Final documentation -

Project Name: ChourangiHealth Video Consultation Platform

Technology Stack: OpenVidu v2.32, FastAPI (Python), React.js, AWS EC2, DuckDNS,

Let's Encrypt SSL Purpose: A secure telemedicine video calling platform where a doctor and patient can join the same video session using a shared Calling ID and consult in real time.

PART 1: PROJECT OVERVIEW AND ARCHITECTURE

We built a two-party video consultation system. The doctor logs in from one machine and the patient logs in from another machine. Both enter the same Calling ID such as 1. The backend creates an OpenVidu session using that Calling ID as the custom session ID. Both parties receive a token for that same session and connect to each other via WebRTC video. The local video appears in a small picture-in-picture box and the remote participant appears in a large video box. Controls for toggling video on or off and muting or unmuting audio are available in the header.

PART 2: BACKEND (FastAPI + OpenVidu)

The backend was built using FastAPI with Python. The project structure contains an app folder with main.py, config.py, dependencies.py, a routers folder containing auth.py, health.py, and video.py, a services folder containing openvidu_service.py and auth_service.py, and a schemas folder containing auth_schema.py and video_schema.py. A .env file, requirements.txt, Dockerfile, and docker-compose.yml are at the root.

The authentication system uses a hardcoded in-memory user database with two users. The doctor user has username doctor and password 123456 with role admin. The patient user has username user and password 123456 with role user. When login is called, the backend authenticates the user and returns a JWT access token signed with HS256 algorithm. The token contains the username and role as claims.

The video session system works as follows. When the frontend sends a POST request to the video session endpoint with a session ID equal to the Calling ID the user entered, the backend first calls the OpenVidu REST API to create a session using that value as the customSessionId. If OpenVidu returns 409 meaning the session already exists, the backend handles it gracefully and continues. Then the backend calls the OpenVidu connection API to generate a fresh token for that session and returns the token and session ID to the frontend. This means both doctor and patient who enter the same Calling ID receive tokens for the same OpenVidu session and can see each other.

The backend uses JWT authentication middleware. Every video session request requires a valid Bearer token in the Authorization header. The dependencies.py file verifies the token and returns the payload or raises 401 errors for expired or invalid tokens.

CORS was configured in main.py to allow requests from the production domain https://chourangi.duckdns.org so the frontend hosted on that domain can communicate with the backend API.

The backend is deployed on the EC2 server and runs using uvicorn with the nohup command so it stays running in the background without requiring any manual intervention. The command used is nohup uvicorn app.main:app --host 0.0.0.0 --port 8000 and output is logged to /tmp/backend.log.

PART 3: FRONTEND (React.js)

The frontend was built using React.js with Create React App. The project structure contains a src folder with App.js, index.js, App.css, an api folder containing api.js, and a pages folder containing LoginPage.js and VideoPage.js.

The api.js file contains two functions. The loginUser function sends a POST request to the backend auth login endpoint with username and password. On success it decodes the JWT token using jwt-decode to extract the role and stores the token and role in localStorage. The createVideoSession function sends a POST request to the backend video session endpoint with the session ID and the JWT token in the Authorization header.

The LoginPage was redesigned with a soft purple and white theme. It features a two panel layout. The left panel shows the ChourangiHealth brand, a hero title saying Healthcare at your fingertips, cartoon SVG illustrations of a doctor and patient facing each other with a signal icon between them, and three feature badges showing encrypted video, low latency, and HIPAA friendly design. The right panel shows a login card with a role toggle between Doctor and Patient which auto fills the username field, input fields for username and password, error display, and a purple gradient Sign In button. The design uses floating blob background effects, drop shadows, and rounded corners.

The VideoPage was redesigned with the same soft purple theme. After login the user sees a join screen with a cartoon avatar of their role, an input field for the Calling ID, and a Start Consultation button. After joining, the page shows a frosted glass header with the brand name, a live call duration timer that starts when both participants are connected, role badge, connection status indicator, video toggle button, audio toggle button, and leave call button. The main video area shows the remote participant in a large 520 pixel height video box with a waiting animation showing bouncing dots and the remote participant avatar while waiting for the other person to join. The local self video appears as a picture in picture overlay in the bottom right corner with a purple border glow.

The streamCreated event in OpenVidu uses a setTimeout of 300 milliseconds before calling subscriber.addVideoElement to ensure the React DOM has rendered the video element before attaching the stream. The publisher is also initialized after a 300 millisecond delay after connecting to the session for the same reason.

PART 4: OPENVIDU SERVER

OpenVidu v2.32 was used as the WebRTC media server. It was already installed on the EC2 instance at the path /opt/openvidu. OpenVidu runs as a set of Docker containers managed by docker-compose. The containers include openvidu-server for the core signaling server, openvidu-proxy which is the Nginx reverse proxy, openvidu-kms which is Kurento Media Server for WebRTC media processing, openvidu-coturn for TURN and STUN server for NAT traversal, and openvidu-app which is the default OpenVidu Call application.

The OpenVidu default Call application was disabled by changing WITH_APP=true to WITH_APP=false in the docker-compose.yml file so that our custom React frontend could serve at the root path.

The OpenVidu REST API is used by the backend to create sessions and generate connection tokens. Session creation uses POST to /openvidu/api/sessions with the customSessionId in the payload. Connection token generation uses POST to /openvidu/api/sessions/SESSION_ID/connection. Both calls use HTTP Basic Auth with username OPENVIDUAPP and the configured secret.

PART 5: SSL CERTIFICATE AND DOMAIN SETUP

The reason SSL was required is that browsers only allow camera and microphone access via WebRTC on HTTPS connections. HTTP is only allowed on localhost. Without SSL the browser blocks the camera and microphone, and users get a security warning every time they visit the OpenVidu server URL and must manually accept the risk before the application works. This is not acceptable for production use.

We chose the Let's Encrypt approach which is free and auto-renewing. A free subdomain was created at DuckDNS. The domain chourangi.duckdns.org was registered and pointed to the EC2 public IP 18.130.149.212. The OpenVidu .env file at /opt/openvidu/.env was updated with three changes. DOMAIN_OR_PUBLIC_IP was changed from the raw IP to chourangi.duckdns.org. CERTIFICATE_TYPE was changed from selfsigned to letsencrypt. LETSENCRYPT_EMAIL was set to the email address for certificate notifications. After restarting OpenVidu, Let's Encrypt automatically issued a trusted SSL certificate for chourangi.duckdns.org. The browser now shows a green padlock and no security warnings appear for any user visiting the domain for the first time.

PART 6: AWS EC2 SETUP AND SECURITY GROUP

The EC2 instance was already running with OpenVidu installed. The Security Group was verified to have all required ports open for inbound traffic. Port 22 TCP is open for SSH access. Port 80 TCP is open for HTTP and Let's Encrypt certificate verification. Port 443 TCP is open for HTTPS web traffic. Port 4443 TCP is open for OpenVidu WebSocket signaling. Port 3478 TCP and UDP is open for TURN and STUN server. Ports 40000 to 57000 UDP are open for WebRTC media streams. Ports 10000 to 20000 UDP are also open for additional media traffic. All ports are open to 0.0.0.0/0 for IPv4 and ::/0 for IPv6.

PART 7: DEPLOYMENT PROCESS

The code was pushed to GitHub. The frontend repository is at https://github.com/AshuuuPatil/openvidu_videocall_microservice_frontend and the backend repository is at https://github.com/AshuuuPatil/openvidu_videocall_microservice_backend. Both were pushed on the master branch.

On the EC2 server, Node.js version 18 was installed using the NodeSource setup script. Both repositories were cloned using git clone with the master branch flag to avoid getting the empty main branch. The frontend was built using npm install followed by npm run build which generates a production optimized build folder. The build files were copied to /opt/openvidu/custom-layout which is the folder mounted inside the OpenVidu Nginx Docker container at the same path.

The backend dependencies were installed using pip install with the break-system-packages and ignore-installed flags to handle Ubuntu system Python conflicts. The backend .env file was updated to point OPENVIDU_URL to https://chourangi.duckdns.org.

A custom Nginx configuration file was created at /opt/openvidu/custom-nginx-locations/frontend.conf. This file tells the OpenVidu Nginx container to serve the React build files from /opt/openvidu/custom-layout for all routes, return 404 for missing static files so React routing works correctly, and proxy all requests to /api/ to the FastAPI backend running on the Docker bridge IP 172.17.0.1 at port 8000.

After all configuration changes, OpenVidu was restarted using ./openvidu restart from the /opt/openvidu directory.

PART 8: FINAL WORKING STATE

The application is now fully deployed and accessible at https://chourangi.duckdns.org. Any user visiting this URL from any device on any network sees the ChourangiHealth login page immediately with no security warnings. The doctor logs in with username doctor and password 123456. The patient logs in with username user and password 123456. Both enter the same Calling ID on the join screen. The backend creates or reuses an OpenVidu session for that Calling ID and returns tokens to both parties. Both connect to the same session and can see and hear each other in real time. Video and audio can be toggled independently. A live call duration timer shows how long the call has been active. Either party can leave the call and return to the login page.

PART 9: KNOWN LIMITATIONS AND RECOMMENDED NEXT STEPS

The current implementation uses hardcoded credentials stored directly in the Python code which is not suitable for a real production system. The recommended next steps are to replace the hardcoded user database with a real database such as PostgreSQL with hashed passwords. Role based access control should be added so doctors have moderator privileges in the OpenVidu session. Session cleanup should be implemented when both participants leave so OpenVidu resources are freed. The frontend should be made responsive for mobile devices. Recording support can be added using the OpenVidu recording API. A proper CI/CD pipeline should be set up so that pushing to GitHub automatically deploys to EC2. An Elastic IP should be assigned to the EC2 instance so the public IP does not change on restart which would break the DuckDNS mapping and SSL certificate.