Федеральное государственное автономное образовательное учреждение высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра «Информатика и информационные технологии»

Направление подготовки/ специальность: _информационные системы и технологии_____

ОТЧЕТ

по Индивидуальному вариативному заданию

Студент: _Кононова Елизавета Андреевна__ Группа: _241-332_____

Место прохождения практики: Московский Политех, кафедра «Информатика и информационные технологии»

Отчет принят с оценкой _____ Дата _____

Руководитель практики: Худайбердиева Гулшат

Москва 2025

Создание HTTP-сервера на Python с нуля

Цель: реализовать простой сервер на Python с использованием сокетов.

Задачи:

1. Изучить HTTP, ключевые компоненты,
2. Изучить работу сервера.
3. Скомпилировать код для работы сервера.

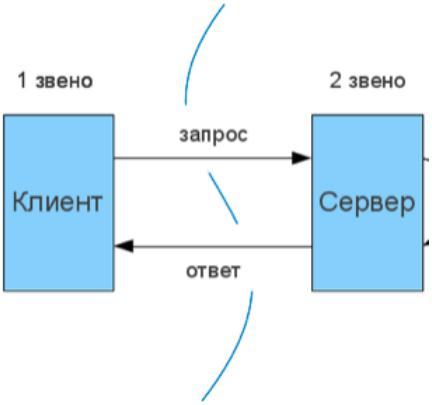
Ход работы:

1. **Исследование предметной области**
   HTTP (HyperText Transfer Protocol) — протокол прикладного уровня для передачи гипертекстовых документов.

Ключевые компоненты:

- Запросы (Request)
- Ответы (Response)
- Методы (GET, POST)
- Коды состояния (200, 404, 500)

Как работает сервер:



2. Практическая реализация

1. Для начала создадим папку для работы с сервером, например my_server в папке пользователя на диске C:\

2. В ней создадим файл server.py с кодом для нашего сервера и папку static для хранения статических файлов.

3. Теперь создадим базовый код:

```
import socket
import os
import threading
from datetime import datetime
class HTTPServer:
def init(self, host='127.0.0.1', port=8080):
```

4. Инициализируем сервер, добавим хост - ip адрес для привязки, порт прослушивания, папку со статическими файлами

```
self.host = host
self.port = port
self.server_socket = None #Основной сокет сервера
self.static_dir = 'static'
self.running = False #Флаг работы сервера
if not os.path.exists(self.static_dir): #Создаем папку static если ее нет
os.makedirs(self.static_dir)
print(f"Создана папка {self.static_dir}")
```

5. Теперь запуск сервера и настройка сокета

```
def start(self):
try:
# 1. Создаем TCP-сокет (IPv4, потоковый)
self.server_socket = socket.socket(
socket.AF_INET, # IPv4
socket.SOCK_STREAM # TCP
)
```

Copy

Download

```
# 2. Разрешаем повторное использование адреса
self.server_socket.setsockopt(
    socket.SOL_SOCKET,
    socket.SO_REUSEADDR,
    1
)

# 3. Привязываем сокет к адресу
self.server_socket.bind((self.host, self.port))

# 4. Начинаем слушать соединения (макс. 5 в очереди)
self.server_socket.listen(5)

self.running = True
print(f"Сервер запущен на http://{self.host}:{self.port}")

# 5. Запускаем основной цикл обработки
self.serve_forever()
```

except Exception as e:
print(f"Ошибка запуска: {e}")
self.stop()

6. Теперь создадим основной цикл обработки соединений

def serve_forever(self):
try:
while self.running:
try:
# Устанавливаем таймаут 1 сек для проверки флага running
self.server_socket.settimeout(1)

Copy
Download

Принимаем новое соединение
client_conn, client_addr = self.server_socket.accept()

# Создаем поток для обработки запроса
thread = threading.Thread(
target=self.handle_request,
args=(client_conn, client_addr),
daemon=True # Поток завершится с основным
)
thread.start()

except socket.timeout:
continue # Таймаут для проверки running

except KeyboardInterrupt:
print("\nСервер останавливается...")
finally:
self.stop()

7. Создадим обработку одного клиентского запроса

```
def handle_request(self, client_conn, client_addr):
try:
# 1. Получаем данные запроса (макс. 4KB)
request_data = client_conn.recv(4096).decode('utf-8')
if not request_data:
return
```

Copy

Download

```
# 2. Парсим первую строку запроса (например: GET / HTTP/1.1)
first_line = request_data.split('\r\n')[0]
parts = first_line.split()
if len(parts) < 2:
    return

method, path = parts[0], parts[1]  # Метод и путь

# 3. Генерируем упрощенный ответ
response = self.generate_simple_response(method, path)

# 4. Отправляем ответ
client_conn.sendall(response.encode('utf-8'))

except Exception as e:
print(f"Ошибка обработки: {e}")
error_msg = "HTTP/1.1 500 Error\r\n\r\nServer Error"
client_conn.sendall(error_msg.encode('utf-8'))
finally:
client_conn.close() # Закрываем соединение
```

8. Наконец создадим генерацию ответов сервера

```
def generate_simple_response(self, method, path):
if method != 'GET':
return "HTTP/1.1 405 Method Not Allowed\r\n\r\n"
```

Нормализуем путь

```
if path == '/':
path = '/index.html'
```

Пытаемся найти файл

```
file_path = os.path.join(self.static_dir, path.lstrip('/'))
if os.path.isfile(file_path):
try:
with open(file_path, 'r') as f:
content = f.read()
```

Copy

Download

```
    # Минимальные необходимые заголовки
    headers = [
        "HTTP/1.1 200 OK",
        f"Content-Length: {len(content)}",
        "Connection: close",
        "\r\n"
    ]

    return "\r\n".join(headers) + content

 except Exception:
    return "HTTP/1.1 500 Error\r\n\r\n"
else:
return "HTTP/1.1 404 Not Found\r\n\r\n"
```

if **name** == "**main**":

server = HTTPServer()

server.start()

Готово!

Примеры работы

В терминале вводим python

```
C:\Users\lizak\my_server>python http_server.py
Сервер запущен на http://127.0.0.1:8080
Нажмите Ctrl+C для остановки
```

Мы видим пустую белую страницу, это означает, что сервер запущен и работает

Пример ответа сервера

```
[2025-05-22 21:14:09.115075] ('127.0.0.1', 54891) - GET /
=== HTTP Response ===
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 0
Connection: close
Server: PythonHTTPServer/1.0
Date: Thu, 22 May 2025 18:14:09 GMT
```

Модификация

Добавим счетчик посещений

def **init**(self, host='127.0.0.1', port=8080):

self.host = host

self.port = port

self.server_socket = None

self.static_dir = 'static'

self.running = False

self.visits = 0

В метод handle_request добавляем увеличение счетчика:

def handle_request(self, client_conn, client_addr):

try:

self.visits += 1 # <- Добавить эту строку (увеличиваем счетчик)

Copy

Download

```
    request_data = client_conn.recv(4096).decode('utf-8')

    if not request_data:
        return
```

В метод generate_response добавляем вывод счетчика в страницу:

if os.path.isfile(file_path):

try:

with open(file_path, 'r') as f:

content = f.read()

Copy

Download

```
    # Добавить эти 2 строки (вставка счетчика в HTML):
    if path.endswith('.html'):
        content = content.replace('</body>', f'<footer>Visits:
```

{self.visits}</footer></body>')

Вывод: изучен HTTP, принцип его работы и способ реализации простого HTTP сервера на Python, добавлена модификация.

Список использованной литературы:

1. Python: Building a basic HTTP Server from scratch in Python https://joaoventura.net/blog/2017/python-webserver/
2. http.server — HTTP servers https://docs.python.org/3/library/http.server.html
3. Python HTTP сервер: обработка GET и POST запросов https://sky.pro/wiki/python/python-http-server-obrabotka-get-i-post-zaprosov/