```python
##  VOICE ASSISTANT  ##

import os
import requests
import cohere
import re
import pyttsx3
import streamlit as st
from bs4 import BeautifulSoup
from audio_recorder_streamlit import audio_recorder
from langchain.chains import RetrievalQA
from langchain_cohere import ChatCohere
from langchain_cohere import CohereEmbeddings
from langchain_community.vectorstores import DeepLake
from langchain.text_splitter import CharacterTextSplitter
from langchain_community.document_loaders import TextLoader

# constants
TEMP_AUDIO_PATH = "temp_audio.wav"
AUDIO_FORMAT = "audio/wav"

cohere_api_key = os.environ.get('COHERE_API_KEY')

my_activeloop_org_id = "ankur82garg"
my_activeloop_dataset_name = "langchain_course_jarvis_assistant"
dataset_path = f'hub://{my_activeloop_org_id}/{my_activeloop_dataset_name}'

embeddings = CohereEmbeddings(model="embed-multilingual-v3.0")

# Create a list of relative URLs
def get_documentation_urls():
    return [
        '/docs/huggingface_hub/guides/overview',
        '/docs/huggingface_hub/guides/download',
        '/docs/huggingface_hub/guides/upload',
        '/docs/huggingface_hub/guides/hf_file_system',
        '/docs/huggingface_hub/guides/repository',
        '/docs/huggingface_hub/guides/search',
    ]

def construct_full_url(base_url, relative_url):
    return base_url + relative_url

# Scrape page content
def scrape_page_content(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text,'html.parser')
    text = soup.body.text.strip()
    text = re.sub(r'[\x00-\x08\x0b-\x0c\x0e-\x1f\x7f-\xff]', '', text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip()

def scrape_all_content(base_url, relative_urls, filename):
    content = []
    for relative_url in relative_urls:
        full_url = construct_full_url(base_url, relative_url)
        scraped_content = scrape_page_content(full_url)
        content.append(scraped_content.rstrip('\n'))

    with open(filename, 'w', encoding='utf-8') as file:
        for item in content:
            file.write("%s\n" % item)

    return content

```

```python
# Loading and Splitting Texts
def load_docs(root_dir, filename):
    docs = []
    try:
        loader = TextLoader(os.path.join(root_dir, filename), encoding='utf8')
        docs.extend(loader.load_and_split())
    except Exception as e:
        pass
    return docs

def split_docs(docs):
    text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    return text_splitter.split_documents(docs)

# Define the main function
def main():
    base_url = 'https://huggingface.co'
    filename='content.txt'
    root_dir ='./'
    relative_urls = get_documentation_urls()
    content = scrape_all_content(base_url, relative_urls, filename)
    docs = load_docs(root_dir, filename)
    texts = split_docs(docs)
    db = DeepLake(dataset_path=dataset_path, embedding_function=embeddings)
    db.add_documents(texts)

# Call the main function if this script is being run as the main program
if __name__ == '__main__':
    main()

# Load Embeddings and Database
def load_embeddings_and_database(active_loop_data_set_path):
    embeddings = CohereEmbeddings(model = 'embed-english-v3.0')
    db = DeepLake(
        dataset_path=active_loop_data_set_path,
        read_only=True,
        embedding_function=embeddings
    )
    return db

# Transcribe audio using Whisper
def transcribe_audio(audio_file_path, cohere_api_key):
    cohere_api_key = cohere_api_key
    try:
        with open(audio_file_path, "rb") as audio_file:
            response = cohere.Audio.transcribe("whisper-1", audio_file)
        return response.get("text")
    except Exception as e:
        print(f"Error transcribing audio: {str(e)}")
        return None

# Record and transcribe audio
def record_and_transcribe_audio():
    audio_bytes = audio_recorder()
    transcription = None
    if audio_bytes:
        st.audio(audio_bytes, format=AUDIO_FORMAT)
        with open(TEMP_AUDIO_PATH, "wb") as f:
            f.write(audio_bytes)

        if st.button("Transcribe"):
            transcription = transcribe_audio(TEMP_AUDIO_PATH, cohere_api_key)
            os.remove(TEMP_AUDIO_PATH)
            display_transcription(transcription)
    return transcription

```

```python
# Display the transcription of the audio
def display_transcription(transcription):
    if transcription:
        st.write(f"Transcription: {transcription}")
        with open("audio_transcription.txt", "w+") as f:
            f.write(transcription)
    else:
        st.write("Error transcribing audio.")

# Get user input from Streamlit text input field
def get_user_input(transcription):
    return st.text_input("", value=transcription if transcription else "", key="input")

# Search the database for a response based on the user's query
def search_db(user_input, db):
    retriever = db.as_retriever()
    retriever.search_kwargs['distance_metric'] = 'cos'
    retriever.search_kwargs['fetch_k'] = 100
    retriever.search_kwargs['k'] = 4
    model = ChatCohere(model = 'command-r7b-12-2024',temperature =0)
    qa = RetrievalQA.from_llm(model, retriever=retriever, return_source_documents=True)
    response=  qa.invoke({'query': user_input})

    return response

# Text-to-Speech with pyttsx3
def text_to_speech_pyttsx3(text):
    # Initialize the TTS engine
    engine = pyttsx3.init()

    # Set properties (optional)
    engine.setProperty('rate', 150)  # Speed of speech
    engine.setProperty('volume', 1)  # Volume (0.0 to 1.0)

    # Save speech to a file
    audio_path = 'output_audio.mp3'
    engine.save_to_file(text, audio_path)
    engine.runAndWait()

    # Read the audio file and return it as a binary stream
    with open(audio_path, 'rb') as f:
        audio_bytes = f.read()

    # Delete the temporary file after reading
    os.remove(audio_path)

    return audio_bytes

# Display conversation history using streamlit messages
def display_conversation(history):
    for i in range(len(history["generated"])):
        # Display user messages
        st.chat_message("user").text(history["past"][i])

        # Display assistant responses
        st.chat_message("assistant").text(history["generated"][i])

        # Voice using pyttsx3 Text-to-Speech
        text = history["generated"][i]
        audio = text_to_speech_pyttsx3(text)
        st.audio(audio, format='audio/mp3')

# Main function to run the loop
def main():
    # Initialize Streamlit app with a title
    st.write("# JarvisBase 🤖")
```

```
198
199     # Load embeddings and the DeepLake database
200     db = load_embeddings_and_database(dataset_path)
201
202     # Record and transcribe audio
203     transcription = record_and_transcribe_audio()
204
205     # Get user input from text input or audio transcription
206     user_input = get_user_input(transcription)
207
208     # Initialize session state for generated responses and past messages
209     if "generated" not in st.session_state:
210         st.session_state.generated = ["I am ready to help you"]
211     if "past" not in st.session_state:
212         st.session_state.past = ["Hey there!"]
213
214     # Search the database for a response based on user input and update the session state
215     if user_input:
216         output = search_db(user_input, db)
217         st.session_state.past.append(user_input)
218         response = str(output["result"])
219         st.session_state.generated.append(response)
220
221     # Display conversation history using Streamlit messages
222     if st.session_state["generated"]:
223         display_conversation(st.session_state)
224
225 # Run the main function when the script is executed
226 if __name__ == "__main__":
227     main()
```