**Управление качеством программных систем**

**Мерзлова Анастасия, ПИ21-2**

**Задание 1**

*git https://github.com/mrayuu/development-management*

Начнем с наших данных. Вставляем их в файл с помощью запроса **post:**

***products.json***

[{"id": "4", "name": "Apple iPhone 11, 64GB", "data": {"price": 389.99, "color": "Purple"}}, {"id": "5", "name": "Samsung Galaxy Z Fold2", "data": {"price": 689.99, "color": "Brown"}}, {"id": "6", "name": "Apple AirPods", "data": {"generation": "3rd", "price": 120}}, {"id": "10", "name": "Apple iPad Mini 5th Gen", "data": {"Capacity": "64 GB", "Screen size": 7.9}}, {"id": "12", "name": "Apple iPad Air", "data": {"Generation": "4th", "Price": "419.99", "Capacity": "64 GB"}}, {"id": "13", "name": "Apple iPad Air", "data": {"Generation": "4th", "Price": "519.99", "Capacity": "256 GB"}}]

***main.py***

import requests

import json

import base64

# данные для добавления новых объектов

new\_data = [

{"id": "4", "name": "Apple iPhone 11, 64GB", "data": {"price": 389.99, "color": "Purple"}},

{"id": "5", "name": "Samsung Galaxy Z Fold2", "data": {"price": 689.99, "color": "Brown"}},

{"id": "6", "name": "Apple AirPods", "data": {"generation": "3rd", "price": 120}},

{"id": "10", "name": "Apple iPad Mini 5th Gen", "data": {"Capacity": "64 GB", "Screen size": 7.9}},

{"id": "12", "name": "Apple iPad Air", "data": {"Generation": "4th", "Price": "419.99", "Capacity": "64 GB"}},

{"id": "13", "name": "Apple iPad Air", "data": {"Generation": "4th", "Price": "519.99", "Capacity": "256 GB"}}

]

additional\_data = [

{"id": "16", "name": "Logitech MX Master 3", "data": {"type": "Wireless", "price": 99.99}},

{"id": "17", "name": "Dell XPS 13", "data": {"Processor": "Intel Core i7", "price": 1299.99}}

]

token = 'ghp\_3AWDtYipigpQG0XsVCIo8nkPx516GD4J3lbb'

# URL для обновления (PUT) файла products.json

url = 'https://api.github.com/repos/mrayuu/development-management/contents/products.json'

headers = {

'Authorization': f'Token {token}',

'Accept': 'application/vnd.github.v3+json'

}

# Получение текущего содержимого файла

response\_get = requests.get(url, headers=headers)

data\_get = response\_get.json()

current\_sha = data\_get['sha']

# Кодирование нового текста в Base64

encoded\_content = base64.b64encode(json.dumps(new\_data, ensure\_ascii=False).encode()).decode()

data = {

'message': 'Обновление файла products.json',

'content': encoded\_content,

'sha': current\_sha

}

# PUT-запрос для обновления файла с новым содержимым

response\_put = requests.put(url, headers=headers, json=data)

if response\_put.status\_code == 200:

print(f"Данные успешно обновлены в файле products.json.")

else:

print(f"Ошибка {response\_put.status\_code}: {response\_put.text}")

# Затем делаем GET-запрос для получения содержимого файла

response\_get = requests.get(url, headers=headers)

data\_get = response\_get.json()

if response\_get.status\_code == 200:

# Декодируем содержимое из Base64

decoded\_content = base64.b64decode(data\_get['content']).decode('utf-8')

print(f"Содержимое products.json перед удалением:\n{decoded\_content}")

# Прочитаем данные и удалим два элемента

products\_data = json.loads(decoded\_content)

products\_data = products\_data[2:] # Удалим два элемента (например, первые два)

# Кодируем обновленные данные в формат JSON с ensure\_ascii=False

updated\_json\_data = json.dumps(products\_data, ensure\_ascii=False)

# Создаем DELETE-запрос для удаления файла

data\_delete = {

'message': 'Удаление двух элементов из файла products.json',

'sha': data\_get['sha']

}

response\_delete = requests.delete(url, headers=headers, json=data\_delete)

if response\_delete.status\_code == 200:

print(f"Два элемента успешно удалены из файла products.json.")

# Создаем POST-запрос для создания файла с обновленными данными

data\_post = {

'message': 'Добавление файла products.json',

'content': base64.b64encode(updated\_json\_data.encode()).decode()

}

response\_post = requests.put(url, headers=headers, json=data\_post)

if response\_post.status\_code == 200:

print(f"Обновленные данные успешно записаны в файл products.json.")

else:

print(f"Ошибка {response\_post.status\_code}: {response\_post.text}")

# В случае ошибки при записи, попробуем восстановить предыдущие данные

restore\_data = {

'message': 'Восстановление предыдущих данных в файл products.json',

'content': data\_get['content'],

'sha': data\_get['sha']

}

requests.put(url, headers=headers, json=restore\_data)

else:

print(f"Ошибка {response\_delete.status\_code}: {response\_delete.text}")

else:

print(f"Ошибка {response\_get.status\_code}: {response\_get.text}")

# Затем делаем GET-запрос для получения содержимого обновленного файла

response\_get\_updated = requests.get(url, headers=headers)

data\_get\_updated = response\_get\_updated.json()

if response\_get\_updated.status\_code == 200:

# Декодируем содержимое из Base64

decoded\_content\_updated = base64.b64decode(data\_get\_updated['content']).decode('utf-8')

print(f"\nОбновленное содержимое products.json:\n{decoded\_content\_updated}")

else:

print(f"Ошибка {response\_get\_updated.status\_code}: {response\_get\_updated.text}")

# Чтение текущего содержимого из Base64

decoded\_content\_before = base64.b64decode(data\_get['content']).decode('utf-8')

print(f"\nСодержимое products.json перед добавлением новых данных:\n{decoded\_content\_before}")

# Добавление новых данных к текущему списку

products\_data\_additional = json.loads(decoded\_content\_before)

products\_data\_additional.extend(additional\_data)

# Кодирование обновленных данных в формат JSON с ensure\_ascii=False

updated\_json\_data\_additional = json.dumps(products\_data\_additional, ensure\_ascii=False)

# Создаем PUT-запрос для обновления файла с новыми данными

data\_additional = {

'message': 'Добавление новых данных в файл products.json',

'content': base64.b64encode(updated\_json\_data\_additional.encode()).decode(),

'sha': data\_get['sha']

}

response\_put\_additional = requests.put(url, headers=headers, json=data\_additional)

if response\_put\_additional.status\_code == 200:

print(f"Новые данные успешно добавлены в файл products.json.")

else:

print(f"Ошибка {response\_put\_additional.status\_code}: {response\_put\_additional.text}")

# Затем делаем GET-запрос для получения содержимого обновленного файла

response\_get\_updated\_additional = requests.get(url, headers=headers)

data\_get\_updated\_additional = response\_get\_updated\_additional.json()

if response\_get\_updated\_additional.status\_code == 200:

# Декодируем содержимое из Base64

decoded\_content\_updated\_additional = base64.b64decode(data\_get\_updated\_additional['content']).decode('utf-8')

print(f"\nОбновленное содержимое products.json после добавления новых данных:\n{decoded\_content\_updated\_additional}")

else:

print(f"Ошибка {response\_get\_updated\_additional.status\_code}: {response\_get\_updated\_additional.text}")

Наш код корректно обрабатывает get, post, delete запросы:

****

******

Далее формируем из наших запросов отдельный файл с проверочными тестами на все использованные запросы: сначала получаем доступ (get), затем вставляем данные (post), далее смотрим на результат (get), удаляем (delete), снова смотрим (get). По какой-то причине не все тесты обрабатываются корретктно

***test\_api\_operations.py***

import requests

import json

import base64

url = 'https://api.github.com/repos/mrayuu/development-management/contents/products.json'

headers = {

'Authorization': f'Token {token}',

'Accept': 'application/vnd.github.v3+json'

}

def test\_api\_connection():

# Тестирование подключения к API (GET)

response = requests.get(url, headers=headers)

assert response.status\_code == 200, f"Ошибка при подключении к API. Код: {response.status\_code}"

def test\_post\_data():

# Тестирование добавления данных (POST)

new\_data = [

{"id": "4", "name": "Apple iPhone 11, 64GB", "data": {"price": 389.99, "color": "Purple"}},

{"id": "5", "name": "Samsung Galaxy Z Fold2", "data": {"price": 689.99, "color": "Brown"}},

{"id": "6", "name": "Apple AirPods", "data": {"generation": "3rd", "price": 120}},

{"id": "10", "name": "Apple iPad Mini 5th Gen", "data": {"Capacity": "64 GB", "Screen size": 7.9}},

{"id": "12", "name": "Apple iPad Air", "data": {"Generation": "4th", "Price": "419.99", "Capacity": "64 GB"}},

{"id": "13", "name": "Apple iPad Air", "data": {"Generation": "4th", "Price": "519.99", "Capacity": "256 GB"}}

]

# Кодирование нового текста в Base64

encoded\_content = base64.b64encode(json.dumps(new\_data, ensure\_ascii=False).encode()).decode()

data = {

'message': 'Добавление тестовых данных в файл products.json',

'content': encoded\_content

}

response\_post = requests.put(url, headers=headers, data=json.dumps(data))

assert response\_post.status\_code == 200, f"Ошибка при добавлении данных. Код: {response\_post.status\_code}"

def test\_get\_data():

# Тестирование получения данных (GET)

response\_get = requests.get(url, headers=headers)

assert response\_get.status\_code == 200, f"Ошибка при получении данных. Код: {response\_get.status\_code}"

def test\_delete\_data():

# Тестирование удаления данных (DELETE)

response\_get = requests.get(url, headers=headers)

assert response\_get.status\_code == 200, f"Ошибка при получении данных перед удалением. Код: {response\_get.status\_code}"

# Декодируем содержимое из Base64

decoded\_content = base64.b64decode(response\_get.content).decode('utf-8', errors='replace')

try:

products\_data = json.loads(decoded\_content)

except json.JSONDecodeError:

assert False, "Ошибка при декодировании JSON."

# Удаляем первые два элемента (например)

products\_data = products\_data[2:]

# Кодируем обновленные данные в формат JSON с ensure\_ascii=False

updated\_json\_data = json.dumps(products\_data, ensure\_ascii=False)

# Получаем SHA текущего файла

sha = response\_get.json()['sha']

# Создаем PUT-запрос для обновления файла с обновленными данными

data\_put = {

'message': 'Обновление файла products.json после удаления данных',

'content': base64.b64encode(updated\_json\_data.encode()).decode(),

'sha': sha

}

response\_put = requests.put(url, headers=headers, data=json.dumps(data\_put))

assert response\_put.status\_code == 200, f"Ошибка при обновлении файла после удаления данных. Код: {response\_put.status\_code}"

def test\_get\_data\_after\_delete():

# Тестирование получения данных после удаления (GET)

response\_get\_after\_delete = requests.get(url, headers=headers)

assert response\_get\_after\_delete.status\_code == 200, f"Ошибка при получении данных после удаления. Код: {response\_get\_after\_delete.status\_code}"

# Декодируем содержимое из Base64 с явным указанием кодировки

decoded\_content\_after\_delete = base64.b64decode(response\_get\_after\_delete.content).decode('utf-8', errors='replace')

try:

products\_data\_after\_delete = json.loads(decoded\_content\_after\_delete)

except json.JSONDecodeError:

assert False, "Ошибка при декодировании JSON."

# Добавьте проверку, что элементы действительно удалены (например, длина списка)

assert len(products\_data\_after\_delete) == len(products\_data) - 2, "Элементы не были удалены успешно."