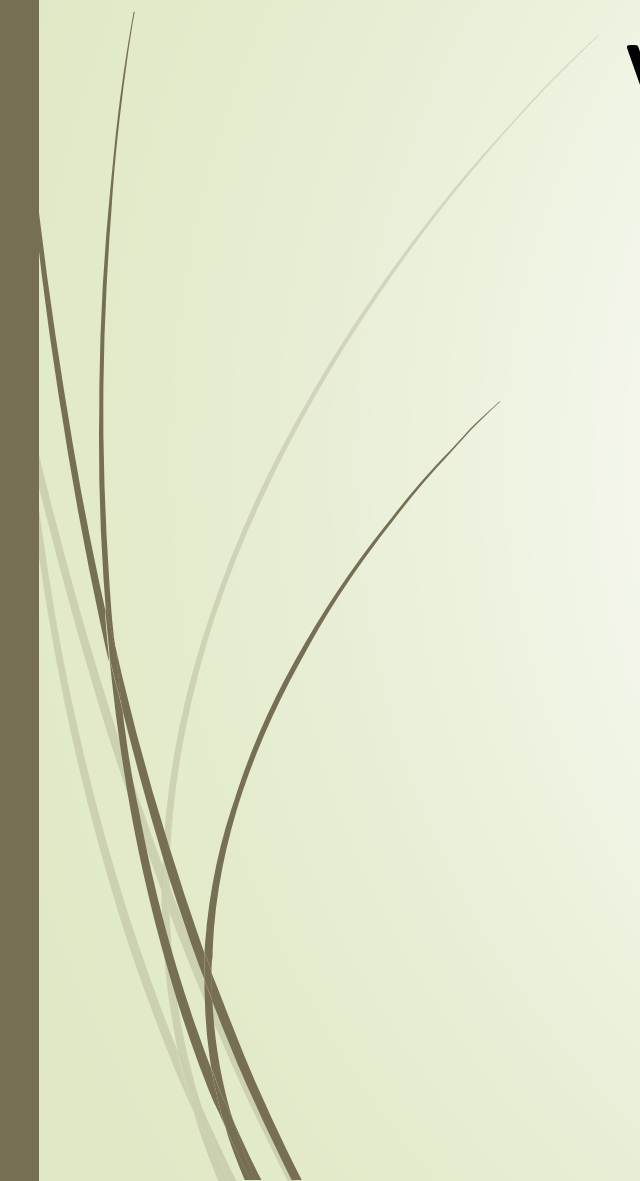




Video Conferencing Web App

Author Mr. Shubham Oulkar

2023-08-14



- 
- 1. Objectives**
 - 2. Functional Features**
 - 3. Non Functional Features**
 - 4. Site map**
 - 5. System design Architecture**
 - 6. What is WebRTC?**
 - 7. How WebRTC works?**
 - 7.1 Signaling**
 - 7.2 Connecting**
 - 7.3 Secure connection**
 - 7.4 Real Time Communication**
 - 8. Deployment on Render**
 - 9. References, Technology, and Tools**



Objectives:

Design an intuitive and user-centric video conferencing web application that provides seamless, high-quality virtual communication experiences, fosters collaboration, and ensures accessibility across devices and platforms. The objective is to create a user-friendly interface, optimize audio and video performance, integrate essential features such as screen sharing , prioritize data security and privacy, and deliver a reliable tool that enhances remote communication for both personal and professional users.



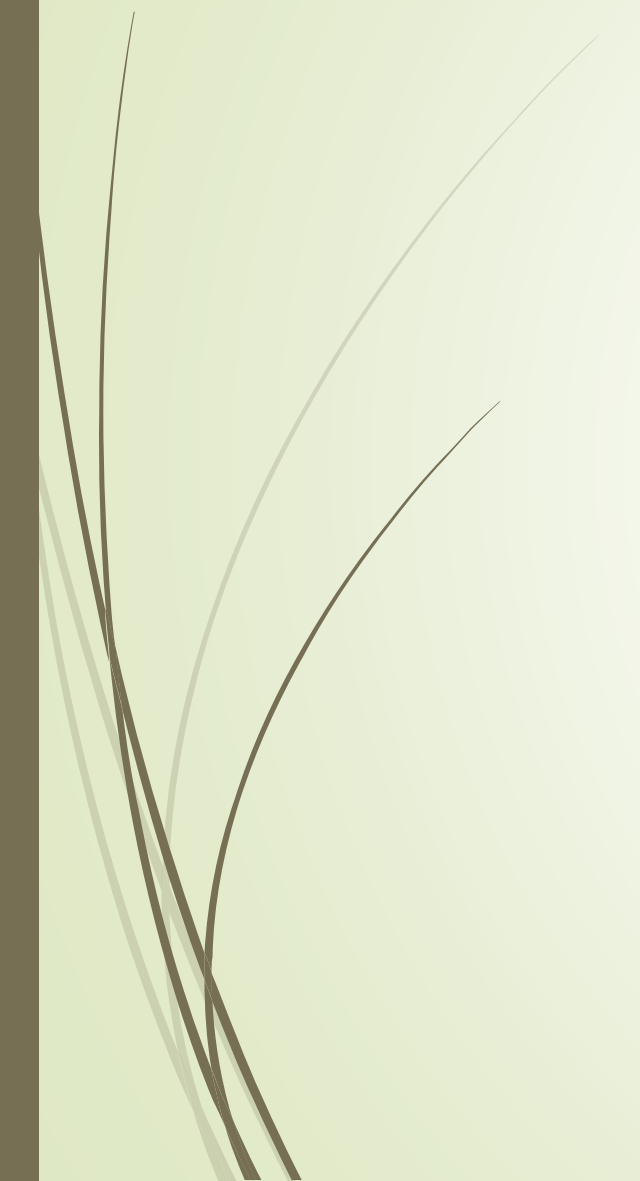
Functional Requirements :

- The system must support 1-to-1 calls
- Calls can be audio or video or screen sharing
- User signup
- Responsive Web Page design

Note: Screen share will be an extension of video call itself, only in case of video call the source of the video is the camera, whereas in screen share source of the video will be the screen.

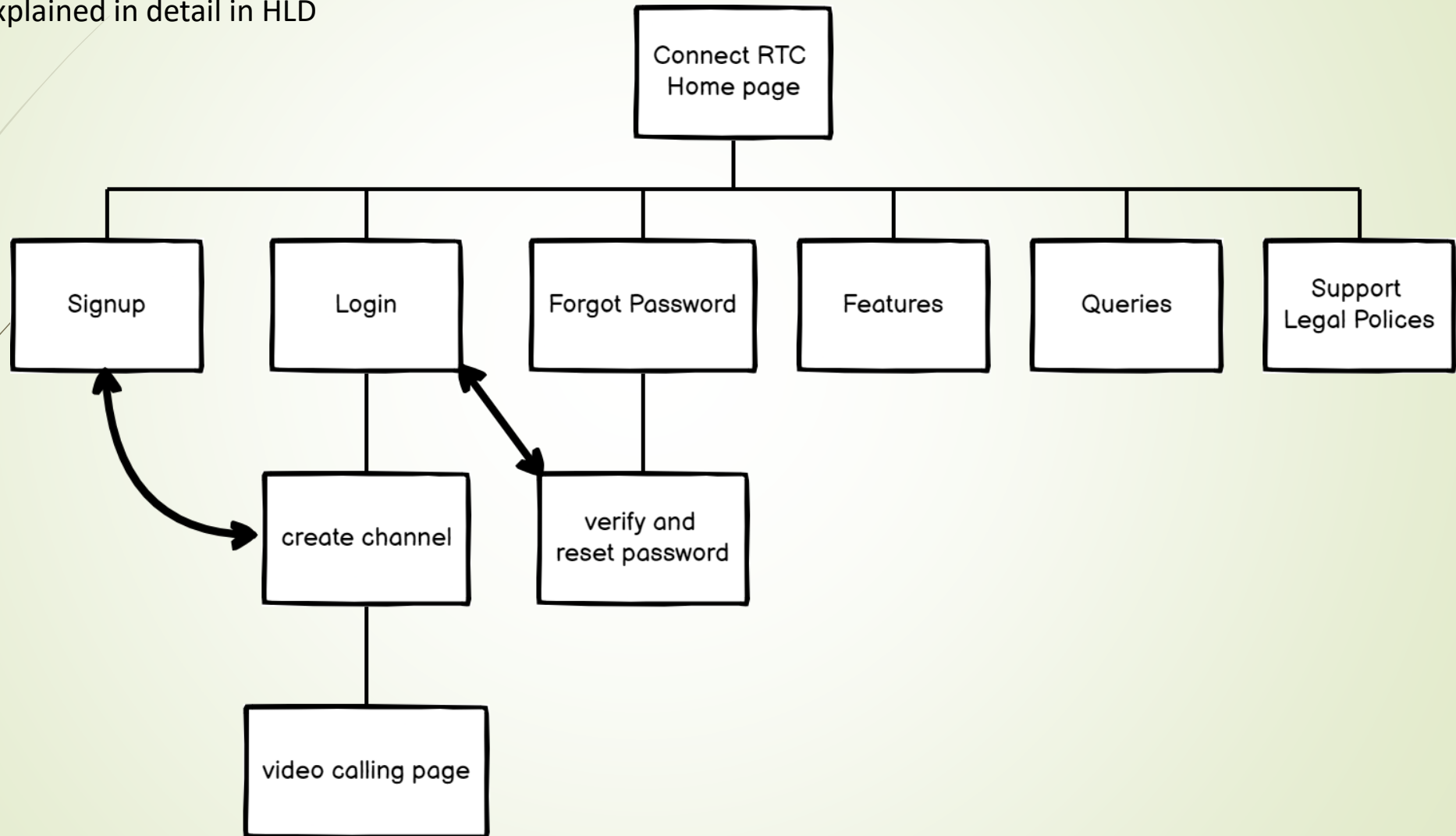


Non Functional Requirements:

- Should be super fast - low latency is not enough
 - High availability
 - Data loss is OK for video/audio/screen sharing
 - works the same in every environment
- 

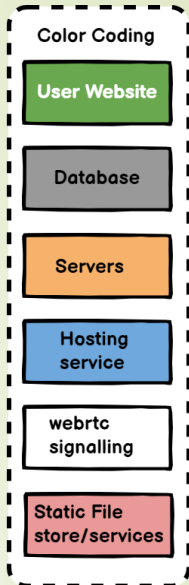
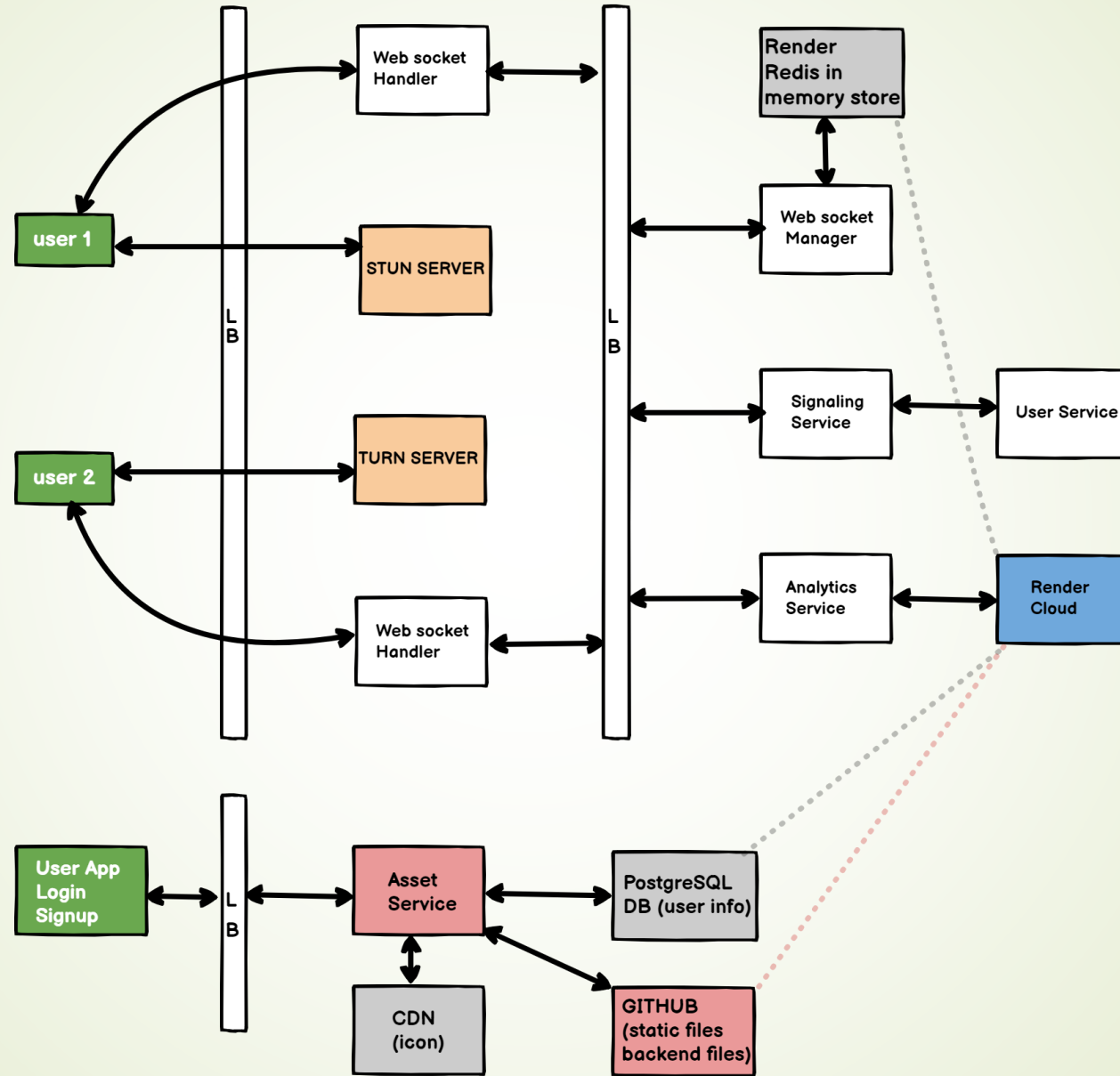
Site map: following's are list of pages for a website

Explained in detail in HLD



Architecture:

Explained in detail in LLD





What is WebRTC ?

WebRTC, short for Web Real-Time Communication, is both an API and a Protocol. The WebRTC protocol is a set of rules for two WebRTC agents to negotiate bi-directional secure real-time communication.

What makes WebRTC special is that once a connection is established; data can be transmitted directly between browsers in real time without touching the server. By bypassing the server we reduce latency since the data doesn't have to go to the server first, this makes WebRTC great for exchanging audio and video.

Four sequential steps in WebRTC connection :

1. Signaling (Django channels, websocket, redis, SDP)
2. Connecting (STUN/TURN, ICE)
3. Securing (SRTP)
4. Communicating (RTP)

How WebRTC works ?

1) Signaling uses an existing, plain-text protocol called SDP (Session Description Protocol).

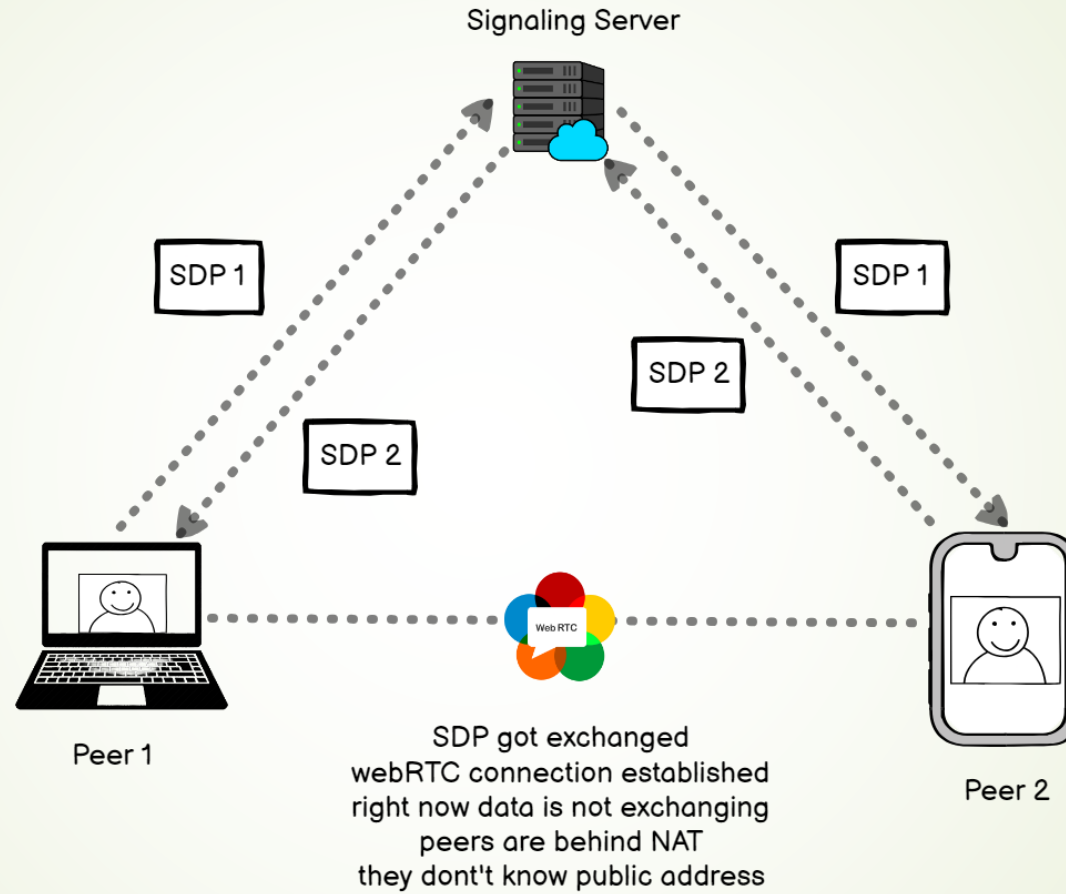
Each SDP message is made up of key/value pairs and contains a list of “media sections”.

The SDP that the two WebRTC agents exchange contains details like:

- The IPs and Ports that the agent is reachable on (candidates).
- The number of audio and video tracks the agent wishes to send.
- The audio and video codecs each agent supports.
- The values used while connecting (uFrag/uPwd).
- The values used while securing (certificate fingerprint).

Our Application don't use WebRTC for signaling messages, generally third party service is used for relaying messages. For this project we are using Django Channels and websockets for signaling SDP to peers.

Exchanging SDP to peers




2) Connecting:

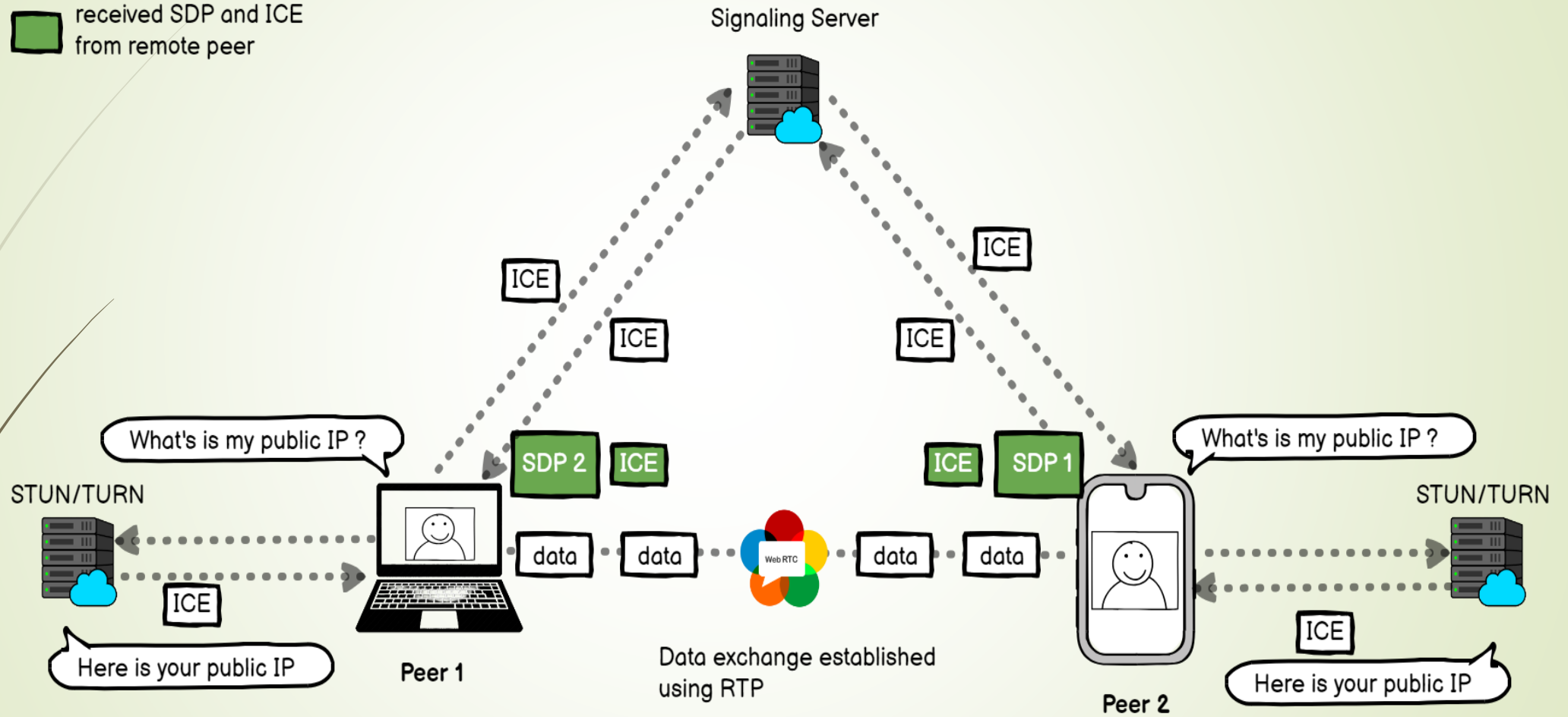
Once two WebRTC agents have exchanged SDPs, they have enough information to attempt to connect to each other. To make this connection happen, WebRTC uses another established technology called ICE (Interactive Connectivity Establishment).


ICE is a protocol that pre-dates WebRTC and allows the establishment of a direct connection between two agents without a central server. These two agents could be on the same network or on the other side of the world.

ICE enables direct connection, but the real magic of the connecting process involves a concept called 'NAT Traversal' and the use of STUN/TURN Servers. ICE is a protocol that tries to find the best way to communicate between two ICE Agents. Each ICE Agent publishes the ways it is reachable, these are known as candidates. A candidate is essentially a transport address of the agent that it believes the other peer can reach. ICE then determines the best pairing of candidates.

Exchange ICE to peers

 received SDP and ICE from remote peer





3) Securing the transport layer:

To make our communication secure! This is done through two more protocols that also pre-date WebRTC; DTLS (Datagram Transport Layer Security) and SRTP (Secure Real-Time Transport Protocol). The first protocol, DTLS, is simply TLS over UDP (TLS is the cryptographic protocol used to secure communication over HTTPS). The second protocol, SRTP, is used to ensure encryption of RTP (Real-time Transport Protocol) data packets.

First, WebRTC connects by doing a DTLS handshake over the connection established by ICE. Unlike HTTPS, WebRTC doesn't use a central authority for certificates. It simply asserts that the certificate exchanged via DTLS matches the fingerprint shared via signaling. This DTLS connection is then used for Data Channel messages.

Next, WebRTC uses the RTP protocol, secured using SRTP, for audio/video transmission. We initialize our SRTP session by extracting the keys from the negotiated DTLS session.



4) Communicating (RTP):

Now that we have two WebRTC agents connected and secure, bi-directional communication established, WebRTC will use two pre-existing protocols: RTP (Real-time Transport Protocol), and SCTP (Stream Control Transmission Protocol). We use RTP to exchange media encrypted with SRTP, and we use SCTP to send and receive DataChannel messages encrypted with DTLS.

RTP is quite a minimal protocol, but it provides the necessary tools to implement real-time streaming. The most important thing about RTP is that it gives flexibility to the developer, allowing them to handle latency, package loss, and congestion as they please.



Deployment on Render:

We use render as hosting platform because, Render allows you to host static sites, web services, PostgreSQL databases, and Redis instances. Its extremely simple UI/UX and great git integration allow you to get an app up and running in minutes. It has native support for Python. Render also provide basic analytics chart like connection time, memory usage, Bandwidth usage, CPU time and many more. Another most important reason to choose Render is it **provide internal Redis server** in production for free. That makes our signaling service super fast.

Reference:

https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

<https://webrtcforthe curious.com/docs/01-what-why-and-how>

<https://www.w3.org/TR/webrtc/>

<https://medium.com/agora-io/how-does-webrtc-work-996748603141>

Technology Used:



UI design tools used:

