МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Лабораторная работа № 15

по дисциплине «WEB технологии»

на тему «HTTP чат. Web-сокеты»

Выполнил

студент группы М3О-125БВ-24

Егоров А.В.

Приняли

ассистент каф. 304 Борисов А.И.

Москва
2025

Содержание:

Постановка задачи:

1. Разработать программу «HTTP-сервер», которая принимает запрос GET от браузера и отправляет в ответ HTML страницу с полем чата, строкой ввода сообщения и кнопкой отправить. В этом HTML-ответе передаются предыдущие 100 сообщений, которые отображаются в браузере пользователя.

На стороне пользователя вводится информация в браузер, и, по нажатию кнопки, отправляется информация (POST запрос) о введенном сообщении. Это сообщение сохраняется на стороне сервера. Ответом на данный запрос будет новая HTML-страница.

2. Провести тестирование данного программного обеспечения на 3-х клиентах и одном сервере. Программный код и результаты тестирования привести в отчете.

3. Доработать HTTP-сервер для общения при помощи web-сокетов. На стороне сервера необходимо предложить соединение через web-сокеты. Отправление данных от клиентов будет аналогичным, но при получении данных соединение не будет разорвано.

Серверу необходимо сохранять все открытые соединения web-сокетов. При GET запросе от клиента, сервер отправляет HTML-страницу чата, JS файл для отображения сообщений и JSON файл, содержащий сообщения.

По результатам получения POST запроса от клиента, web-сервер дополняет JSON файл (максимум 100 сообщений) и рассылает его всем клиентам.

4. Провести тестирование данного программного обеспечения на 3-х клиентах и одном сервере. Программный код и результаты тестирования привести в отчете.

Код

chat_http.py

```
from flask import Flask, request, redirect, render_template_string
import json
import os

# Создаём экземпляр Flask-приложения
app = Flask(__name__)

# Имя файла для хранения сообщений
MESSAGES_FILE = "laba_6/messages.json"

def load_messages():
    """Загружает сообщения из JSON-файла, если он существует."""
    if os.path.exists(MESSAGES_FILE):
        try:
            with open(MESSAGES_FILE, "r", encoding="utf-8") as f:
                return json.load(f)
        except Exception:
            return []
    return []

def save_messages(messages):
    """Сохраняет сообщения в JSON-файл."""
    os.makedirs(os.path.dirname(MESSAGES_FILE), exist_ok=True)
    with open(MESSAGES_FILE, "w", encoding="utf-8") as f:
        json.dump(messages, f, ensure_ascii=False)

# Список для хранения последних 100 сообщений чата
messages = load_messages()

# HTML-шаблон страницы чата с формой для ввода сообщений
HTML = '''
<!DOCTYPE html>
<html>
<head>
    <title>Chat</title>
    <style>
        body { font-family: Arial, sans-serif; }
        #chat-box { height: 250px; overflow-y: auto; border: 1px solid
#ccc; padding: 8px; margin-bottom: 10px; background: #f9f9f9; }
        .msg { margin-bottom: 4px; }
        form { display: flex; gap: 8px; }
        input[type="text"] { flex: 1; padding: 6px; }
        button { padding: 6px 16px; }
    </style>
</head>
<body>
    <h2>HTTP Chat</h2>
    <div id="chat-box">
        {% for msg in messages %}
            <div class="msg">{{ msg }}</div>
        {% endfor %}
```

```
    </div>
    <form action="/send" method="post" autocomplete="off">
        <input type="text" name="message" placeholder="Type your
message..." autofocus required maxlength="200">
        <button type="submit">Send</button>
    </form>
</body>
</html>
'''

@app.route('/', methods=['GET'])
def index():
    """
    Обрабатывает GET-запрос на главную страницу.
    Возвращает HTML-страницу с последними 100 сообщениями.
    """
    return render_template_string(HTML, messages=messages[-100:])

@app.route('/send', methods=['POST'])
def send():
    """
    Обрабатывает POST-запрос с новым сообщением.
    Добавляет сообщение в список, ограничивает список 100 сообщениями,
    сохраняет в JSON-файл, затем перенаправляет пользователя обратно на
главную страницу.
    """
    msg = request.form.get('message', '').strip()
    if msg:
        messages.append(msg)
        if len(messages) > 100:
            messages.pop(0)
        save_messages(messages)
    return redirect('/')

if __name__ == '__main__':
    # При запуске сервера загружаем сообщения из файла (на случай изменений
вне процесса)
    messages = load_messages()
    # Запуск сервера Flask на всех интерфейсах, порт 8080
    app.run(host='0.0.0.0', port=8080)
```

chat_ws.py

```
from flask import Flask, render_template_string, jsonify
from flask_socketio import SocketIO, emit
import json
import os

# Создаём Flask-приложение
app = Flask(__name__)

# Инициализируем SocketIO для поддержки WebSocket соединений
socketio = SocketIO(app, cors_allowed_origins="*")

# Имя файла для хранения сообщений
MESSAGES_FILE = "laba_6/messages.json"

# Загрузка сообщений из файла
def load_messages():
    if os.path.exists(MESSAGES_FILE):
        try:
            with open(MESSAGES_FILE, "r", encoding="utf-8") as f:
                return json.load(f)
        except Exception:
            return []
    return []

# Сохранение сообщений в файл
def save_messages(msgs):
    # Создаём директорию, если её нет
    os.makedirs(os.path.dirname(MESSAGES_FILE), exist_ok=True)
    with open(MESSAGES_FILE, "w", encoding="utf-8") as f:
        json.dump(msgs, f, ensure_ascii=False, indent=2)

# Список для хранения последних 100 сообщений чата
messages = load_messages()

# HTML-шаблон страницы чата с подключением к WebSocket через Socket.IO
HTML = '''
<!DOCTYPE html>
<html>
<head>
    <title>WebSocket Chat</title>
    <script src="https://cdn.socket.io/4.0.1/socket.io.min.js"></script>
</head>
<body>
    <h2>WebSocket Chat</h2>
    <div id="chat" style="height:200px;overflow:auto;border:1px solid
#ccc;margin-bottom:10px;"></div>
    <input id="msg" type="text" autofocus>
    <button onclick="sendMsg()">Send</button>
    <script>
        // Подключение к серверу через Socket.IO
        var socket = io();
        // Функция отправки сообщения на сервер
        function sendMsg() {
```

```
            var m = document.getElementById('msg').value;
            if (m.trim() === "") return;
            socket.emit('message', m);
            document.getElementById('msg').value = '';
        }
        // Обработка получения новых сообщений от сервера
        socket.on('messages', function(msgs){
            var chat = document.getElementById('chat');
            chat.innerHTML = msgs.map(m => '<div>'+m+'</div>').join('');
            chat.scrollTop = chat.scrollHeight;
        });
        // Первоначальная загрузка истории сообщений через fetch и запрос к
серверу
        fetch('/messages.json')
            .then(r=>r.json())
            .then(msgs=>socket.emit('messages', msgs));
    </script>
</body>
</html>
'''

# Главная страница: отдаёт HTML с чатом
@app.route('/')
def index():
    return render_template_string(HTML)

# Эндпоинт для получения последних 100 сообщений в формате JSON
@app.route('/messages.json')
def get_messages():
    # Загружаем актуальные сообщения из файла
    msgs = load_messages()
    return jsonify(msgs[-100:])

# Обработка события "message" от клиента (новое сообщение)
@socketio.on('message')
def handle_message(msg):
    msg = str(msg).strip()
    if msg:
        msgs = load_messages()
        msgs.append(msg)
        # Ограничиваем историю 100 сообщениями
        if len(msgs) > 100:
            msgs = msgs[-100:]
        save_messages(msgs)
        # Рассылаем обновлённую историю всем подключённым клиентам
        socketio.emit('messages', msgs[-100:])

# При подключении клиента отправляем ему последние 100 сообщений
@socketio.on('connect')
def handle_connect():
    # Загружаем актуальные сообщения из файла при подключении клиента
    msgs = load_messages()
    emit('messages', msgs[-100:])
```

```
# Обработка события "messages" — отправка истории сообщений клиенту по
запросу
@socketio.on('messages')
def send_messages(msgs):
    # Загружаем актуальные сообщения из файла по запросу клиента
    msgs = load_messages()
    emit('messages', msgs[-100:])

# Запуск сервера
if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0', port=8080)
```

Работа программы

Серверная часть

Разработано серверное приложение на Python, реализующее чат с двумя интерфейсами взаимодействия: через HTTP (GET/POST) и через WebSocket. В качестве основы выбран фреймворк Flask, для поддержки WebSocket — расширение Flask-SocketIO. История сообщений хранится централизованно в JSON-файле на сервере.

Хранение сообщений

Все сообщения сохраняются в файл **messages.json** в формате JSON.

- При запуске сервера история сообщений загружается из этого файла.
- При добавлении нового сообщения файл обновляется.
- Для предотвращения ошибок при отсутствии файла или директории реализована автоматическая их инициализация.

HTTP-чат

GET-запрос:

Сервер возвращает HTML-страницу, содержащую форму для ввода сообщений и область для отображения последних 100 сообщений, загруженных из файла.

POST-запрос:

Сервер принимает новое сообщение, добавляет его в историю, сохраняет в файл и перенаправляет пользователя обратно на главную страницу.

Обработка сообщений:

Все операции с историей сообщений выполняются через функции загрузки и сохранения JSON-файла.

WebSocket-чат

Установка соединения:

При подключении клиента сервер автоматически отправляет ему последние 100 сообщений из файла.

Обработка новых сообщений:

При получении сообщения от клиента оно добавляется в историю, сохраняется в файл и рассылается всем подключённым клиентам через WebSocket.

Доступ к истории:

История сообщений также доступна по HTTP-эндпоинту `/messages.json` для возможного использования сторонними клиентами.

Клиентская часть

HTTP-версия

HTML-страница содержит форму для ввода сообщений и область для отображения истории чата. После отправки сообщения страница обновляется, и пользователь видит актуальную историю.

WebSocket-версия

HTML-страница подключает внешний JavaScript-код, который устанавливает WebSocket-соединение с сервером. Сообщения отправляются на сервер и отображаются у всех клиентов в реальном времени без перезагрузки страницы. При открытии страницы клиент сразу получает актуальную историю сообщений.

Вывод по работе:

В ходе выполнения проекта был осоен навык в создании многопользовательских веб-приложений с использованием протоколов HTTP и WebSocket. Разработка чат-сервера позволила на практике изучить особенности обработки запросов, организации обмена сообщениями в реальном времени и хранения данных на сервере.