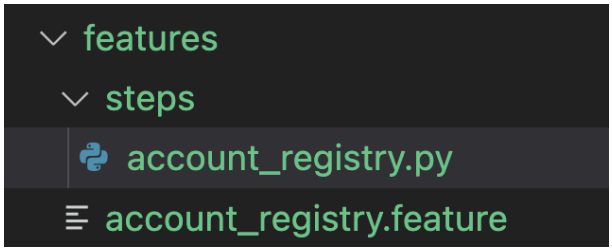
BDD

Na dzisiejszych zajęciach postaramy się zautomatyzować testy API używając języka Gherkin.
Teoria - https://www.softwaretestinghelp.com/cucumber-bdd-tutorial/
Buzzwordy:

- BDD – behavior driven development
- Cucumber – framework używany w BDD
- Gherkin – język opisu scenariuszy
- Given/When/Then/And

Wykonaj poniższe polecenia:

1. Instalacja biblioteki: pip3 install behave
2. Instalacja biblioteki z asercjami: pip3 install unittest_assertions
3. W swoim projekcie stwórz folder **features** a w nim folder **steps**



4. W folderze features stwórz plik **account_registry.feature**:

```
Feature: Account registry

 Scenario: User is able to create a new account
   Given Number of accounts in registry equals: "0"
   When I create an account using name: "kurt", last name: "cobain", pesel: "89092909876"
   Then Number of accounts in registry equals: "1"
   And Account with pesel "89092909876" exists in registry

#  Scenario: User is able to create a second account
   # TODO

 Scenario: User is able to update name of already created account
   Given Account with pesel "89092909876" exists in registry
   When I update "name" of account with pesel: "89092909876" to "russell"
   Then Account with pesel "89092909876" has "name" equal to "russell"

#  Scenario: User is able to update surname of already created account
  # TODO

 Scenario: User is able to delete already created account
   Given Account with pesel "89092909876" exists in registry
   When I delete account with pesel: "89092909876"
```

```
  Then Account with pesel "89092909876" does not exist in registry
  And Number of accounts in registry equals: "0"
   #TODO change to 1 when second account is created

 # Scenario: User is able to delete last account
  # TODO
```

5. W folderze **steps** stwórz plik **account_registry.py**:

```
from behave import *
import requests
from unittest_assertions import AssertEqual

assert_equal = AssertEqual()
URL = "http://localhost:5000"

@when('I create an account using name: "{name}", last name: "{last_name}", pesel: "{pesel}"')
def create_account(context, name, last_name, pesel):
    json_body = { "name": f"{name}",
    "surname": f"{last_name}",
    "pesel": pesel
    }
    create_resp = requests.post(URL + "/api/accounts", json = json_body)
    assert_equal(create_resp.status_code, 201)

@step('Number of accounts in registry equals: "{count}"')
def is_account_count_equal_to(context, count):
    #TODO

@step('Account with pesel "{pesel}" exists in registry')
def check_account_with_pesel_exists(context, pesel):
    #TODO

@step('Account with pesel "{pesel}" does not exist in registry')
def check_account_with_pesel_does_not_exist(context, pesel):
    response = requests.get(URL + f"/api/accounts/{pesel}")
    assert_equal(response.status_code, 404)

@when('I delete account with pesel: "{pesel}"')
def delete_account(context, pesel):
    #TODO

@when('I update "{field}" of account with pesel: "{pesel}" to "{value}"')
def update_field(context, field, pesel, value):
    if field not in ["name", "surname"]:
        raise ValueError(f"Invalid field: {field}. Must be 'name' or 'surname'.")
    json_body = { f"{field}": f"{value}" }
    response = requests.patch(URL + f"/api/accounts/{pesel}", json = json_body)
    assert_equal(response.status_code, 200)

@then('Account with pesel "{pesel}" has "{field}" equal to "{value}"')
def field_equals_to(context, pesel, field, value):
    #TODO
```

6. Uruchom flaska i bazę danych w swoim projekcie
7. TODO:
    a. Dostosuj już istniejący kod w **account_registry.py**
    b. Zaimplementuj brakujący kod w **account_registry.py**
    c. Używając „stepów" zdefiniowanych w **account_registry.py** uzupełnij brakujące scenariusze w pliku **account_registry.feature**

Scenariusze uruchamiamy poleceniem **behave.**

Dla osób zainteresowanych oceną 4 lub wyżej:
Zaimplementuj test BDD który sprawdzi funkcjonalność przelewów (wykonywanie przelewów wychodzących, przychodzących i ekspresowych). Stwórz oddzielne pliki *.features* i *.py*

Przykład funkcjonalności opisanej w Gherkinie:

***Feature**: Wypłata gotówki z bankomatu*
***Scenario**: Udana wypłata gotówki*
***Given** Użytkownik ma konto bankowe*
***And** Użytkownik ma wystarczające środki na koncie*
***And** Bankomat ma wystarczającą ilość gotówki*
***When** Użytkownik wkłada kartę*
***And** Wprowadza poprawny PIN*
***And** Próbuje wypłacić 100 zł*
***Then** Bankomat powinien wypłacić 100 zł*
***And** Saldo użytkownika na koncie powinno zostać zmniejszone o 100 zł*
***And** Bankomat powinien wydrukować potwierdzenie*

***Scenario**: Niewystarczające środki na koncie*
***Given** Użytkownik ma konto bankowe*
***And** Użytkownik ma 50 zł na swoim koncie*
***And** Bankomat ma wystarczającą ilość gotówki*
***When** Użytkownik wkłada kartę*
***And** Wprowadza poprawny PIN*
***And** Próbuje wypłacić 100 zł*
***Then** Bankomat powinien wyświetlić komunikat "Niewystarczające środki"*
***And** Bankomat nie powinien wypłacić żadnej gotówki*
***And** Saldo użytkownika na koncie powinno pozostać bez zmian*

And Bankomat nie powinien wydrukować potwierdzenia