VIDEO CHAT APPLICATION

Sandeepsivaram S Sashikumar N

Visweswaran A Srinath M

Venkat Raj V

ABSTRACT

Chatting is now-a-days very useful to express our ideas as well as receive others ideas on any topic. Chats reflect the recent trends of the society. Sometimes, it is possible to meet eminent people in chatting and have their advice.

Our application makes video calling a pleasant experience. It has excellent features that make any user do whatever he wants while chatting.

The demand for social networking sites is increasing day by day. A social networking site that allows you to video chat online is the primary inspiration for my project. The goal of my project is to build an online video chatting tool that enables users to join real-time streaming video chat rooms where users can share their video with another person.

Some of the existing video chat applications are Stickam, YouCam, ichat, blogtv. My online video chat tool offers similar functionality like these social networking sites, and it is simple to execute and doesn’t have to rely on any third-party sites. Users can directly enter webcam chat room. Also, this video chat application doesn't require any additional software installation on the client side.

TABLE OF CONTENTS

INTRODUCTION

Video calls have become an integral part of today's communication with over 175% increase in regular live video usage among millennials in just the last 3 years, according to tokbox. We can see that video chat apps are growing in popularity with incredible speed both for businesses as well as personal use.

Moreover, respondents in a 2019 surveyreported an increase in cloud-based video conferencing application needs. These stats are telling. Video conferencing app development is on the rise.

Hence, the following report will be about how to create a video chat application using HTML, CSS and JavaScript. The existing and proposed system will be listed below.

System Analysis

Existing System:

The existing communication system is not built as a software application. Everybody communicates with others physically or through the mails. To make this complex communication job simple and allows the users to participate in live communication and save unproductive time it is to be built as a software application.
Each and every user or employee of an organization has to register, get into his inbox and check for his mail which doesn't provide live communication resemblance to the user. This facility does not categorize the users depending on their interests. This type of communication channel fails in providing effective user-friendly communication between the users. If this channel grows up to some extent then it will be harder to place some restrictions on the users. As a result, ineffective communication wastes the user time.

Proposed System:

The first step of analysis process involves the identification of need. The success of a system depends largely on how accurately a problem is defined, thoroughly investigated and satisfying the customer needs by providing user friendly environment

This system has been developed in order to overcome the difficulties encountered while using the mailing system for communication between the users. Providing user friendly communication channel, live communication facility, categorizing the users, logging the communication transaction, sending public & private messages, sending instant & offline messages, graphical communication are motivating factors for the development of this system.

Scope and Objectives:

- This can be treated as a product mainly used in chatting kind of communication products to exchange the information between the by storing the users info and his connection details, chatroom control
- Panel (invite, ban and allow) details and chatroom management details. It also helps the administrator to monitor the chatroom by generating different kinds of reports like users currently available in the chatroom, banned list of users and allows the users to view the offline messages individually. Lot of effort was put to make it user friendly.
- Optimum utilization of system is possible. All basic functionalities are provided.
- Reduces the user manual communication work.
- The wastage of time is reduced.
- It also helps in providing instant and offline communication.

More flexible, it means we can continue to use the same system even the no of users up to maximum level.

1. Design constraints

1.1 Software constraints :

Operating System: Windows 7 and above

Other Software's: Visual Studio code

1.2 Hardware Constraints

Pentium Processor	**:**	Pentium IV
RAM	**:**	8 MB minimum
Hard Disk	**:**	500 GB minimum
CD/ROM Drive	**:**	52X
VDU	**:**	VGA
Key Board	**:**	101 Standard

METHODOLOGY

SOURCE CODE

HTML FILE:

```
<html>
<head>
<script
src='https://cdn.scaledrone.com/scaledrone.min.js'></sc
ript>
<meta charset="utf-8">
<meta name="viewport" content="width=device-
width">
<style>
    body {
      background: #0098ff;
      display: flex;
      height: 100vh;
      margin: 0;
      align-items: center;
      justify-content: center;
      padding: 0 50px;
      font-family: -apple-system, BlinkMacSystemFont,
sans-serif;
    }
    video {
      max-width: calc(50% - 100px);
      margin: 0 50px;
      box-sizing: border-box;
      border-radius: 2px;
      padding: 0;
      background: white;
    }
    .copy {
      position: fixed;
      top: 10px;
      left: 50%;
      transform: translateX(-50%);
      font-size: 16px;
color: white;
    }
</style>
</head>
```

```
<body>
<div class="copy">Send your URL to a friend to start a
video call</div>
<video id="localVideo" autoplay muted></video>
<video id="remoteVideo" autoplay></video>

<script src="script.js"></script>
</body>S
</html>
```

JAVASCRIPT FILE:

```
// Generate random room name if needed
if (!location.hash) {
location.hash = Math.floor(Math.random() *
0xFFFFFF).toString(16);
}
constroomHash = location.hash.substring(1);

// TODO: Replace with your own channel ID
const drone = new
ScaleDrone('2CgoHIvhJGEDU3wE');
```

```
// Room name needs to be prefixed with 'observable-'
constroomName = 'observable-' + roomHash;
const configuration = {
iceServers: [{
urls: 'stun:stun.l.google.com:19302'
  }]
};
let room;
let pc;


function onSuccess() {};
function onError(error) {
console.error(error);
};

drone.on('open', error => {
  if (error) {
    return console.error(error);
  }
  room = drone.subscribe(roomName);
```

```
room.on('open', error => {
    if (error) {
onError(error);
    }
  });
  // We're connected to the room and received an array
of 'members'
  // connected to the room (including us). Signaling
server is ready.
room.on('members', members => {
    console.log('MEMBERS', members);
    // If we are the second user to connect to the room we
will be creating the offer
constisOfferer = members.length === 2;
startWebRTC(isOfferer);
  });
});

// Send signaling data via Scaledrone
function sendMessage(message) {
drone.publish({
    room: roomName,
    message
  });
}

function startWebRTC(isOfferer) {
  pc = new RTCPeerConnection(configuration);

  // 'onicecandidate' notifies us whenever an ICE agent
needs to deliver a
  // message to the other peer through the signaling
server
pc.onicecandidate = event => {
    if (event.candidate) {
sendMessage({'candidate': event.candidate});
    }
  };

  // If user is offerer let the 'negotiationneeded' event
create the offer
  if (isOfferer) {
pc.onnegotiationneeded = () => {
pc.createOffer().then(localDescCreated).catch(onError);
```

```
    }
  }

  // When a remote stream arrives display it in the
#remoteVideo element
pc.onaddstream = event => {
remoteVideo.srcObject = event.stream;
  };

navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
  }).then(stream => {
    // Display your local video in #localVideo element
localVideo.srcObject = stream;
    // Add your stream to be sent to the conneting peer
pc.addStream(stream);
  }, onError);

  // Listen to signaling data from Scaledrone
room.on('data', (message, client) => {
    // Message was sent by us
    if (client.id === drone.clientId) {
      return;
    }

    if (message.sdp) {
      // This is called after receiving an offer or answer
from another peer
pc.setRemoteDescription(new
RTCSessionDescription(message.sdp), () => {
        // When receiving an offer lets answer it
        if (pc.remoteDescription.type === 'offer') {
pc.createAnswer().then(localDescCreated).catch(onErro
r);
        }
      }, onError);
    } else if (message.candidate) {
      // Add the new ICE candidate to our connections
remote description
pc.addIceCandidate(
        new RTCIceCandidate(message.candidate),
onSuccess, onError
```

```
      );
    }
  });
}

function localDescCreated(desc) {
pc.setLocalDescription(
desc,
    () =>sendMessage({'sdp': pc.localDescription}),
onError
  );
}
```

CONCLUSION

Thus, a video chat application was created using the given code and executed successfully. The application can be used to communicate between two individuals by sharing their url after opening the link if they have a webcam and a mic.