P2P Architecture

QUICKCONNECT APP

OSMAN GHANI GRANDAY FA21-BSE-024
ABDULLAH FA21-BSE-016
ABID SP21-BSE-085

QuickConnect P2P Video Call App Documentation

Overview

QuickConnect is a simple, lightweight peer-to-peer (P2P) video calling application built using WebRTC technology. It allows users to initiate video calls without the need for registration or complex setup. The app uses a WebSocket signaling server to facilitate connection establishment between peers.

Features

- **Peer-to-Peer Connection:** Establishes a direct, encrypted connection between devices.
- **WebRTC Powered:** Real-time, low-latency video and audio communication.
- **No Registration Required:** Users can start calls by sharing unique room links.
- **Cross-Platform Compatibility:** Works on both desktop and mobile browsers.
- **Simplified Interface:** Minimalist design for ease of use.

Project Structure

QuickConnect/

├── client/

│ ├── index.html

│ ├── style.css

│ └── script.js

├── server/

│ └── server.js

├── .gitignore

├── package.json

└── README.md

Dependencies

- Node.js
- WebSocket (ws) library for signaling server

Setting Up the Project

1. **Install Node.js:**
    - Download and install Node.js from https://nodejs.org/.
2. **Set Up the Project Directory:**
    - Create the project directory and navigate into it.

3. **Initialize npm and Install Dependencies:**

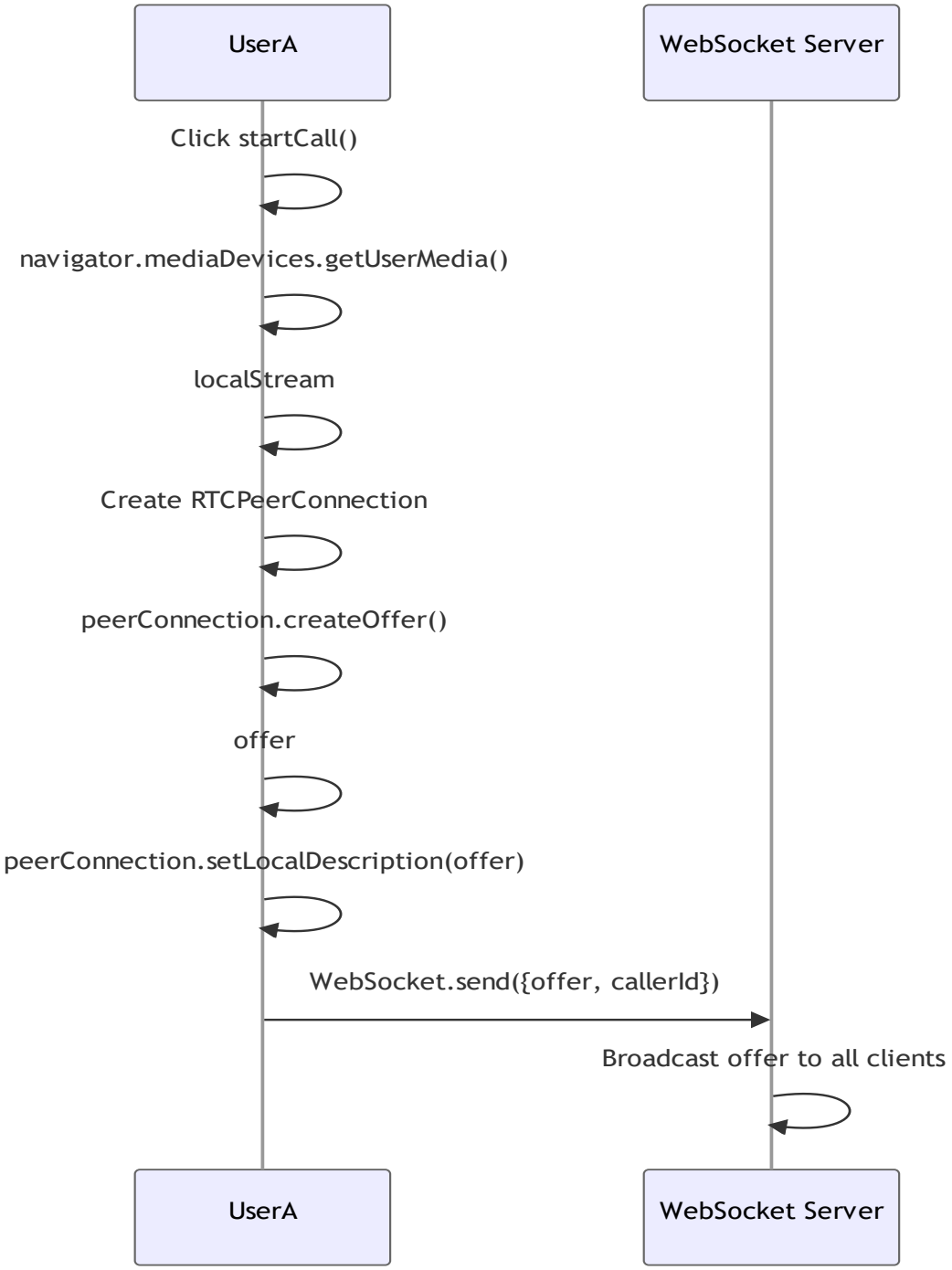
npm init -y

npm install ws

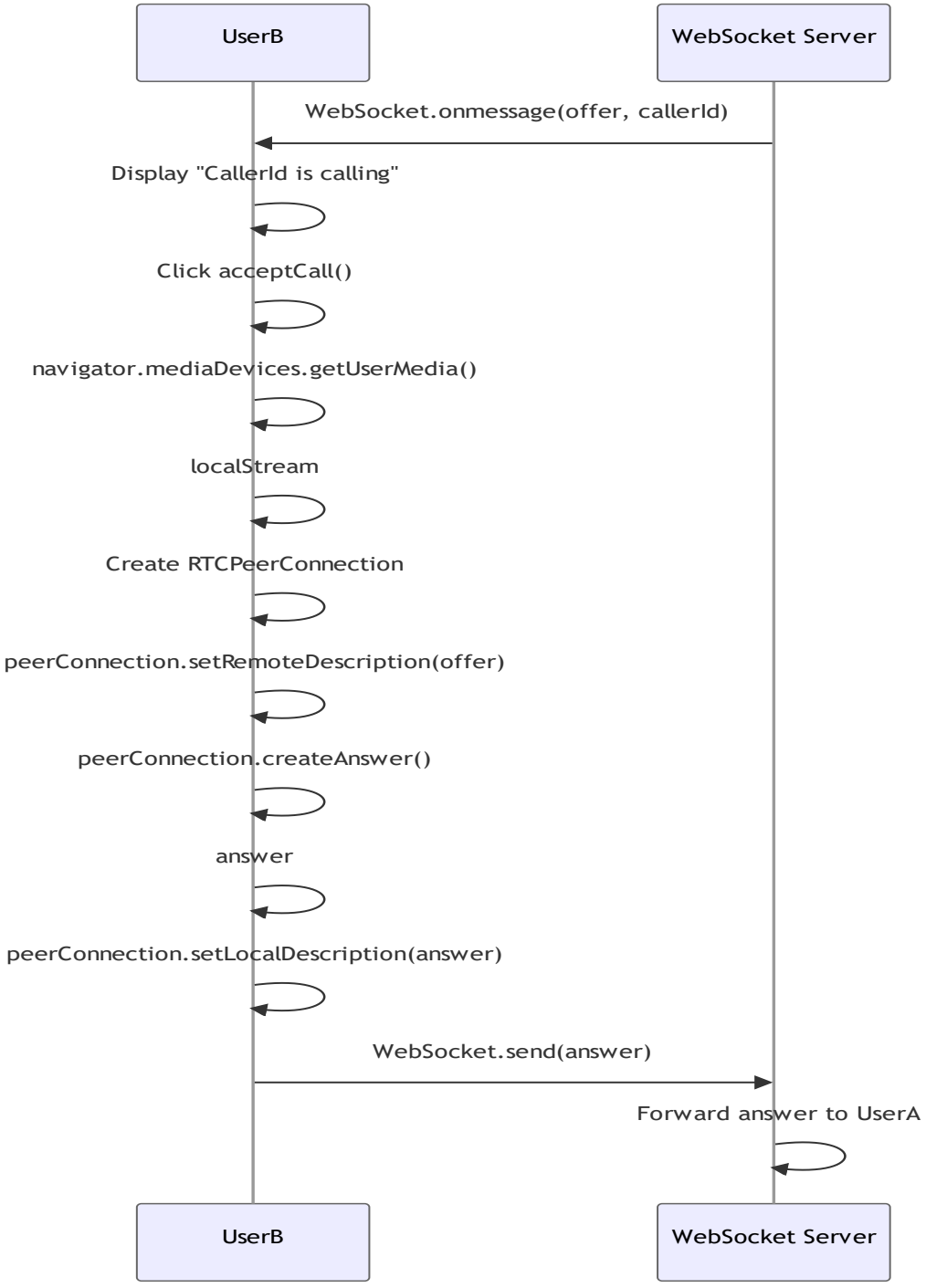
4. **Create Project Files:**
    - server.js: Signaling server script.
    - index.html: HTML file for the client-side UI.
    - style.css: CSS file for styling.
    - script.js: JavaScript file for client-side logic.

How It Works

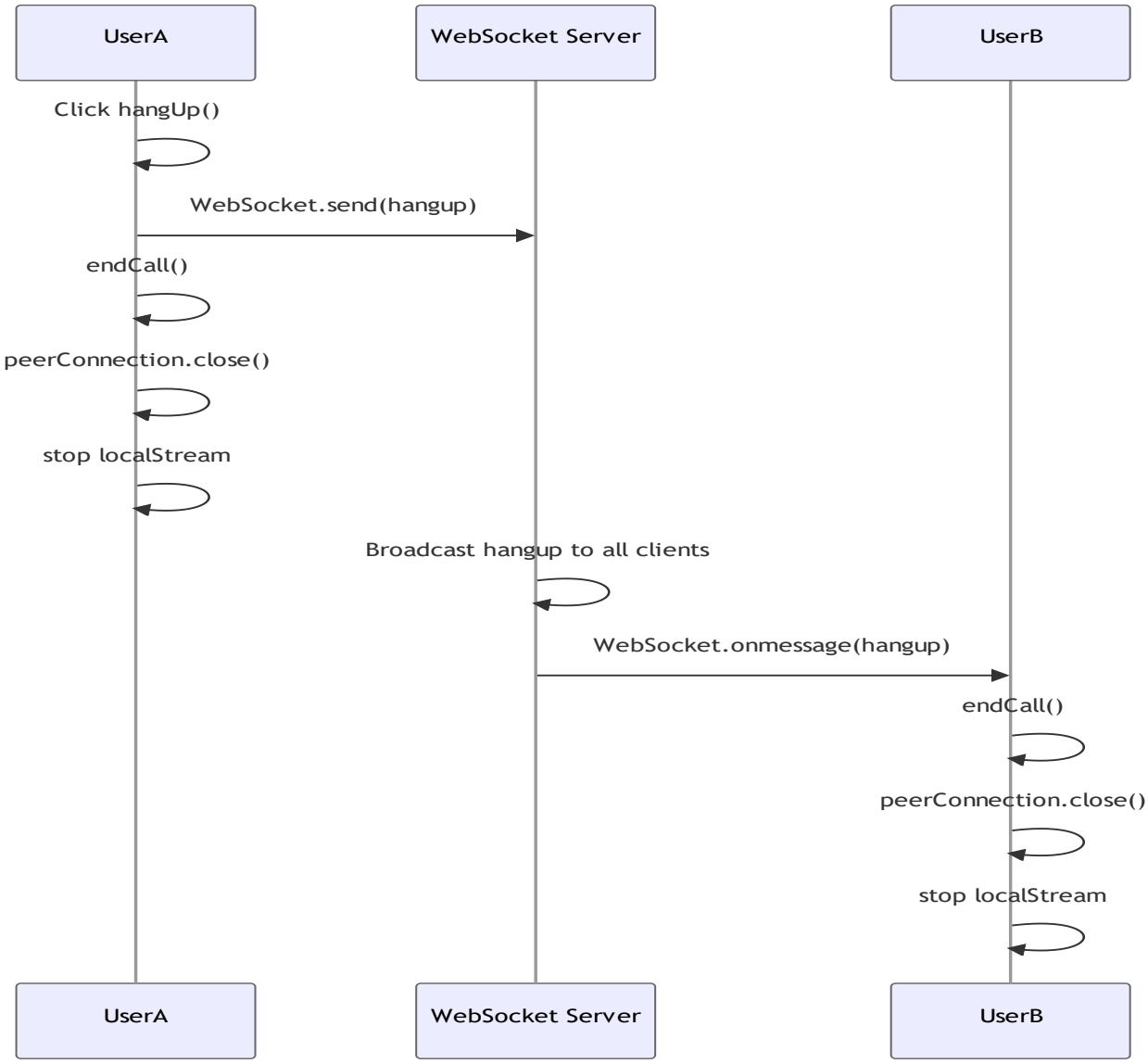
1. **Starting a Call:**
    - When the user clicks the start button, the application accesses the local media devices (camera and microphone) to capture video and audio streams.
    - A new RTCPeerConnection is created to handle the WebRTC connection.
    - An offer is created and sent to the signaling server via WebSocket.
    - The local video stream is displayed on the user's screen.



2. **Accepting a Call:**
    - When the user receives a call, a notification is displayed with the caller's ID.
    - The user can accept the call, which initiates the process to capture their local media stream and add it to the RTCPeerConnection.
    - An answer is created and sent to the signaling server via WebSocket.
    - The remote video stream is displayed on the user's screen.



3. **Handling ICE Candidates:**
    - As part of the WebRTC connection setup, ICE candidates are exchanged between peers via the signaling server.
    - These candidates are added to the RTCPeerConnection to establish a direct connection.
4. **Ending a Call:**
    - When the user clicks the hang-up button, a hangup message is sent to the signaling server.
    - The WebRTC connection is closed, and the local and remote video streams are stopped.
    - The video elements are cleared.



Running the Project

1. **Start the Signaling Server:**

node server/server.js

   - Ensure the server is running at ws://localhost:8080.

2. **Start a Local HTTP Server to Serve the Client Files:**

npx http-server client -p 8081

   - Serve the client files at http://localhost:8081.

3. **Access the Application:**

   - Open a browser and navigate to http://localhost:8081.
   - Share the link with another user to start a video call.