

Lab03 Synchroniczny interfejs Web WTI - lab03

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/7fad2652-a53d-4b18-917c-48bca972551/lab_WTI_cw.3.pdf

1. Używając biblioteki Pandas przygotuj funkcję łączącą (join) dwie tabele (klasy DataFrame): tabelę zawierającą zawartość pliku userRatedMovies.dat i tabelę zawierającą zawartość pliku movieGenres.dat. Wiersze połączonej tabeli (klasy DataFrame) powinny reprezentować oceny udzielone przez danego użytkownika filmowi (niekoniecznie jawnie określone) o danym zbiorze gatunków. Zawartość powstałej tabeli powinna umożliwiać (w ramach realizacji następnego ćwiczenia lab WTI - patrz uwaga poniżej) sukcesywną (realizowaną "wiersz po wierszu") aktualizację (przez aplikację Web API) średnich ocen udzielonych filmom poszczególnych gatunków (tj. aktualizację m.in. średniej oceny udzielonej komediom, itp.) i na tej podstawie - aktualizację profilu danego użytkownika, przy czym profil ten powinien być zbiorem różnic między średnią oceną filmów danego gatunku a średnią oceną filmów danego gatunku udzieloną przez danego użytkownika. Element (tj. zmienna typu prostego, zmiennoprzecinkowego) reprezentujący ocenę gatunku filmu nie ocenionego przez danego użytkownika powinien mieć wartość równą zero. Uwaga: Wszelkie funkcje inne niż funkcje przygotowania danych i funkcje komunikacyjne (np. funkcje przetwarzania statystycznego) stanowią przedmiot innego ćwiczenia - w wykonywanym ćwiczeniu należy przygotować jedynie ich tzw. zaślepki.

```
# zad01
def getRatedMoviesAndMovieGenres(self):
    ratedMovies = pd.read_csv(
        './userRatedMovies.dat', nrows=100, header=0, delimiter='\t', usecols=['userID', 'movieID', 'rating'])
    movieGenres = pd.read_csv(
        './movieGenres.dat', nrows=100, header=0, delimiter='\t')

    movieGenres['dummyColumn'] = 1
    movieGenresPivoted = movieGenres.pivot_table(
        index='movieID', columns='genre', values='dummyColumn')
    movieGenresPivoted = movieGenresPivoted.fillna(0)

    ratedMoviesAndMovieGenres = pd.merge(
        ratedMovies, movieGenresPivoted, on='movieID')

    return ratedMoviesAndMovieGenres
```

2. Na podstawie wyników poprzedniego punktu ćwiczenia przygotuj funkcję dostarczającą tabelę typu DataFrame. Użyj tej funkcji w funkcji głównej (stanowiącej pomoc przy realizacji - w kolejnych punktach ćwiczenia - aplikacji klienckiej) wykorzystującej metodę iterrows do stworzenia iteratora służącego sukcesywnej konwersji wierszy tabeli na dokumenty w formacie JSON (sukcesywnie "drukowane" w konsoli).

```

if __name__ == '__main__':
    mergedTables = RatedMoviesAndMovieGenresDataFrame()
    data = mergedTables.getMergedTables()

    row_iterator = data.iterrows()

    for row in row_iterator:
        row_as_dict = row[1].to_dict()
        row_as_json = json.dumps(row_as_dict)
        print(row_as_json)

```

3. Używając biblioteki Flask przygotuj uproszczoną, jednowątkową implementację interfejsu typu Web API (zbliżonego do usługi REST, ale nie spełniającej założenia HATEOAS) bazującego na reprezentacji danych w formacie JSON. Interfejs ten (tj. API) powinien umożliwiać zapis (z użyciem metody POST do punktu końcowego /rating) kolejnych wierszy tabeli uzyskanej w poprzednim kroku ćwiczenia, odczyt (z użyciem metody GET, z punktu końcowego usługi /ratings), kasowanie (z użyciem metody DELETE w punkcie końcowym usługi /ratings) oraz odczyt (z użyciem metody GET, z punktu końcowego usługi wspólnego dla wszystkich użytkowników /avg-genre-ratings/all-users) słownika zawierającego “zaślepkowe” (np. losowe) “średnie” oceny udzielone (przez wszystkich użytkowników) filmom poszczególnych gatunków, jak również odczyt aktualnego (również zaślepkowego, np. o losowych wartościach “udających” średnie oceny udzielone filmom poszczególnych gatunków) profilu użytkownika z użyciem metody GET, z punktu końcowego usługi o adresie właściwym dla danego identyfikatora użytkownika (identyfikatora wg. danych pliku userRatedMovies.dat) /avg-genre-ratings/user.

```

from API_SERVICE import API_SERVICE
from flask import Flask, jsonify, request

app = Flask(__name__)
app.config["DEBUG"] = True

class wtiproj03_API:

    @app.route('/rating/<id>', methods=['GET'])
    def getRating(id):
        return jsonify(API_SERVICE().get(id))

    @app.route('/ratings', methods=['GET'])
    def getRatings():
        return jsonify(API_SERVICE().getAll())

    @app.route('/ratings', methods=['DELETE'])
    def deleteRatings():
        return jsonify(API_SERVICE().delete())

    @app.route('/rating', methods=['POST'])
    def createRating():
        return API_SERVICE().create(request.json)

    @app.route('/avg-genre-ratings/all-users', methods=['GET'])
    def getAvgRatings():
        return API_SERVICE().getAvgRatings()

    @app.route('/avg-genre-ratings/user/<user_id>', methods=['GET'])
    def getAvgRating(user_id):

```

```

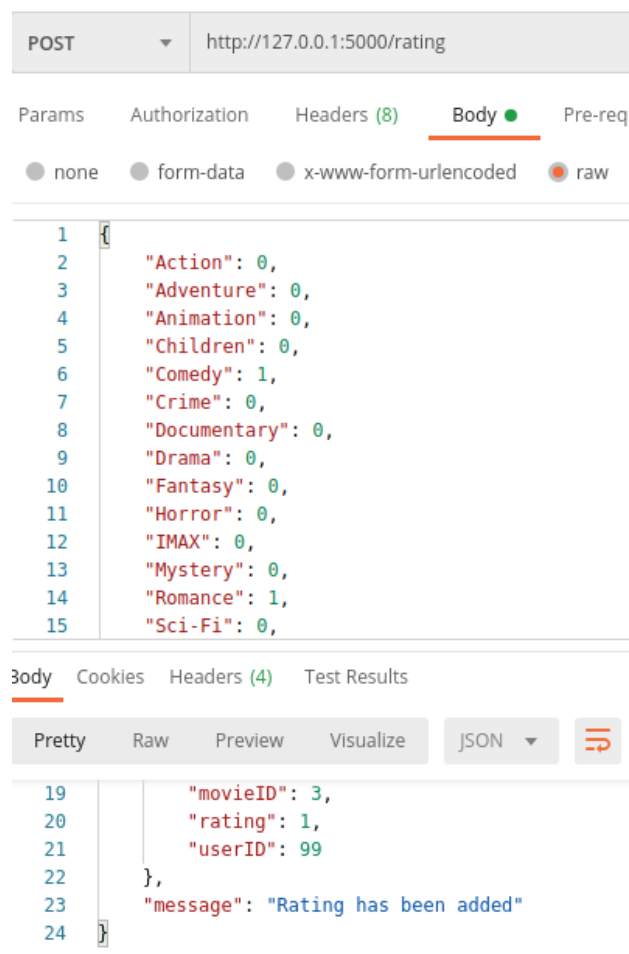
        return API_SERVICE().getAvgRating(user_id)

if __name__ == '__main__':
    wtiproj03_API()
    app.run()

```

4. Przetestuj przygotowaną implementację interfejsu typu Web API (wszystkie punkty końcowe i wszystkie metody HTTP) z użyciem aplikacji Postman. Jako treść ciała komunikatu HTTP POST użyj przykładowego dokumentu JSON wygenerowanego z użyciem implementacji przygotowanej w punkcie 2 ćwiczenia. Zwróć uwagę na poprawne zastosowanie pola nagłówkowego Content-Type.

1. Komunikat HTTP POST do 127.0.0.1:9875/rating



2. Komunikat HTTP GET do 127.0.0.1:9875/ratings

GET http://127.0.0.1:5000/ratings Send Save

Params Authorization **Headers (7)** Body Pre-request Script Tests Settings Cookies Code

Hide auto-generated headers

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Postman-Token	<calculated when request is sent>				
<input checked="" type="checkbox"/> Host	<calculated when request is sent>				
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.26.8				
<input checked="" type="checkbox"/> Accept	*/*				
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br				
<input checked="" type="checkbox"/> Connection	keep-alive				
<input checked="" type="checkbox"/> Content-Type	application/json				
Key	Value	Description			

Body Cookies Headers (4) Test Results Status: 200 OK Time: 26 ms Size: 2.48 KB Save Response

Pretty **Raw** Preview Visualize

```
{
  "Action": 0.0,
  "Adventure": 0.0,
  "Animation": 0.0,
  "Children": 0.0,
```

3. Komunikat HTTP DELETE do 127.0.0.1:9875/ratings

DELETE http://127.0.0.1:5000/ratings

Params Authorization **Headers (6)** Body Pre-request Script Tests Settings

<input checked="" type="checkbox"/> Accept	*/*
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br
<input checked="" type="checkbox"/> Connection	keep-alive
Key	Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ≡

1 []

4. Komunikat HTTP GET na 127.0.0.1:9875/avg-genre-ratings/all-users

GET

http://127.0.0.1:5000/avg-genre-ratings/all-users

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Headers

Hide auto-generated headers

	KEY	VALUE
<input checked="" type="checkbox"/>	Postman-Token ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	Host ⓘ	<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent ⓘ	PostmanRuntime/7.26.8
<input checked="" type="checkbox"/>	Accept ⓘ	*/*
<input checked="" type="checkbox"/>	Accept-Encoding ⓘ	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection ⓘ	keep-alive

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

Visualize

JSON

```


1  {
2    "genre-Action": 0.9405725905029553,
3    "genre-Adventure": 0.4713245210189484,
4    "genre-Animation": 0.8036375619049675,
5    "genre-Children": 0.07678446180418652,
6    "genre-Comedy": 0.10732800457520975,
7    "genre-Crime": 0.9009335036890221,
8    "genre-Documentary": 0.9075237309793135,
9    "genre-Drama": 0.7314669905328793,
10   "genre-Fantasy": 0.7806586962393711,
11   "genre-Horror": 0.3309467740429818,
12   "genre-IMAX": 0.932343704284553,
13   "genre-Mystery": 0.47941784167859613,
14   "genre-Romance": 0.2538637824354,
15   "genre-Sci-Fi": 0.5275028358730901,
16   "genre-Thriller": 0.08445075166426241,
17   "genre-War": 0.5472205251625385
18 }







```

5. Komunikat HTTP GET na 127.0.0.1:9875/avg-genre-ratings/78


GET http://127.0.0.1:5000/avg-genre-ratings/user/78

Params Authorization **Headers (6)** Body Pre-request Script Tests Set

Headers  Hide auto-generated headers

	KEY	VALUE
<input checked="" type="checkbox"/>	Postman-Token 	<calculated when request is sent>
<input checked="" type="checkbox"/>	Host 	<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent 	PostmanRuntime/7.26.1
<input checked="" type="checkbox"/>	Accept 	*/*
<input checked="" type="checkbox"/>	Accept-Encoding 	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection 	keep-alive
	Key	Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON 

```

7      "genre-Animation": 0.449770111081097,
8      "genre-Children": 0.7826158884520569,
9      "genre-Comedy": 0.8997388985426368,
10     "genre-Crime": 0.6617384244040447,
11     "genre-Documentary": 0.6952939253203911,
12     "genre-Drama": 0.9014299306057656,
13     "genre-Fantasy": 0.1724915404199604,
14     "genre-Horror": 0.2315147004125171,
15     "genre-IMAX": 0.6652606540852848,
16     "genre-Mystery": 0.06151094402868418,
17     "genre-Romance": 0.6140516058829094,
18     "genre-Sci-Fi": 0.03406087245707423,
19     "genre-Thriller": 0.04073310921081297,
20     "genre-War": 0.43254752870447266
21 },
22 "user_id": "78"
23 }
24 }
```

```

import json
from flask import Flask
from wtiproj03 import RatedMoviesAndMovieGenresDataFrame
import random

app = Flask(__name__)
app.config["DEBUG"] = True

class API_SERVICE:

    def __init__(self):
        self.rawRatingData = self.getAll()

    def getAll(self):
```

```

data = RatedMoviesAndMovieGenresDataFrame().getRatedMoviesAndMovieGenres()

return data.to_dict('records')

def delete(self):
    self.rawRatingData = []
    return self.rawRatingData

def create(self, json_data):
    response = {}
    response["message"] = "Rating has been added"
    response["data"] = json_data

    return response

def getAvgRatings(self):
    dummy_avg_genre_ratings = {}

    for rawRatingDataItem in self.rawRatingData:
        rawRatingDataItemKeys = list(rawRatingDataItem)
        for rawRatingDataItemKey in rawRatingDataItemKeys:
            if rawRatingDataItemKey[:6] == "genre-":
                dummy_avg_genre_ratings[rawRatingDataItemKey] = random.random()

    return dummy_avg_genre_ratings

def getAvgRating(self, user_id):
    response = {}

    dummy_avg_genre_ratings_for_user = {}

    for rawRatingDataItem in self.rawRatingData:
        if rawRatingDataItem["userID"] == int(user_id):
            rawRatingDataItemKeys = list(rawRatingDataItem)
            for rawRatingDataItemKey in rawRatingDataItemKeys:
                if rawRatingDataItemKey[:6] == "genre-":
                    dummy_avg_genre_ratings_for_user[rawRatingDataItemKey] = random.random()

    response["data"] = {
        "user_id": user_id,
        "ratings": dummy_avg_genre_ratings_for_user,
        "message": "User not exists" if not dummy_avg_genre_ratings_for_user.keys() else ""
    }

    return response

if __name__ == '__main__':
    api_service = API_SERVICE()

```

5. Używając biblioteki Requests przygotuj testową aplikację klienta przygotowanego wcześniej interfejsu typu API wyposażonego w funkcję “drukowania w konsoli” danych wszystkich zapytań i wszystkich odpowiedzi. Wprowadź odstęp czasowy 10 ms między wysłaniem dwóch kolejnych wiadomości (time.sleep(0.01)).

```

import requests
import time

class wtproj03_API_CLIENT:

```

```

def __init__(self):
    self.endpoint = "http://127.0.0.1:5000/"

def getAll(self):
    r = requests.get(self.endpoint + "ratings")
    print('ratings/', r.status_code)

def add(self):
    r = requests.post(self.endpoint + "rating", data={
        "Action": 0,
        "movieID": 3,
        "rating": 1,
        "userID": 99
    })
    print('add rating/', r.status_code)

def delete(self):
    r = requests.delete(self.endpoint + "ratings")
    print('delete ratings/', r.status_code)

def getAvgRatings(self):
    r = requests.get(self.endpoint + "avg-genre-ratings/all-users")
    print('avg-genre-ratings/all-users/', r.status_code)

def getAvgRating(self, id):
    r = requests.get(self.endpoint + "avg-genre-ratings/user/" + str(id))
    print('avg-genre-ratings/user/' + str(id), r.status_code)

if __name__ == '__main__':
    client = wtproj03_API_CLIENT()
    client.getAll()
    time.sleep(0.01)
    client.add()
    time.sleep(0.01)
    client.delete()
    time.sleep(0.01)
    client.getAvgRatings()
    time.sleep(0.01)
    client.getAvgRating(78)

```

```

ratings/ 200
add ratings/ 200
delete ratings/ 200
avg-genre-ratings/all-users/ 200
avg-genre-ratings/user/78 200

```

6. Zademonstruj wykorzystanie w aplikacji klienckiej wszystkich funkcji przygotowanego wcześniej interfejsu typu Web API.

ODP. Screeny z Postman w zadaniu nr.4