**🧱 Project Overview: “SIPComm”**

A lightweight, SIP-based calling app (web/mobile) that lets users register with a SIP server and make audio calls to each other. Later, you can expand it with video, presence, chat, and hardware integration.

**🖥️ Components & Architecture**

1. **Frontend (Client)**
   * React or React Native app
   * Uses JsSIP for SIP signaling over WebRTC (browser or mobile-friendly)
2. **Backend (Optional for MVP)**
   * Node.js server to manage authentication, logging, and call state (expandable)
   * Database: Firebase (easy setup) or PostgreSQL (structured)
3. **SIP Server**
   * Use Asterisk, FreeSWITCH, or a hosted SIP provider like Kamailio or Twilio SIP Domain
   * Handles user registration and SIP signaling
4. **Media Handling**
   * Peer-to-peer via WebRTC for audio (handled by JsSIP)
   * ICE/STUN/TURN setup for NAT traversal

**🔧 MVP Features**

| **Feature** | **Description** |
| --- | --- |
| SIP Registration | User logs in and registers with SIP credentials |
| Contact List | Display hardcoded or registered peers |
| Outgoing Call | User can initiate SIP audio call (INVITE) |
| Incoming Call | UI to answer/reject when another peer calls |
| Call Status UI | Show ringing, connected, and end states |
| Audio Stream | Use WebRTC for real-time audio |
| Basic Call Log | Save recent call data (localStorage or DB) |

**🪄 Getting Started**

1. **Set Up SIP Server**
   * Use a cloud-based Asterisk or a hosted SIP provider
   * Create SIP extensions for at least 2 users
2. **Build the Client**
   * Use JsSIP with WebRTC
   * Register SIP URI like sip:alice@yourserver.com
3. **Implement Call Control**
   * Initiate call with ua.call('sip:bob@yourserver.com')
   * Handle SIP events: accepted, ended, failed
4. **Test Call Functionality**
   * Run two instances (or different browsers/devices)
   * Verify audio connectivity between users

**🧠 Future Feature Ideas**

* Video calling support
* Call recording and playback
* Presence (subscribe to availability)
* Chat messaging (SIP SIMPLE)
* Push notifications for mobile calls

If you’d like, I can help you scaffold the folder structure, write the registration logic, or walk you through setting up JsSIP with a real server. Want to build the SIP client first or set up the server side?

Can we start right now using VS Code?

Absolutely, Jagdev—we’re ready to roll! If you've got Visual Studio Code open, let’s take the first step toward building your SIP-based app. Here’s what we’ll do to kick things off:

**🧰 Step 1: Set Up Your Project Environment**

1. **Create a Project Folder** In your terminal or file explorer:

bash

mkdir sipcomm-client

cd sipcomm-client

1. **Initialize a New Web Project** Let’s use Vite + React for fast setup:

bash

npm create vite@latest

# Choose 'React' and name it (e.g., "sipcomm-client")

cd sipcomm-client

npm install

1. **Open in VS Code** If you’re not there already:

bash

code .

**📦 Step 2: Add JsSIP Library**

This library handles SIP over WebSocket using WebRTC:

bash

npm install jssip

Then inside your React app (e.g., App.jsx), you can import it:

js

import JsSIP from 'jssip';

**🧪 Step 3: Create a Simple SIP Registration Script**

Let’s start with just connecting to the SIP server. Here's a quick boilerplate you can try in App.jsx once you’ve got everything installed:

js

import { useEffect } from 'react';

import JsSIP from 'jssip';

function App() {

useEffect(() => {

const socket = new JsSIP.WebSocketInterface('wss://your-sip-server.com/ws');

const configuration = {

sockets: [socket],

uri: 'sip:your-username@your-sip-server.com',

password: 'your-password',

};

const ua = new JsSIP.UA(configuration);

ua.on('connected', () => console.log('Connected to SIP server'));

ua.on('registered', () => console.log('SIP account registered'));

ua.on('registrationFailed', (e) => console.error('Registration failed:', e));

ua.start();

return () => ua.stop();

}, []);

return <h1>🚀 SIPComm Client Ready</h1>;

}

export default App;

If your SIP server isn't ready yet, I can walk you through setting one up (like Asterisk or a free SIP testing service). Want to try SIP registration first, or do you have a SIP server ready to test against?

How I can make use of SIP H.248 from Richerlink Device

To use SIP and H.248 protocols on a Richerlink device like the RL841GWV-D or RL821GWV, you’ll need to configure it for VoIP services, typically for voice over fiber (FTTH) deployments. Here’s a simplified guide to get you started:

**🛠️ Step-by-Step Setup Overview**

1. **Access the Device Interface** Log in to the Richerlink ONU’s web interface using its IP address (usually something like 192.168.1.1) via a browser.
2. **Enable VoIP Services** Navigate to the VoIP or Voice settings section. You’ll typically find options to enable either SIP or H.248 protocol.
3. **Configure SIP Settings** If using SIP:
   * Enter the SIP server IP or domain
   * Set the SIP port (default is usually 5060)
   * Input your VoIP account credentials (username, password)
   * Set codecs (e.g., G.711, G.729) and DTMF mode
4. **Configure H.248 Settings** If using H.248:
   * Enter the MGC (Media Gateway Controller) IP address
   * Set the H.248 port (default is 2944)
   * Configure the terminal ID and profile
   * Ensure the device is in H.248 AG mode (this may require CLI access or firmware support)
5. **Save and Reboot** After configuration, save the settings and reboot the device to apply changes.
6. **Verify Registration** Check the status LEDs or the web interface to confirm that the device has successfully registered with the SIP server or MGC.

**📌 Notes**

* Richerlink ONUs support both SIP and H.248, but only one protocol is active at a time.
* Make sure your ISP or VoIP provider supports the protocol you choose and provides the necessary credentials.
* For H.248, advanced configuration may require CLI access or vendor-specific tools.

You can find more technical details in the Richerlink RL841GWV-D product page or explore Huawei’s H.248 AG configuration guide for similar setups.

If you tell me your exact model and what you're trying to achieve (like voice calls, triple-play services, etc.), I can tailor the steps even more.

What are the advantages of using SIP over H.248?

Great question, Jagdev. SIP (Session Initiation Protocol) and H.248 (also known as Megaco) serve different architectural roles in VoIP, but SIP has several advantages that make it more popular in many modern deployments:

**🌐 Key Advantages of SIP over H.248**

* **Peer-to-Peer Architecture** SIP is a decentralized, peer-to-peer signaling protocol, which means endpoints (like IP phones or soft clients) can initiate and manage sessions directly. H.248, on the other hand, uses a master-slave model where a Media Gateway Controller (MGC) controls Media Gateways (MGs).
* **Simplicity and Flexibility** SIP is text-based and modeled after HTTP, making it easier to read, debug, and extend. Developers can quickly implement new features or integrate SIP with web-based services.
* **Interoperability and Extensibility** SIP is widely adopted and supported across a broad range of devices and platforms. It’s also highly extensible, allowing for integration with presence, messaging, and video conferencing systems.
* **Lower Cost and Complexity** SIP endpoints can be simpler and cheaper because they don’t require a centralized controller like H.248 does. This makes SIP ideal for distributed networks and smaller deployments.
* **Better Fit for Modern Networks** SIP works well with NAT traversal, mobile networks, and cloud-based VoIP services. It’s also the backbone of many unified communications platforms today.
* **Developer-Friendly** Because SIP is modular and well-documented, it’s easier for developers outside the traditional telecom space to build applications around it.

That said, H.248 still has its place—especially in large-scale carrier-grade networks where centralized control of media gateways is essential. But for most enterprise and residential VoIP setups, SIP is the go-to.

If you're considering which to use on your Richerlink device, SIP is usually the more straightforward and future-proof choice. Want help configuring it?

Edit in a page

What are some practical uses of SIP in VoIP?

SIP (Session Initiation Protocol) is the backbone of many VoIP systems, and its practical uses go far beyond just making phone calls. Here’s how it powers real-world communication:

**📞 Everyday Uses of SIP in VoIP**

* **Voice Calls Over the Internet** SIP initiates, manages, and terminates voice calls between users—whether it’s a softphone app, IP desk phone, or browser-based client.
* **Video Conferencing** SIP supports multimedia sessions, so it’s used to set up and manage video calls and virtual meetings across platforms.
* **Unified Communications (UC)** SIP enables integration of voice, video, messaging, and presence into a single platform—think Microsoft Teams or Zoom Phone.
* **Call Centers and IVR Systems** SIP routes calls intelligently based on rules, enabling features like auto-attendants, call queues, and skill-based routing.
* **Remote Work Enablement** Employees can use SIP-enabled apps to make and receive calls from anywhere with an internet connection—no need for physical phone lines.
* **SIP Trunking** Businesses use SIP trunks to connect their on-premise PBX systems to the public telephone network via the internet, replacing traditional ISDN lines.
* **Mobile VoIP Apps** Apps like WhatsApp, Viber, and many enterprise tools use SIP under the hood to manage call sessions and messaging.
* **Disaster Recovery and Redundancy** SIP allows for failover routing—if one server goes down, calls can be rerouted to another, ensuring business continuity.
* **Presence and Instant Messaging** SIP supports presence information (e.g., available, busy, away) and real-time messaging, which is key for collaboration tools.
* **Cost Optimization** SIP reduces the need for physical infrastructure and long-distance charges, making it ideal for startups and global teams.

If you're thinking about deploying SIP in your setup, I can help you map out the architecture or choose the right tools. Want to dive into SIP trunking or explore how it fits into a hybrid cloud setup?

Can you help me develop a software project that can make use of SIP compared to VOIp

Absolutely, Jagdev. SIP *is* a part of VoIP—but if you want a software project that really emphasizes SIP’s signaling capabilities over generic “VoIP,” there are some cool angles we can take. Here's a practical project idea and a blueprint to build on:

**🧠 Project Idea: SIP-Powered Smart Intercom System**

Imagine a cross-platform desktop/mobile app that lets users:

* Initiate voice/video calls between rooms or locations
* Use presence detection (online, offline, DND)
* Route calls intelligently based on availability
* Record and store conversations
* Connect to SIP-based door entry hardware or desk phones

Instead of building raw VoIP transport (like media codecs and RTP), you’d focus on SIP signaling: establishing, modifying, and terminating sessions. The media would piggyback on existing libraries.

**🛠️ Tech Stack Suggestion**

* **Frontend**: React (Web), React Native or Flutter (Mobile)
* **Backend**: Node.js or Python (for managing users, state, and logs)
* **SIP Stack**:
  + JSSIP (for browser-based SIP over WebRTC)
  + PJSIP (for mobile/native support)
  + Or Asterisk as a SIP server with REST hooks
* **Database**: PostgreSQL or Firebase (depending on real-time vs structured data needs)
* **Media Relay (optional)**: FreeSWITCH or Janus Gateway

**🔄 Core Features to Build**

1. **SIP Account Registration** Users create their VoIP profiles and authenticate to a SIP server.
2. **Contact Directory & Presence** Show who's online, away, or offline via SIP OPTIONS or SUBSCRIBE/NOTIFY.
3. **Call Control** Build UI for incoming/outgoing calls using SIP INVITE, BYE, etc.
4. **Media Handling** Use WebRTC or native clients to establish actual audio/video streams post-SIP negotiation.
5. **Voicemail & Call Logging** Record missed calls, store voicemails (optionally using Asterisk’s backend).
6. **Cross-Device Sync** Support logging in from multiple devices and syncing call history.

**✨ Why SIP (Not Just VoIP)?**

* You’ll focus on **protocol-level logic**, not just media transport.
* Enables **carrier-level features** like call transfer, DND, hold, and BLF (Busy Lamp Field).
* Prepares you for scalable SIP trunking integration with telecom networks.
* You can test with **real SIP providers** and even deploy on IP phones.

If this sounds like a direction you’d want to pursue, I can help scaffold the architecture, write a basic project outline or even walk through specific modules like user registration or signaling flow. Want to sketch out a first version of the system or pick a SIP library to start with?

Yes want to sketch out first version

Awesome, Jagdev! Let’s sketch out the first version of your SIP-powered smart communication system. Think of this as your project’s MVP—Minimum Viable Product—that you can build on in iterations.

**🧱 Project Overview: “SIPComm”**

A lightweight, SIP-based calling app (web/mobile) that lets users register with a SIP server and make audio calls to each other. Later, you can expand it with video, presence, chat, and hardware integration.

**🖥️ Components & Architecture**

1. **Frontend (Client)**
   * React or React Native app
   * Uses JsSIP for SIP signaling over WebRTC (browser or mobile-friendly)
2. **Backend (Optional for MVP)**
   * Node.js server to manage authentication, logging, and call state (expandable)
   * Database: Firebase (easy setup) or PostgreSQL (structured)
3. **SIP Server**
   * Use Asterisk, FreeSWITCH, or a hosted SIP provider like Kamailio or Twilio SIP Domain
   * Handles user registration and SIP signaling
4. **Media Handling**
   * Peer-to-peer via WebRTC for audio (handled by JsSIP)
   * ICE/STUN/TURN setup for NAT traversal

**🔧 MVP Features**

| **Feature** | **Description** |
| --- | --- |
| SIP Registration | User logs in and registers with SIP credentials |
| Contact List | Display hardcoded or registered peers |
| Outgoing Call | User can initiate SIP audio call (INVITE) |
| Incoming Call | UI to answer/reject when another peer calls |
| Call Status UI | Show ringing, connected, and end states |
| Audio Stream | Use WebRTC for real-time audio |
| Basic Call Log | Save recent call data (localStorage or DB) |

**🪄 Getting Started**

1. **Set Up SIP Server**
   * Use a cloud-based Asterisk or a hosted SIP provider
   * Create SIP extensions for at least 2 users
2. **Build the Client**
   * Use JsSIP with WebRTC
   * Register SIP URI like sip:alice@yourserver.com
3. **Implement Call Control**
   * Initiate call with ua.call('sip:bob@yourserver.com')
   * Handle SIP events: accepted, ended, failed
4. **Test Call Functionality**
   * Run two instances (or different browsers/devices)
   * Verify audio connectivity between users

**🧠 Future Feature Ideas**

* Video calling support
* Call recording and playback
* Presence (subscribe to availability)
* Chat messaging (SIP SIMPLE)
* Push notifications for mobile calls

If you’d like, I can help you scaffold the folder structure, write the registration logic, or walk you through setting up JsSIP with a real server. Want to build the SIP client first or set up the server side?

Can we start right now using VS Code

Absolutely, Jagdev—we’re ready to roll! If you've got Visual Studio Code open, let’s take the first step toward building your SIP-based app. Here’s what we’ll do to kick things off:

**🧰 Step 1: Set Up Your Project Environment**

1. **Create a Project Folder** In your terminal or file explorer:

bash

mkdir sipcomm-client

cd sipcomm-client

1. **Initialize a New Web Project** Let’s use Vite + React for fast setup:

bash

npm create vite@latest

# Choose 'React' and name it (e.g., "sipcomm-client")

cd sipcomm-client

npm install

1. **Open in VS Code** If you’re not there already:

bash

code .

**📦 Step 2: Add JsSIP Library**

This library handles SIP over WebSocket using WebRTC:

bash

npm install jssip

Then inside your React app (e.g., App.jsx), you can import it:

js

import JsSIP from 'jssip';

**🧪 Step 3: Create a Simple SIP Registration Script**

Let’s start with just connecting to the SIP server. Here's a quick boilerplate you can try in App.jsx once you’ve got everything installed:

js

import { useEffect } from 'react';

import JsSIP from 'jssip';

function App() {

useEffect(() => {

const socket = new JsSIP.WebSocketInterface('wss://your-sip-server.com/ws');

const configuration = {

sockets: [socket],

uri: 'sip:your-username@your-sip-server.com',

password: 'your-password',

};

const ua = new JsSIP.UA(configuration);

ua.on('connected', () => console.log('Connected to SIP server'));

ua.on('registered', () => console.log('SIP account registered'));

ua.on('registrationFailed', (e) => console.error('Registration failed:', e));

ua.start();

return () => ua.stop();

}, []);

return <h1>🚀 SIPComm Client Ready</h1>;

}

export default App;

If your SIP server isn't ready yet, I can walk you through setting one up (like Asterisk or a free SIP testing service). Want to try SIP registration first, or do you have a SIP server ready to test against?

Today

For Vite Project name suggest a suitable name in realtion to this sipcom-client as installed the step 2 commands inside the cureent project

Since your current Vite project houses the frontend portion and you're calling it sipcom-client, it makes sense to name the Vite project something that reflects that role while tying into the overall system name. Here are a few clean, relevant suggestions:

**🔧 Project Name Suggestions for Vite Frontend**

* sipcomm-ui — emphasizes the user interface layer
* sipcomm-portal — if it’s meant as the main access point
* sipcomm-dialer — gives a softphone or call-oriented feel
* sipcomm-front — simple, direct, and dev-friendly
* sipcomm-console — if it’s more dashboard-like or admin-focused
* callwave-client — a bit more stylish while staying on theme
* voxmesh-ui — futuristic branding with "voice" and "mesh" hints

If you’re keeping it literal and aligned with your current folder structure, sipcomm-client works fine too. Want to give the whole system a brand identity while we're at it? I can help design a logo, tagline, or color scheme next.

What framework i will choose from the list provided

It depends a bit on your comfort level and goals, but let’s break down the main frameworks from the Vite project options to help you choose wisely:

**⚙️ Common Vite Project Options (and when to choose them)**

| **Framework** | **Best for…** |
| --- | --- |
| **React** | Building dynamic UIs with component-based architecture. Great community, excellent for modern web apps. |
| **Vue** | Similar to React, but with a more approachable syntax and simpler learning curve. Very beginner-friendly. |
| **Svelte** | Super lightweight and reactive. Ideal if you want optimal performance with minimal boilerplate. |
| **Vanilla JS** | Raw JavaScript with no framework overhead—best for learning or very lightweight apps. |
| **Lit** | Creating fast Web Components. Excellent if you want reusable, framework-agnostic elements. |
| **Preact** | A lightweight alternative to React, with a smaller bundle size. Great for low-resource environments. |

**🧠 My Recommendation for sipcomm-client**

You’re building a modern, SIP-enabled communication interface—probably with real-time features, WebRTC integration, and dynamic UI updates. That makes **React** a strong fit:

* Seamless integration with JsSIP and WebRTC
* Rich ecosystem (hooks, state managers like Redux or Zustand)
* React + Vite makes for a lightning-fast dev environment

So unless you're drawn to Vue or Svelte, go with **React** for this one—it’ll give you power, flexibility, and tons of community support.

Want me to help you set up the file structure or add routing and state management next?

What variant to select next

Once you've chosen **React** as your framework in Vite, the next step in the setup wizard usually asks what variant of React you'd like to use:

**⚙️ React Variant Options in Vite**

You’ll likely see two main choices:

1. **JavaScript**
   * Use this if you want to move fast and keep things simple.
   * No need to worry about typing—great for quick prototyping.
   * Easier if you’re still getting used to React or SIP integration.
2. **TypeScript**
   * Adds static typing, which helps catch errors early and improves code completion.
   * Highly recommended for production-scale apps or if you plan to expand this project long-term.
   * Slight learning curve if you're not used to types, but definitely worth it.

**✅ My Suggestion**

Go with **React + TypeScript** unless you're in a hurry to prototype or you're new to both React and TypeScript. Since SIP and WebRTC can involve complex data (e.g. session states, codecs, ICE candidates), TypeScript can help you manage it all safely and cleanly.

If you're ready, select that variant in the terminal prompt and we’ll dive into writing your first SIP registration component! Want to go with TypeScript? I can guide you through organizing your folder structure next.

typscript +SWC is also available. Is it more advantageous or just going to prove cumbersome.

Great observation, Jagdev. Choosing **TypeScript + SWC** as your Vite project setup can actually give you the best of both worlds—developer friendliness *and* lightning-fast builds.

Here’s a quick breakdown to help you decide:

**⚡ What is SWC?**

SWC (Speedy Web Compiler) is a Rust-based compiler that replaces Babel or TSC (TypeScript Compiler) for transforming and bundling your TypeScript and JavaScript code. It’s designed for performance—**blazingly fast**, especially in large projects.

**🚀 Benefits of TypeScript + SWC**

* **Superfast Build & HMR Times** Compiles and hot-reloads significantly faster than traditional setups with Babel or TSC.
* **Modern Syntax Support** Handles modern JS and TS syntax just as well—if not better—with less overhead.
* **Ideal for Larger Projects** If you plan to grow SIPComm into a multi-feature platform, SWC’s performance pays off over time.
* **Native Vite Support** Vite works great with SWC now, so there’s minimal config effort. You won’t feel the "cumbersome" part.

**👀 What’s the Catch?**

* **No Type Checking (by Default)** SWC only handles code transformation—not type-checking. You’ll still need to run tsc --noEmit or use your IDE for TypeScript errors.
* **Some Plugin Limitations** If you're relying on advanced Babel plugins, SWC may not fully support them (though this is rare in a React+Ts setup).

**✅ Verdict**

If you're aiming for a smooth development experience with quick reloads and a clean React + TypeScript stack, **go ahead and choose TypeScript + SWC**. It’ll feel snappy, modern, and forward-looking.

Once you’ve selected it, let’s wire up your first JsSIP session in TypeScript. Ready to lay out your SIPClient.ts service next?

INVITE sip:[service]@[remote\_ip]:[remote\_port] SIP/2.0

Via: SIP/2.0/[transport] [local\_ip]:[local\_port];branch=[branch]

From: sipp <sip:sipp@[local\_ip]:[local\_port]>;tag=[call\_number]

To: sut <sip:[service]@[remote\_ip]:[remote\_port]>

Call-ID: [call\_id]

CSeq: 1 INVITE

Contact: sip:sipp@[local\_ip]:[local\_port]

Max-Forwards: 70

Subject: Performance Test

Content-Type: application/sdp

Content-Length: [len]

v=0

o=user1 53655765 2353687637 IN IP[local\_ip\_type] [local\_ip]

s=-

c=IN IP[media\_ip\_type] [media\_ip]

t=0 0

m=audio [media\_port] RTP/AVP 0

a=rtpmap:0 PCMU/8000

On http://localhost:5173 What should I write inside the url as specific

If you're asking what you should type into the **text input field** on your SIPComm app at http://localhost:5173, that’s where you enter a **valid SIP URI**—like an address for the user you want to call.

Here are a few examples depending on what service or test setup you're using:

**✅ If You're Using a Free SIP Testing Service:**

* **Sonetel Echo Bot**:
* sip:echo@sip.sonetel.com

(This will call an echo test—you’ll hear your voice played back.)

* **Test call with someone else on same SIP server**:
* sip:friend@sip.sonetel.com

**🔧 If You're Hosting Your Own SIP Server (e.g., Asterisk or FreeSWITCH):**

* Replace with your SIP extension and server:
* sip:101@your-sip-domain.com
* Or if you’re testing on localhost:
* sip:101@127.0.0.1

Just make sure:

* The SIP URI **matches an actual user** registered on that server
* Your SIP account is **successfully registered** (watch the browser console for “✅ Registered”)

Would you like help finding a real SIP URI to call based on the server you’ve set up or testing tool you're using? I can guide you through a real-time SIP call next! 🎧📞

Using Port Forwarding as public will it work at some other end

Yes, Jagdev — using **port forwarding** can absolutely make your local SIP/WebRTC app accessible to remote users, but there are some caveats and tweaks needed to make it truly work end-to-end:

**🌍 What Port Forwarding Does in SIP/WebRTC**

When you forward ports on your router (for example, your home or test network), you allow traffic from the public internet to be sent directly to your local machine. For SIP and media:

| **Purpose** | **Typical Protocol** | **Port (Default)** |
| --- | --- | --- |
| SIP signaling (UDP/TCP) | SIP | 5060 |
| SIP over WebSocket | WS/WSS | 5066 / 7443 |
| RTP (media) | UDP | 10000–20000 |
| STUN/TURN (for NAT) | UDP | 3478, 5349 |

**✅ It Will Work If:**

* Your **public IP** is reachable (you can use a dynamic DNS if it's not static).
* The correct SIP ports are forwarded to your local device.
* You're using **STUN/TURN servers** for NAT traversal (especially with WebRTC).
* Your SIP/WebRTC stack (e.g., JsSIP) is configured to know the external IP address of your machine or uses ICE candidates effectively.
* Remote clients are connecting to ws://your-public-ip:port or via your domain with proper certificates (for wss://).

**⚠️ Common Pitfalls**

* **NAT issues**: Media might not flow even if signaling does (use STUN or TURN).
* **Firewall blocks**: Local firewall might block inbound SIP/RTP traffic.
* **ISP restrictions**: Some ISPs block SIP ports like 5060 to deter VoIP abuse.
* **No SSL cert**: For wss:// (secure WebSocket), remote browsers will need HTTPS with a valid certificate.

**🧪 How to Test from the Outside**

1. Open your SIP client locally.
2. On your router:
   * Forward 5060 → your local IP (for SIP)
   * Forward 10000–20000 → for RTP (media)
   * Optionally forward 7443 or 5066 for WebSocket SIP
3. From another network (or mobile browser), access:
4. wss://your.public.ip:7443

Or whatever port your SIP WebSocket server is listening on.

1. In your App.tsx, point JsSIP here:

ts

const socket = new JsSIP.WebSocketInterface('wss://your-public-ip:7443');

If you’re curious, I can help you:

* Detect your public IP
* Set up free STUN/TURN servers
* Or use services like **ngrok** or **Cloudflare Tunnel** to avoid manual port forwarding altogether

Want help tunneling your SIP app securely without router headaches? 😊

I like help finding a real SIP URI to call based on the server you’ve set up or testing tool you're using? I can guide you through a real-time SIP call next! 🎧📞

Perfect, Jagdev! Let’s get you a **real SIP URI** you can dial right now from your SIPComm client to test your setup.

**✅ Public SIP URIs You Can Call for Testing**

Here are some **free and reliable SIP addresses** you can use to test audio, echo, and call flow:

| **SIP URI** | **Description** |
| --- | --- |
| sip:echo@iptel.org | Echo test — speaks back whatever you say |
| sip:music@iptel.org | Plays music on hold — great for audio quality testing |
| sip:904@mouselike.org | Mouselike echo test — includes audio loopback |
| sip:echo@sip.sonetel.com | Sonetel echo bot — works well with JsSIP |
| sip:lenny@sip.itslenny.com | “Lenny” bot — a fun voice that talks back (great for testing call flow) |

These are **publicly accessible**, so you don’t need to register an account with those domains to call them. Just make sure your SIP UA is registered with a valid SIP provider (like Sonetel, sip2sip.info, or your own Asterisk server).

**🧪 How to Use It in Your App**

In your SIPComm dialer input field, try pasting:

sip:echo@iptel.org

Then hit **📞 Call**. You should hear your own voice echoed back if your microphone and WebRTC are working correctly.

Want help registering a free SIP account (like with sip2sip.info or Sonetel) so you can receive calls too? Or shall we test incoming call handling next? 🔁📲

I am using this at http://127.0.0.1:5173

Perfect—that means your SIPComm client is running **locally**, which is ideal while you're developing and debugging. However, if you want to **connect with a real-world SIP server** or **test calls to public SIP URIs**, there are a few things to be aware of when running from http://127.0.0.1:5173:

**🌐 Local vs Remote SIP Communication**

| **Scenario** | **What Works** | **What May Need Extra Setup** |
| --- | --- | --- |
| Testing JsSIP client UI | ✅ Works locally |  |
| SIP registration with public provider | ✅ Works if WebSocket URI is reachable |  |
| Calling public SIP URIs (e.g., sip:echo@iptel.org) | ✅ Works, assuming proper config |  |
| Receiving calls from outside | ⚠️ Only works if SIP/WebSocket endpoint is publicly accessible (via port forwarding or tunneling) |  |
| Media (audio) streaming | ⚠️ May require STUN/TURN servers due to NAT |  |

**✅ How to Make It Work from Localhost**

Here’s what to check:

1. **Your SIP UA (**startSipUA**) is pointed to a valid SIP WebSocket server**:

ts

wsServers: 'wss://sip.sonetel.com/ws' // or your provider's WS URI

uri: 'sip:your-username@sipserver.com'

1. **You are calling a public SIP URI**, like:
2. sip:echo@iptel.org
3. **You’re running the app in Chrome or Firefox**, which support WebRTC and JsSIP.

**🧠 Bonus: Want to Expose Your App to the Internet?**

If you want someone *outside your network* to access your SIPComm app and call you:

* Use a tunneling service like ngrok:

bash

npx ngrok http 5173

This gives you a public HTTPS link (e.g., https://abc123.ngrok.io)

* Update your SIP WebSocket config to use your **public SIP server**
* Share that ngrok URL with your testing partner

Let’s do a test call next! Do you want to:

* Try one of those echo bots?
* Or register your own free SIP account so others can call *you*?