УДК 004

Разработка системы видеочата

Анишкова Анастасия Сергеевна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Целью исследования является разработка видеочата. Для реализации использовались языки программирование такие как HTML и Python. Полученный результат позволяет осуществлять организацию видеоконференций.
Ключевые слова: Сервер, видеочат, HTML, Python, Socket.IO.

Development of a video chat system

Anishkova Anastasia Sergeevna
Sholom Aleichem Priamurskiy State University
Student

Abstract
The purpose of the research is to develop a video chat. Programming languages such as HTML and Python were used for implementation. The result obtained allows you to organize video conferences.
Key words: Server, Video chat, HTML, Python, Socket.IO.

1 Введение

1.1 Актуальность

Тема разработки видеочата является актуальной и востребованной, особенно в наше время, когда все больше людей работает удаленно и проводит много времени в интернете. Видеочаты становятся основным средством общения для многих людей, позволяя им визуально общаться с семьей, друзьями, коллегами и партнерами по бизнесу, которые находятся на другом конце мира.

1.2 Обзор исследований

К. Сухов описал WebSockets-стандарт современного веба[1], Веб-сокеты, как система взаимодействия клиент-сервер и клиент-сервер-клиент описал А. С. Гавриленко[2], Н. В. Ким пишет о проектирование и разработке компонентов для браузерного видео чата[3], применение дейтаграммных сокетов для реализации многопользовательского сетевого видеочата описывает Е. В. Вершинин, Р. А. Поляков[4], А. А. Денищенко, А. Р. Нехаев описал разработку веб приложения видеочат[5], Видеочат без плагинов.

Юзаем WebRTC+ сокеты для звонков из чистого браузера продемонстрирвал А. С Лыкошин [6].

1.3 Цель исследования

Целью исследования является создание системы видеочата для организации видеоконференций.

2 Материалы и методы

В данном исследование используются языки программирования HTML и Python.

3 Результаты

Видеочат — это современное средство коммуникации, которое позволяет людям общаться в режиме реального времени, визуально оценивать выражение лица и жесты собеседника, а также передавать эмоции через голос и мимику. Это удобный и эффективный способ общения на расстоянии, особенно для тех, кто не может встретиться лично. Видеочаты стали широкодоступными благодаря развитию технологий и созданию удобных платформ для общения. Они помогают людям поддерживать связь с близкими, друзьями и коллегами, независимо от географического расположения. Видеочаты также позволяют проводить деловые встречи и конференции в режиме онлайн, что экономит время и ресурсы на поездки. В целом, видеочаты обеспечивают более близкий и непосредственный контакт между людьми, что способствует улучшению качества коммуникации.

Разработку веб-приложения для организации видеоконференций можно разделить на 2 части: создание HTML страницы и написание серверной составляющей. HTML должен включать в себя следующие элементы: контейнер, в котором будут создаваться элементы для транслирования изображения с веб-камеры пользователя, а также элемент «audio», воспроизводящий звук с микрофона пользователя.

HTML страница в начальной реализации представлена следующим образом:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta    name="viewport"    content="width=device-width,
initial-scale=1.0">
    <title>Video Chat</title>
</head>
<body>
    <h1>Video Chat</h1>
    <div id="videos-container"></div>
    <audio id="audio" autoplay></audio>
</body>
</html>
```

На данном этапе отсутствует взаимодействие с серверной стороной приложения. Для организации передачи информации между клиентом и сервером будет использоваться модуль «Socket.IO». Данный модуль разработан для работы с веб-сокетами (WebSocket) и другими транспортными средствами для обеспечения двусторонней связи в реальном времени между клиентом и сервером в веб-приложениях.

Модуль «Socket.IO» поддерживается большим количеством языков программирования, в том числе Python. Из достоинств модуля можно выделить следующие ключевые характеристики:

1. Socket.IO обеспечивает двустороннюю связь в реальном времени, что позволяет мгновенно передавать данные между клиентом и сервером.

2. Socket.IO включает в себя встроенную поддержку автоматического восстановления соединения, что позволяет приложению восстанавливаться после временных сбоев в сети.

3. Socket.IO предоставляет простой синтаксис для отправки и получения сообщений между клиентом и сервером.

Модуль можно подключить через тег «script», указав в качестве источника ссылку на подключение через AJAX:

```
<script src =
"https://cdnjs.cloudflare.com/ajax/libs/socket.io/3.1.3/socket.io.js"> </script>
```

После подключения модуля, на стороне HTML становится доступным функционал Socket.IO. Перед написанием необходимых функций и процедур, стоит рассмотреть серверную сторону приложения.

Как и было описано в теоретической части, реализован сервер на Python. Для реализации работы сервера в веб-ориентированной среде используется фреймворк Flask. Данный фреймворк был разработан для написания клиент-серверных решений на Python и обладает базовым, но достаточным функционалом для разработки веб-ориентированных приложений. Фреймворк можно легко расширять, также он поддерживает работу с модулем Socket.IO.

Приложения, разработанные на Flask, строятся на принципах функционального программирования. То есть все поступающие на сервер элементы и данные должны быть обработаны в отдельных функциях. Нужно это для обеспечения доступности и простоты кода, а также для структуризации программы на отдельные функциональные блоки.

Базовая структура приложения, написанного на Flask, выглядит следующим образом:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def main_app():
  pass
if __name__ == '__main__':
  app.run(debug=True)
```

Здесь можно выделить следующие основные части данного кода:

1. Для создания приложения используется метод «Flask()», в скобках которого указана специальная переменная в Python «__name__», которая автоматически устанавливается интерпретатором в зависимости от контекста выполнения. Этот метод реализует все основные функциональности, необходимые для обработки HTTP-запросов, управления маршрутами и т.д.

2. Функции, напрямую связанные с отображением страниц или обработкой данных с этих страниц, заключаются в декораторы. Декораторы позволяют изменить или расширить поведение функции без изменения ее кода. То есть за счёт отдельных функций можно расширить уже созданную ранее. В примере выше определена функция «main_app», в которой должна обрабатываться информация веб-приложения. В представленном примере, @app.route('/') является декоратором функции «main_app», который привязывает функцию, к корневому URL-маршруту. Это означает, что если кто-то посещает корневой URL веб-приложения, то будет вызвана указанная функция.

3. Запуск приложения осуществляется через метод «run()», в скобках которого указан параметр, включающий режим отладки. В режиме отладки Flask предоставляет дополнительные инструменты и информацию об ошибках для облегчения разработки. В режиме отладки также сервер автоматически перезапускается после изменения кода.

Помимо Flask используется также следующие библиотеки:

1. CV2 – модуль для компьютерного зрения. Используется для получения изображения с веб-камеры пользователя.

2. Base64 – модуль для кодирования данных в Base64 формат. Нужен для передачи изображений в зашифрованном виде для последующего отображения на стороне клиента.

3. Threading – модуль для создания потоков в приложении. Данный модуль необходим для организации подключения большого количества пользователей (без многопоточности подключиться сможет только 1 человек).

4. NumPy – модуль для работы с математическими структурами и данными. Он нужен для работы с массивами при обработке звука на стороне сервера.

На стороне клиента необходимо реализовать следующий функционал: подключение и отключение пользователя, вывод изображения с его веб-камеры, воспроизведение звука на стороне клиента.

1. Подключение пользователя. На стороне клиента подключение реализуется в одну строку:

```
    socket = io.connect('http://' + document.domain + ':' +
location.port);
```

Создаётся переменная «socket», в которую записывается информация о подключении к серверу через функцию «io.connect()».

Подключение на стороне сервера работает следующим образом: создаётся переменная client_id, в которую записывается id пользователя, который подключился в данный момент времени. Полученный id сохраняется в отдельный словарь clients, где в качестве ключа выступают id пользователей, а значениями ключей являются объекты, предназначенные для отображения изображения с веб-камеры. Такие объекты можно создать через модуль CV2, нужно использовать класс «VideoCapture()», в скобках которого указан номер подключаемой камеры (по умолчанию 0) и конфигурация камеры (опционально, но рекомендуется для встроенных камер):

```
clients[client_id] = cv2.VideoCapture(0,cv2.CAP_DSHOW)
```

После внесения нового пользователя в словарь, запускается поток, в котором запускается функция «video_thread», которая предназначения для вывода изображения с веб-камеры пользователя на страницу сайта:

```
video_thread_instance = threading.Thread(
  target=video_thread, args=(client_id, clients[client_id])
)
video_thread_instance.start()
```

Также создаётся аудио-канал, который передаёт звук с микрофона пользователя. Для создания звукового потока используется функция «InputStream». Аргументами являются функция, которую необходимо запустить, количество каналов и тип записи. «InputStream» является частью модуля «SoundDevice»:

```
if audio_stream is None:
  audio_stream = sd.InputStream(callback=audio_callback,
channels=1, dtype=np.int16)
  audio_stream.start()
```

Итоговый результат функции подключения пользователя выглядит следующим образом:

```
@socketio.on('connect')
def handle_connect():
    global audio_stream
    client_id = request.sid
    clients[client_id] = cv2.VideoCapture(0,cv2.CAP_DSHOW)
    print(clients)
    if audio_stream is None:
        audio_stream =
sd.InputStream(callback=audio_callback, channels=1,
dtype=np.int16)
        audio_stream.start()
    video_thread_instance = threading.Thread(
```

```
            target=video_thread, args=(client_id,
clients[client_id])
        )
        video_thread_instance.start()
```

Для отключения пользователя на стороне клиента достаточно перезагрузить страницу:

```
socket.on('disconnect', function () {
     location.reload();
});
```

На стороне сервера также достаточно удалить пользователя из словаря clients и вызвать функцию по удалению элемента с изображением его веб-камеры со страницы:

```
@socketio.on('disconnect')
def handle_disconnect():
    client_id = request.sid
    if client_id in clients:
        if clients[client_id].isOpened():
            clients[client_id].release()
        del clients[client_id]
        socketio.emit('remove_video', {'client_id':
client_id})
```

Функция удаления элемента со страницы работает следующим образом: в качестве аргумента функция получает id пользователя. Далее необходимо получить ссылку на данный элемент по id элемента на странице (все элементы имеют id пользователей по умолчанию) и если данный элемент существует (проверка нужна для того, чтобы не допустить прерываний работы сервера из-за попытки сослаться на несуществующий элемент), то его нужно удалить:

```
socket.on('remove_video', function (data) {
  var clientId = data.client_id;
  var imgElement = document.getElementById(clientId);
  if (imgElement) {
     imgElement.remove();
  }
});
```

Функция, «video_thread» не привязана ни к каким декораторам и может быть вызвана только внутри потока во время подключения пользователя к сайту. Внутри функции запускается бесконечный цикл, который будет обрабатывать каждый кадр и выводить его на экран в закодированном в Base64 формате. Обработка кадров происходит с помощью методов чтения информации с объекта, за которым была ранее закреплена веб-камера

пользователя. Для вывода изображения используется функция «video», которая определена на стороне клиента:

```
socket.on('video', function (data) {
  var clientId = data.client_id;
  var frameAsText = data.frame;
  var imgElement = document.getElementById(clientId);
  if (!imgElement) {
     imgElement = document.createElement('img');
     imgElement.id = clientId;
     document.getElementById('videos-
container').appendChild(imgElement);
  }
  imgElement.src = '' + frameAsText;
});
```

Функция создаёт новый элемент «img» с id пользователя, если такой отсутствует. В этот элемент помещается изображение с веб-камеры в Base64 формате.

На стороне сервера функция «video_thread» реализована следующим образом:

```
def video_thread(client_id, video_capture):
  try:
     while True:
          ret, frame = clients[client_id].read()
          if not ret or frame is None:
                    continue
          _, buffer = cv2.imencode('.jpg', frame)
          if buffer is None or buffer.size == 0:
                    continue
          frame_as_text =
base64.b64encode(buffer).decode('utf-8')
          socketio.emit('video', {'client_id': client_id,
'frame': frame_as_text})
```

Стоит упомянуть про функцию отображения страницы. Она привязана к декоратору загрузки корневого каталога сайта и предназначена для отображения HTML шаблона, который был создан ранее:

```
@app.route('/')
def index():
    return render_template('index.html')
```

Все функции приложения написаны и сервер готов к запуску. Во время работы Flask-приложения, в командную консоль будет выводиться отладочная информация: логи о подключении пользователей, передаче информации между страницами. Также выводится информация, которая указана в функциях «print()» (см. рис. 1).

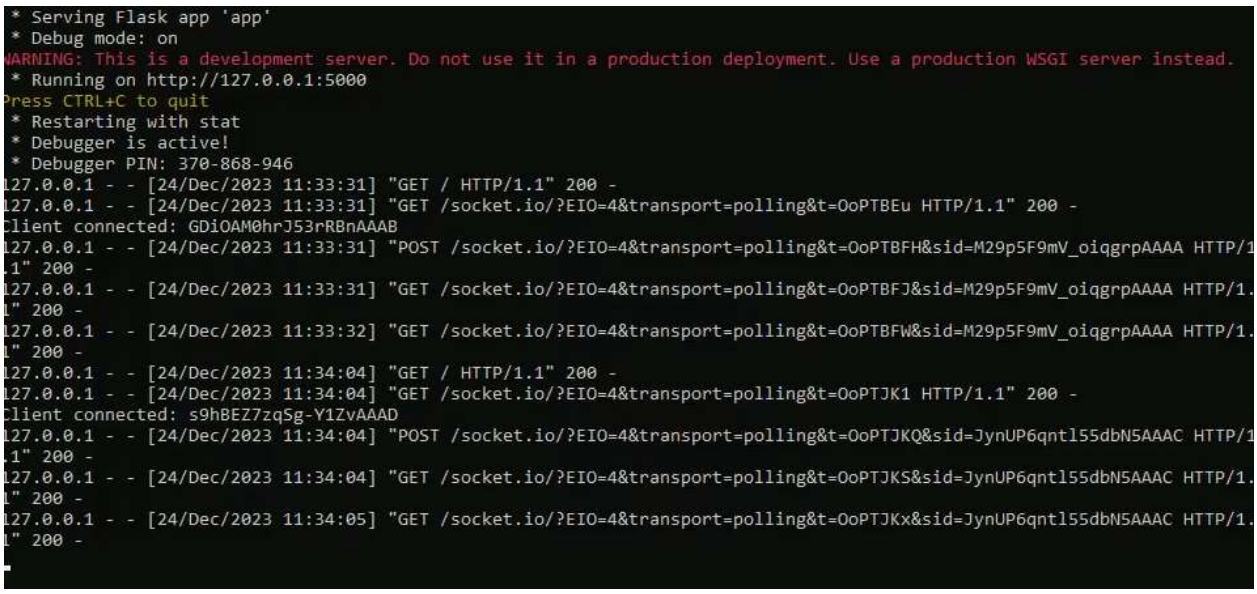

Рисунок 1 – Информация о работе сервера в командной консоли ОС

Если во время запуска сервера не возникло ошибок (выводятся также в консоль в виде исключений Python-формата), то приложение готово к работе.

К серверу может подключиться сколько угодно человек – все пользователи будут отображаться в отдельных окнах (см. рис. 2). Если пользователь выйдет из приложения, то окно с изображением его веб-камеры будет удалено. Подключаться можно с разных устройств, но нужно учитывать, что не все браузеры поддерживают работу с протоколом «WebSocket», что может затруднить работу приложения и привести к некорректному отображению элементов на странице.

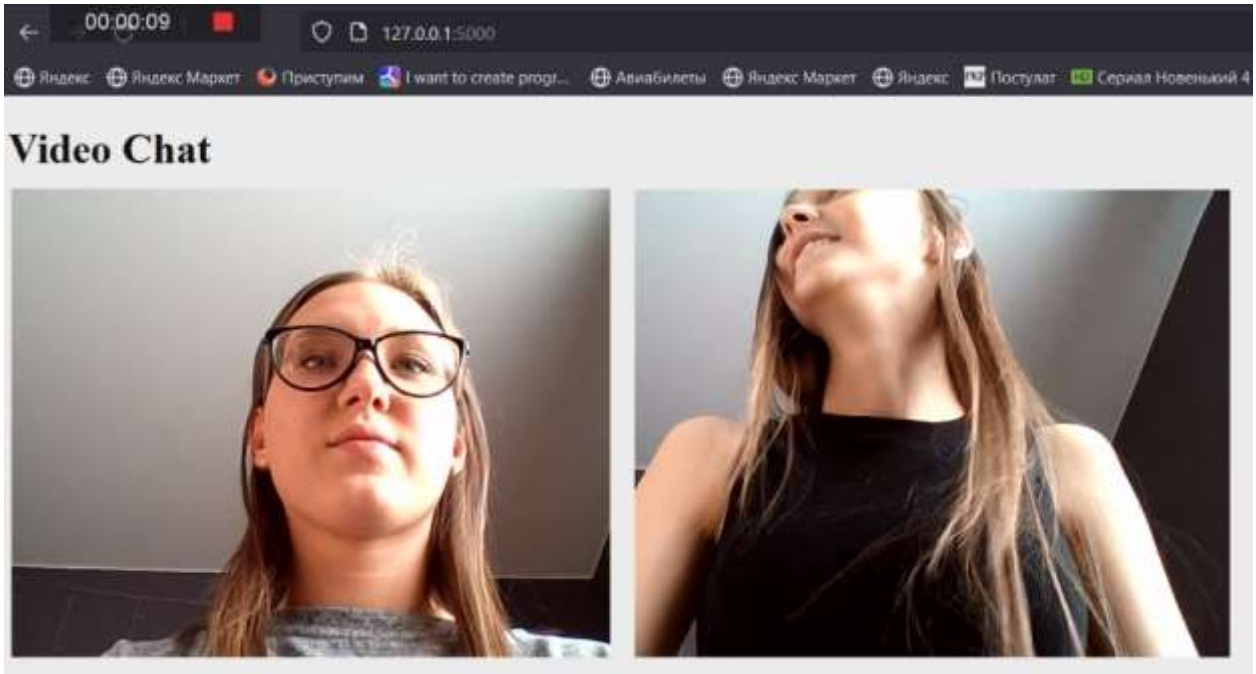

Рисунок 2 – Пример работы приложения

Выводы

Разработка системы видеочата является актуальной и востребованной задачей в современном мире информационных технологий. В процессе выполнения были рассмотрены основные аспекты разработки такой системы, включая выбор подходящих технологий и инструментов и реализацию функциональности.

В результате работы была создана функциональная система видеочата, способная обеспечить коммуникацию в режиме реального времени между пользователями. Она позволяет передавать видео и аудио данные.

Выполнение позволило углубиться в проблематику разработки систем видеочата и приобрести ценный опыт в данной области.

Библиографический список

1. Сухов К. WebSockets-стандарт современного веба. Часть 2: Socket. io-веб-сокеты для всех! //Системный администратор. 2021. №. 7-8. С. 95-101.
2. Гавриленко А. С. Веб-сокеты, как система взаимодействия клиент-сервер и клиент-сервер-клиент // 55-я юбилейная конференция аспирантов, магистрантов и студентов учреждения образования «Белорусский государственный университет информатики и радиоэлектроники». Минск, Беларусь: БГУИР, 2020. С. 22-23.
3. Ким Н. В. Проектирование и разработка компонентов для браузерного видео чата: бакалаврская работа / Национальный исследовательский Томский политехнический университет (ТПУ), Инженерная школа информационных технологий и робототехники (ИШИТР), Отделение информационных технологий (ОИТ); науч. рук. Ф. В. Саврасов. Томск, 2022, 65 с.
4. Вершинин Е. В., Поляков Р. А. Применение дейтаграммных сокетов для реализации многопользовательского сетевого видеочата // Вектор научной мысли: научный журнал. СПб: МИПИ им. Ломоносова, 2021. С. 63-66.
5. Денищенко А. А., Нехаев А. Р. Разработка веб приложения видеочат // Вектор научной мысли: научный журнал. Воронеж: ВГУ, 2021. С. 96-102.
6. Лыкошин А. С. Видеочат без плагинов. Юзаем WebRTC+ сокеты для звонков из чистого браузера //Хакер. 2020. №. 176. С. 112-117.