WebRTC Peer-to-Peer Video Conferencing

Overview

Attribute	Value
Difficulty	⭐⭐⭐ Medium
Estimated Time	6-8 hours
Primary Skills	WebSocket, JavaScript, WebRTC APIs

Background

Intellicon wants to explore browser-native video calling capabilities without relying on third-party services like Twilio or Zoom. A lightweight P2P solution could be embedded directly into the contact center for agent-to-agent or agent-to-supervisor video check-ins.

User Story

As a support agent working remotely,
I want to quickly start a video call with my supervisor by sharing a simple link,
So that we can discuss complex customer issues face-to-face without leaving the browser.

Acceptance Criteria

#	Criteria	Priority
1	User lands on homepage and enters their display name	Must Have
2	System generates a unique meeting room URL	Must Have
3	User can copy/share the meeting link	Must Have

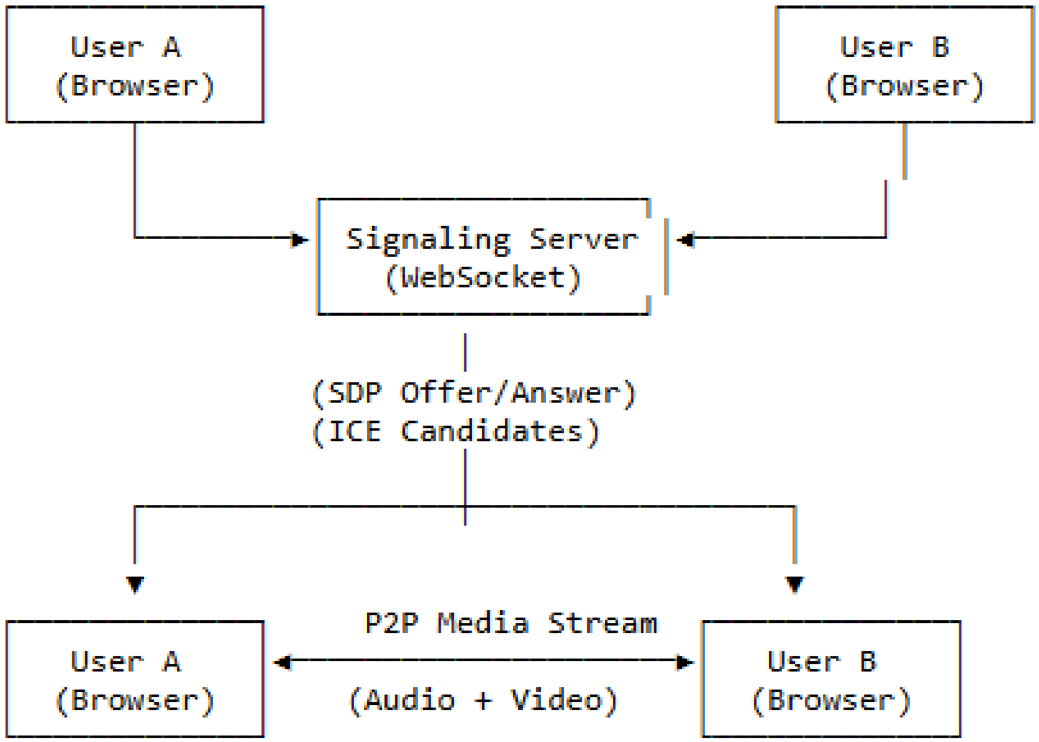
4	Second user opens link, enters their name, and joins the room	Must Have
5	Both users see each other's video and hear audio	Must Have
6	Either user can mute audio or disable video	Must Have
7	Either user can end the call	Must Have
8	Works on Chrome and Firefox (minimum)	Must Have
9	Graceful handling when one user disconnects	Should Have
10	Mobile-responsive UI	Should Have

Technical Requirements

Core Components

1. **Signaling Server** - WebSocket or Socket.io for connection negotiation
2. **STUN Server** - Google's public STUN servers are acceptable (`stun:stun.l.google.com:19302`)
3. **ICE Candidate Exchange** - Proper handling of network traversal
4. **Media Stream Handling** - `getUserMedia` API for camera/microphone access

Architecture Diagram



Recommended Stack

- **Frontend**: Vanilla JS, React, or Vue
- **Signaling Server**: Node.js with Socket.io or `ws` library
- **Styling**: Tailwind CSS or Bootstrap

Bonus Features (Extra Points)

Feature	Points
Screen sharing capability	+15
Text chat alongside video	+10
Recording the call locally	+20
Support for 3+ participants (mesh or SFU)	+25
Virtual background or blur effect	+15

Connection quality indicator	+10
Reconnection on network drop	+10

Deliverables Checklist

- Working web application
- README.md with:
  - Setup instructions
  - Dependencies list
  - How to run locally
  - Known limitations
- Architecture diagram showing signaling flow
- Demo video describing the working of the project
- Chat link/screenshots of the prompts with the AI tool
- Name of the Ai tool used for development

Evaluation Rubric

Criteria	Weight	0 Points	5 Points	10 Points
Core Functionality	30%	Does not work	Basic calling works with issues	Flawless video/audio calling
Code Quality	20%	Messy, no structure	Organized, some comments	Clean, well-documented, modular
UI/UX Design	15%	Basic/ugly	Functional and clean	Polished, intuitive, responsive
Error Handling	15%	Crashes on errors	Basic error messages	Graceful recovery, helpful messages
Innovation/Bonus	10%	No extras	1-2 bonus features	3+ bonus features implemented
Presentation	10%	Poor demo	Clear explanation	Engaging demo, good Q&A

Total: 100 points

Resources

Documentation

- WebRTC API (MDN)
- WebRTC Samples
- Socket.io Documentation

STUN Servers (Free)

```javascript
const iceServers = [
  { urls: 'stun:stun.l.google.com:19302' },
  { urls: 'stun:stun1.l.google.com:19302' }
];
```

Code Snippets

Getting User Media:

```javascript
const stream = await navigator.mediaDevices.getUserMedia({
  video: true,
  audio: true
});
```

Creating Peer Connection:

```javascript
const peerConnection = new RTCPeerConnection({ iceServers });
stream.getTracks().forEach(track => {
  peerConnection.addTrack(track, stream);
});
```

Common Pitfalls to Avoid

1. **Not handling ICE candidate timing** - Candidates may arrive before or after SDP exchange
2. **Forgetting to handle browser permissions** - Always wrap `getUserMedia` in try/catch
3. **Not cleaning up connections** - Remove tracks and close connections on disconnect
4. **Ignoring HTTPS requirement** - WebRTC requires secure context (HTTPS or localhost)
5. **Hardcoding room IDs** - Generate unique IDs for each room

---

Testing Checklist

- Test with same network (local)
- Test across different networks (NAT traversal)
- Test with camera off / mic off combinations
- Test with one user leaving mid-call
- Test with slow network (throttle in DevTools)
- Test on mobile browser
