

Frontend Documentation

React Frontend Implementation Guide

ChourangiHealth Video Consultation Platform

Version 1.0 | February 2026

1. Overview

The frontend is a React.js single page application that provides the video consultation interface for doctors and patients. It uses the OpenVidu Browser SDK to handle WebRTC connections and the custom FastAPI backend for authentication and session management.

1.1 Key Libraries

- React 18: UI framework
- openvidu-browser: WebRTC SDK that connects to OpenVidu server
- jwt-decode: Decodes JWT token to read role claim without backend call
- No UI library used: All styling is custom inline CSS JavaScript objects

2. Project Structure

```
openvidu-frontend-poc/
  public/
    index.html      # HTML entry point
  src/
    App.js         # Root component, handles auth routing
    index.js        # React DOM render entry point
    index.css       # Global styles (overflow:hidden for fixed layout)
    api/
      api.js        # All HTTP calls to backend
    pages/
      LoginPage.js   # Login screen with role selection
      VideoPage.js    # Main video consultation page
  package.json
  .gitignore
```

3. api.js - API Layer

All backend API calls are centralized in src/api/api.js. The API_BASE_URL constant controls which server the app talks to.

3.1 URL Configuration

```
// For EC2 production:  
const API_BASE_URL = "https://chourangi.duckdns.org";  
  
// For local development (backend on same machine):  
// const API_BASE_URL = "http://localhost:8000";  
  
// For local development (backend accessible from other devices):  
// const API_BASE_URL = "http://192.168.1.62:8000";
```

NOTE: Always switch back to the production URL before building and deploying to EC2.

3.2 API Functions

loginUser(username, password)

Calls POST /api/auth/login. Stores JWT token in localStorage as jwt. Decodes the token using jwt-decode to extract the role claim and stores it in localStorage as role.

createVideoSession(token, sessionId)

Calls POST /api/video/session with the calling ID. Returns sessionId and token needed to connect to the OpenVidu session.

startRecording(token, sessionId)

Calls POST /api/video/recording/start with session_id as query parameter. Returns recordingId and status.

stopRecording(token, recordingId)

Calls POST /api/video/recording/stop with recording_id as query parameter. Returns recordingId, status, and url of the MP4 file.

downloadRecording(token, recordingId)

Calls GET /api/video/recording/{recordingId}/download. Receives binary MP4 data, creates a blob URL using window.URL.createObjectURL, creates a temporary anchor element, triggers a click to start the download, then cleans up the blob URL and anchor element. This gives a seamless download experience without any browser popup or credential request.

4. App.js - Authentication Routing

App.js manages the authentication state. If a JWT token exists in localStorage the user sees VideoPage. Otherwise they see LoginPage. The onLogout function clears localStorage and shows LoginPage again.

5. LoginPage.js

The login page has a styled form with username and password fields. On submit it calls loginUser from api.js. On success it calls the onLogin prop function which updates App.js state to show VideoPage. The page displays ChourangiHealth branding.

6. VideoPage.js - Main Video Page

6.1 State Variables

- callingId: The room ID entered by user (e.g. 1)
- session: The OpenVidu Session object
- publisher: The local Publisher stream object
- connected: Boolean whether WebSocket to OpenVidu is connected
- remoteJoined: Boolean whether another participant has joined
- hasJoined: Boolean whether current user has joined a session
- videoEnabled: Boolean local video on/off state
- audioEnabled: Boolean local audio mute/unmute state
- callDuration: Integer seconds since both participants connected
- messages: Array of chat message objects
- newMessage: String current text in chat input
- isChatOpen: Boolean whether chat panel is visible
- unreadCount: Integer count of unread messages when chat is closed
- isRecording: Boolean whether recording is currently in progress
- recordingId: String ID of current or last recording
- recordingUrl: String MP4 URL returned after recording stops

6.2 Refs Used

- localVideoRef: Attached to local video element for publisher stream
- remoteVideoRef: Attached to remote video element for subscriber stream
- timerRef: Holds the setInterval reference for call duration timer
- isChatOpenRef: Mirror of isChatOpen state for use inside signal event listener closure
- sessionRef: Mirror of session state for use in cleanup effects and signal sending
- chatMessagesRef: Attached to chat messages container for scroll control

6.3 joinSession Function

This is the main function that connects to OpenVidu. Steps:

- Step 1: Calls `createVideoSession` backend API to get OpenVidu token
- Step 2: Creates OpenVidu and Session objects from `openvidu-browser` SDK
- Step 3: Registers event listeners on the session for `streamCreated`, `streamDestroyed`, `reconnecting`, `reconnected`, `sessionDisconnected`, and `signal:chat`
- Step 4: Calls `mySession.connect(token)` to establish WebSocket connection to OpenVidu
- Step 5: After 300ms delay, initializes publisher with camera and microphone
- Step 6: Publishes local stream to the session

NOTE: The 300ms delay before publisher initialization is needed to ensure the DOM video element is rendered before attaching the stream.

6.4 Chat Implementation

Chat uses the OpenVidu Signal API which piggybacks on the existing WebSocket connection. No additional server or socket library is needed.

- Sending: `sessionRef.current.signal({ data: JSON.stringify(msg), to: [], type: 'chat' })`
- Receiving: `mySession.on('signal:chat', handler)` registered during `joinSession`
- The sender adds their own message to state immediately without waiting for the signal echo
- The signal listener checks `event.from.connectionId` against `mySession.connection.connectionId` to skip own messages
- `isChatOpenRef` is used instead of `isChatOpen` state in the listener to avoid stale closure
- Scroll: `chatMessagesRef.current.scrollTop = chatMessagesRef.current.scrollHeight` in `useEffect` when messages change

NOTE: Do not use `messagesEndRef.scrollIntoView` as it scrolls the entire page. Use `scrollTop` on the container div instead.

6.5 Recording UI Flow

Three buttons share the same position in the header and only one is visible at a time based on state:

- Record button (red): visible when `isRecording` is false and `recordingUrl` is null. Calls `handleStartRecording`.
- Stop Rec button (orange, pulsing): visible when `isRecording` is true. Calls `handleStopRecording`.
- Download button (green): visible when `recordingUrl` is not null. Calls `handleDownloadRecording`.

After download completes, `setRecordingUrl(null)` and `setRecordingId(null)` are called to reset back to the Record button state, allowing repeated recording cycles.

6.6 Video Layout

The layout uses CSS position fixed with height 100vh and overflow hidden to prevent any page scrolling. This is needed because mobile browsers resize the viewport when keyboard opens which can cause layout shifts.

- Header: Fixed at top with all control buttons
- Main area: Flex row containing video section and optional chat panel
- Video section: Takes full width when chat is closed, reduces when chat is open
- Remote video: Full size in a rounded card with dark background
- Local video PiP: Positioned absolutely in bottom-left corner, 180x135px with purple border
- Waiting overlay: Shows avatar animation when remote participant has not yet joined
- Chat panel: Fixed 340px wide, slides alongside video

6.7 Avatars

Custom SVG avatars are defined as React components at the top of VideoPage.js. DoctorAvatar shows a person in a white coat with a stethoscope. PatientAvatar shows a person with a heart symbol. These render on the join screen and in the waiting overlay.

7. Build and Deployment

7.1 Local Development

```
cd openvidu-frontend-poc  
npm install  
npm start
```

App runs at `http://localhost:3000`. To test from another device on the same WiFi, run:

```
npm start -- --host 0.0.0.0
```

Then access from other device at `http://192.168.1.62:3000`

NOTE: When testing locally with another device, change `API_BASE_URL` in `api.js` to `http://192.168.1.62:8000` so both devices reach the backend. Run backend with: `unicorn app.main:app --host 0.0.0.0 --port 8000`

7.2 Production Build and Deploy

```
# Build  
npm run build  
  
# On EC2: Pull latest and deploy  
cd /home/ubuntu/test-1/frontend  
git fetch origin master  
git reset --hard origin/master  
npm run build  
cp -r build/* /opt/openvidu/custom-layout/
```

7.3 Git Repository

- Repository URL:
`https://github.com/AshuuuPatil/openvidu_videocall_microservice_frontend`
- Branch: master
- `.gitignore` includes: `node_modules/`, `build/`, `.env`

End of Frontend Documentation