

Министерство образования республики Молдова
Технический Университет Молдовы
Департамент Программной Инженерии и Автоматики

О т ч ё т

Лабораторная работа №3

По предмету: Programarea in retea

Выполнил студ. гр. SI-202

Абабий Эдуард

Проверил

Лях Аркадий

Кишинёв – 2023

Для начала создал файл server.py, (локальный сервер), который будет обрабатывать запросы GET POST HEAD OPTIONS

```
from flask import Flask, request, Response, jsonify
import json
from datetime import datetime
```

Импортируем все необходимые зависимости, для создания сервера использую инструмент flask

```
@app.route('/get_data_guid', methods=['GET'])
def get_data_guid():
    guid = request.args.get('guid')
    with open('data.json') as f:
        data = json.load(f)
    for object in data['users']:
        if object['id'] == guid:
            return {'process': True,
                    'content': object}
    return {
        'process': False,
        'content': ''
    }
```

В данной функции определен обработчик маршрута /get_data_guid, который принимает запросы только методом GET. Ожидается, что в параметрах запроса будет передан параметр guid.

При обработке запроса, сначала получается значение параметра guid через объект request.args. Затем открывается файл data.json и загружается в объект data.

Далее, выполняется цикл по всем объектам, содержащимся в массиве users из объекта data. Если находится объект с id, соответствующим переданному параметру guid, то он возвращается в виде JSON-объекта с помощью функции return. Если ни один объект не соответствует переданному guid, то возвращается JSON-объект с ключом process, равным False, и пустым значением в ключе content.

Таким образом, данная функция ищет объект с заданным guid в массиве объектов users из data.json и возвращает его в JSON-формате, если он найден. Если же объект не найден, возвращается JSON-объект с флагом process, указывающим на неудачу, и пустым содержимым.

```
@app.route('/process_data', methods=['POST'])
def process_data():
    guid = request.args.get('guid')
    data = request.get_json()
    print(data)
    print(guid)

    with open('data.json') as f:
        db = json.load(f)
        for object in db['users']:
            if object['id'] == guid:
                object['name'] = data['name']
                object['nick_name'] = data['userName']
                object['phone'] = data['phone']
                object['address'] = data['address']
                object['pass id'] = data['pass_id']
            f.close()
        with open('data.json', 'w') as f:
            json.dump(db, f, indent=4)
        return {
            'process': True,
            'content': ''
        }
    return {
        'process': False,
        'content': ''
    }
```

Это обработчик POST-запроса на адрес '/process_data'. При получении такого запроса сервер должен прочитать параметр 'guid' из URL-адреса, данные в формате JSON из тела запроса, и затем обновить запись о пользователе с соответствующим идентификатором (guid) в базе данных, используя данные из тела запроса.

Затем происходит следующее:

Сначала читается файл 'data.json' с помощью оператора 'with open() as f', чтобы получить доступ к данным в файле.

Затем данные в файле загружаются в переменную db с помощью функции json.load().

После этого сервер перебирает все объекты в массиве 'users' и, если находит объект с идентификатором, соответствующим guid, обновляет его поля с помощью данных из тела запроса.

После этого файл закрывается с помощью f.close() и данные сохраняются в файл с помощью json.dump() в том же формате, что и изначально (с отступами indent=4).

В конце функция возвращает JSON-объект, который сообщает клиенту о том, прошло ли обновление успешно. Если обновление было успешно, значение 'process' равно True, иначе False.

```
@app.route('/get_head', methods=['HEAD'])
def get_head():
    current_time_micro = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
    headers = {'time': current_time_micro}
    return Response('', headers=headers)
```

Данный код является обработчиком маршрута для HTTP-метода HEAD в рамках веб-приложения Flask. HTTP-метод HEAD используется для получения только заголовков ответа от сервера без тела ответа.

При обращении к данному маршруту методом HEAD сервер возвращает пустой ответ, но в заголовки ответа добавляется пользовательский заголовок 'time', который содержит текущее время в микросекундах в определенном формате.

Таким образом, когда пользователь обратится к данному маршруту методом HEAD, сервер вернет пустой ответ с добавленным пользовательским заголовком 'time', который содержит текущее время в микросекундах в определенном формате.

```

@app.route('/get_options', methods=['OPTIONS'])
def get_options():
    response = jsonify({
        'message': 'This is an OPTIONS request!',
        'allowed_methods': ['GET', 'POST', 'PUT', 'DELETE']
    })
    response.headers.add('Access-Control-Allow-Origin', '*')
    response.headers.add('Access-Control-Allow-Headers', 'Content-Type,Authorization')
    response.headers.add('Access-Control-Allow-Methods', 'OPTIONS,HEAD,GET,POST,PUT,DELETE')
    return response

```

Когда пользователь обращается к данному маршруту методом OPTIONS, сервер возвращает JSON-объект в ответе, содержащий сообщение 'This is an OPTIONS request!' и список доступных методов ('GET', 'POST', 'PUT', 'DELETE').

Для того, чтобы клиентский код мог обращаться к данному маршруту с других доменов, сервер добавляет в заголовки ответа следующие поля:

'Access-Control-Allow-Origin': '*' - разрешает доступ к ресурсу со всех доменов

'Access-Control-Allow-Headers': 'Content-Type,Authorization' - разрешает использование заголовков 'Content-Type' и 'Authorization'

'Access-Control-Allow-Methods': 'OPTIONS,HEAD,GET,POST,PUT,DELETE' - разрешает использование HTTP-методов 'OPTIONS', 'HEAD', 'GET', 'POST', 'PUT' и 'DELETE'

Таким образом, при обращении к данному маршруту методом OPTIONS, сервер вернет JSON-объект с информацией о доступных методах и свойствах ресурса, а также добавит в заголовки ответа разрешения для обращения к ресурсу из других доменов.

```

{
  "users": [
    {
      "id": "3d627c0b-362a-4ff7-a5ff-628e5f2d34c3",
      "email": "ababiy495@gmail.com",
      "nick_name": "JonyHowman",
      "name": "Edward",
      "phone": "+37360597059",
      "address": "Ulita",
      "pass id": "357 918 345"
    }
  ]
}

```

Данный код представляет собой JSON-объект, содержащий список пользователей с их информацией.

Каждый пользователь представлен в объекте в виде словаря, который содержит следующие поля:

"id": уникальный идентификатор пользователя

"email": адрес электронной почты пользователя

"nick_name": никнейм пользователя

"name": имя пользователя

"phone": номер телефона пользователя

"address": адрес пользователя

"pass id": идентификационный номер паспорта пользователя

Объект содержит список из трех пользователей. Каждый пользователь имеет уникальный идентификатор "id". Каждое поле содержит соответствующую информацию о пользователе.

Данный объект может использоваться для передачи информации о пользователях между приложениями или компонентами программного обеспечения. Он может быть обработан и представлен в удобном виде с помощью кода на разных языках программирования, например, на Python, JavaScript, Java и др.

Я его создал, как псевдо базу данных, которая будет обрабатываться во время запросов

Затем я взялся за создание небольшого GUI приложения

MainWindow

guid

Get

Post

Head

Options

user name

phone

name

address

pass id

У нас есть несколько полей ввода с placeholder.

Guid – поле ввода для идентификатор пользователя

User name – поле для ввода ника пользователя (для дальнейшей смены)

Phone - поле для ввода телефона пользователя (для дальнейшей смены)

Name - поле для ввода имени пользователя (для дальнейшей смены)

Address – поле для ввода адреса пользователя (для дальнейшей смены)

Pass id – поле для ввода паспортных данных (для дальнейшей смены)

Большое поле, которое пустое, оно нам понадобится для вывода результата запросов

Думаю объяснять смысл кнопок не нужно, каждая создаёт определенный запрос

```
def get_data(self):
    guid = self.GuidInput.toPlainText()
    response = requests.get(f"http://localhost:5000/get_data_guid?guid={guid}")
    if not response.json()['process']:
        self.OutputContainer.setPlainText("This user doesn't exist")
    else:
        data = response.json()['content']
        self.OutputContainer.setPlainText(f'Name: {data["name"]}\n'
                                          f'User name: {data["nick_name"]}\n'
                                          f'Email: {data["email"]}\n'
                                          f'Address: {data["address"]}\n'
                                          f'Pass ID: {data["pass_id"]}\n'
                                          f'Phone: {data["phone"]}\n')
```

Данный код является методом класса и представляет собой клиентскую часть для получения данных о пользователе из удаленного сервера.

Метод принимает GUID пользователя в виде строки, которая извлекается из виджета GuidInput. Затем метод отправляет GET-запрос на сервер с параметром guid, содержащим переданный GUID.

Если на сервере не найден пользователь с переданным GUID, метод выводит сообщение об этом в виджет Output Container. В противном случае метод получает данные о пользователе в формате JSON из поля "content" в ответе сервера и выводит информацию о пользователе в виджет OutputContainer в виде строки с помощью f-строки, содержащей соответствующие поля: имя, никнейм, адрес, идентификационный номер паспорта, электронная почта и телефон.

```
def post_data(self):
    guid = self.GuidInput.toPlainText()
    data = {
        "name": self.NameInput.toPlainText(),
        "userName": self.UserNameInput.toPlainText(),
        "address": self.AddressInput.toPlainText(),
        "phone": self.PhoneInput.toPlainText(),
        "pass_id": self.PassInput.toPlainText()
    }
    response = requests.post(f'http://localhost:5000/process_data?guid={guid}', json=data)
    if not response.json()['process']:
        self.OutputContainer.setPlainText("This user doesn't exist")
    else:
        self.OutputContainer.setPlainText("Data was changed")
```


Данный код является методом класса и представляет собой клиентскую часть для отправки данных о пользователе на удаленный сервер.

Метод принимает GUID пользователя в виде строки, который извлекается из виджета GuidInput, а также данные о пользователе, которые заполняются в соответствующих виджетах NameInput, UserNameInput, AddressInput, PhoneInput и PassInput. Далее данные пользователя упаковываются в формат JSON и отправляются на сервер методом POST с указанием GUID в параметре запроса.

Если на сервере не найден пользователь с переданным GUID, метод выводит сообщение об этом в виджет Output Container. В противном случае метод выводит сообщение об успешной отправке данных в виджет OutputContainer.

```
def get_head(self):  
    response = requests.head('http://localhost:5000/get_head')  
    self.OutputContainer.setPlainText(str(response)+' / '+str(response.headers.get('time')))
```

Данный метод отправляет HTTP-запрос методом HEAD на адрес http://localhost:5000/get_head.

В ответ на этот запрос сервер должен вернуть только заголовки ответа без тела.

Затем, метод получает ответ от сервера, извлекает значение заголовка time и выводит его в Output Container.

Заголовок time был добавлен на сервере в ответ на запрос методом HEAD, вернув пустое тело ответа, чтобы клиент мог узнать текущее время на сервере.

```
def get_options(self):  
    response = requests.options('http://localhost:5000/get_options')  
    self.OutputContainer.setPlainText(str(response)+' '+str(response.headers.get('Access-Control-Allow-Methods')))
```

Этот метод отправляет HTTP-запрос методом OPTIONS на адрес http://localhost:5000/get_options.

Метод OPTIONS используется для запроса информации о доступных методах для определенного ресурса на сервере.

В ответ на этот запрос сервер должен вернуть список разрешенных методов и другую информацию.

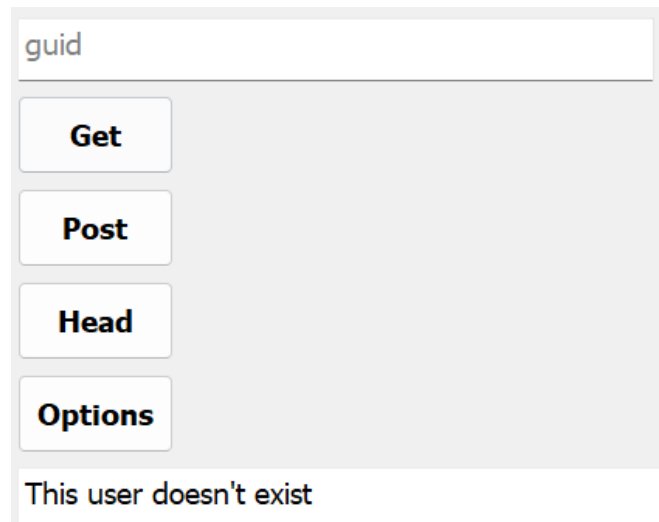
Затем, метод получает ответ от сервера, извлекает значение заголовка Access-Control-Allow-Methods и

выводит его в OutputContainer.

Этот заголовок был добавлен на сервере в ответ на запрос методом OPTIONS, чтобы указать, какие методы HTTP разрешены для данного ресурса и могут быть использованы клиентом.

Результат работы запросов:

Обработка запроса GET без указания GUID пользователя



guid

Get

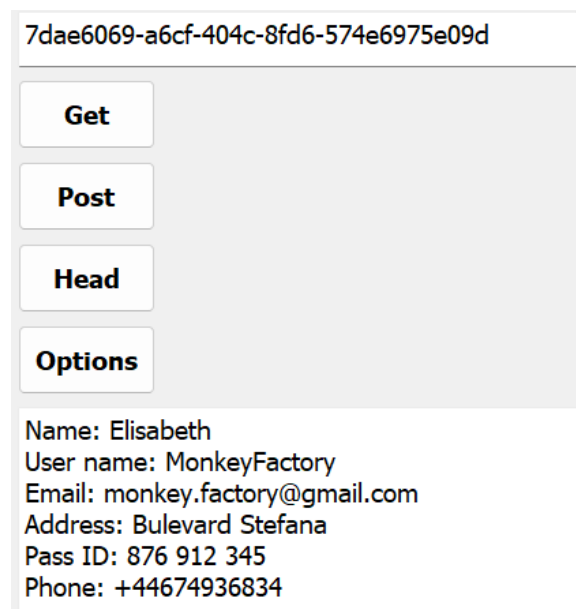
Post

Head

Options

This user doesn't exist

С указанием GUID пользователя



7dae6069-a6cf-404c-8fd6-574e6975e09d

Get

Post

Head

Options

Name: Elisabeth
User name: MonkeyFactory
Email: monkey.factory@gmail.com
Address: Bulevard Stefana
Pass ID: 876 912 345
Phone: +44674936834

7dae6069-a6cf-404c-8fd6-574e6975e09d	Pavel Volea
Get	+7384859483
Post	Papa Francisco
Head	Vatikan
Options	777 777 777

Data was changed

Обработка запроса POST с данными
Результат запроса

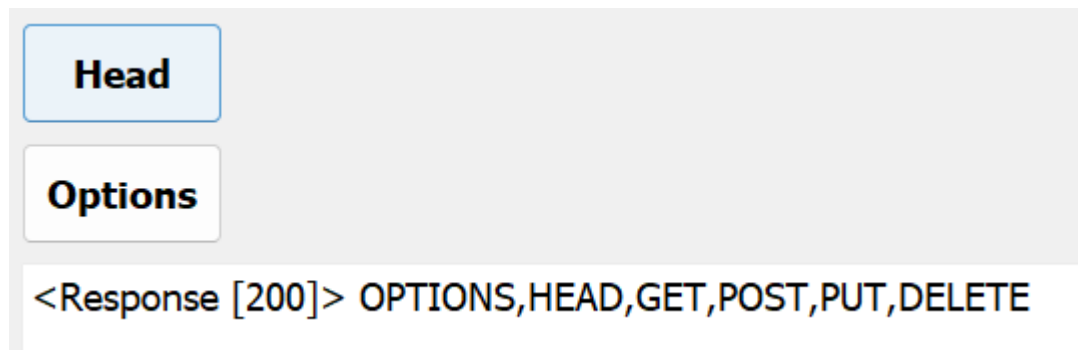
```
{
  "id": "7dae6069-a6cf-404c-8fd6-574e6975e09d",
  "email": "monkey.factory@gmail.com",
  "nick_name": "Pavel Volea",
  "name": "Papa Francisco",
  "phone": "+7384859483",
  "address": "Vatikan",
  "pass id": "777 777 777"
}
```

Результат запроса HEAD

Head
Options

<Response [200]> / 2023-04-02 22:09:54.135785

Результат запроса OPTIONS



Вывод: В данной лабораторной работе из нового для себя я узнал инструмент flask для создания локального сервера обработчика запросов, но с ним ещё предстоит познакомиться получше. Создал небольшое GUI приложение для обработки запросов GET POST HEAD OPTIONS

