HEISENBURGERS

Jaxon AI Voice Doorbell

Yousef Amirghofran, Anze Zgonc, Mia Dragovic, Borja Albers, Lucas Van Zyl

Problem

With the number of short-term rental properties like Airbnb's increasing rapidly and many hosts managing multiple properties, the entrance of people (guests, staff, etc.) and the logistical operations of the property (handling deliveries, taking messages, etc) has become difficult and there hasn't been an easy-to-use solution to fix this problem. Property managers have tried to handle this situation manually but it hasn't worked out.

Some of the problems that hosts and guests encounter are:

1. Keys getting lost or stolen
2. Hosts having to be physically present to hand over the keys
3. Previous guests accessing the property without booking
4. Delays in check-in for guests
5. Guests feeling unsafe in the property

Considering these challenges, there is a clear need in the market for a more advanced, secure, and user-friendly access control solution.

Purpose of the Project

Our project aims to address the limitations of traditional access control systems by developing an intelligent, voice-activated door lock system. Our main objectives are:

1. We want to enhance security by using advanced authentication to make sure that only authorized guests can enter the property. This would reduce the risk associated with lost or stolen keys.
2. We want to provide a smooth and convenient method for users to unlock doors using voice commands, which removes the need to carry physical keys.
3. We want to implement a system that can monitor entry attempts in real-time and send notifications via email, which would improve situational awareness and security management.

4. We want to create a system that integrates with existing smart home appliances, using AI technologies for decision-making, speech recognition, and database management for guest and access control.

Project Goals

1. Developing a voice-activated entry system that uses a USB microphone to capture user voice commands, and convert them to text using the OpenAI whisper-1 model.
2. Implementing AI-based authentication by using an AI model to process the transcribed speech and make decisions on whether to grant or deny access based on a database of authorized users.
3. Automating door-lock mechanism by controlling a solenoid lock via a relay, which is activated when the passcode is validated by the system.
4. Sending real-time notifications about entry attempts, successful entries, deliveries, and messages.
5. Designing a website using React and ShadcnUI components for administrators to manage the system, which includes adding or removing users, monitoring logs, and configuring settings.

Our Jaxon AI Voice Doorbell is designed to be a complete solution that addresses all the identified problems. It combines advanced security features with the convenience of artificial intelligence, tailored specifically for the needs of Airbnb properties.

Functionality of the circuit

The circuit is the physical implementation of our AI-powered smart lock system. It uses a USB microphone and speaker, connected to a Raspberry Pi to communicate with the user attempting to enter. From that communication, the Raspberry Pi uses an AI model to extract and validate the user's given password. Once the Raspberry Pi validates the password the lock contracts and the door can be opened.

The circuit consists of:

- **The inputs**: Button, which when pressed starts the passcode validation program. Passcode provided (Handled by Raspberry Pi, connected to a microphone)

- **The output**: A solenoid lock that is unlocked when the Raspberry Pi validates the user's password.
- A transistor and a relay are used to control the solenoid lock

I1(Power button)	**I2(Password validation)**	**O1(Lock)**
ON	Correct password	Unlocked
OFF	Incorrect password	Locked

Circuit design

The functionality of the circuit can be described by the following diagram, where the signal from the Raspberry Pi GPIO pin is amplified from 3.3V to 5V using an NPN BJT transistor. The amplified 5V signal is then used as an input to a relay, which controls whether or not the solenoid lock is unlocked, by acting as a switch connecting the lock to a 12V power supply.

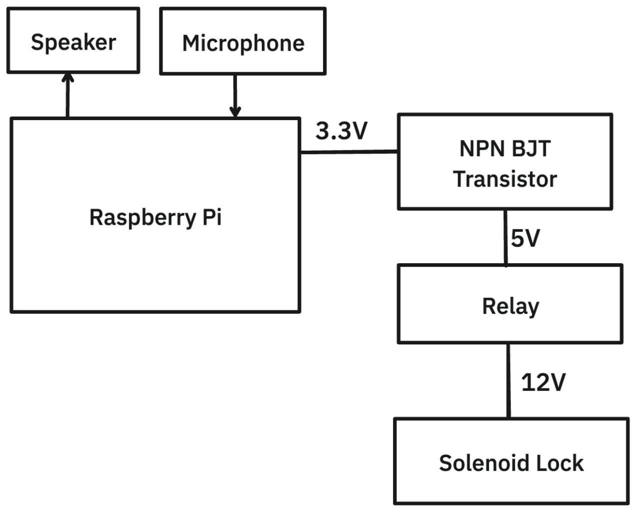


Figure 1: Functionality of the circuit

Break down each part of the circuit

Raspberry Pi

Figure 2: Raspberry Pi 4 Figure 3: Speaker and USB microphone

We are using a *Raspberry Pi 4 Model B*, as the control module for our circuit, allowing us to connect the circuit to the internet. Internet connection is important, as it allows us to implement IOT features, which are crucial to our solution. The Raspberry Pi is also connected to a USB microphone and a speaker, which allows it to communicate with the user. The pin GPIO 17 is connected to a button, which is used to run the password validation program when pressed. The pin GPIO 18, outputs a 3.3V signal to the circuit if the password from the user is correct, otherwise it does not output any voltage. Since the relay we are using to control the solenoid lock is a 5V relay, we have used a BJT transistor (2N222A) to amplify the signal from 3.3V to 5V.

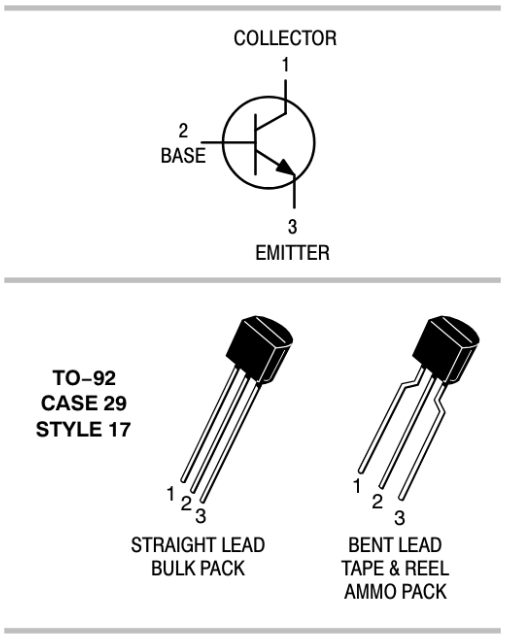


Figure 4: 2N222A BJT Transistor

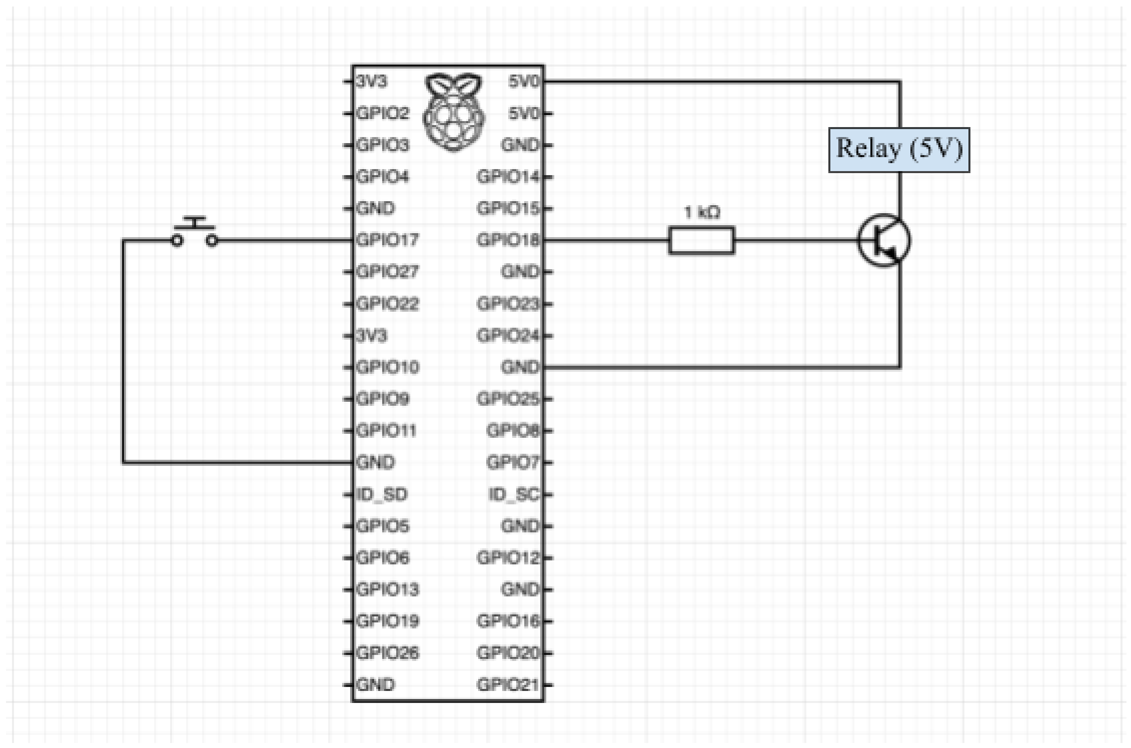


Figure 5: Raspberry Pi Subcircuit

Figure 5 shows the subcircuit described above which also includes a 1kΩ resistor that is there to protect the transistor and the Raspberry Pi pins from a high current, which could damage them. When the voltage at the base is high a current travels through the circuit and a current flows through the relay, which activates the rest of the circuit.

Solenoid lock:

Figure 6: Relay Figure 7: Solenoid lock

For our locking mechanism, we are using a solenoid lock (12V/350mA Solenoid lock), which is controlled by a relay connected to the Raspberry Pi. The relay we are using is a

5V relay (KF301 5V 1-Channel Relay), so when it receives a voltage, it will connect the solenoid lock to a higher voltage source, consequently unlocking the lock.

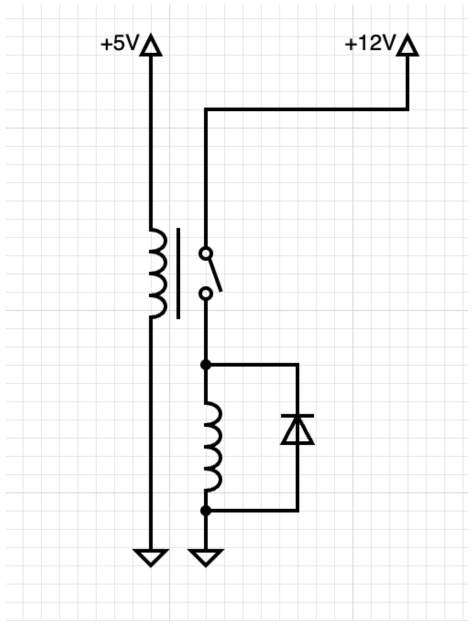
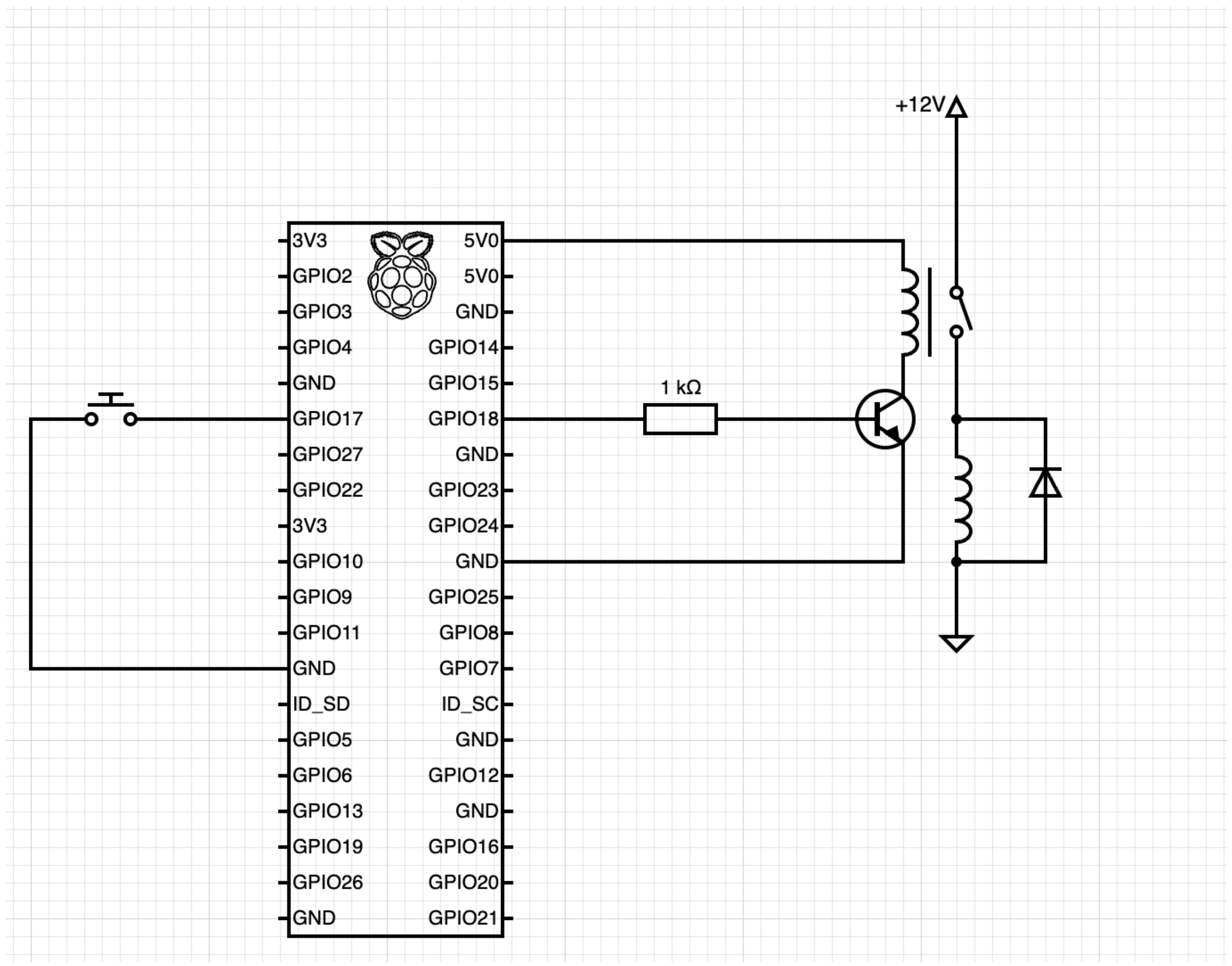


Figure 8: Solenoid lock subcircuit

Figure 8 shows the circuit described above, with the addition of a flyback diode to prevent voltage spikes as a result of back emf, when the circuit is quickly turned off.

Final circuit

Figure 9: Entire circuit



In Figure 9 the positive end of the 12V power supply is modeled by the 12V voltage source and the negative end is modeled by the ground at the end of the circuit.

Components:

- Raspberry Pi 4 Model B
- 1kΩ Resistor
- 2N222A Transistor
- KF301 5V 1-Channel Relay
- Breadboard
- Cables
- 12V power supply
- AUX Speaker
- USB Microphone

Software

The software part of our project comprises the following parts:

- A frontend website built with ReactJS
- An SQL database built with Supabase
- OpenAI Whisper model for speech-to-text
- Llama3 running on Groq for LLM interactions
- Resend for email-sending.

We used React as the framework to build our website that would act as the dashboard for the property owner. Through this dashboard, they can add/remove/edit guests and staff members, and see their notifications like guest entrances, deliveries, and messages.

We used Supabase as the SQL database for our project which would store our guests' information and notifications. The Python script would then query this database to validate passcodes/one-time codes, and send the notifications for messages and deliveries.

We used the OpenAI Whisper model through their provided Python SDK to transcribe the user's voice to text for analysis. The Python script recorded their audio for as long as they spoke, with a mechanism that stopped after they stopped speaking. The audio would then be sent to the OpenAI SDK to be transcribed and it would return the transcribed text. The recorded audio would then be deleted to not occupy unnecessary storage space.

We used Meta's recently released open-source generative AI model called Llama3. We used Groq as our inference engine (which did the operations) because of its industry-leading speed and free cost. We used this LLM for operations such as classifying the intent of the user at the door from their transcribed response (e.g. to enter, deliver a package, or leave a message) and extracting the information we are interested in from their responses (e..g their passcode, their message, and what they are delivering).

Finally, we used Resend Python SDK for sending one-time codes to the users' emails.

Validation

We have recorded a video showing how our product works (circuit + software) available to view at the following link:

amrgh.me/physics-project-video

Some safety checks? Improvements?

We have made sure to protect our circuit components by using a resistor to limit current and a flyback diode to prevent any damage from the solenoid lock.

The product as a whole is fully functional and achieves its purpose without any issues like heating up or glitches. It is also not over-complicated. However, there is room to improve the circuit. For instance, we could add a capacitor to reduce the noise of the signal sent to the relay or use a MOSFET for better efficiency.

References

1. OpenAI. (n.d.). Whisper. GitHub. Retrieved May 23, 2024, from https://github.com/openai/whisper

2. Hugging Face. (n.d.). Meta-Llama-3-8B. Retrieved May 23, 2024, from https://huggingface.co/meta-llama/Meta-Llama-3-8B

3. Hugging Face. (n.d.). Meta-Llama-3-70B. Retrieved May 23, 2024, from https://huggingface.co/meta-llama/Meta-Llama-3-70B

4. Groq. (n.d.). Documentation. WOW. Retrieved May 23, 2024, from https://wow.groq.com/docs/

5. Resend. (n.d.). Retrieved May 23, 2024, from https://resend.com

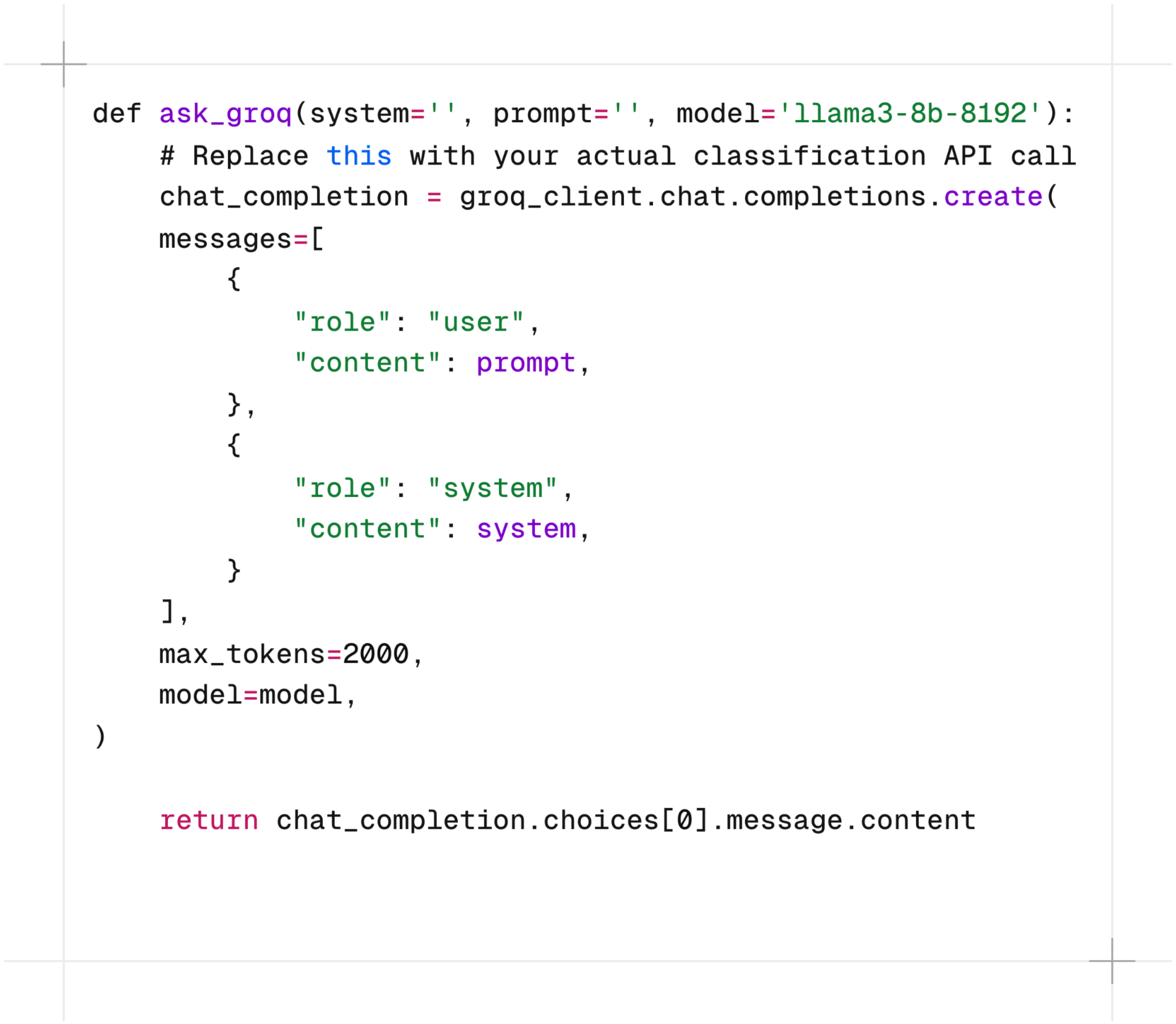
6. Groq. (2024, March 21). Groq tensor streaming processor deep dive from https://wow.groq.com/docs/

7. How To Control A Solenoid With A Raspberry Pi Using a Relay, from https://www.youtube.com/watch?v=BVMeVGET_Ak&t=210s

8. Amplifier Transistors from https://www.onsemi.com/pdf/datasheet/p2n2222a-d.pdf

Appendix

Here are some of the code snippets from the most important parts of the software. The entire GitHub repository can be found at: amrgh.me/physics-project-GitHub



```python
def ask_groq(system='', prompt='', model='llama3-8b-8192'):
    # Replace this with your actual classification API call
    chat_completion = groq_client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": prompt,
        },
        {
            "role": "system",
            "content": system,
        }
    ],
    max_tokens=2000,
    model=model,
)

    return chat_completion.choices[0].message.content
```

```python
# Parameters
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
FRAME_DURATION_MS = 30  # Duration of a frame in milliseconds
FRAMES_PER_BUFFER = int(RATE * FRAME_DURATION_MS / 1000)
VAD_MODE = 2  # Aggressiveness mode of the VAD (0-3)
SILENCE_LIMIT = 80  # Number of silent frames to keep recording after silence is detected

# Voice Activity Detector
vad = webrtcvad.Vad(VAD_MODE)

def is_speech(frame, vad):
    return vad.is_speech(frame, RATE)

def record_audio(filename, duration=10):
    # Audio Stream
    audio = pyaudio.PyAudio()
    stream = audio.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        frames_per_buffer=FRAMES_PER_BUFFER)
    #print("Recording...")

    frames = []
    ring_buffer = collections.deque(maxlen=SILENCE_LIMIT)
    silent_frames = 0
    recording = False
    start_time = time.time()

    while True:
        frame = stream.read(FRAMES_PER_BUFFER)
        is_speaking = is_speech(frame, vad)

        if is_speaking:
            if not recording:
                recording = True
                #print("Start speaking detected.")
            frames.append(frame)
            silent_frames = 0
        else:
            if recording:
                ring_buffer.append(frame)
                silent_frames += 1
                if silent_frames >= SILENCE_LIMIT:
                    #print("Silence detected, stopping recording.")
                    frames.extend(ring_buffer)
                    break
            else:
                ring_buffer.clear()

        # Check if the duration limit has been reached
        if duration and (time.time() - start_time) >= duration:
            if recording:
                frames.extend(ring_buffer)
            break

    #print("Recording stopped.")
    stream.stop_stream()
    stream.close()
    audio.terminate()

    # Save the recorded frames as a .wav file
    with wave.open(filename, 'wb') as wf:
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(audio.get_sample_size(FORMAT))
        wf.setframerate(RATE)
        wf.writeframes(b''.join(frames))

    #print(f"Audio saved as {filename}")
    return filename
```

```python
# Function to send audio to the API
def speech_to_text(file):
    #print("Sending to API...")
    audio_file = open(file, 'rb')
    transcription = client.audio.transcriptions.create(model="whisper-1", file=audio_file)
    #print(transcription.text)
    return transcription.text
```