Fixing Groq Whisper API 404 Error

Problem Analysis

Your error shows:

- Frontend is calling `127.0.0.1:8000/api/transcribe` (404 Not Found)
- Backend has `/api/upload-audio` endpoint (not `/api/transcribe`)
- The backend is supposed to handle transcription automatically after upload

Solution: Update Frontend to Match Backend Architecture

Option 1: Fix Frontend to Use Correct Endpoint (Recommended)

The backend already handles transcription in the background worker. Update your `AudioRecorder.tsx`:

Current Problematic Code

```
// AudioRecorder.tsx - WRONG
const triggerTranscription = async (audioId: string) => {
const response = await fetch(${API_URL}/api/transcribe, {
method: 'POST',
body: JSON.stringify({ audio_id: audioId })
});
// This endpoint doesn't exist!
}
```

Fixed Code

```
// AudioRecorder.tsx - CORRECT
import { useState, useRef } from 'react';
import axios from 'axios';

const API_URL = import.meta.env.VITE_API_URL || 'http://127.0.0.1:8000';

export default function AudioRecorder({ userId }: { userId: string }) {
const [isRecording, setIsRecording] = useState(false);
const [audioBlob, setAudioBlob] = useState<Blob | null>(null);
const [status, setStatus] = useState<string>('');
const [progress, setProgress] = useState<string>('');

const mediaRecorderRef = useRef<MediaRecorder | null>(null);
const chunksRef = useRef<Blob[]>([]);

const startRecording = async () => {
try {
const stream = await navigator.mediaDevices.getUserMedia({
audio: {
```

echoCancellation: true,
noiseSuppression: true,
autoGainControl: true,
sampleRate: 44100
}
});

```
    const mediaRecorder = new MediaRecorder(stream, {
      mimeType: 'audio/webm;codecs=opus'
    });

    mediaRecorderRef.current = mediaRecorder;
    chunksRef.current = [];

    mediaRecorder.ondataavailable = (e) => {
      if (e.data.size > 0) {
        chunksRef.current.push(e.data);
      }
    };

    mediaRecorder.onstop = () => {
      const blob = new Blob(chunksRef.current, { type: 'audio/webm' });
      setAudioBlob(blob);
      stream.getTracks().forEach(track => track.stop());
    };

    mediaRecorder.start(100);
    setIsRecording(true);
    setStatus('recording');
  } catch (error) {
    console.error('Microphone error:', error);
    setStatus('error');
    setProgress('Unable to access microphone');
  }
```

};

const stopRecording = () => {
if (mediaRecorderRef.current && isRecording) {
mediaRecorderRef.current.stop();
setIsRecording(false);

setStatus('stopped');
}
};

const uploadAudio = async () => {
if (!audioBlob) return;

```
setStatus('uploading');
setProgress('Uploading audio...');

const formData = new FormData();
formData.append('file', audioBlob, 'recording.webm');

try {
  // Step 1: Upload audio - backend handles transcription automatically
  const response = await axios.post(
    `${API_URL}/api/upload-audio`,
    formData,
    {
      params: { user_id: userId },
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    }
  );

  const { audio_id, job_id } = response.data;
  console.log('Upload successful:', { audio_id, job_id });

  setProgress('Transcribing with Groq Whisper...');

  // Step 2: Poll for completion (backend processes in background)
  await pollTranscriptionStatus(audio_id);

} catch (error) {
  console.error('Upload failed:', error);
  setStatus('error');
  setProgress(`Upload failed: ${error.response?.data?.detail || error.message}`);
}
```

};

const pollTranscriptionStatus = async (audioId: string) => {
const maxAttempts = 60; // 2 minutes max
let attempts = 0;

```
const checkStatus = async (): Promise<boolean> => {
  try {
    const response = await axios.get(
      `${API_URL}/api/audio/${audioId}/status`
    );

    const { status: audioStatus } = response.data;
    console.log(`Status check ${attempts + 1}:`, audioStatus);

    if (audioStatus === 'completed') {
      setStatus('completed');
      setProgress('Plan generated! ✓');
      return true;
    } else if (audioStatus === 'failed') {
      setStatus('error');
      setProgress('Processing failed. Please try again.');
      return true;
    } else {
      // Still processing
      setProgress(`Processing... (${audioStatus})`);
    }

    attempts++;
    if (attempts >= maxAttempts) {
      setStatus('error');
      setProgress('Processing timeout. Please try again.');
      return true;
    }

    return false;
  } catch (error) {
    console.error('Status check failed:', error);
    attempts++;
    if (attempts >= maxAttempts) {
      setStatus('error');
      setProgress('Status check failed');
      return true;
    }
    return false;
  }
};

// Poll every 2 seconds
while (attempts < maxAttempts) {
  const done = await checkStatus();
  if (done) break;
  await new Promise(resolve => setTimeout(resolve, 2000));
}
```

```
};

return (
<div className="flex flex-col items-center gap-6 p-8">
{/* Recording Button */}
<button
onClick={isRecording ? stopRecording : startRecording}
disabled={status === 'uploading'}
className={ w-32 h-32 rounded-full font-bold text-white text-lg transition-all duration-300
shadow-lg ${isRecording ? 'bg-red-500 hover:bg-red-600 animate-pulse' : 'bg-blue-500
hover:bg-blue-600' } ${status === 'uploading' ? 'opacity-50 cursor-not-allowed' : ''} }
>
{isRecording ? '□ Stop' : '□ Record'}
</button>
```

```
  {/* Status */}
  {progress && (
    <div className="text-center">
      <p className="text-gray-700 font-medium">{progress}</p>
    </div>
  )}

  {/* Upload Button */}
  {audioBlob && status === 'stopped' && (
    <button
```

```
      onClick={uploadAudio}
      className="px-8 py-3 bg-green-500 hover:bg-green-600 text-white rounded-
    >
      Upload & Process
    </button>
  )}
</div>
```

);
}

Option 2: Verify Backend Endpoints

Make sure your backend/main.py has these endpoints:

backend/main.py

from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from supabase import create_client
from redis import Redis
from rq import Queue
import os
import uuid

app = FastAPI()

CORS Configuration

app.add_middleware(
CORSMiddleware,
allow_origins=[
"http://localhost:5173", # Vite dev server
"http://127.0.0.1:5173",
"https://your-app.vercel.app"
],
allow_credentials=True,
allow_methods=[""*],*
allow_headers=[""],
)

Initialize services

supabase = create_client(
os.getenv("SUPABASE_URL"),
os.getenv("SUPABASE_KEY")
)
redis_conn = Redis.from_url(os.getenv("REDIS_URL", "redis://localhost:6379"))
q = Queue(connection=redis_conn)

@app.get("/")
async def root():
return {"status": "ok", "message": "Mental Health Assistant API"}

@app.post("/api/upload-audio")
async def upload_audio(
file: UploadFile = File(...),
user_id: str = None
):
"""Upload audio and queue for Groq transcription"""
try:
# Generate unique ID
file_id = str(uuid.uuid4())
file_path = f"audio/{user_id}/{file_id}.webm"

```
    # Read file
    content = await file.read()
    print(f"Received audio file: {len(content)} bytes")

    # Upload to Supabase Storage
    storage_response = supabase.storage.from_("audio-files").upload(
        file_path,
        content,
        {"content-type": file.content_type or "audio/webm"}
    )

    print(f"Uploaded to storage: {file_path}")

    # Create database record
    audio_log = supabase.table("audio_logs").insert({
        "id": file_id,
        "user_id": user_id,
        "storage_path": file_path,
        "status": "uploaded"
    }).execute()

    print(f"Created audio log: {file_id}")

    # Queue transcription job
    job = q.enqueue(
        "worker.process_audio_groq",
        audio_log.data[0]["id"],
        job_timeout=300
    )

    print(f"Queued job: {job.id}")

    return {
        "audio_id": audio_log.data[0]["id"],
        "status": "queued",
        "job_id": job.id
    }

except Exception as e:
    print(f"Upload error: {str(e)}")
    raise HTTPException(status_code=500, detail=str(e))
```

@app.get("/api/audio/{audio_id}/status")
async def get_audio_status(audio_id: str):
"""Check transcription status"""
try:
audio = supabase.table("audio_logs")
.select("*")
.eq("id", audio_id)
.single()
.execute()

```
        if not audio.data:
            raise HTTPException(status_code=404, detail="Audio not found")

        return {
            "status": audio.data["status"],
            "audio_id": audio_id
        }
```

```
except Exception as e:
    print(f"Status check error: {str(e)}")
    raise HTTPException(status_code=500, detail=str(e))
```

Test endpoint for Groq API

@app.get("/api/test-groq")
async def test_groq():
"""Test Groq API connection"""
try:
from groq import Groq
client = Groq(api_key=os.getenv("GROQ_API_KEY"))

```
    # Test with a simple request
    models = client.models.list()

    return {
        "status": "success",
        "message": "Groq API connected",
        "models": [model.id for model in models.data]
    }
except Exception as e:
    return {
        "status": "error",
        "message": str(e)
    }
```

if **name** == "**main**":
import uvicorn
uvicorn.run(app, host="0.0.0.0", port=8000)

Step 3: Create/Update Worker File

Ensure your `backend/worker.py` handles Groq transcription:

backend/worker.py

import os
import tempfile
from supabase import create_client
from groq import Groq
from textblob import TextBlob
import requests

Initialize

supabase = create_client(
os.getenv("SUPABASE_URL"),
os.getenv("SUPABASE_KEY")
)

groq_client = Groq(api_key=os.getenv("GROQ_API_KEY"))

def process_audio_groq(audio_id: str):
"""Process audio with Groq Whisper API"""
print(f"Starting transcription for {audio_id}")

```
try:
    # Update status
    supabase.table("audio_logs").update({
        "status": "processing"
    }).eq("id", audio_id).execute()

    # Get audio record
    audio = supabase.table("audio_logs") \
        .select("*") \
        .eq("id", audio_id) \
        .single() \
        .execute()

    storage_path = audio.data["storage_path"]
    print(f"Downloading from: {storage_path}")

    # Download audio
    audio_data = supabase.storage \
        .from_("audio-files") \
        .download(storage_path)
```

```
# Save to temp file
with tempfile.NamedTemporaryFile(delete=False, suffix=".webm") as tmp_file
    tmp_file.write(audio_data)
    tmp_path = tmp_file.name

print(f"Temp file created: {tmp_path}")

# Transcribe with Groq
with open(tmp_path, "rb") as file:
    transcription = groq_client.audio.transcriptions.create(
        file=(tmp_path, file.read()),
        model="whisper-large-v3",
        response_format="json",
        language="en"
    )

transcript_text = transcription.text
print(f"Transcription complete: {len(transcript_text)} chars")

# Simple mood analysis
blob = TextBlob(transcript_text)
mood_vector = {
    "polarity": blob.sentiment.polarity,
    "subjectivity": blob.sentiment.subjectivity
}

# Save transcript
transcript_record = supabase.table("transcripts").insert({
    "audio_id": audio_id,
    "text": transcript_text,
    "mood_vector": mood_vector
}).execute()

print(f"Transcript saved: {transcript_record.data[0]['id']}")

# Update status to completed
supabase.table("audio_logs").update({
```

```
            "status": "completed"
        }).eq("id", audio_id).execute()

        print(f"Processing complete for {audio_id}")

        # Cleanup
        os.remove(tmp_path)

        return {"status": "success", "transcript_id": transcript_record.data[0]['id']}

    except Exception as e:
        print(f"Processing error for {audio_id}: {str(e)}")

        # Update status to failed
        supabase.table("audio_logs").update({
            "status": "failed",
            "error_message": str(e)
        }).eq("id", audio_id).execute()

        raise e
```

Testing Steps

1. Test Backend Endpoints

Start backend

cd backend
python3 -m uvicorn main:app --reload --host 0.0.0.0 --port 8000

In another terminal, test Groq connection

curl http://127.0.0.1:8000/api/test-groq

Should return:

{"status": "success", "message": "Groq API connected", "models": [...]}

2. Start Redis and Worker

Terminal 1: Start Redis

redis-server

Terminal 2: Start RQ worker

cd backend
python3 -m rq worker --with-scheduler

3. Test Frontend

cd frontend
npm run dev

Open browser to http://localhost:5173

Try recording and uploading

4. Check Logs

Monitor all three terminals:

- **Backend**: Should show upload request
- **Worker**: Should show transcription progress
- **Browser console**: Should show status updates

Common Issues & Solutions

Issue 1: Groq API Key Invalid

Verify your .env file

cat backend/.env | grep GROQ_API_KEY

Test manually

python3

> > > from groq import Groq
> > > client = Groq(api_key="your_key_here")
> > > client.models.list()

Issue 2: CORS Error

Update `backend/main.py` CORS origins to include your frontend URL:

allow_origins=[
"http://localhost:5173",
"http://127.0.0.1:5173",
"http://localhost:3000", # Add if needed
]

Issue 3: Redis Not Running

Check if Redis is running

redis-cli ping

Should return: PONG

If not, start Redis

redis-server

Issue 4: Supabase Storage Bucket Missing

Create the `audio-files` bucket in Supabase:

1. Go to Supabase Dashboard → Storage
2. Create new bucket: `audio-files`
3. Set to **Public** or configure policies
4. Test upload manually

Issue 5: Worker Not Processing Jobs

Check RQ dashboard

rq info

Should show:

- Worker count

- Queue jobs

- Failed jobs

If jobs stuck, flush and restart

```
redis-cli FLUSHALL
python3 -m rq worker --with-scheduler
```

Environment Variables Checklist

Create `backend/.env`:

Supabase

```
SUPABASE_URL=https://xxxxx.supabase.co
SUPABASE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Groq

```
GROQ_API_KEY=gsk_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Redis

```
REDIS_URL=redis://localhost:6379
```

Optional (for later)

```
AGENT_ROUTER_API_KEY=ar_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
TWILIO_ACCOUNT_SID=ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
TWILIO_AUTH_TOKEN=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Create `frontend/.env`:

```
VITE_API_URL=http://127.0.0.1:8000
VITE_SUPABASE_URL=https://xxxxx.supabase.co
VITE_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Quick Debugging Script

Save this as backend/test_groq.py:

import os
from groq import Groq
from dotenv import load_dotenv

load_dotenv()

def test_groq():
"""Test Groq API with a sample audio file"""
api_key = os.getenv("GROQ_API_KEY")

```
if not api_key:
    print("✖ GROQ_API_KEY not found in .env")
    return

print(f"✓ API Key found: {api_key[:10]}...")

try:
    client = Groq(api_key=api_key)

    # List available models
    models = client.models.list()
    print(f"✓ Connected to Groq API")
    print(f"✓ Available models: {[m.id for m in models.data]}")

    # Test transcription with a test file (if you have one)
    # Uncomment when you have a test audio file
    # with open("test.webm", "rb") as file:
    #     transcription = client.audio.transcriptions.create(
    #         file=("test.webm", file.read()),
    #         model="whisper-large-v3"
    #     )
    #     print(f"✓ Transcription test: {transcription.text}")

except Exception as e:
    print(f"✖ Error: {str(e)}")
```

if **name** == "**main**":
test_groq()

Run it:

```
cd backend
python3 test_groq.py
```

Summary

The 404 error occurs because:

1. ❌ Frontend calls `/api/transcribe` (doesn't exist)
2. ✅ Backend has `/api/upload-audio` (correct endpoint)

Fix: Update `AudioRecorder.tsx` to use `/api/upload-audio` and poll `/api/audio/{id}/status` for completion.

The backend worker handles Groq transcription automatically in the background - no separate transcribe endpoint needed!

Fixing Groq Whisper API 404 Error

Problem Analysis

Your error shows:

- Frontend is calling `127.0.0.1:8000/api/transcribe` (404 Not Found)
- Backend has `/api/upload-audio` endpoint (not `/api/transcribe`)
- The backend is supposed to handle transcription automatically after upload

Solution: Update Frontend to Match Backend Architecture

Option 1: Fix Frontend to Use Correct Endpoint (Recommended)

The backend already handles transcription in the background worker. Update your `AudioRecorder.tsx`:

Current Problematic Code

```
// AudioRecorder.tsx - WRONG
const triggerTranscription = async (audioId: string) => {
const response = await fetch(${API_URL}/api/transcribe, {
method: 'POST',
body: JSON.stringify({ audio_id: audioId })
});
// This endpoint doesn't exist!
}
```

Fixed Code

```
// AudioRecorder.tsx - CORRECT
import { useState, useRef } from 'react';
import axios from 'axios';

const API_URL = import.meta.env.VITE_API_URL || 'http://127.0.0.1:8000';

export default function AudioRecorder({ userId }: { userId: string }) {
const [isRecording, setIsRecording] = useState(false);
const [audioBlob, setAudioBlob] = useState<Blob | null>(null);
const [status, setStatus] = useState<string>('');
const [progress, setProgress] = useState<string>('');

const mediaRecorderRef = useRef<MediaRecorder | null>(null);
const chunksRef = useRef<Blob[]>([]);

const startRecording = async () => {
try {
const stream = await navigator.mediaDevices.getUserMedia({
audio: {
echoCancellation: true,
noiseSuppression: true,
autoGainControl: true,
sampleRate: 44100
}
});
```

```
  const mediaRecorder = new MediaRecorder(stream, {
    mimeType: 'audio/webm;codecs=opus'
  });

  mediaRecorderRef.current = mediaRecorder;
  chunksRef.current = [];

  mediaRecorder.ondataavailable = (e) => {
    if (e.data.size > 0) {
      chunksRef.current.push(e.data);
    }
  };

  mediaRecorder.onstop = () => {
    const blob = new Blob(chunksRef.current, { type: 'audio/webm' });
    setAudioBlob(blob);
    stream.getTracks().forEach(track => track.stop());
  };

  mediaRecorder.start(100);
  setIsRecording(true);
  setStatus('recording');
} catch (error) {
  console.error('Microphone error:', error);
  setStatus('error');
  setProgress('Unable to access microphone');
}
```

};

const stopRecording = () => {
if (mediaRecorderRef.current && isRecording) {
mediaRecorderRef.current.stop();
setIsRecording(false);
setStatus('stopped');
}
};

const uploadAudio = async () => {
if (!audioBlob) return;

```
  setStatus('uploading');
  setProgress('Uploading audio...');

  const formData = new FormData();
  formData.append('file', audioBlob, 'recording.webm');

  try {
    // Step 1: Upload audio - backend handles transcription automatically
    const response = await axios.post(
      `${API_URL}/api/upload-audio`,
      formData,
      {
        params: { user_id: userId },
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      }
    );

    const { audio_id, job_id } = response.data;
    console.log('Upload successful:', { audio_id, job_id });

    setProgress('Transcribing with Groq Whisper...');

    // Step 2: Poll for completion (backend processes in background)
    await pollTranscriptionStatus(audio_id);

  } catch (error) {
    console.error('Upload failed:', error);
    setStatus('error');
    setProgress(`Upload failed: ${error.response?.data?.detail || error.message}`);
  }
```

};

const pollTranscriptionStatus = async (audioId: string) => {
const maxAttempts = 60; // 2 minutes max
let attempts = 0;

```
  const checkStatus = async (): Promise<boolean> => {
    try {
      const response = await axios.get(
        `${API_URL}/api/audio/${audioId}/status`
      );

      const { status: audioStatus } = response.data;
      console.log(`Status check ${attempts + 1}:`, audioStatus);

      if (audioStatus === 'completed') {
        setStatus('completed');
        setProgress('Plan generated! ✓');
        return true;
      } else if (audioStatus === 'failed') {
        setStatus('error');
        setProgress('Processing failed. Please try again.');
        return true;
      } else {
        // Still processing
        setProgress(`Processing... (${audioStatus})`);
      }

      attempts++;
      if (attempts >= maxAttempts) {
        setStatus('error');
        setProgress('Processing timeout. Please try again.');
        return true;
      }

      return false;
    } catch (error) {
      console.error('Status check failed:', error);
      attempts++;
      if (attempts >= maxAttempts) {
        setStatus('error');
        setProgress('Status check failed');
        return true;
      }
      return false;
    }
  };

  // Poll every 2 seconds
  while (attempts < maxAttempts) {
    const done = await checkStatus();
    if (done) break;
    await new Promise(resolve => setTimeout(resolve, 2000));
  }
```

};

return (
<div className="flex flex-col items-center gap-6 p-8">
{/* Recording Button */}
<button
onClick={isRecording ? stopRecording : startRecording}
disabled={status === 'uploading'}

```
className={ w-32 h-32 rounded-full font-bold text-white text-lg transition-all duration-300
shadow-lg ${isRecording ? 'bg-red-500 hover:bg-red-600 animate-pulse' : 'bg-blue-500
hover:bg-blue-600' } ${status === 'uploading' ? 'opacity-50 cursor-not-allowed' : ''} }
>
{isRecording ? '□ Stop' : '□ Record'}
</button>
```

```
    {/* Status */}
    {progress && (
      <div className="text-center">
        <p className="text-gray-700 font-medium">{progress}</p>
      </div>
    )}

    {/* Upload Button */}
    {audioBlob && status === 'stopped' && (
      <button
        onClick={uploadAudio}
        className="px-8 py-3 bg-green-500 hover:bg-green-600 text-white rounded-
      >
        Upload & Process
      </button>
    )}
  </div>
```

```
);
}
```

Option 2: Verify Backend Endpoints

Make sure your backend/main.py has these endpoints:

backend/main.py

```
from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from supabase import create_client
from redis import Redis
from rq import Queue
import os
import uuid

app = FastAPI()
```

CORS Configuration

app.add_middleware(
CORSMiddleware,
allow_origins=[
"http://localhost:5173", # Vite dev server
"http://127.0.0.1:5173",
"https://your-app.vercel.app"
],
allow_credentials=True,
allow_methods=["*"],
allow_headers=["*"],
)

Initialize services

supabase = create_client(
os.getenv("SUPABASE_URL"),
os.getenv("SUPABASE_KEY")
)
redis_conn = Redis.from_url(os.getenv("REDIS_URL", "redis://localhost:6379"))
q = Queue(connection=redis_conn)

@app.get("/")
async def root():
return {"status": "ok", "message": "Mental Health Assistant API"}

@app.post("/api/upload-audio")
async def upload_audio(
file: UploadFile = File(...),
user_id: str = None
):
"""Upload audio and queue for Groq transcription"""
try:
# Generate unique ID
file_id = str(uuid.uuid4())
file_path = f"audio/{user_id}/{file_id}.webm"

```
    # Read file
    content = await file.read()
    print(f"Received audio file: {len(content)} bytes")

    # Upload to Supabase Storage
    storage_response = supabase.storage.from_("audio-files").upload(
        file_path,
        content,
        {"content-type": file.content_type or "audio/webm"}
    )

    print(f"Uploaded to storage: {file_path}")

    # Create database record
    audio_log = supabase.table("audio_logs").insert({
        "id": file_id,
        "user_id": user_id,
        "storage_path": file_path,
        "status": "uploaded"
    }).execute()

    print(f"Created audio log: {file_id}")

    # Queue transcription job
    job = q.enqueue(
        "worker.process_audio_groq",
        audio_log.data[0]["id"],
        job_timeout=300
    )

    print(f"Queued job: {job.id}")

    return {
        "audio_id": audio_log.data[0]["id"],
        "status": "queued",
        "job_id": job.id
    }

except Exception as e:
    print(f"Upload error: {str(e)}")
    raise HTTPException(status_code=500, detail=str(e))
```

@app.get("/api/audio/{audio_id}/status")
async def get_audio_status(audio_id: str):
"""Check transcription status"""
try:
audio = supabase.table("audio_logs")

.select("*")
.eq("id", audio_id)
.single()
.execute()

```
    if not audio.data:
        raise HTTPException(status_code=404, detail="Audio not found")

    return {
        "status": audio.data["status"],
        "audio_id": audio_id
    }

except Exception as e:
    print(f"Status check error: {str(e)}")
    raise HTTPException(status_code=500, detail=str(e))
```

Test endpoint for Groq API

@app.get("/api/test-groq")
async def test_groq():
"""Test Groq API connection"""
try:
from groq import Groq
client = Groq(api_key=os.getenv("GROQ_API_KEY"))

```
    # Test with a simple request
    models = client.models.list()

    return {
        "status": "success",
        "message": "Groq API connected",
        "models": [model.id for model in models.data]
    }
except Exception as e:
    return {
        "status": "error",
        "message": str(e)
    }
```

if **name** == "**main**":
import uvicorn
uvicorn.run(app, host="0.0.0.0", port=8000)

Step 3: Create/Update Worker File

Ensure your backend/worker.py handles Groq transcription:

backend/worker.py

import os
import tempfile
from supabase import create_client
from groq import Groq
from textblob import TextBlob
import requests

Initialize

supabase = create_client(
os.getenv("SUPABASE_URL"),
os.getenv("SUPABASE_KEY")
)

groq_client = Groq(api_key=os.getenv("GROQ_API_KEY"))

def process_audio_groq(audio_id: str):
"""Process audio with Groq Whisper API"""
print(f"Starting transcription for {audio_id}")

```
try:
    # Update status
    supabase.table("audio_logs").update({
        "status": "processing"
    }).eq("id", audio_id).execute()

    # Get audio record
    audio = supabase.table("audio_logs") \
        .select("*") \
        .eq("id", audio_id) \
        .single() \
        .execute()

    storage_path = audio.data["storage_path"]
```

```
print(f"Downloading from: {storage_path}")

# Download audio
audio_data = supabase.storage \
    .from_("audio-files") \
    .download(storage_path)

# Save to temp file
with tempfile.NamedTemporaryFile(delete=False, suffix=".webm") as tmp_file
    tmp_file.write(audio_data)
    tmp_path = tmp_file.name

print(f"Temp file created: {tmp_path}")

# Transcribe with Groq
with open(tmp_path, "rb") as file:
    transcription = groq_client.audio.transcriptions.create(
        file=(tmp_path, file.read()),
        model="whisper-large-v3",
        response_format="json",
        language="en"
    )

transcript_text = transcription.text
print(f"Transcription complete: {len(transcript_text)} chars")

# Simple mood analysis
blob = TextBlob(transcript_text)
mood_vector = {
    "polarity": blob.sentiment.polarity,
    "subjectivity": blob.sentiment.subjectivity
}

# Save transcript
transcript_record = supabase.table("transcripts").insert({
    "audio_id": audio_id,
    "text": transcript_text,
    "mood_vector": mood_vector
```

```
        }).execute()

        print(f"Transcript saved: {transcript_record.data[0]['id']}")

        # Update status to completed
        supabase.table("audio_logs").update({
            "status": "completed"
        }).eq("id", audio_id).execute()

        print(f"Processing complete for {audio_id}")

        # Cleanup
        os.remove(tmp_path)

        return {"status": "success", "transcript_id": transcript_record.data[0]['id']}

    except Exception as e:
        print(f"Processing error for {audio_id}: {str(e)}")

        # Update status to failed
        supabase.table("audio_logs").update({
            "status": "failed",
            "error_message": str(e)
        }).eq("id", audio_id).execute()

        raise e
```

Testing Steps

1. Test Backend Endpoints

Start backend

cd backend
python3 -m uvicorn main:app --reload --host 0.0.0.0 --port 8000

In another terminal, test Groq connection

curl http://127.0.0.1:8000/api/test-groq

Should return:

{"status": "success", "message": "Groq API connected", "models": [...]}

2. Start Redis and Worker

Terminal 1: Start Redis

redis-server

Terminal 2: Start RQ worker

cd backend
python3 -m rq worker --with-scheduler

3. Test Frontend

cd frontend
npm run dev

Open browser to http://localhost:5173

Try recording and uploading

4. Check Logs

Monitor all three terminals:

- **Backend**: Should show upload request
- **Worker**: Should show transcription progress
- **Browser console**: Should show status updates

Common Issues & Solutions

Issue 1: Groq API Key Invalid

Verify your .env file

cat backend/.env | grep GROQ_API_KEY

Test manually

python3

> > > from groq import Groq
> > > client = Groq(api_key="your_key_here")
> > > client.models.list()

Issue 2: CORS Error

Update backend/main.py CORS origins to include your frontend URL:

allow_origins=[
"http://localhost:5173",
"http://127.0.0.1:5173",
"http://localhost:3000", # Add if needed
]

Issue 3: Redis Not Running

Check if Redis is running

redis-cli ping

Should return: PONG

If not, start Redis

redis-server

Issue 4: Supabase Storage Bucket Missing

Create the audio-files bucket in Supabase:

1. Go to Supabase Dashboard → Storage
2. Create new bucket: audio-files
3. Set to **Public** or configure policies
4. Test upload manually

Issue 5: Worker Not Processing Jobs

Check RQ dashboard

```
rq info
```

Should show:

- Worker count

- Queue jobs

- Failed jobs

If jobs stuck, flush and restart

```
redis-cli FLUSHALL
python3 -m rq worker --with-scheduler
```

Environment Variables Checklist

Create `backend/.env`:

Supabase

```
SUPABASE_URL=https://xxxxx.supabase.co
SUPABASE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Groq

```
GROQ_API_KEY=gsk_xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Redis

```
REDIS_URL=redis://localhost:6379
```

Optional (for later)

```
AGENT_ROUTER_API_KEY=ar_xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
TWILIO_ACCOUNT_SID=ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
TWILIO_AUTH_TOKEN=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Create `frontend/.env`:

VITE_API_URL=http://127.0.0.1:8000
VITE_SUPABASE_URL=https://xxxxx.supabase.co
VITE_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Quick Debugging Script

Save this as backend/test_groq.py:

import os
from groq import Groq
from dotenv import load_dotenv

load_dotenv()

def test_groq():
"""Test Groq API with a sample audio file"""
api_key = os.getenv("GROQ_API_KEY")

```
if not api_key:
    print("✗ GROQ_API_KEY not found in .env")
    return

print(f"✓ API Key found: {api_key[:10]}...")

try:
    client = Groq(api_key=api_key)

    # List available models
    models = client.models.list()
    print(f"✓ Connected to Groq API")
    print(f"✓ Available models: {[m.id for m in models.data]}")

    # Test transcription with a test file (if you have one)
    # Uncomment when you have a test audio file
    # with open("test.webm", "rb") as file:
    #     transcription = client.audio.transcriptions.create(
    #         file=("test.webm", file.read()),
    #         model="whisper-large-v3"
    #     )
    #     print(f"✓ Transcription test: {transcription.text}")
except Exception as e:
    print(f"❌ Error: {str(e)}")
```

if **name** == "**main**":
test_groq()

Run it:

cd backend
python3 test_groq.py

Summary

The 404 error occurs because:

1. ❌ Frontend calls `/api/transcribe` (doesn't exist)
2. ✅ Backend has `/api/upload-audio` (correct endpoint)

Fix: Update `AudioRecorder.tsx` to use `/api/upload-audio` and poll `/api/audio/{id}/status` for completion.

The backend worker handles Groq transcription automatically in the background - no separate transcribe endpoint needed!