Feature 15 Mikroserwisy:

Nasz kod działa i jest używany przez nasz bank jako „dependency” w systemie monolitowym. Dobra robota!
W tym miesiącu dołączył do naszego departamentu nowy architekt – zwolennik mikroserwisów. Po burzliwej dyskusji zmieniamy podejście do udostępniania naszych funkcjonalności. Musimy „wystawić” trzy API endpointy:

- POST „/accounts” – stworzy konto osobiste i doda je do rejestru. Przykładowe body:

```
{
    "imie": "james",
    "nazwisko": "hetfield",
    "pesel": "89092909825"
}
```

- GET „/accounts/count” - zwraca ilość kont zapisanych w rejestrze
- GET “/accounts/<pesel>” - zwróci dane konta (imię, nazwisko, pesel, saldo) z podanym peselm. Jeżeli konta z podanym peselem nie ma w rejestrze zwracamy 404

Będziemy używali biblioteki Flask - https://flask.palletsprojects.com/en/2.2.x/
Staramy się aby response code naszych endpointów był poprawny. Ściąga - https://http.cat/
Kod odpowiedzialny za API umieśćmy w pliku **app/api.py**
Do wystartowania naszego mikroserwisu wykorzystamy komendę:
flask --app app/api.py --debug run
lub
python3 -m flask --app api.py run

Do stworzenia API możesz użyć wzoru:

```
from flask import Flask, request, jsonify
from app.AccountRegistry import AccountRegistry
from app.PersonalAccount import PersonalAccount

app = Flask(__name__)

@app.route("/api/accounts", methods=['POST'])
def create_account():
   data = request.get_json()
   print(f"Create account request: {data}")
   konto = PersonalAccount(data["name"], data["surname"], data["pesel"])
   AccountRegistry.add_account(konto)
   return jsonify({"message": "Account created"}), 201

@app.route("/api/accounts/<pesel>", methods=['GET'])
def get_account_by_pesel(pesel):
    #implementacja powinna znaleźć się tutaj i powinna zwracać dane konta
```

```
    return jsonify({"imie": "imie"}), 200

@app.route("/api/accounts/<pesel>", methods=['PATCH'])
def update_account(pesel):
    #implementacja powinna znaleźć się tutaj
    return jsonify({"message": "Account updated"}), 200

@app.route("/api/accounts/<pesel>", methods=['DELETE'])
def delete_account(pesel):
    #implementacja powinna znaleźć się tutaj
    return jsonify({"message": "Account deleted"}), 200
```

1. Upewnij się (wysyłając requesty manualnie) czy endpoint do tworzenie konta działa
2. Stwórz test integracyjny który stworzy konto używając API
3. Zaimplementuj pozostałe metody naszego CRUD
4. Stwórz testy automatyczne:
    a. Test, który wyśle zapytanie GET o wyszukanie konta z peselem
    b. Test, który sprawdzi czy zwracamy 404 gdy konta o podanym peselu nie ma w rejestrze
    c. Test na update konta
    d. Test na delete konta

Testy umieszczamy w oddzielnym folderze (np. api_test).
Pamiętajmy w jakiej kolejności odpalają się metody testowe. Postaraj się pisać atomowe testy
Testy API odpalamy komendą np.: ***python3 -m unittest app/api_test/account_crud.py***
Do pisania testów API użyjemy biblioteki unittest oraz **requests**.

uwaga: Operacja PATCH powinna nadpisać tylko pola które otrzyma w body. Reszta parametrów konta pozostaje bez zmian. Jeżeli dopisaliśmy kod aplikacji proszę dodać odpowiednie unit testy.