# **Project API Documentation**

This document outlines the RESTful API endpoints and WebSocket communication channels available in our real-time speech processing application.

## **1. Audio Upload Endpoint**

This endpoint is used for uploading audio files for asynchronous transcription processing.

* **URL:** /api/upload-audio/
* **Method:** POST
* **Description:** Accepts an audio file, assigns a session ID, and queues it for transcription by a Celery worker. The actual transcription and subsequent alerts/transcriptions are sent via WebSocket.
* **Request Body:** multipart/form-data
  + audio: (File) The audio file to be transcribed (e.g., .wav, .mp3, .webm). **Required.**
  + session\_id: (String, Optional) A unique identifier for the session. If not provided, a UUID will be generated.
  + speaker\_type: (String, Optional) The type of speaker for this audio ('prospect', 'agent', or 'unknown'). Defaults to 'prospect'.
* **Example Request (using curl):**  
  curl -X POST \  
   http://localhost:8000/api/upload-audio/ \  
   -H 'Content-Type: multipart/form-data' \  
   -F 'audio=@/path/to/your/audio.wav' \  
   -F 'session\_id=my\_custom\_session\_123' \  
   -F 'speaker\_type=prospect'
* **Example Success Response (HTTP 202 Accepted):**  
  {  
   "message": "Audio file received and queued for processing.",  
   "session\_id": "my\_custom\_session\_123"  
  }
* **Example Error Responses:**
  + **HTTP 400 Bad Request:**  
    {  
     "error": "No audio file provided."  
    }
  + **HTTP 500 Internal Server Error:**  
    {  
     "error": "Failed to process audio file."  
    }

## **2. Keyword Management Endpoints**

These endpoints allow for managing keywords used in real-time detection.

### **2.1. List All Keywords / Create New Keyword**

* **URL:** /api/keywords/
* **Method:** GET (To retrieve all keywords)
* **Method:** POST (To create a new keyword)

#### **GET Request (List Keywords)**

* **Description:** Retrieves a list of all configured keywords.
* **Example Success Response (HTTP 200 OK):**  
  [  
   {  
   "id": 1,  
   "word": "pricing",  
   "talking\_point": "Discuss competitive pricing strategies.",  
   "is\_active": true,  
   "created\_at": "2025-07-08T12:00:00.000000Z",  
   "updated\_at": "2025-07-08T12:00:00.000000Z"  
   },  
   {  
   "id": 2,  
   "word": "integration",  
   "talking\_point": "Highlight seamless integration capabilities.",  
   "is\_active": true,  
   "created\_at": "2025-07-08T12:05:00.000000Z",  
   "updated\_at": "2025-07-08T12:05:00.000000Z"  
   }  
  ]

#### **POST Request (Create Keyword)**

* **Description:** Creates a new keyword entry.
* **Request Body:** application/json
  + word: (String) The keyword to detect. **Required.**
  + talking\_point: (String, Optional) An optional talking point or suggestion associated with the keyword.
  + is\_active: (Boolean, Optional) Whether this keyword is active for detection. Defaults to true.
* **Example Request:**  
  {  
   "word": "budget",  
   "talking\_point": "Explore the prospect's budget constraints and solutions.",  
   "is\_active": true  
  }
* **Example Success Response (HTTP 201 Created):**  
  {  
   "id": 3,  
   "word": "budget",  
   "talking\_point": "Explore the prospect's budget constraints and solutions.",  
   "is\_active": true,  
   "created\_at": "2025-07-08T12:10:00.000000Z",  
   "updated\_at": "2025-07-08T12:10:00.000000Z"  
  }
* **Example Error Response (HTTP 400 Bad Request):**  
  {  
   "word": [  
   "This field may not be blank."  
   ]  
  }

### **2.2. Retrieve, Update, or Delete a Specific Keyword**

* **URL:** /api/keywords/<int:pk>/ (where <int:pk> is the keyword's ID)
* **Method:** GET (Retrieve)
* **Method:** PUT (Full Update)
* **Method:** PATCH (Partial Update)
* **Method:** DELETE (Delete)

#### **GET Request (Retrieve Keyword)**

* **Description:** Retrieves details of a single keyword by its ID.
* **Example Success Response (HTTP 200 OK):**  
  {  
   "id": 1,  
   "word": "pricing",  
   "talking\_point": "Discuss competitive pricing strategies.",  
   "is\_active": true,  
   "created\_at": "2025-07-08T12:00:00.000000Z",  
   "updated\_at": "2025-07-08T12:00:00.000000Z"  
  }
* **Example Error Response (HTTP 404 Not Found):**  
  {  
   "detail": "Not found."  
  }

#### **PUT Request (Full Update Keyword)**

* **Description:** Updates an existing keyword entirely. All fields must be provided.
* **Request Body:** application/json (same as POST for creating a keyword)
* **Example Request:**  
  {  
   "word": "pricing\_updated",  
   "talking\_point": "Review current pricing models.",  
   "is\_active": false  
  }
* Example Success Response (HTTP 200 OK):  
  (Returns the updated keyword object, similar to GET response)

#### **PATCH Request (Partial Update Keyword)**

* **Description:** Partially updates an existing keyword. Only provide the fields you want to change.
* **Request Body:** application/json (subset of keyword fields)
* **Example Request:**  
  {  
   "is\_active": false  
  }
* Example Success Response (HTTP 200 OK):  
  (Returns the partially updated keyword object)

#### **DELETE Request (Delete Keyword)**

* **Description:** Deletes a keyword by its ID.
* **Example Success Response (HTTP 204 No Content):** (No content returned on successful deletion)

## **3. WebSocket Communication**

The WebSocket channel is used for real-time bidirectional communication, primarily for streaming audio chunks and receiving live transcriptions and alerts.

* **URL:** ws://localhost:8000/ws/audio/<session\_id>/
* **Protocol:** WebSocket
* **Description:** Clients connect to this endpoint, providing a unique session\_id in the URL path.
  + **Client to Server:** Send binary audio chunks (ArrayBuffer or Blob) from the microphone. Can also send JSON messages to set speaker\_type.
  + **Server to Client:** Receives JSON messages containing transcription updates and keyword alerts.

### **Client-to-Server Messages (Examples)**

1. **Audio Chunk (Binary):**
   * Sent directly as ArrayBuffer or Blob from MediaRecorder.
   * Example (JavaScript): ws.send(event.data);
2. **Set Speaker Type (JSON):**
   * Sent as a JSON string.
   * Example (JavaScript):  
     ws.send(JSON.stringify({  
      type: 'set\_speaker\_type',  
      speaker\_type: 'agent' // or 'prospect', 'unknown'  
     }));

### **Server-to-Client Messages (Examples)**

1. **Transcription Update:**
   * type: transcription
   * data:
     + text: (String) The transcribed text segment.
     + speaker\_type: (String) The type of speaker ('prospect', 'agent', 'unknown').
     + timestamp: (String) Timestamp of the transcription.
   * Example:  
     {  
      "type": "transcription",  
      "data": {  
      "text": "Hello, this is a test transcription.",  
      "speaker\_type": "prospect",  
      "timestamp": "2025-07-08T12:30:45.123Z"  
      }  
     }
2. **Keyword Alert:**
   * type: alert
   * data:
     + keyword: (String) The detected keyword.
     + talking\_point: (String) The associated talking point for the keyword.
     + full\_text: (String) The full transcribed text where the keyword was found.
     + speaker\_type: (String) The type of speaker who said the keyword.
   * Example:  
     {  
      "type": "alert",  
      "data": {  
      "keyword": "pricing",  
      "talking\_point": "Discuss competitive pricing strategies.",  
      "full\_text": "So, about your pricing, how does that work?",  
      "speaker\_type": "prospect"  
      }  
     }
3. **Self-Test Response (for internal testing):**
   * type: self\_test\_response
   * data: (String) A message confirming the consumer's self-test.
   * Example:  
     {  
      "type": "self\_test\_response",  
      "data": "Consumer self-test message received!"  
     }