Phát live với âm thanh

Hiện tôi đang làm dự án phát live sử dụng webrtc, tôi sử dụng musetalk để sinh ảnh lipsync xong rồi sẽ gửi generated frame đến audio queue và sử dụng video track để lấy nó gửi cho frontend. queue sẽ được put vào (frame_idx, frame). Và hiện tôi muốn phát cả âm thanh nữa. Tuy nhiên tôi có ý tưởng là tôi sẽ gửi file âm thanh (audio_path), và sẽ phát nó cùng với quá trình stream ảnh.
Tôi đang định thực hiện phát audio với 2 điều kiện sau:
1. Audio chỉ bắt đầu khi frame đầu tiên được chiếu (có frame_idx)
2. Tại thời điểm (ví dụ audio giây thứ 2) cần phải check xem đã có frame ở thời gian đó hay chưa. Tức là abs(audio_time - frame_idx/video_fps) < threshold.

Dự án hiện tại đang có trong repo sau:

Dựa trên mã nguồn hiện tại trong repo (đặc biệt là **`src/services/webrtc.py`**), luồng WebRTC sử dụng hai hàng đợi (queue) để đẩy dữ liệu video và audio:

- **`VideoTrack`** lấy một tuple **`(frame_idx, frame_bgr)`** từ **`video_queue`**, chuyển frame sang **`VideoFrame`** và gán **`pts = frame_idx`**, **`time_base = 1/fps`**GitHub. Vì thế, khung hình được xác định bởi chỉ số frame index.
- **`AudioTrack`** lấy một tuple **`(idx, samples)`** từ **`audio_queue`**, nhưng chỉ dùng **`samples`**; nó tự cộng dồn **`_pts_samples`** để gán **`pts`** và **`time_base = 1/sample_rate`**GitHub. Do đó giá trị **`idx`** kèm theo audio khối hiện nay không được dùng để đồng bộ audio với video.
- Mỗi phiên WebRTC có **`video_queue`** và **`audio_queue`**; producer chỉ cần **`put((frame_idx, frame))`** và **`put((frame_idx, audio_chunk))`**GitHub.

Với nhu cầu phát âm thanh theo file bên cạnh luồng hình ảnh được sinh từ MuseTalk, ý tưởng của bạn – "audio chỉ bắt đầu khi frame đầu tiên được chiếu và trước khi gửi mỗi đoạn audio cần kiểm tra đã có frame ở thời gian tương ứng chưa" – là hợp lý. Dưới đây là cách triển khai mà không cần can thiệp vào lớp **`AudioTrack`**:

1. **Chuẩn bị audio:** đọc file âm thanh (wav/mp3) thành mảng mẫu đơn kênh, ví dụ dùng **`wave`** hoặc **`soundfile`**. Tính **`samples_per_frame = sample_rate / fps`** để biết mỗi khung hình sẽ tương ứng bao nhiêu mẫu âm thanh.
2. **Luồng sinh frame:** giữ nguyên việc gọi **`musetalk_service.generate_frames_for_webrtc(audio_path, video_queue, dummy_audio_queue, fps…)`** để MuseTalk chỉ đẩy frame vào **`video_queue`** (đưa một **`audio_queue`** phụ để bỏ qua audio do MuseTalk sinh ra).
3. **Luồng phát audio:** tạo một thread riêng đọc audio theo từng đoạn nhỏ. Trước khi gửi đoạn audio đầu tiên, đợi cho tới khi **`video_queue`** có ít nhất một frame (sử dụng **`video_queue.empty()`** hoặc xem phần tử cuối **`video_queue.queue[-1]`**). Với mỗi đoạn audio i:

```
python


while True:
    # đọc frame index lớn nhất đã có trong video_queue
    with video_queue.mutex:
        last_idx = video_queue.queue[-1][0] if video_queue.queue else -1
    audio_time = (i * samples_per_frame) / sample_rate
    frame_time = last_idx / fps
    # chỉ gửi nếu |audio_time - frame_time| < threshold
    if abs(audio_time - frame_time) < 0.1:  # threshold 100 ms
        break
    time.sleep(0.01)

audio_queue.put((i, audio_chunk))
```

Như vậy, audio sẽ bắt đầu khi có frame đầu tiên, và trong quá trình phát sẽ luôn kiểm tra xem khung hình tương ứng đã có chưa. Chỉ khi điều kiện lệch thời gian nhỏ hơn ngưỡng (**`threshold`**) thì đoạn audio mới được đưa vào **`audio_queue`**.

4. **Thay đổi API:** cập nhật endpoint **`/api/webrtc/realtime/start`** để nhận thêm tham số **`audio_path`**, rồi truyền nó vào hàm **`start_realtime_session`**. Trong **`start_realtime_session`**, sau khi chuẩn bị avatar bạn lấy **`video_queue, audio_queue = webrtc_service.get_producer_queues(session_id)`**, khởi tạo thread tạo frame (gọi MuseTalk) và thread phát audio như trên.

Vì **`AudioTrack.recv`** tích lũy **`_pts_samples`** thay vì dùng **`idx`**GitHub, việc chèn âm thanh trễ sẽ không gây ra "khoảng trống" trong luồng audio: khi thread audio chưa push dữ liệu, **`audio_queue.get()`** sẽ block và luồng client sẽ yên lặng cho tới khi có audio.

Nếu file âm thanh dài hơn số frame hiện có (ví dụ do khung hình bị chậm), cơ chế trên sẽ tự động đợi cho đến khi có đủ frame để phát tiếp audio. Còn nếu video kết thúc trước khi audio hết, bạn có thể dừng luồng audio sau khi phát xong.

Cách triển khai này không cần thay đổi nhiều trong lớp **`WebRTCService`**; bạn chỉ cần sửa **`start_realtime_session`** (và endpoint API tương ứng) để đọc **`audio_path`**, khởi chạy hai thread và đẩy audio theo điều kiện như mô tả ở trên.

Sources

> Tôi không có ý định chia audio thành chunks để gửi vào queue nữa nên audio queue có vẻ không cần nữa. Mà tôi sẽ gửi toàn bộ audio luôn

Khi không chia nhỏ tệp âm thanh mà gửi cả file một lần, hàng đợi audio (**`audio_queue`**) gần như không còn giá trị sử dụng – **`AudioTrack.recv()`** của aiortc chỉ lấy ra một mảng mẫu audio duy nhất và chuyển thành một **`AudioFrame`** với toàn bộ độ dài âm thanh. Trong mã hiện tại, biến **`idx`** đi kèm với audio chunk hoàn toàn bị bỏ qua (nó chỉ được gán vào biến **`_idx`** rồi vứt đi) và **`pts`** được tính bằng tổng số mẫu đã gửi GitHub. Vì vậy, nếu bạn đẩy toàn bộ file vào queue, **`AudioTrack`** sẽ phát nó liên tục ngay sau khi nhận được mà không hề quan tâm tới số frame hay thời gian.

Với cách làm "gửi nguyên file":

- Bạn vẫn có thể trì hoãn việc phát bằng cách **đợi khi hàng đợi video có frame đầu tiên** rồi mới `put()` dữ liệu âm thanh vào queue (ví dụ: `while video_queue.empty(): time.sleep(…)`). Điều này đáp ứng được điều kiện thứ nhất (chỉ bắt đầu khi có frame đầu).
- Tuy nhiên, **không còn cách nào để kiểm tra và "dừng lại" ở các mốc thời gian sau đó**. Một khi cả dải âm thanh đã được đưa vào queue và bắt đầu được truyền, luồng audio sẽ chạy đến hết vì `AudioTrack` không có cơ chế tạm dừng dựa trên `idx`GitHub.

Nếu bạn vẫn muốn đảm bảo điều kiện thứ hai (đến giây thứ N phải kiểm tra xem đã có frame tương ứng hay chưa) thì buộc phải chia nhỏ âm thanh để có thể quyết định gửi hay tạm dừng tại các mốc thời gian. Hai lựa chọn khả thi:

1. **Giữ audio queue và chia tệp thành nhiều đoạn nhỏ**: Bạn gửi từng đoạn theo thời gian thực; trước mỗi đoạn, kiểm tra `last_frame_idx / fps` so với `audio_time` giống như gợi ý ban đầu. Đây là cách duy nhất cho phép tạm dừng hoặc làm chậm audio nếu video bị trễ.
2. **Gửi toàn bộ file đến client và điều khiển trên frontend**: Trả về URL của file âm thanh, tạo một thẻ `<audio>` ở client, nhưng chỉ gọi `audio.play()` khi đã nhận được frame đầu tiên. Trong quá trình phát, bạn có thể dùng API `currentTime` của HTML5 audio kết hợp với số frame đã hiển thị (truy cập qua một endpoint trạng thái) để tự tạm dừng audio nếu nó chạy nhanh hơn video. Cách này không dùng `audio_queue` nhưng yêu cầu bạn phải cập nhật mã JavaScript để đồng bộ audio với video ở phía trình duyệt.

Tóm lại, **nếu bỏ cơ chế chia nhỏ audio**, bạn chỉ còn khả năng trì hoãn việc phát tới lúc có frame đầu. Điều kiện kiểm tra tại các mốc thời gian sau đó là không thực hiện được ở phía server mà không chia nhỏ hoặc điều khiển client.

Sources

↪

Gửi toàn bộ file đến client và điều khiển trên frontend: Trả về URL của file âm thanh, tạo một thẻ `<audio>` ở client, nhưng chỉ gọi `audio.play()` khi đã nhận được frame đầu tiên. Trong quá trình phát, bạn có thể dùng API `currentTime` của HTML5 audio kết hợp với số frame đã hiển thị (truy cập qua một endpoint trạng thái) để tự tạm dừng audio nếu nó chạy nhanh hơn video. Cách này không dùng `audio_queue` nhưng yêu cầu bạn phải cập nhật mã JavaScript để đồng bộ audio với video ở phía trình duyệt.

> Tôi cũng có ý này, tôi nên implement thế nào nhỉ, trước tiên cứ trình bày đầy đủ hướng implement trước, còn code thì cứ từ từ chưa vội

Để triển khai hướng **gửi toàn bộ tệp âm thanh đến client và điều khiển phát trên frontend**, bạn cần sửa lại cả backend và frontend, nhưng cách làm sẽ đơn giản hơn so với việc chia nhỏ âm thanh. Dưới đây là một lộ trình cụ thể để thực hiện:

---

1. Backend: cung cấp file âm thanh cho client

1. **Lưu đường dẫn âm thanh vào dữ liệu phiên**
   Khi khởi tạo phiên realtime, bạn cần biết file âm thanh sẽ dùng. Có thể lấy từ `StreamProduct.audio_path` hoặc truyền qua tham số API. Lưu đường dẫn này vào trạng thái phiên để client có thể truy cập.
2. **Trả về URL âm thanh**
   - Bạn có thể thêm một trường `audio_url` vào response của API khởi động realtime. Ví dụ, endpoint `/api/webrtc/realtime/start` trả về:

     ```
     json

     {
       "status": "realtime_started",
       "audio_url": "/static/audio/answer_12_3.mp3",
       "fps": 25
     }
     ```

   - Hoặc tạo một endpoint riêng để client yêu cầu đường dẫn âm thanh khi cần.
3. **Cung cấp thông tin đồng bộ video**
   Để cho frontend biết video đã phát đến frame nào, backend cần expose một endpoint trả về "frame index hiện tại". Một phương án nhẹ nhàng:
   - Khi producer đẩy frame vào `video_queue`, hãy cập nhật một biến toàn cục `last_frame_idx`.
   - Tạo API `GET /api/webrtc/status/{session_id}` trả về `last_frame_idx` và `fps` (nếu chưa có). Người dùng có thể dựa vào `video_time = last_frame_idx / fps` để so sánh với `audio.currentTime`.

   Trong mã hiện tại, `VideoTrack` gán `pts` cho mỗi frame bằng `idx` và `time_base` là `1/fps`GitHub, vì vậy bạn hoàn toàn có thể dùng `frame_idx / fps` để tính thời gian đã phát.

---

2. Frontend: điều khiển thẻ `<audio>`

1. **Tạo và preload thẻ audio**
   Trong `live.js`, khi nhận được `audio_url` từ backend:

   ```
   javascript

   const audioEl = document.createElement('audio');
   audioEl.src = audioUrlFromApi;
   audioEl.preload = 'auto';
   audioEl.autoplay = false;
   document.body.appendChild(audioEl);
   ```

2. **Chờ video frame đầu tiên rồi mới phát audio**
   - Khi WebRTC nhận track video đầu tiên (`pc.ontrack` với `track.kind === 'video'`), bạn biết rằng đã có frame. Lúc này gọi `audioEl.play()`. Điều này đáp ứng điều kiện "audio chỉ bắt đầu khi frame đầu tiên được chiếu".
3. **Theo dõi tiến độ phát để giữ đồng bộ**
   - Định kỳ (ví dụ mỗi 200 ms) gọi tới endpoint trạng thái backend để lấy `last_frame_idx` và `fps`.
   - Tính thời gian của video: `const videoTime = lastFrameIdx / fps`.

- Lấy thời gian hiện tại của audio: **`const audioTime = audioEl.currentTime`**.
- Nếu **`audioTime`** > **`videoTime + threshold`** (ví dụ **`threshold = 0.1`** giây), tạm dừng audio: **`audioEl.pause()`**.
- Nếu audio đang tạm dừng và **`audioTime <= videoTime`**, tiếp tục phát: **`audioEl.play()`**.

Pseudocode:

```
javascript

let syncInterval = setInterval(async () => {
  const status = await fetch(`/api/webrtc/status/${sessionId}`).then(r => r.json());
  const videoTime = status.last_frame_idx / status.fps;
  const audioTime = audioEl.currentTime;
  if (audioTime > videoTime + 0.1) {
    audioEl.pause();
  } else if (audioEl.paused && audioTime <= videoTime) {
    audioEl.play();
  }
}, 200);
```

4. **Xử lý khi kết thúc**
   Khi video kết thúc hoặc người dùng dừng stream, gọi **`clearInterval(syncInterval)`** và dừng audio.

---

3. Ưu và nhược điểm

- Ưu điểm: không phải chia nhỏ âm thanh, không cần hàng đợi audio ở backend, logic đồng bộ nằm hoàn toàn ở frontend.
- Nhược điểm: cần thêm một endpoint để frontend biết frame hiện tại; việc polling thường xuyên có thể gây tải nhẹ lên server. Ngoài ra, nếu video bị trễ quá lâu, audio sẽ liên tục bị tạm dừng và người xem có thể thấy giật.

Với lộ trình này, bạn vẫn giữ nguyên kiến trúc WebRTC và chỉ mở rộng API/JS để điều khiển playback. Sau khi triển khai xong cơ chế lấy **`last_frame_idx`** trên server và thêm logic kiểm soát **`audio.currentTime`** trên client, bạn có thể điều chỉnh thêm ngưỡng **`threshold`** hoặc khoảng polling để đạt trải nghiệm mượt mà nhất.

Sources

↪

Định kỳ (ví dụ mỗi 200 ms) gọi tới endpoint trạng thái backend để lấy `last_frame_idx` và `fps`.
Tôi nghĩ cái này có thể lấy ra từ video queue hoặc video track mà

Bạn hoàn toàn đúng khi nói rằng thông tin “video đang chạy tới đâu” trên lý thuyết có thể có sẵn ở phía backend thông qua hàng đợi video hoặc ở phía client thông qua luồng video. Tuy nhiên, cần phân biệt rõ hai trường hợp:

- **Hàng đợi video (`video_queue`) hay biến `last_frame_idx` nằm trong process server** – đó là nơi producer đẩy `(frame_idx, frame)`GitHub. Trình duyệt không thể trực tiếp truy cập vào queue này; muốn sử dụng nó bạn phải cung cấp một API để gửi giá trị hiện tại ra ngoài (như ví dụ `status` trong gợi ý trước). Nếu không có API, client không biết hàng đợi đã phát tới index bao nhiêu.
- **Client nhận luồng video qua WebRTC** – trình duyệt chỉ nhận các gói RTP đã mã hóa và dựng thành hình ảnh. Thư viện WebRTC trên trình duyệt không expose `frame_idx` hay `pts` của mỗi frame; vì vậy bạn không thể đọc trực tiếp số khung đã hiển thị từ đối tượng `MediaStreamTrack`.

Tuy nhiên, bạn vẫn có một lựa chọn đơn giản mà không cần gọi API: sử dụng thuộc tính `currentTime` của thẻ `<video>` để ước lượng thời gian phát của luồng video. Khi server gán `pts` và `time_base` cho mỗi `VideoFrame`GitHub, trình duyệt sẽ xây dựng một mốc thời gian nội bộ; `videoEl.currentTime` đại diện cho số giây kể từ lúc track bắt đầu phát. Nếu bạn khởi động audio cùng lúc với việc hiển thị frame đầu tiên, thì `videoEl.currentTime` và `audioEl.currentTime` nằm cùng một gốc thời gian. Do đó bạn có thể đồng bộ như sau:

```javascript
// khi nhận video frame đầu, bắt đầu audio:
pc.ontrack = (e) => {
  if (e.track.kind === 'video') {
    // ... gắn track vào videoEl.srcObject ...
    if (!audioStarted) {
      audioStarted = true;
      audioEl.play();
    }
  }
};

// định kỳ so sánh video và audio
setInterval(() => {
  const videoTime = videoEl.currentTime;     // thời gian của luồng video
  const audioTime = audioEl.currentTime;     // thời gian của file audio
  if (audioTime > videoTime + 0.1) {         // audio chạy nhanh hơn video
    audioEl.pause();
  } else if (audioEl.paused && audioTime <= videoTime) {
    audioEl.play();
  }
}, 200);
```

Cách làm này không cần truy cập vào queue backend hay gọi API; bạn dựa vào `currentTime` của thẻ video để biết video đã phát tới đâu và điều chỉnh audio tương ứng. Điểm cần lưu ý:

- `videoEl.currentTime` có thể hơi lệch so với server do độ trễ mạng, nhưng thông thường chúng tương đối ổn định vì WebRTC sử dụng timestamp (`pts`) gửi từ serverGitHub.
- Bạn vẫn cần đảm bảo audio bắt đầu phát đúng lúc – tức là đợi tới khi `ontrack` nhận được frame video đầu tiên rồi mới gọi `audio.play()`.

Nếu muốn chính xác hơn (ví dụ để hiển thị trạng thái đồng bộ cho admin), bạn vẫn có thể cập nhật server để lưu `last_frame_idx` và cung cấp qua API. Nhưng với yêu cầu của bạn – phát nguyên tệp âm thanh và chỉ cần tránh việc audio chạy quá nhanh so với video – việc dùng thuộc tính `currentTime` của thẻ `<video>` là cách nhẹ nhàng và không cần can thiệp sâu vào backend.

Sources

> ừm, thế thì dùng cách cũ vẫn chính xác hơn nhỉ. Có giải pháp nào khác không?

Nếu bạn cảm thấy việc polling API trạng thái từ backend là nặng nề, có một vài cách khác để lấy thông tin "video đã phát tới đâu" mà không phải dựa vào **`currentTime`** của **`<video>`**:

1. Sử dụng `requestVideoFrameCallback` trên thẻ video

Các trình duyệt hiện đại (Chrome, Edge, Firefox) hỗ trợ phương thức **`videoEl.requestVideoFrameCallback()`**. Mỗi khi một khung hình mới sắp được hiển thị, callback này sẽ nhận được một đối tượng **`VideoFrameMetadata`** có các trường như **`presentedFrames`**, **`expectedDisplayTime`** và **`presentationTime`**. Bạn có thể dùng **`metadata.presentedFrames`** để biết chính xác đã hiển thị bao nhiêu frame kể từ lúc video bắt đầu mà không cần hỏi backend.

Ví dụ:

```
javascript


let presentedFrames = 0;

function onFrame(now, metadata) {
  presentedFrames = metadata.presentedFrames;
  // Đăng ký callback cho frame tiếp theo
  videoEl.requestVideoFrameCallback(onFrame);
}

// Gọi lần đầu khi video bắt đầu phát
videoEl.requestVideoFrameCallback(onFrame);

// Trong vòng lặp đồng bộ audio:
const videoTime = presentedFrames / fps; // fps lấy từ API khởi tạo
const audioTime = audioEl.currentTime;
if (audioTime > videoTime + 0.1) {
  audioEl.pause();
} else if (audioEl.paused && audioTime <= videoTime) {
  audioEl.play();
}
```

Ưu điểm của cách này là toàn bộ logic nằm ở phía client: bạn không cần endpoint để trả về **`last_frame_idx`** và không phải ước lượng bằng **`currentTime`**. Giá trị **`presentedFrames`** tăng đúng theo số frame hiển thị nên độ chính xác cao hơn.

2. Dùng WebRTC data channel để truyền thông tin đồng bộ

Thay vì HTTP polling, bạn có thể tạo một **data channel** trong cùng phiên WebRTC. Ở phía server, mỗi khi producer đẩy một frame vào **`video_queue`** thì gửi một gói dữ liệu rất nhỏ (chẳng hạn **`{frame_idx: idx}`**) qua data channel. Ở phía client, lắng nghe sự kiện **`onmessage`** của data channel để cập nhật **`lastFrameIdx`**. Sau đó bạn thực hiện so sánh giữa **`audio.currentTime`** và **`lastFrameIdx / fps`**.

Ưu điểm của data channel là:

- Không phát sinh thêm request HTTP; sử dụng cùng kết nối WebRTC hiện có.
- Độ trễ thấp và dữ liệu nhỏ (chỉ vài byte cho mỗi frame hoặc vài lần mỗi giây).

Mẫu mã giả (client):

```
javascript

// Khi khởi tạo WebRTC
const dc = pc.createDataChannel('sync');
dc.onmessage = (event) => {
  const data = JSON.parse(event.data);
  lastFrameIdx = data.frame_idx;
};

// Đồng bộ audio
setInterval(() => {
  const videoTime = lastFrameIdx / fps;
  const audioTime = audioEl.currentTime;
  // so sánh như trước...
}, 200);
```

Server (producer thread):

```
python

# sau khi push frame vào video_queue
data_channel.send(json.dumps({"frame_idx": idx}))
```

Cách này yêu cầu bạn thêm một kênh dữ liệu và gửi thông tin định kỳ, nhưng nó loại bỏ nhu cầu polling và giữ cho đồng bộ chính xác hơn so với chỉ dựa trên thời gian hiển thị.

3. Ghép audio vào chính luồng WebRTC

Phương án "cổ điển" là giữ nguyên mô hình audio queue và để aiortc quản lý đồng bộ. Nếu bạn push toàn bộ âm thanh vào **`audio_queue`** ngay khi frame đầu tiên xuất hiện, WebRTC sẽ tự đồng bộ audio và video theo **`pts`** của khung hìnhGitHub. Tuy nhiên bạn sẽ mất khả năng tạm dừng audio khi video chậm vì audio sẽ tiếp tục chạy. Muốn dừng giữa chừng thì buộc phải chia nhỏ (hoặc đẩy các đoạn silent) – đây là lý do vì sao giải pháp "đẩy chunk ngắn và kiểm tra" được khuyến khích nếu cần kiểm soát chặt chẽ.

Tóm lại, nếu muốn tránh việc gọi API mà vẫn đồng bộ chính xác, hai giải pháp thực tiễn là:

- **`requestVideoFrameCallback`** – hoàn toàn client-side, dễ triển khai và đủ chính xác trong hầu hết trường hợp.
- **Data channel để gửi `frame_idx`** – yêu cầu sửa backend để gửi thông tin đồng bộ qua WebRTC nhưng loại bỏ hoàn toàn HTTP polling.

Bạn có thể chọn giải pháp phù hợp với nhu cầu và độ phức tạp chấp nhận được trong dự án của mình.

Sources

Phân chia hợp lý để code các module, chỉ rõ là ở thành phần nào, thực hiện việc gì

Để triển khai chức năng "phát cả video và audio đồng bộ" một cách rõ ràng, dễ bảo trì, bạn nên tách thành các mô-đun độc lập ở cả backend và frontend. Dưới đây là một gợi ý phân chia và vai trò từng

thành phần:

Backend (Python/FastAPI)

1. `webrtc_service.py` (đã có)

- **Vai trò:** Quản lý các phiên WebRTC: tạo/kết thúc `RTCPeerConnection`, giữ `video_queue` và `audio_queue` cho mỗi `session_id`.
- **Sửa/Thêm:**
  - Bổ sung khả năng tạo **data channel** nếu chọn phương án đồng bộ qua data channel. Khi nhận request tạo offer, hãy `pc.createDataChannel('sync')` và lưu lại để producer có thể gửi thông điệp `frame_idx`.
  - Cung cấp phương thức `send_sync_message(session_id, frame_idx)` gửi frame index qua data channel tương ứng.

2. `musetalk_service.py` (đã có)

- **Vai trò:** Gọi MuseTalk để sinh các khung hình theo audio, prepare avatar...
- **Sửa/Thêm:** Nếu dùng data channel, trong hàm sinh frame (hoặc trong producer thread của `StreamProcessor`) sau mỗi lần `video_queue.put((idx, frame))` hãy gọi `webrtc_service.send_sync_message(session_id, idx)`.

3. `stream_processor.py` (đã có)

- **Vai trò:** Điều phối toàn bộ phiên livestream: chuẩn bị avatar, gọi MuseTalk sinh frame, push vào hàng đợi, v.v.
- **Sửa/Thêm:**
  - Khi khởi động realtime, trả về `audio_url` và `fps` để client biết file audio và tốc độ video (nếu dùng phương án `requestVideoFrameCallback`).
  - Nếu chọn đồng bộ bằng data channel thì không cần API `status` nữa; ngược lại, nếu chọn polling, thêm biến `last_frame_idx` và endpoint `GET /api/webrtc/status/{session_id}` trả về `last_frame_idx` và `fps`.

4. `api/webrtc.py` (đã có)

- **Vai trò:** Định nghĩa các endpoint `/offer`, `/realtime/start`, `/status`, v.v.
- **Sửa/Thêm:**
  - Endpoint `/api/webrtc/realtime/start` nên trả về thêm `audio_url` (đường dẫn tệp âm thanh) và `fps`.
  - Endpoint `/api/webrtc/status/{session_id}` (nếu dùng polling) trả về `last_frame_idx` và `fps` để client tính thời gian video.

Frontend (JavaScript)

1. Module WebRTC

- **Vai trò:** Thiết lập `RTCPeerConnection`, gửi offer, nhận track video (và audio nếu dùng audio queue), mở data channel.
- **Sửa/Thêm:**
  - Nếu dùng data channel: `const dc = pc.createDataChannel('sync')` (hoặc lắng nghe data channel được tạo sẵn) và cài `dc.onmessage = (evt) => { lastFrameIdx = JSON.parse(evt.data).frame_idx; }`.
  - Khi nhận được track video đầu tiên (`pc.ontrack`), gắn vào thẻ `<video>` và bắt đầu audio (nếu đã sẵn sàng).

2. Module AudioPlayer

- **Vai trò:** Quản lý thẻ `<audio>`: tải file âm thanh (`audio_url`), chờ sẵn, phát/dừng.
- **Sửa/Thêm:**
  - Khi `WebRTC` báo đã nhận frame đầu tiên, gọi `audio.play()`.
  - Cung cấp phương thức `syncWith(videoTime)` để so sánh và tạm dừng hoặc tiếp tục audio.

3. Module SyncController

- **Vai trò:** Quyết định khi nào tạm dừng hoặc phát lại audio dựa trên tiến độ video.
- **Cách triển khai tùy phương án:**
  - **Phương án polling API:** Định kỳ (200 ms) gọi `/api/webrtc/status/{session_id}` để lấy `last_frame_idx` và `fps`, sau đó tính `videoTime = last_frame_idx / fps`, so với `audio.currentTime`.
  - **Phương án `requestVideoFrameCallback`:** Đăng ký `videoEl.requestVideoFrameCallback()` để nhận `metadata.presentedFrames` liên tục, tính `videoTime = presentedFrames / fps` mà không cần backend.
  - **Phương án data channel:** Sử dụng `lastFrameIdx` cập nhật từ `dc.onmessage` rồi tính `videoTime = lastFrameIdx / fps`. Khi `audio.currentTime > videoTime + threshold` thì `audio.pause()`, ngược lại nếu audio đang pause và `audio.currentTime <= videoTime` thì `audio.play()`.

Lựa chọn phương án đồng bộ

Phương án	Backend cần làm	Frontend cần làm	Độ chính xác / độ phức tạp
Polling API	Cập nhật `last_frame_idx` và cung cấp endpoint `/status`	Định kỳ fetch, so sánh `last_frame_idx / fps` với `audio.currentTime`	Dễ cài đặt, nhưng có độ trễ và sinh nhiều request
requestVideoFrameCallback	Trả về `fps` khi khởi tạo	Dùng `videoEl.requestVideoFrameCallback()`	Hoàn toàn

Phương án	Backend cần làm	Frontend cần làm	Độ chính xác / độ phức tạp
		để lấy **`presentedFrames`**, so sánh với **`audio.currentTime`**	client-side, không cần endpoint; phụ thuộc hỗ trợ trình duyệt
Data channel	Tạo data channel sync, gửi **`{frame_idx}`** mỗi khi push frame	Lắng nghe data channel, cập nhật **`lastFrameIdx`**, so sánh với **`audio.currentTime`**	Đồng bộ chính xác, không cần polling; phức tạp hơn chút khi thiết lập WebRTC

Với cách tổ chức này, mỗi phần có trách nhiệm rõ ràng:

- **Musetalk và producer** tập trung sinh frame.
- **WebRTC service** quản lý kết nối, hàng đợi, và (nếu có) kênh dữ liệu đồng bộ.
- **API layer** cung cấp điểm vào cho client: khởi động phiên, lấy thông tin audio/fps, hoặc trạng thái frame.
- **Frontend** tách biệt giữa lớp hiển thị WebRTC, lớp phát audio và lớp đồng bộ, giúp dễ thay đổi một thành phần mà không ảnh hưởng tới phần khác.

Bạn có thể bắt đầu bằng cách chọn phương án đồng bộ phù hợp, rồi hiện thực từng module theo cấu trúc trên. Khi đã sẵn sàng với thiết kế, việc chuyển sang hiện thực mã sẽ thuận tiện và ít phát sinh lỗi hơn.