy folder ukryty `.git`.
2. **git config** Konfiguracja ustawień globalnych gita, takich jak nazwa użytkownika czy email.
3. **git clone** Kopiuje istniejące repozytorium z serwera na PC. Przyjmuje m.in. adres https repozytorium.
4. **git add** Dodaje zmodyfikowane pliki do obszaru staging, faza przed commit
5. **git commit** Zapisuje zmiany z obszaru staging w historii repozytorium. Wymusza komentarz.
6. **git status** Wyświetla aktualny stan repozytorium, pokazując zmodyfikowane, dodane i nieśledzone pliki. Pomaga kontrolować, co zostanie zapisane w kolejnym commicie.
7. **git remote** Zarządza połączeniami z repozytoriami zdalnymi. Pozwala dodawać, usuwać i sprawdzać adresy repozytoriów takich jak origin.
8. **git push** Wysyła lokalne commity do repozytorium zdalnego. Umożliwia synchronizację zmian z serwerem, np. GitHubem. Wymusza zalogowanie do serwisu usługodawcy gita.
9. **git branch** Wyświetla listę gałęzi lub tworzy nową gałąź w repozytorium. Gałęzie pozwalają pracować nad różnymi wersjami projektu równolegle.
10. **git checkout** Przełącza między branchami.
11. **git merge** Łączy zmiany z jednej gałęzi do drugiej.

3 Konfiguracja lokalnego repozytorium

Konfigurujemy lokalne repozytorium git zgodnie z topologią.

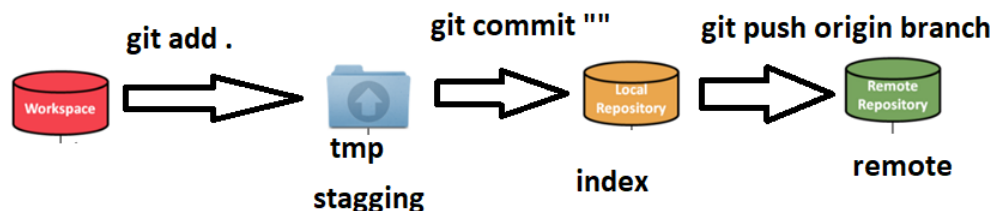


```
vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$ git config --list
user.email=wrx93174@student.wroclaw.merito.pl
user.name=Mkowal
core.editor=nano
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/MaciejK11/2025_L_II_NWh_INFI_IAMm.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
branch.master.vscode-merge-base=origin/master
```

Rysunek 1: Enter Caption

Jest to konfiguracja wykonana przy zadaniach z forkiem, origin to sklonowane repozytorium(`git clone http`). Na Windowsie miałem problem z ustawieniem notepad++ jako `core.editor`, `git push` zawieszał `git bash`, więc zmieniłem na notepad i działało, powyższy przykład `config -list` to linux bash, zatem edytor ustawiłem po prostu jako nano.

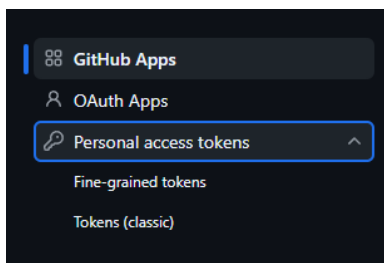
3.1 Schemat



Rysunek 2: Schemat pull request

Wszystkie zmiany jakie wykonamy w repozytorium zostają zapisane w przestrzeni roboczej(może być też `tmp/temporary` ale nazwałem tak `staging` więc niech tak zostanie), komenda `git add example` dodaje zmiany wykonane w `example` do folderu pośredniego nazwanego przeze mnie `tmp` (`staging`), następnie indeksowanie polega na zapisaniu zmian z folderu pośredniego w repozytorium lokalnym, zmiany wykonane są na określonych gałęziach, które można tworzyć, mergować i zmieniać. Dopiero `git push origin`(`http/ssh` repozytorium na serwerze usługodawcy `git`) przenosi nas do strony logowania, a nasz `commit` do repozytorium zdalnego.

3.2 tokeny



Rysunek 3: Opcja utworzenia tokenu

W GitHubie można generować tokeny, które mogą być używane zamiast hasła, mają czas wygaśnięcia i są ogólnie bezpieczniejsze, ponieważ mogą zostać usunięte w dowolnym momencie, wyświetlają się jednak tylko raz, dlatego są raczej jednorazowe.

3.3 Branch

Branch to wersja repozytorium lokalnego, którą można edytować niezależnie od innych gałęzi.

3.4 Konflikt

Konflikty podczas `merge`(scalania) mogą uniemożliwić wykonanie `commita` do momentu ich rozstrzygnięcia.

```

student-wroclaw@MX1-Docker-17 MINGW64 ~/Desktop/MaciejK11/se_nauka_gita (master)
$ git branch
  konflikkt
* master

student-wroclaw@MX1-Docker-17 MINGW64 ~/Desktop/MaciejK11/se_nauka_gita (master)
$ git merge konflikkt
Auto-merging avg.php
CONFLICT (content): Merge conflict in avg.php
Automatic merge failed; fix conflicts and then commit the result.

```

Rysunek 4: Konflikt gałęzi

Konflikt pomiędzy branchami powoduje modyfikację plików których dotyczy i wygląda następująco:

```

k? php

function srednia (a, b)
{
    return (a+b)/2
}

function wazona (a, b)
{
<<<<<<< HEAD
    return (5*a+9*b)/2
=====
    return (7*a+11*b)/2
>>>>>>> konflikkt
}

?>

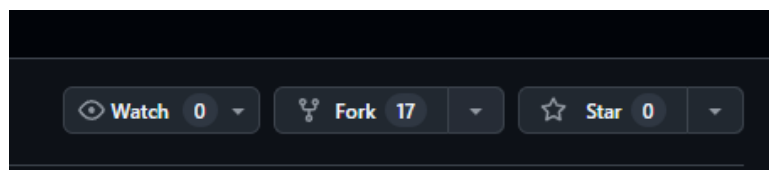
```

Rysunek 5: Komentarze po merge

W takim przypadku kod należy zmienić ręcznie.

4 Fork

Na laboratorium wykonany został fork(kopia) repozytorium podanego przez wykładowcę.



Rysunek 6: Przycisk Fork

Repozytoium zawiera program uruchamiający serwer w terminalu na danym porcie, do którego można się podłączyć z innego terminala(podając adres IP+port) i otrzymać tekst output zależny od wywołanej komendy.

4.1 Flask

Flask jest lekkim frameworkiem webowym w Pythonie(po prostu biblioteka). Służy do tworzenia aplikacji internetowych i API oraz pozwala definiować routing (@app.route), obsługiwać żądania HTTP i zwracać odpowiedzi (HTML, JSON). W naszym przypadku zwracany będzie .json.

4.2 Format .json

JSON to po prostu format zapisu danych. Plik z rozszerzeniem .json przechowuje dane w postaci struktury klucz:wartość.

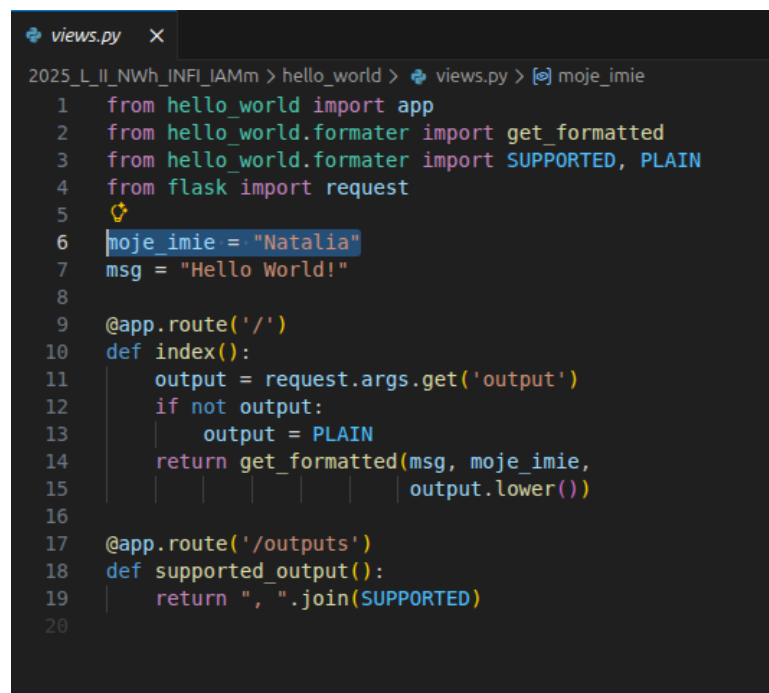
4.3 Markdown

Format Markdown(.md) to format pliku do dokumentacji tekstowej, zawiera specjalne znaki określające pogrubienia oraz podobne elementy możliwe do reprezentacji graficznej.

4.4 Błędy

lista znalezionych błędów

1. format_to_json - zwraca 'mgs' co jest niezgodne z formatem .json zatem zostaje zmienione na 'msg'.
2. stała moje_imie - nie jest błędem ale jej wartość zostaje zmieniona.
3. test_msg_with_output - w wersji programu z 'mgs' .assertEqual działało poprawnie, jednak w nowej wersji zostaje zmienione zgodnie z format_to_json
4. 'README.md' - błędna ścieżka aktywacji .venv, w Source nie ma funkcji activate, znajduje się ona w bin dla linuxa oraz Scripts dla Win10/11.



```
views.py X
2025_L_I_NWh_INFI_IAMm > hello_world > views.py > moje_imie
1 from hello_world import app
2 from hello_world.formater import get_formatted
3 from hello_world.formater import SUPPORTED, PLAIN
4 from flask import request
5
6 moje_imie = "Natalia"
7 msg = "Hello World!"
8
9 @app.route('/')
10 def index():
11     output = request.args.get('output')
12     if not output:
13         output = PLAIN
14     return get_formatted(msg, moje_imie,
15                           output.lower())
16
17 @app.route('/outputs')
18 def supported_output():
19     return ", ".join(SUPPORTED)
20
```

Rysunek 7: Zmiana imienia

```

2025_L_II_NWh_INF_IAMm > test > test_views.py > FlaskTestCase
1  import unittest
2  from hello_world import app
3  from hello_world.formater import SUPPORTED
4
5
6  class FlaskTestCase(unittest.TestCase):
7      def setUp(self):
8          app.config['TESTING'] = True
9          self.app = app.test_client()
10
11     def test_outputs(self):
12         rv = self.app.get('/outputs')
13         s = str(rv.data)
14         ' '.join(SUPPORTED) in s
15
16     def test_msg_with_output(self):
17         rv = self.app.get('/?output=json')
18         self.assertEqual(b'{"imie": "Natalia", "mgs": "Hello World!"}', rv.data)
19

```

Rysunek 8: Zmiana Ensure

```

1  # Simple Flask App
30  $ PYTHONPATH=. FLASK_APP=hello_world flask run
31  ...
32
33  - Uruchamianie testów (see: http://doc.pytest.org/en/latest/cap)
34  ...
35
36  $ PYTHONPATH=. py.test
37  $ PYTHONPATH=. py.test --verbose -s
38  ...
39
40  - Kontynuując pracę z projektem, aktywowanie hermetycznego środowiska
41  ...
42
43  # deaktywacja
44  $ deactivate
45  ...
46
47  ...
48  ...
49
50  # aktywacja
51  $ source .venv/Source/activate
52  ...
53
54  - Integracja z TravisCI:
55  ...
56
57  # miejsce na twoje notatki
58  ...
59
60  # Pomocnicze
61

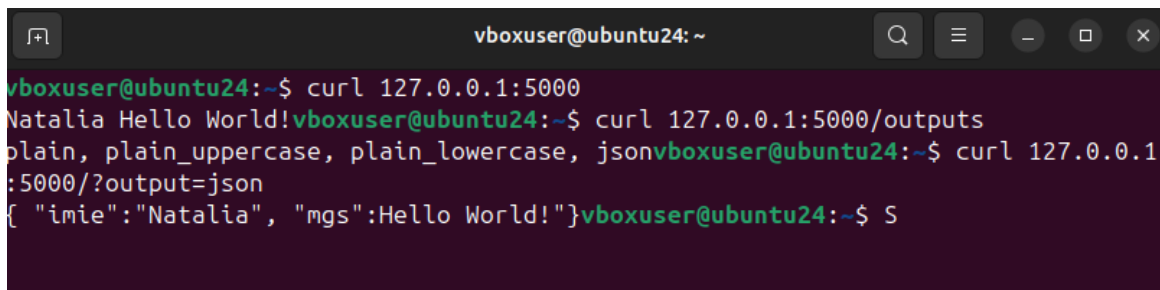
```

Rysunek 9: Markdown bug

Powyższy przykład to nie jest do końca błąd w kodzie zatem go nie zmieniłem, jest to błąd dokumentacji.

4.5 Działanie

Do uruchomienia programu wykorzystujemy albo przeglądarkę podając IP oraz nr portu, lub narzędzie curl spełniające tę samą funkcję (zwraca dane tekstowe, w układzie JSON).



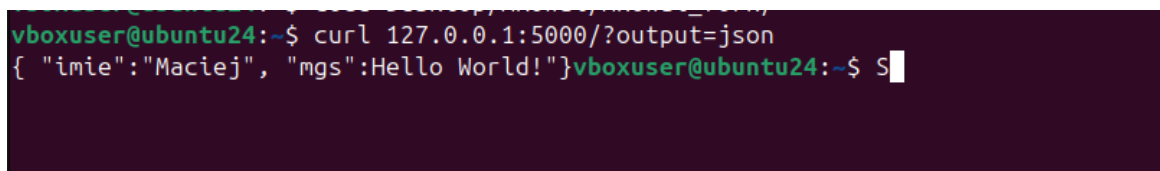
```
vboxuser@ubuntu24: ~
vboxuser@ubuntu24:~$ curl 127.0.0.1:5000
Natalia Hello World!vboxuser@ubuntu24:~$ curl 127.0.0.1:5000/outputs
plain, plain_uppercase, plain_lowercase, jsonvboxuser@ubuntu24:~$ curl 127.0.0.1:5000/?output=json
{"imie":"Natalia", "mgs":"Hello World!"}vboxuser@ubuntu24:~$ S
```

Rysunek 10: Aplikacja przed zmianami stałych



```
vboxuser@ubuntu24: ~
vboxuser@ubuntu24:~$ curl 127.0.0.1:5000
curl: (7) Failed to connect to 127.0.0.1 port 5000 after 0 ms: Couldn't connect to server
vboxuser@ubuntu24:~$ curl 127.0.0.1:5000
Maciej Hello World!vboxuser@ubuntu24:~$
```

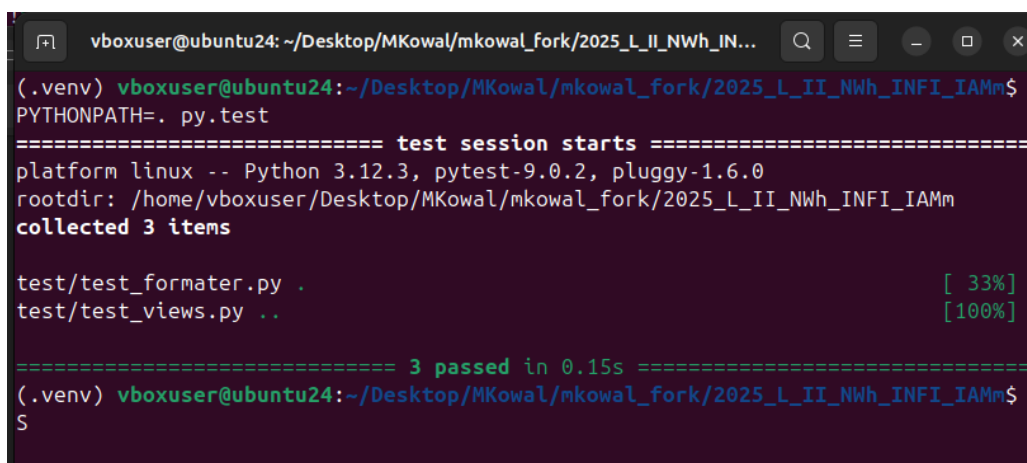
Rysunek 11: Aplikacja po zmianach stałych



```
vboxuser@ubuntu24:~$ curl 127.0.0.1:5000/?output=json
{"imie":"Maciej", "mgs":"Hello World!"}vboxuser@ubuntu24:~$ S
```

Rysunek 12: Zwrócony format .json(mgs jeszcze nieporawione)

4.6 Testowanie

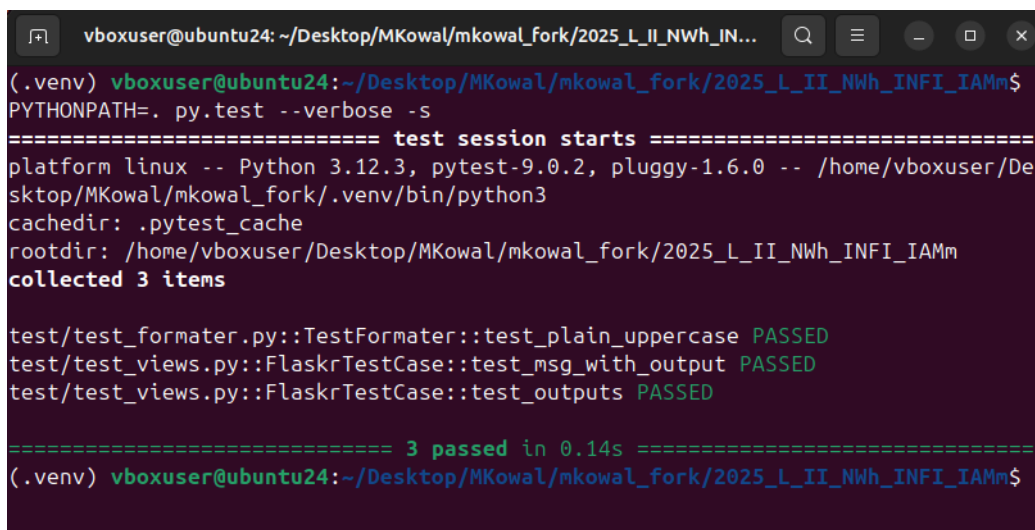


```
vboxuser@ubuntu24: ~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$
PYTHONPATH=. py.test
===== test session starts =====
platform linux -- Python 3.12.3, pytest-9.0.2, pluggy-1.6.0
rootdir: /home/vboxuser/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm
collected 3 items

test/test_formatter.py . [ 33%]
test/test_views.py .. [100%]

===== 3 passed in 0.15s =====
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$
S
```

Rysunek 13: Test nr 1



```

vboxuser@ubuntu24: ~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$
PYTHONPATH=. py.test --verbose -s
===== test session starts =====
platform linux -- Python 3.12.3, pytest-9.0.2, pluggy-1.6.0 -- /home/vboxuser/De
sktop/MKowal/mkowal_fork/.venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/vboxuser/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm
collected 3 items

test/test_formatter.py::TestFormatter::test_plain_uppercase PASSED
test/test_views.py::FlaskrTestCase::test_msg_with_output PASSED
test/test_views.py::FlaskrTestCase::test_outputs PASSED

===== 3 passed in 0.14s =====
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMm$

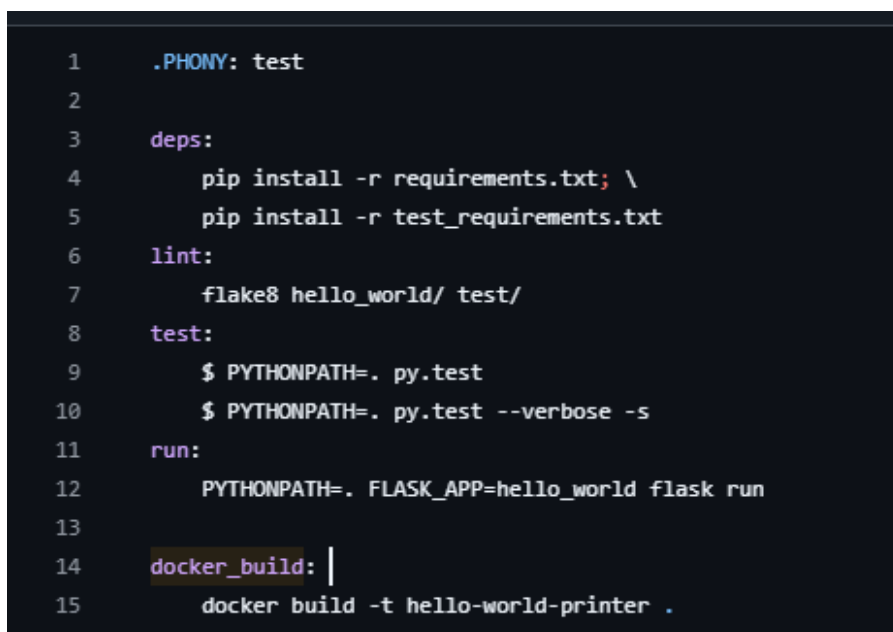
```

Rysunek 14: Test nr2

5 Makefile

Makefile jest typem pliku umożliwiającym uruchamianie określonych komend, służących do kompilacji oraz testowania programów. Wymaga instalacji programu make, skrypty przechowywane w Makefile. **Elementy naszego Makefile**

1. `deps`: pobiera pakiety pythona określone w plikach
2. `lint`: linter, nie testuje oprogramowania, ale znajduje potencjalne błędy(m. in. uznaje `import views` za niepotrzebny `import`)
3. `test`: wywołuje testy programu
4. `run`: uruchamia program
5. `docker_build`: tworzy obraz dockera o nazwie `hello-world-printer` w tym repozytorium na podstawie `Dockerfile`(omówienie w dalszej części)



```

1  .PHONY: test
2
3  deps:
4      pip install -r requirements.txt; \
5      pip install -r test_requirements.txt
6  lint:
7      flake8 hello_world/ test/
8  test:
9      $ PYTHONPATH=. py.test
10     $ PYTHONPATH=. py.test --verbose -s
11  run:
12     PYTHONPATH=. FLASK_APP=hello_world flask run
13
14  docker_build: |
15     docker build -t hello-world-printer .

```

Rysunek 15: Makefile

5.1 Dlaczego ważne jest single point of entry?

Ogólnie sprowadza się to do tego, że cała komunikacja w programie przechodzi przez jedno miejsce, dzięki czemu możliwa jest walidacja argumentów oraz weryfikacja uprawnień czy testowanie (te same testy są wywoływane dla różnych instancji, dodatkowa walidacja).

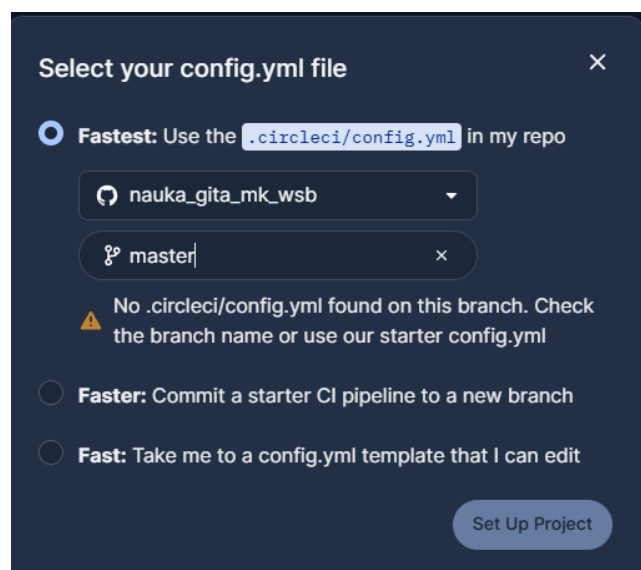
6 CircleCI

CircleCI jest serwisem umożliwiającym automatyczne testowanie oraz buildowanie (budowanie?) aplikacji na podstawie podanego repozytorium. Jest zintegrowany m. in. z makiem (Makefile) oraz Dockerem.

6.1 YAML

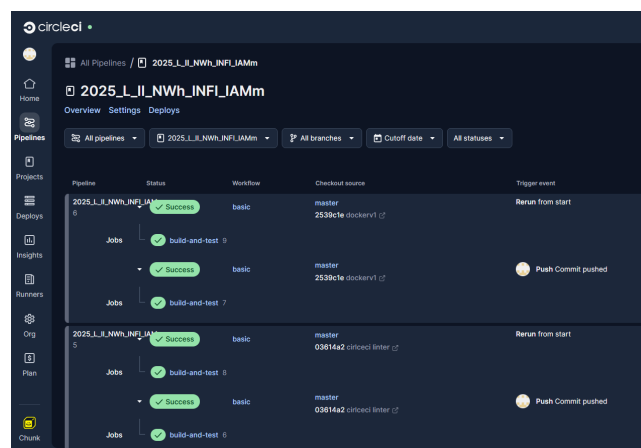
Format .yaml to czytelny dla człowieka format do zapisu danych w postaci tekstowej. Używa się go do konfiguracji aplikacji

6.2 Setup

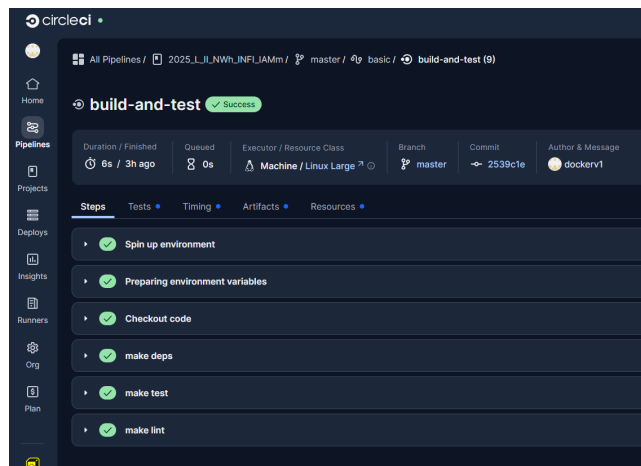


Rysunek 16: Przykładowy setup na CircleCI

Powyżej przykładowy setup projektu na CircleCI, akurat w tym projekcie nie mam config.yml.



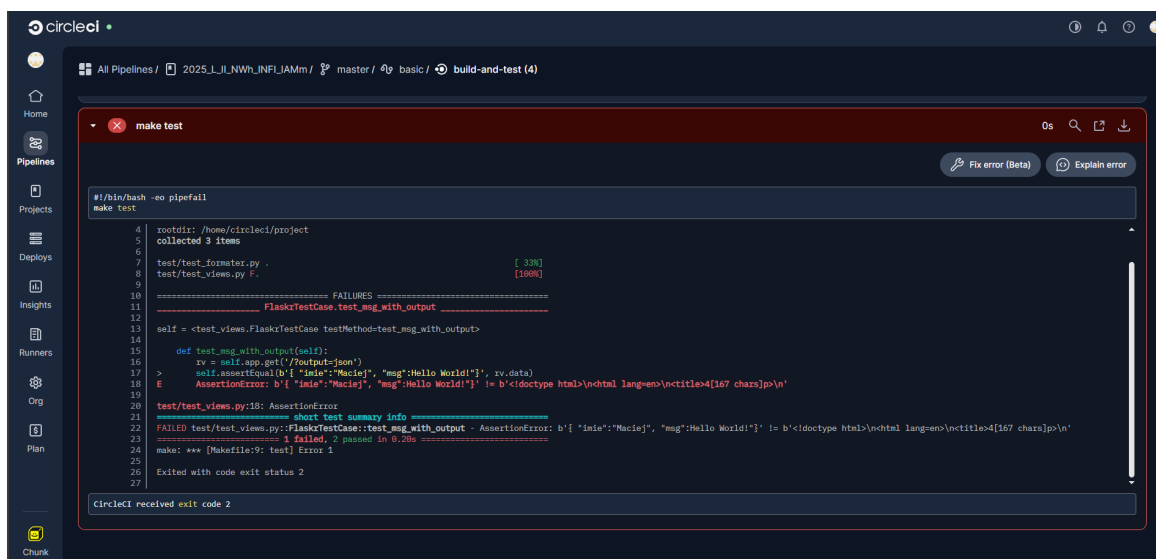
Rysunek 17: Udane testy projektu



Rysunek 18: Udane testy projektu



Rysunek 19: Nieudane testy



Rysunek 20: Logi nieudanego testu

Wiele buildów nie udało się na początku z powodu złej kombinacji znaków białych w plikach Makefile oraz config.yml (znak biały po \ dodatkowa spacja przed nazwą argumentu).

7 Docker

Docker to platforma do tworzenia, uruchamiania i zarządzania kontenerami. Kontener to lekka, przenośna jednostka oprogramowania, która zawiera aplikację wraz ze wszystkimi zależnościami, takimi jak biblioteki i konfiguracje, dzięki czemu działa identycznie na każdym systemie.

7.1 Dockerfile

W naszym projekcie tworzymy Dockerfile, czyli instrukcję jak zbudować obraz dockera (obraz to kopia której nie można zmieniać).

```
vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMn$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMn$
```

Rysunek 21: Lista kontenerów dockera

```
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMn$ make docker_build
docker build -t hello-world-printer .
[+] Building 0.6s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 272B                                0.0s
=> [internal] load metadata for docker.io/library/python:3        0.2s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/7] FROM docker.io/library/python:3@sha256:151ab3571dad616bb831852e86411e2165295c7f67 0.0s
=> => resolve docker.io/library/python:3@sha256:151ab3571dad616bb831852e86411e2165295c7f67 0.0s
=> [internal] load build context                                   0.0s
=> => transferring context: 463B                                      0.0s
=> CACHED [2/7] WORKDIR /tmp                                       0.0s
=> CACHED [3/7] ADD requirements.txt /tmp/requirements.txt        0.0s
=> CACHED [4/7] RUN pip install -r /tmp/requirements.txt          0.0s
=> CACHED [5/7] RUN mkdir -p /usr/src/hello_world_printer        0.0s
=> CACHED [6/7] ADD hello_world/ /usr/src/hello_world_printer/hello_world/ 0.0s
=> CACHED [7/7] ADD main.py /usr/src/hello_world_printer          0.0s
=> exporting to image                                              0.1s
=> => exporting layers                                              0.0s
=> => exporting manifest sha256:f9e000e9e75870dc6948f01b02653039b91a4260b21f5539f32b918924 0.0s
=> => exporting config sha256:ac9783b0fb368b6fd2c633d1cc8c19c3661220273f184d2c067cfbaf5e9af 0.0s
=> => exporting attestation manifest sha256:6140334fc0b3070f08089567cb959d2ac8920ce1970a0b 0.0s
=> => exporting manifest list sha256:69b0d90b6c56ac0a99db8f83b846d0e4248f70355c47caf702cb 0.0s
=> => naming to docker.io/library/hello-world-printer:latest      0.0s
=> => unpacking to docker.io/library/hello-world-printer:latest   0.0s
```

Rysunek 22: Docker build

Make `docker_build` poprawnie tworzy obraz dockera.

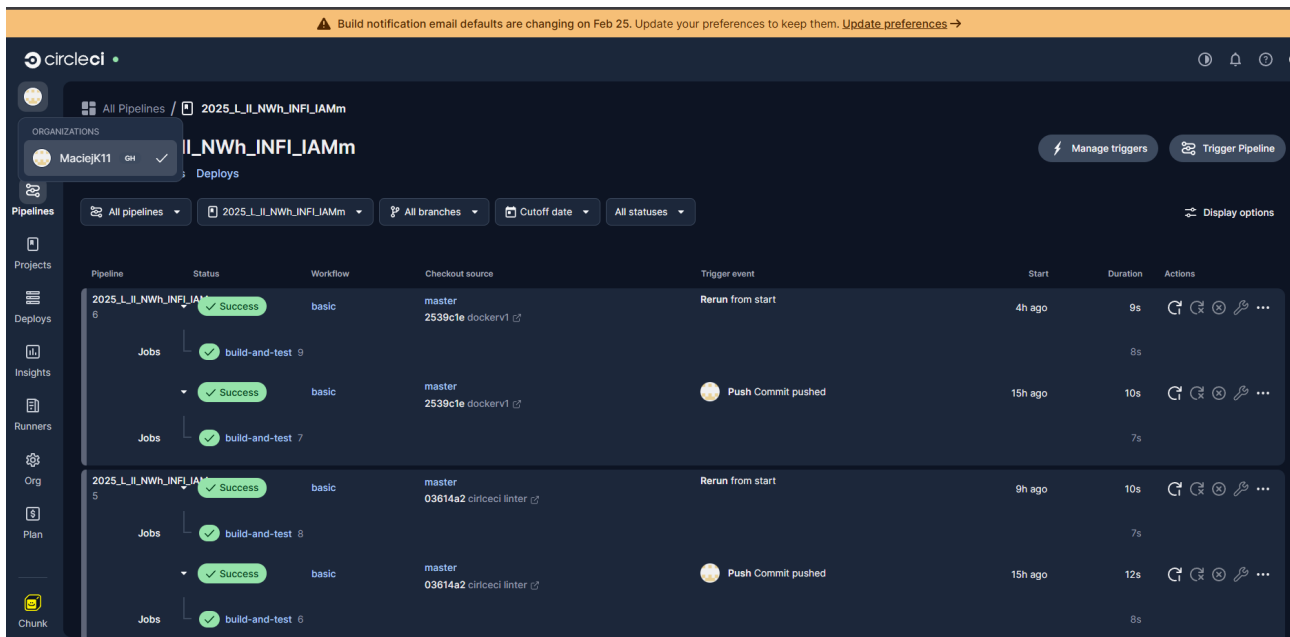
```
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMn$ make docker_run
docker build -t hello-world-printer .
[+] Building 0.7s (12/12) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 279B                                0.0s
=> [internal] load metadata for docker.io/library/python:3        0.4s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/7] FROM docker.io/library/python:3@sha256:151ab3571dad616bb831852e86411e2165295c7f67e 0.0s
=> => resolve docker.io/library/python:3@sha256:151ab3571dad616bb831852e86411e2165295c7f67e 0.0s
=> [internal] load build context                                   0.0s
=> => transferring context: 463B                                      0.0s
=> CACHED [2/7] WORKDIR /tmp                                       0.0s
=> CACHED [3/7] ADD requirements.txt /tmp/requirements.txt        0.0s
=> CACHED [4/7] RUN pip install -r /tmp/requirements.txt          0.0s
=> CACHED [5/7] RUN mkdir -p /usr/src/hello_world_printer        0.0s
=> CACHED [6/7] ADD hello_world/ /usr/src/hello_world_printer/hello_world/ 0.0s
=> CACHED [7/7] ADD main.py /usr/src/hello_world_printer          0.0s
=> exporting to image                                              0.1s
=> => exporting layers                                              0.0s
=> => exporting manifest sha256:f9e000e9e75870dc6948f01b02653039b91a4260b21f5539f32b918924 0.0s
=> => exporting config sha256:ac9783b0fb368b6fd2c633d1cc8c19c3661220273f184d2c067cfbaf5e9af 0.0s
=> => exporting attestation manifest sha256:7bb823d72b4883efa28e51991ab6e4d0856c7143807b36 0.0s
=> => exporting manifest list sha256:b6fa1f22fb85d509a2af16475ffb3db34421fa8202c3fcbdd3ac6f 0.0s
=> => naming to docker.io/library/hello-world-printer:latest      0.0s
=> => unpacking to docker.io/library/hello-world-printer:latest   0.0s
docker rm -f hello-world-printer-dev || true
hello-world-printer-dev
docker run \
  --name hello-world-printer-dev \
  -p 5000:5000 \
  -d hello-world-printer
2f61a5c8f836cc08a463bc9a8216aa5719d60087143638ac526e38d0272c6a
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMn$ docker run hello-world-printer:latest
(.venv) vboxuser@ubuntu24:~/Desktop/MKowal/mkowal_fork/2025_L_II_NWh_INFI_IAMn$
```

Rysunek 23: Docker nie uruchamia aplikacji

Jednak docker nie uruchamia zbudowanej aplikacji, przez co dlasza część zadania jest nie możliwa do wykonania. Próbowałem zmian znaków białych, które mogły być przyczyną błędów na początku, jednak ostatecznie wykonuje obraz tworzony jest bez błędów, co nie zmienia faktu, że cały build jest bez sensu, ponieważ nie udało mi się utworzyć działającej aplikacji.

8 Screenshoty

8.1 CircleCI

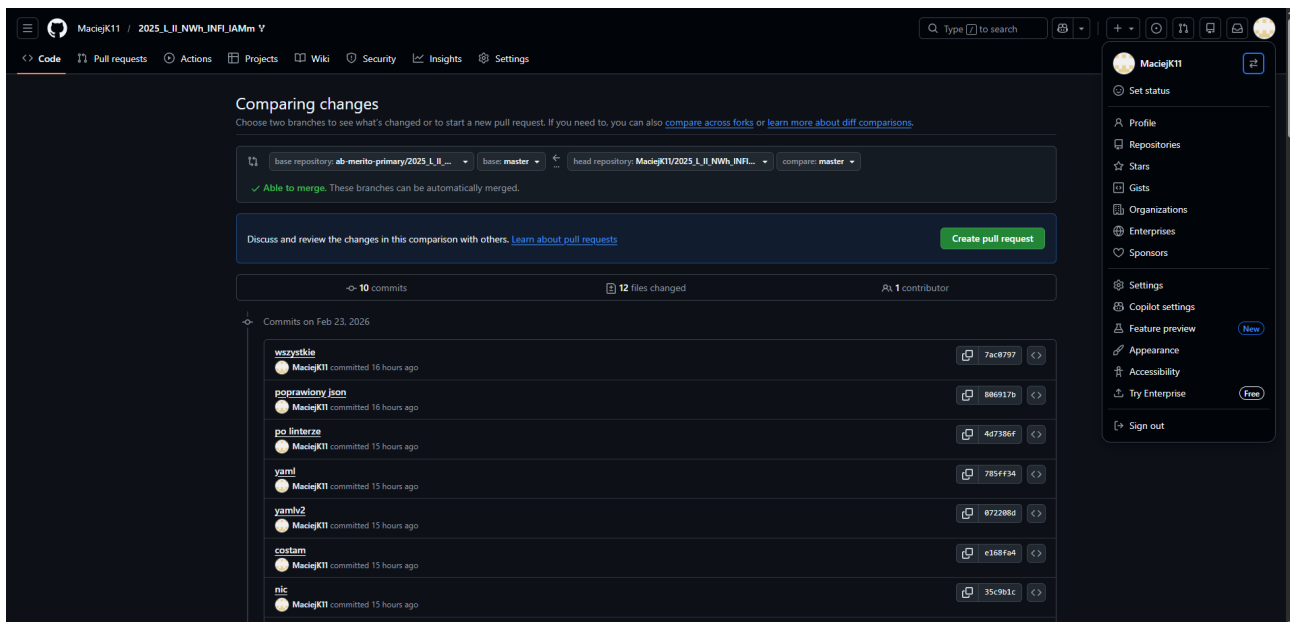


Rysunek 24: CircleCI screenshot

8.2 Dockerhub

Nie wykonano przez problemy.

8.3 Comitty Github



Rysunek 25: Github screenshot