Documentacion

1) Descripción general

La integración de HeyGen Streaming API con LiveKit v2 permite que un avatar IA se **conecte en tiempo real** a un canal WebRTC basado en LiveKit, reciba texto (o audio) en tiempo real, procese/ande con lógica (por ejemplo un LLM o repetición) y emita vídeo + audio con muy baja latencia. (HeyGen API Documentation)

Principales características

- Uso de WebRTC (vía LiveKit) para streaming de video en tiempo real. (HeyGen API Documentation)
- Soporte para diferentes "task types": por ejemplo *talk* (avatar habla usando LLM) o *repeat* (repite texto exacto). (HeyGen API Documentation)
- Opción de usar la infraestructura de LiveKit gestionada por HeyGen **o** usar tu propia instancia de LiveKit para mayor control. (HeyGen API Documentation)
- Soporte para SDK de HeyGen (Streaming Avatar SDK) que encapsula gran parte de este flujo. (HeyGen API Documentation)

Caso de uso típico

Por ejemplo, quieres un "asistente virtual" con cara de avatar IA que interactúe con un usuario en directo (voz + vídeo). El usuario se conecta a una sala LiveKit, el avatar también, y se envían mensajes de texto o voz (STT) para que el avatar responda en vivo.

La integración facilita: inicio de sesión, creación de sesión avatar, arranque del streaming, envío de tareas (texto), recepción de vídeo/audio.

Beneficios

- Muy baja latencia: al usar WebRTC/LiveKit.
- Experiencia real-time: ideal para reuniones, soporte en vivo, escenarios interactivos.

- Flexibilidad: seleccionar modo “gestionar tu propia LiveKit” o “usar infraestructura de HeyGen”.
- Escalabilidad: al basarse en LiveKit, se puede integrar con infraestructura ya existente.

---

2) Flujo de funcionamiento

Aquí el flujo general que debes implementar:

1. Autenticación: obtener token/clave para usar la API de HeyGen.
2. Crear una sesión streaming: llamar al endpoint para crear la sesión (v2).
3. Iniciar la sesión: activar el streaming, vincular con LiveKit (URL, access_token).
4. Conectarse al room de LiveKit desde cliente: con SDK de LiveKit o directamente WebRTC.
5. Interacción: enviar texto (o audio) para que el avatar hable.
6. Opcional: WebSocket de control/monitorización de eventos.
7. Cierre de sesión: detener el streaming, desconectar de room, limpiar recursos.

---

3) Endpoints y parámetros (documentación técnica)

Basado en la guía oficial, aquí tienes los endpoints relevantes y sus parámetros críticos.

3.1 `/v1/streaming.create_token`

- **Método**: `POST`
- **Propósito**: generar un token de sesión para autenticación. (HeyGen API Documentation)
- **Headers**:
  - `X-Api-Key`: tu API Key de HeyGen (algunas guías indican también “Authorization: Bearer …”).
- **Body**: usualmente vacío o mínimo (según documentación).

- **Respuesta**: devuelve algo como `{ data: { token: "..." } }`. (HeyGen API Documentation)

3.2 `/v1/streaming.new`

- **Método**: `POST`
- **Propósito**: crear una nueva sesión de streaming para el avatar. (HeyGen API Documentation)
- **Headers**: `Content-Type: application/json`, `Authorization: Bearer <session_token>` (o `X-Api-Key` según flujo).
- **Body (ejemplo mínimos)**:

```
{
  "version": "v2",
  "avatar_id": "YOUR_AVATAR_ID",
  "quality": "high",                // opcional: e.g., "high" / "medium" / "low"
  "video_encoding": "H264",         // opcional
  "voice": {
    "voice_id": "YOUR_VOICE_ID",
    "rate": 1.0,
    "emotion": "FRIENDLY"           // opcional
  },
  "knowledge_id": "YOUR_KNOWLEDGE_ID", // opcional
  "language": "en"
}
```

  (HeyGen API Documentation)

- **Respuesta**:

```
{
  "session_id": "xxx",
  "access_token": "yyy",
  "url": "wss://... (LiveKit websocket URL)",
  "is_paid": true,
  "session_duration_limit": 600
}
```

(Nota: si usas tu propia instancia de LiveKit, `access_token` / `url` pueden variar). (HeyGen API Documentation)

3.3 `/v1/streaming.start`

- **Método**: `POST`
- **Propósito**: iniciar el streaming una vez creada la sesión. (HeyGen API Documentation)
- **Headers**: `Authorization: Bearer <session_token>`
- **Body**:

```
{
  "session_id": "<session_id>"
}
```

  (en algunos casos también `"session_token": "<session_token>"`, `"silence_response": false`, `"stt_language": "en"`). (HeyGen API Documentation)

- **Respuesta**: información de conexión para LiveKit (URL + access_token) o confirmación de arranque.

3.4 `/v1/streaming.task`

- **Método**: `POST`
- **Propósito**: enviar una tarea al avatar (texto para hablar). (HeyGen API Documentation)
- **Headers**: `Authorization: Bearer <session_token>`
- **Body**:

```
{
  "session_id": "<session_id>",
  "text": "Hello world",
  "task_type": "talk" // o "repeat"
}
```

- **Respuesta**: usualmente `200 OK` + details o `{ success: true }`.

3.5 `/v1/streaming.stop`

- **Método**: `POST`
- **Propósito**: detener la sesión de streaming, liberar recursos. (HeyGen API Documentation)
- **Headers**: `Authorization: Bearer <session_token>`
- **Body**:

```
{
  "session_id": "<session_id>"
}
```

- **Respuesta**: confirmación de parada.

---

4) Flujo de implementación (Ejemplo JS/TS)

Aquí tienes un ejemplo (cliente web básica) que refleja el flujo descrito:

```
// config.ts
export const API_CONFIG = {
  serverUrl: "https://api.heygen.com",
  token: "YOUR_API_KEY_OR_ACCESS_TOKEN",
};

// streaming.ts
import { API_CONFIG } from "./config";

let sessionInfo: { session_id: string; access_token: string; url: string } | null
= null;

export async function createSession(avatarId: string, voiceId: string) {
  const resp = await fetch(`${API_CONFIG.serverUrl}/v1/streaming.new`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${API_CONFIG.token}`,
    },
    body: JSON.stringify({
      version: "v2",
```

```
      avatar_id: avatarId,
      voice: {
        voice_id: voiceId,
        rate: 1.0
      },
      quality: "high",
      video_encoding: "H264",
    }),
  });
  const data = await resp.json();
  sessionInfo = {
    session_id: data.session_id,
    access_token: data.access_token,
    url: data.url,
  };
  return sessionInfo;
}

export async function startStreaming() {
  if (!sessionInfo) throw new Error("Session not created");
  const resp = await fetch(`${API_CONFIG.serverUrl}/v1/streaming.start`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${API_CONFIG.token}`,
    },
    body: JSON.stringify({
      session_id: sessionInfo.session_id,
    }),
  });
  return resp.json();
}

export async function sendTextTask(text: string, taskType: "talk"|"repeat"
= "talk") {
  if (!sessionInfo) throw new Error("Session not created");
  await fetch(`${API_CONFIG.serverUrl}/v1/streaming.task`, {
    method: "POST",
```

```
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${API_CONFIG.token}`,
    },
    body: JSON.stringify({
      session_id: sessionInfo.session_id,
      text,
      task_type: taskType,
    }),
  });
}

export async function stopStreaming() {
  if (!sessionInfo) throw new Error("Session not created");
  await fetch(`${API_CONFIG.serverUrl}/v1/streaming.stop`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${API_CONFIG.token}`,
    },
    body: JSON.stringify({
      session_id: sessionInfo.session_id,
    }),
  });
  sessionInfo = null;
}
```

Luego en tu cliente (frontend) te conectas al room de LiveKit usando `sessionInfo.url` y `sessionInfo.access_token`, por ejemplo con el SDK de LiveKit:

```
import { Room } from "livekit-client";

const room = new Room();
await room.connect(sessionInfo.url, sessionInfo.access_token);
// Al recibir pistas de vídeo/audio:
room.on(Room.TrackSubscribed, (track) => {
  if (track.kind === "video") {
    const mediaStream = new MediaStream([track.mediaStreamTrack]);
```

```
    videoElement.srcObject = mediaStream;
  }
});
```

Y después envías tareas con `sendTextTask("Hola ¿cómo estás?")`.

---

5) Implementación avanzada: usar instancia propia de LiveKit

Si prefieres controlar totalmente la infraestructura de LiveKit (por ejemplo por motivos de cumplimiento, personalización, permisos), puedes usar la guía “Using your own LiveKit Instance”. (HeyGen API Documentation)

Puntos clave de esta variante

- Tu backend genera **tokens de acceso para LiveKit** tanto para el avatar (HeyGen) como para los clientes.
- Cuando llamas a `streaming.new`, debes incluir un campo `livekit_settings` que contiene:

  ```
  {
    "url": "wss://your-livekit.domain",
    "room": "room_name",
    "token": "<token_for_avatar>"
  }
  ```

- La respuesta ya **no** devuelve `access_token` / `url` para LiveKit, porque tú lo manejas. (HeyGen API Documentation)
- Tu cliente frontend usa los tokens que tú generas para conectarse al room.

Cuándo usarlo

- Cuando tienes requisitos de cumplimiento o certificación que requieren infraestructura propia.
- Cuando ya usas LiveKit para otras funciones (vídeo conferencia, agentes multi-participantes).
- Cuando necesitas control granular del room (múltiples agentes IA, moderadores, bots).

---

6) Checklist de implementación

Para que verifiques paso a paso:

- [ ] Obtienes API key/token de HeyGen.
- [ ] Llamas a `streaming.create_token` (si aplica).
- [ ] Creas sesión `streaming.new` con parámetros correctos (avatar_id, calidad, voz).
- [ ] Llamas `streaming.start` y obtienes URL + access_token para LiveKit (o configuras tu propia instancia).
- [ ] En el cliente, usas SDK LiveKit para conectar al room (usando url y token).
- [ ] Una vez conectado, envías tareas con `streaming.task` (texto), y la pista de vídeo del avatar aparece en el cliente.
- [ ] Cuando termines, llamas `streaming.stop` para liberar la sesión.
- [ ] Manejas errores, eventos de pista de vídeo/audio, reconexiones, desconexiones.
- [ ] Si usas tu propia LiveKit, generas tokens backend para avatar y cliente, configuras `livekit_settings`.
- [ ] Pruebas en distintos escenarios de calidad (“high”, “medium”, “low”), parámetros de voz, idiomas.
- [ ] Verificas latencia, calidad de vídeo, audio, experiencia de usuario.