**LOW LEVEL DESIGN DOCUMENT**

**Video Conferencing App**

Contents

Abstract

1. Introduction
   1. Why this low-level design documentation?
   2. Scope
   3. Constraints
2. Technical specifications
   1. Libraries Used
   2. Input Schema
   3. Logging
   4. Database
3. Technology Stack
4. Proposed Solution
5. User I/O workflow

Abstract

Video conferencing is an online technology that allows users in different locations to hold face-to-face meetings without having to move to a single location together. We have seen during pandemic that only source we had to conduct business meets, work from home, impart education and so much was only possible because of tech and apps like zoom, Google meet, teams and much more. So as a techie What can be better than making your own video conferencing application and interact with your friends.

1. **Introduction**
   1. **Why this Low-Level Design Document?**

The purpose of this Low-Level Design (LLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The main objective of this project is to make a make a application that allows to share audio and video live stream data and also allows screen sharing at same time

* 1. **Scope**

The LLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow and technology architecture. The LLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system. This software system will be a Web application this system will be designed to make video chat application.

1.3 **Constraints**

There will be limitation i.e as of now only a single room is implemented and only in that room many peers can join.

1. **Technical specifications**
   1. **Library Used**

**Django Channels** – Channels is a project that takes Django and extends its abilities beyond HTTP - to handle WebSockets, chat protocols, IoT protocols, and more. It’s built on a Python specification called [ASGI](http://asgi.readthedocs.io/) Channels builds upon the native ASGI support available in Django since v3.0, and provides an implementation itself for Django v2.2. Django still handles traditional HTTP, whilst Channels give you the choice to handle other connections in either a synchronous or asynchronous style.

**WebRTc and Websockets** - With WebRTC, you can add real-time communication capabilities to your application that works on top of an open standard. It supports video, voice, and generic data to be sent between peers, allowing developers to build powerful voice- and video-communication solutions. The technology is available on all modern browsers as well as on native clients for all major platforms. The technologies behind WebRTC are implemented as an open web standard and available as regular JavaScript APIs in all major browsers. For native clients, like Android and iOS applications, a library is available that provides the same functionality. The WebRTC project is [open-source](https://webrtc.googlesource.com/src/) and supported by Apple, Google, Microsoft and Mozilla, amongst others. This page is maintained by the Google WebRTC team.

* 1. **Logging**

We should be able to log every activity done by the incidents.

* The System identifies at what step logging required
* The System should be able to log each and every system flow.
* Developers can choose logging methods. You can choose database logging/ File logging as well.
* System should not be hung even after using so many loggings. Logging just because we can easily debug issues so logging is mandatory to do.

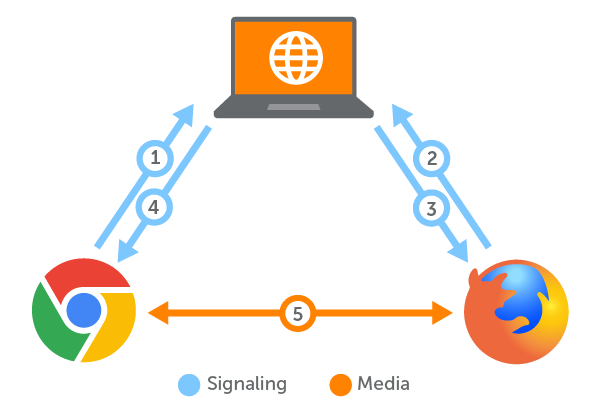
1. **Technology Stack**

|  |  |
| --- | --- |
| **Front End** | HTML/CSS/JS |
| **Backend** | Python /Django, JS |
| **Deployment** | AWS, Heroku |
| **Version control** | GitHub |

1. **Proposed Solution**

In order to make a peer-to-peer connection those peers will be connected using SDP i.e Session Description Protocol – Information about a peer that the other peer needs in order to make p2p connection. Each peer will send their sdp to other peer using signalling server and both peer will connect to signalling server using web sockets and when signalling server knows the location of each peer it can relay messages between the two peers and that way the two peers will exchange their sdp’s with each other so once those peers have each other’s sdp’s then they can establish a peer to peer connection and after that they don’t require a signalling server because then they can connect to each other using peer to peer connection and here we are implementing signalling server using django channels So here the laptop is the django channel signalling server once the browsers connect to the django channel signalling server using web sockets and after that they exchange their sdp’s and now once they have their sdp’s they establish a peer to peer connection with one another and then they will communicate with each other directly without the django channel signalling server

1. **User Input Output WorkFlow**



So here the laptop is the django channel signalling server once the browsers connect to the django channel signalling server using web sockets and after that they exchange their sdp’s and now once they have their sdp’s they establish a peer-to-peer connection with one another and then they will communicate with each other directly without the django channel signalling server

