

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ № 1

по теме: **РАБОТА С СОКЕТАМИ.**

по дисциплине: Web-программирование

Специальность:

45.03.04 Интеллектуальные системы в гуманитарной сфере

Проверил:

Говоров А.И. _____

Дата: «25» сентября 2021г.

Оценка _____

Выполнила:

студентка

группы К33422

Редичкина А.М

Санкт-Петербург

2021

ЦЕЛЬ РАБОТЫ

Овладеть практическими навыками и умениями реализации web-серверов и использования сокетов.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

1. Реализовать клиентскую и серверную часть приложения. Клиент посылает серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ посылает клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.

Содержимое файла client.py:

```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(("127.0.0.1", 5635))
conn.send(b"Hello, server! \n")
data = conn.recv(1024)
print(data.decode("utf-8"))
conn.close()
```

Содержимое файла server.py:

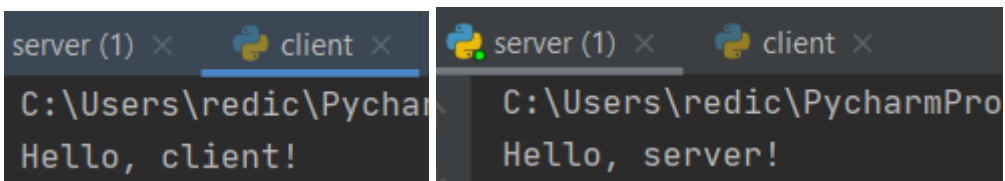
```

import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(("127.0.0.1", 5635))
sock.listen(10)
while True:
    conn, addr = sock.accept()
    data = conn.recv(1024)
    udata = data.decode("utf-8")
    print(udata)
    if not data:
        break
    conn.send(b"Hello, client! \n")
    conn.close()

```

Результат выполнения:



2. Реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры, которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту. Вариант № 1 (Теорема Пифагора).

Содержимое файла client.py:

```

import socket

conn = socket.socket()
conn.connect(("127.0.0.1", 6635))
a = input('Введите длину первого катета: ')
b = input('Введите длину второго катета: ')
catets = ' '.join([str(a), str(b)])
conn.send(catets.encode())
data = conn.recv(1024)
print('Гипотенуза равна: ' + data.decode("utf-8"))
conn.close()

```

После подключения к серверу клиент получает данные для нахождения гипотенузы по теореме Пифагора, после чего отправляет их на сервер. Затем выводит результат, полученный от сервера.

Содержимое файла server.py:

```

import math
import socket

sock = socket.socket()
sock.bind("", 6635)
sock.listen(10)
while True:
    conn, addr = sock.accept()
    data = conn.recv(1024)
    udata = data.decode("utf-8")
    a, b = udata.split()
    c = math.sqrt(int(a) ** 2 + int(b) ** 2)
    print(f"Первый катет равен {a}, второй равен {b}")
    if not data:
        break
    conn.send(str(c).encode())
conn.close()

```

Результат выполнения программы:

```
server (2) × client (1) ×
C:\Users\redic\PycharmProjects\WEB\ve
Введите длину первого катета: 3
Введите длину второго катета: 4
Гипотенуза равна: 5.0

Process finished with exit code 0
|
```

```
server (2) × client (1) ×
C:\Users\redic\PycharmProjects\WEB\venv\S
Первый катет равен 3, второй равен 4
```

3. Содержимое файла server.py:

```
import json
import socket
from email.parser import Parser
from functools import lru_cache
from urllib.parse import parse_qs, urlparse

MAX_LINE = 64 * 1024
MAX_HEADERS = 100

class MyHTTPServer:
    def __init__(self, host, port, server_name):
        self._host = host
        self._port = port
        self._server_name = server_name
        self._students = {}

    def serve_forever(self):
        serv_sock = socket.socket(
            socket.AF_INET,
            socket.SOCK_STREAM,
            proto=0)
        try:
            serv_sock.bind((self._host, self._port))
            serv_sock.listen(100)

            while True:
                conn, _ = serv_sock.accept()
                try:
```

```

        self.serve_client(conn)
    except Exception as e:
        print('Client serving failed', e)
    finally:
        serv_sock.close()

def serve_client(self, conn):
    try:
        req = self.parse_request(conn)
        resp = self.handle_request(req)
        self.send_response(conn, resp)
    except ConnectionResetError:
        conn = None
    except Exception as e:
        self.send_error(conn, e)

    if conn:
        req.rfile.close()
        conn.close()

def parse_request(self, conn):
    rfile = conn.makefile('rb')
    method, target, ver = self.parse_request_line(rfile)
    headers = self.parse_headers(rfile)
    host = headers.get('Host')
    if not host:
        raise HTTPError(400, 'Bad request',
                        'Host header is missing')
    return Request(method, target, ver, headers, rfile)

```

```
def parse_request_line(self, rfile):
    raw = rfile.readline(MAX_LINE + 1)
    if len(raw) > MAX_LINE:
        raise HTTPError(400, 'Bad request',
                        'Request line is too long')

    req_line = str(raw, 'iso-8859-1')
    words = req_line.split()
    if len(words) != 3:
        raise HTTPError(400, 'Bad request',
                        'Malformed request line')

    method, target, ver = words
    if ver != 'HTTP/1.1':
        raise HTTPError(505, 'HTTP Version Not Supported')
    return method, target, ver

def parse_headers(self, rfile):
    headers = []
    while True:
        line = rfile.readline(MAX_LINE + 1)
        if len(line) > MAX_LINE:
            raise HTTPError(494, 'Request header too large')

        if line in (b'\r\n', b'\n', b''):
            break

        headers.append(line)
        if len(headers) > MAX_HEADERS:
```



```
def handle_request(self, req):
    if req.path == '/students' and req.method == 'POST':
        return self.handle_post_student(req)

    if req.path == '/students' and req.method == 'GET':
        return self.handle_get_student(req)

    raise HTTPError(404, 'Not found')

def send_response(self, conn, resp):
    wfile = conn.makefile('wb')
    status_line = f'HTTP/1.1 {resp.status} {resp.reason}\r\n'
    wfile.write(status_line.encode('iso-8859-1'))

    if resp.headers:
        for (key, value) in resp.headers:
            header_line = f'{key}: {value}\r\n'
            wfile.write(header_line.encode('iso-8859-1'))

    wfile.write(b'\r\n')

    if resp.body:
        wfile.write(resp.body)

    wfile.flush()
    wfile.close()
```

```
def send_error(self, conn, err):
    try:
        status = err.status
        reason = err.reason
        body = (err.body or err.reason).encode('utf-8')
    except:
        status = 500
        reason = b'Internal Server Error'
        body = b'Internal Server Error'
    resp = Response(status, reason,
                    [('Content-Length', len(body))],
                    body)
    self.send_response(conn, resp)

def handle_post_student(self, req):
    self._students[req.query['student'][0]] = req.query['grade'][0]
    return Response(202, 'Accepted')

def handle_get_student(self, req):
    accept = req.headers.get('Accept')
    if 'text/html' in accept:
        contentType = 'text/html; charset=utf-8'
        body = '<html><head></head><body>'
        body += f'<div>Студенты (количество:{len(self._students)})</div>'
        body += '<ul>'
        for st in self._students.keys():
            body += f'<li>{st}: {self._students[st]}</li>'
        body += '</ul>'
        body += '</body></html>'
    else:
        body = json.dumps(self._students)
```

```
elif 'application/json' in accept:
    contentType = 'application/json; charset=utf-8'
    body = json.dumps(self._students)

else:
    return Response(406, 'Not Acceptable')

body = body.encode('utf-8')
headers = [('Content-Type', contentType),
            ('Content-Length', len(body))]
return Response(200, 'OK', headers, body)
```

```
class Request:
    def __init__(self, method, target, version, headers, rfile):
        self.method = method
        self.target = target
        self.version = version
        self.headers = headers
        self.rfile = rfile

    @property
    def path(self):
        return self.url.path

    @property
    @lru_cache(maxsize=None)
    def query(self):
        return parse_qs(self.url.query)
```

```

class Response:
    def __init__(self, status, reason, headers=None, body=None):
        self.status = status
        self.reason = reason
        self.headers = headers
        self.body = body

class HTTPError(Exception):
    def __init__(self, status, reason, body=None):
        super()
        self.status = status
        self.reason = reason
        self.body = body

if __name__ == '__main__':
    host = "127.0.0.1"
    port = 5500
    name = "server.local"

    serv = MyHTTPServer(host, port, name)
    try:
        serv.serve_forever()
    except KeyboardInterrupt:
        pass

```

Результат выполнения программы:

Status: 202 (Accepted) Time: 311 ms Size: 0.00 kb

Authorization

Content

Headers

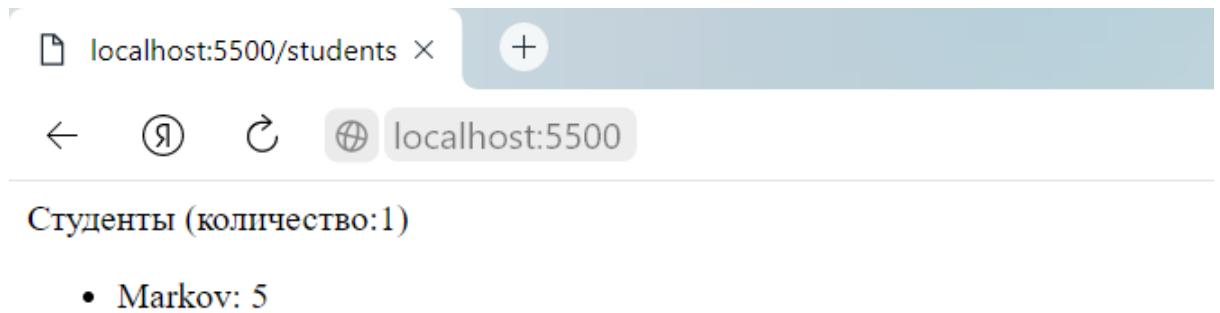
Raw (4)

Content

Headers

Raw (1)

☒ Bearer Token
☐ Basic Auth
☐ Custom



4. Реализовать многопользовательский чат.

Содержимое файла server.py:

```
import socket

from threading import Thread

users = []
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('', 55551))
sock.listen(10)
sock.setblocking(False)

def chat_users():
    while True:
        sock.setblocking(True)
        con, addr = sock.accept()
        sock.setblocking(False)
        con.send(b'Enter your name: \n')
        name = con.recv(1024)
        con.send(b'You can send messages now. Press q if you want to leave the chat. \n')
        name = name.decode("utf-8")
        print(name, 'has joined the chat')
        if con not in users:
            users.append((con, name))
```

```

def message():
    while True:
        try:
            for user in users:
                text = user[0].recv(1024).decode('utf-8')
                if text == "q":
                    user[0].close()
                    print('{} has left the chat'.format(user[1]))
                else:
                    print(str(user[1]) + ':' + text)
                    for send_user in users:
                        if send_user[0] != user[0]:
                            data = f'{user[1]}: ' + text
                            send_user[0].sendall(data.encode('utf8'))
        except socket.error:
            print('No one is in the chat. Waiting for new users...')
            break

user_thread = Thread(target=chat_users)
message_thread = Thread(target=message)

user_thread.start()
message_thread.start()

```

Содержимое файла cli.py:

```
def send_message():
    while True:
        text = input()
        sock.sendall(bytes(text, "utf-8"))
        if text == "q":
            sock.close()
            break

def receive_message():
    try:
        while True:
            data = sock.recv(1024)
            udata = data.decode("utf-8")
            print(udata)
    except ConnectionAbortedError:
        print('You left the chat')

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 55551))
send_thread = Thread(target=send_message)
get_thread = Thread(target=receive_message)

send_thread.start()
get_thread.start()
```

Результат выполнения программы:

```
server × cli × cli2 ×
C:\Users\redic\PycharmProjects\WEB\venv\Scripts\pyt
Anna has joined the chat
Maksim has joined the chat
Anna:Hi!
Maksim:Hi! How are you?
Anna:I'm good. And you?
Maksim:I'm fine
Anna:Sorry, I need to go now. Bye(
Maksim has left the chat
Anna has left the chat
No one is in the chat. Waiting for new users...
```



```
server x cli x cli2 x
C:\Users\redic\PycharmProjects\WEB\venv\Scripts\python.exe C:/Users/redic
Enter your name:

Maksim
You can send messages now. Press q if you want to leave the chat.

Anna: Hi!
Hi! How are you?
Anna: I'm good. And you?
I'm fine
Anna: Sorry, I need to go now. Bye(
q
You left the chat

Process finished with exit code 0
```

```
server x cli x cli2 x
C:\Users\redic\PycharmProjects\WEB\venv\Scripts\python.exe C:/Users/redic
Enter your name:

Anna
You can send messages now. Press q if you want to leave the chat.

Hi!
Maksim: Hi! How are you?
I'm good. And you?
Maksim: I'm fine
Sorry, I need to go now. Bye(
Maksim: q
q
You left the chat

Process finished with exit code 0
```