Overview of ResuWhisper AI

"ResuWhisper AI" is a Streamlit-based web application that allows users to build a resume by answering questions via voice recordings, uploaded audio, or text input. It uses Google's Gemini AI to transcribe and process responses, stores data in a MySQL database, and generates resumes in Word or PDF format based on templates suited to different career stages (Fresher, Intermediate, Veteran). The app includes user authentication, language selection, and a live resume editor.

1. Importing Libraries

The code begins by bringing in tools (libraries) we need to make the app work. Think of these as toolboxes with specific gadgets inside.

```
import streamlit as st
```

- What it does: `streamlit` is the main tool to build the web app. It lets us create buttons, text boxes, and display things on the screen easily. We nickname it `st` for short.

```
import google.generativeai as genai
```

- What it does: This connects us to Google's AI tools, which can understand voice or text and give smart answers. We call it `genai`.

```
import tempfile
```

- What it does: Helps create temporary files (like a scratch pad) to store audio recordings while we work with them.

```
import sounddevice as sd
```

- What it does: Records audio from your microphone. We nickname it `sd`.

```
import numpy as np
```

- What it does: A math tool to handle audio data as numbers. We call it `np` for convenience.

```
import wave
```

- What it does: Lets us save audio recordings as WAV files (a common audio format).

```
import threading
```

- What it does: Allows the app to do multiple things at once, like recording audio while showing the screen.

```
import time
```

- What it does: Gives us tools to pause or time things, like waiting during recording.

```
import os
```

- What it does: Helps us work with files on the computer, like checking if an audio file exists.

```
from pathlib import Path
```

- What it does: Makes handling file paths (locations) easy and works on any computer.

```
import base64
```

- What it does: Turns files (like PDFs) into text so we can download them in the app.

```
from docx import Document as DocxDocument
```

- What it does: Creates and edits Word documents. We rename it `DocxDocument` to avoid confusion.

```
from docx.shared import Pt, Inches, RGBColor
```

- What it does: Tools to set font sizes (`Pt`), margins (`Inches`), and colors (`RGBColor`) in Word docs.

```
from docx.oxml.ns import qn
```

- What it does: Helps tweak low-level Word document settings (like XML stuff).

```
from docx.enum.text import WD_PARAGRAPH_ALIGNMENT
```

- What it does: Controls text alignment in Word docs (left, center, right).

```
import docx2pdf
```

- What it does: Converts Word documents to PDFs.

```
from reportlab.lib.pagesizes import letter
```

- What it does: Sets the page size (like letter size) for PDFs.

```
from reportlab.pdfgen import canvas
```

- What it does: Draws content (text, shapes) onto PDFs.

```
from reportlab.lib.colors import black, HexColor
```

- What it does: Defines colors for PDFs (`black` is black, `HexColor` uses color codes).

```
from io import BytesIO
```

- What it does: Lets us create files in memory (not on disk) to work with them quickly.

---

2. Setting Up the Google AI API

This part connects the app to Google's AI brain.

```
Google_API_Key = "AIzaSyCw8tLyHeobBO65GGnkLUVCGSMLdg-HsBw"
```

- What it does: This is a key (like a password) to use Google's AI. You'd replace it with your own key.

```
genai.configure(api_key=Google_API_Key)
```

- What it does: Tells the `genai` tool to use this key to access the AI.

```
model = genai.GenerativeModel('gemini-1.5-flash')
```

- What it does: Picks a specific AI model (`gemini-1.5-flash`) to process our voice and text.

---

3. Defining Constants

These are lists and dictionaries we'll use later, like a recipe book for the app.

```
resume_templates = {
    "Fresher": "Ideal for 0-2 years of experience. Maximum 1
page.",
    "Intermediate": "Best for 3-7 years of experience. Maximum 1
page.",
    "Veteran": "For 7+ years of experience. Maximum of 2 pages."
}
```

- What it does: A dictionary with three resume styles based on experience level.

```
languages = [
    "English", "Hindi", "Konkani", "Kannada", "Dogri", "Bodo",
"Urdu", "Tamil",
```

```
    "Kashmiri", "Assamese", "Bengali", "Marathi", "Sindhi",
"Maithili",
    "Punjabi", "Malayalam", "Manipuri", "Telugu", "Sanskrit",
"Nepali",
    "Santali", "Gujarati", "Odia"
]
```

- What it does: A list of languages you can speak in when answering questions.

```
questions = [
    "What is your full name, age, address, phone number, email,
and LinkedIn or GitHub profile link (if any)?",
    "What are your career goals, key strengths, and professional
personality traits?",
    "What is your work experience? Include job titles, company
names, employment dates, responsibilities, and achievements.",
    "Tell us about the projects you have done. Include project
names, descriptions, and your contributions (optional).",
    "What is your educational background? Include degrees,
institutions, graduation years, and any relevant coursework or
honors.",
    "What are your hard skills (e.g., technical skills) and soft
skills (e.g., communication, teamwork)?",
    "What certifications do you have? Include the certification
name, issuing organization, and date received.",
    "What are your extracurricular activities or recognitions?
Include activities, organizations, dates, and achievements."
]
```

- What it does: Eight questions you'll answer to build your resume.

```
section_headers = [
    "📋 Personal Information",
    "💼 Professional Summary",
    "👔 Work Experience",
    "🚀 Projects",
    "🎓 Education",
    "🛠️ Skills",
    "🏆 Certifications",
    "🌟 Extracurricular Activities"
]
```

- What it does: Titles for each resume section, matching the questions.

---

4. Simulated User Database

This keeps track of users in the app.

```
if "users_db" not in st.session_state:
    st.session_state["users_db"] = {}
```

- What it does: Creates an empty dictionary to store usernames, passwords, and resume data. `st.session_state` keeps stuff saved while the app runs.

---

5. Setting Up Session State

This sets up variables the app needs to remember.

```
def init_session_state():
```

- What it does: Defines a function to set initial values.

```
defaults = {
     "page": "login",
     "authenticated": False,
     "username": None,
     "selected_language": None,
     "consent_given": False,
     "resume_template": None,
     "current_question_index": 0,
     "responses": {},
     "recording_state": False,
     "audio_file": None,
     "stop_event": None,
     "current_response": None,
```

- What it does: A dictionary with starting values:
- `"page"`: Starts at `"login"`.
- `"authenticated"`: `False` means not logged in yet.
- `"username"`: No user yet (`None`).
- `"selected_language"`: No language picked (`None`).
- `"consent_given"`: `False` means no permission yet.
- `"resume_template"`: No template chosen (`None`).
- `"current_question_index"`: Starts at question 1 (index 0).

- `"responses"`: Empty dictionary for answers.
- `"recording_state"`: `False` means not recording.
- `"audio_file"`: No audio file yet (`_required`).
- `"stop_event"`: No stop signal for recording (`None`).
- `"current_response"`: No answer yet (`None`).

```
    "resume_data": {
        "personal_info": {
            "full_name": "",
            "degree": "",
            "phone": "",
            "email": "",
            "linkedin": "",
            "github": "",
            "address": ""
        },
        "summary": "",
        "qualifications": [],
        "certifications": [],
        "skills": [],
        "experience": [],
        "projects": [],
        "positions": []
    },
```

- What it does: A nested dictionary to store resume info, all empty to start.

```
    "transcribed_once": {},
     "translated_questions": {}
}
```

- What it does:
- `"transcribed_once"`: Tracks which audio files we've already turned into text.
- `"translated_questions"`: Stores questions translated into other languages.

```
for key, value in defaults.items():
     if key not in st.session_state:
         st.session_state[key] = value
```

- What it does: Adds these values to `st.session_state` if they're not already there.

```
init_session_state()
```

- What it does: Runs the function to set everything up.

---

6. Recording Audio

This function captures your voice.

```
def record_audio(filename, stop_event: threading.Event,
samplerate=44100):
```

- What it does: Defines a function to record audio.
- `filename`: Where to save the audio.
- `stop_event`: A signal to stop recording.
- `samplerate`: How clear the audio is (44100 is standard).

```
audio_data = []
```

- What it does: An empty list to collect audio chunks.

```
def callback(indata, frames, time, status):
    if not stop_event.is_set():
        audio_data.append(indata.copy())
    else:
        raise sd.StopStream()
```

- What it does: A mini-function that grabs audio data while recording, stops if signaled.

```
try:
    with sd.InputStream(samplerate=samplerate, channels=1,
dtype=np.int16, callback=callback):
        st.write("🎤 Recording... Press 'Stop Recording' to
end.")
        while not stop_event.is_set():
            time.sleep(0.1)
```

- What it does: Starts recording, shows a message, and waits until stopped.

```
    st.write("✅ Recording Completed.")
    audio_data = np.concatenate(audio_data, axis=0)
```

- What it does: Says recording is done, combines audio chunks into one piece.

```
    with wave.open(filename, 'wb') as wf:
        wf.setnchannels(1)
        wf.setsampwidth(2)
        wf.setframerate(samplerate)
```

```
            wf.writeframes(audio_data.tobytes())
```

- What it does: Saves the audio as a WAV file with 1 channel, 2-byte width, and the set sample rate.

```
        if os.path.exists(filename) and os.path.getsize(filename)
> 0:
            st.success(f"✅ Recording saved to {filename}!")
        else:
            st.error("Audio file was not created or is empty")
```

- What it does: Checks if the file saved correctly and shows a success or error message.

```
    except sd.StopStream:
        pass
    except Exception as e:
        st.error(f"Recording error: {str(e)}")
```

- What it does: Handles errors, like stopping or other problems.

---

7. Talking to Google AI

This function asks the AI for help.

```
def get_gemini_response(input_msg, audio_path=None,
mime_type="audio/wav"):
```

- What it does: Sends a message or audio to the AI.
- `input_msg`: The text or question.
- `audio_path`: Audio file location (optional).
- `mime_type`: File type (default is WAV).

```
    try:
        if audio_path:
            with open(audio_path, "rb") as f:
                audio = genai.upload_file(audio_path,
mime_type=mime_type)
            response = model.generate_content([audio,
input_msg])
        else:
            response = model.generate_content([input_msg])
```

- What it does: If there's audio, uploads it and sends it with the message; otherwise, just sends the message.

```
return response.text if response else None
```

- What it does: Returns the AI's text answer or `None` if no answer.

```
except Exception as e:
     st.error(f"Error with Gemini: {str(e)}")
     return None
```

- What it does: Shows an error if something goes wrong and returns `None`.

---

8. Translating Questions

This turns questions into your chosen language.

```
def translate_questions(language):
```

- What it does: Defines a function to translate questions.

```
if language not in st.session_state["translated_questions"]:
```

- What it does: Checks if we've already translated for this language.

```
    prompt = f"""
     Translate the following 8 English questions into
{language}. Provide the output as a numbered list, with each
translation corresponding to the original question. Do not add
extra commentary or modify the questions beyond translation.

     1. What is your full name, age, address, phone number,
email, and LinkedIn or GitHub profile link (if any)?
     ... (and the other 7 questions)
     """
```

- What it does: Creates a message asking the AI to translate the 8 questions.

```
    response = get_gemini_response(prompt)
```

- What it does: Sends the request to the AI.

```
    if response:
         translated = [line.split(". ", 1)[1] for line in
response.strip().split("\n") if line.strip()]
         st.session_state["translated_questions"][language] =
translated[:8]
```

```
        else:
            st.session_state["translated_questions"][language] =
questions
```

- What it does: If the AI answers, splits the response into a list of 8 translated questions; if not, uses English questions.

```
    return st.session_state["translated_questions"][language]
```

- What it does: Returns the translated questions.

---

9. Processing Responses with AI

This turns your answers into resume-ready text.

```
def process_response_with_gemini(question_index, response):
```

- What it does: Takes your answer and the question number to process it.
- `question_index`: Which question (0-7).
- `response`: Your answer.

```
    prompts = [
        f"""
        **Your Goal**: Your goal is to meticulously extract and
enhance personal details...
        ... (long instructions for question 0)
        """,
        ... (similar prompts for questions 1-7)
    ]
```

- What it does: A list of 8 detailed instructions (prompts) for the AI, one for each question, telling it how to format your answer.

```
    prompt = prompts[question_index]
    result = get_gemini_response(prompt)
    return result
```

- What it does: Picks the right prompt, sends it to the AI with your answer, and returns the result.

---

10. Updating Resume Data

This saves your processed answers into the resume.

```
def update_resume_data(question_index, response):
```

- What it does: Updates the resume with your answer.

```
   result = process_response_with_gemini(question_index,
response)
    if not result:
        st.error("Failed to process response with Gemini")
        return
```

- What it does: Processes the answer; if it fails, shows an error and stops.

(For each question index, it parses the result differently)

- Question 0 (Personal Info):

```
   if question_index == 0:
        lines = result.split("\n")
        personal_info =
st.session_state["resume_data"]["personal_info"]
        for line in lines:
            if "Full Name:" in line:
                personal_info["full_name"] = line.split(": ")[1]
if len(line.split(": ")) > 1 else ""
            ... (similar for address, phone, email, linkedin,
github)
```

- What it does: Splits the AI's response into lines and fills in personal info fields.
- Question 1 (Summary):

```
   elif question_index == 1:
        for line in result.split("\n"):
            if "Enhanced Summary:" in line:
                summary = line.split(": ")[1] if
len(line.split(": ")) > 1 else ""
                st.session_state["resume_data"]["summary"] =
summary
                break
```

- What it does: Grabs the summary text.
- Question 2 (Experience):

```
   elif question_index == 2:
        experience = []
        current_exp = {}
        for line in result.split("\n"):
```

```
            if line.strip() == "---" and current_exp:
                experience.append(current_exp)
                current_exp = {}
            elif "Job Title:" in line:
                current_exp["job_title"] = line.split(": ")[1]
if len(line.split(": ")) > 1 else ""
            ... (similar for company, dates, responsibilities,
achievements)
        if current_exp:
            experience.append(current_exp)
        st.session_state["resume_data"]["experience"] =
experience
```

- What it does: Builds a list of job experiences, splitting multiple jobs by `---`.
- Questions 3-7 (Projects, Qualifications, Skills, Certifications, Positions):

```
elif question_index == 3:  # Projects
    projects = []
    for line in result.split("\n"):
        if line.startswith("-"):
            projects.append(line.strip("- ").strip())
    st.session_state["resume_data"]["projects"] = projects
... (similar for 4-7)
```

- What it does: Makes lists for each section from lines starting with `-`.

---

11. Generating Resume Documents

These functions create your resume in Word or PDF.

Fresher Template (Word)

```
def generate_word_resume(return_pdf=False):
```

- What it does: Makes a Word doc for beginners; can convert to PDF if `return_pdf` is `True`.

```
resume_data = st.session_state["resume_data"]
doc = DocxDocument()
```

- What it does: Gets resume data and starts a new Word doc.

```
sections = doc.sections
for section in sections:
    section.top_margin = Inches(0.3)
```

```
        ... (sets bottom, left, right margins)
```

- What it does: Sets small margins (0.3 inches) on all sides.

```
   header = doc.add_paragraph()
    header.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
    run =
header.add_run(resume_data["personal_info"]["full_name"] or
"Full Name Not Provided")
    run.bold = True
    run.font.size = Pt(16)
```

- What it does: Adds your name, centered, bold, and big (16pt).

```
   if resume_data["personal_info"]["degree"]:
        run =
header.add_run(f"\n{resume_data['personal_info']['degree']}")
    ... (similar for contact info, address)
```

- What it does: Adds degree, contact info, and address below the name.

```
   doc.add_heading("Summary", level=1)
    p = doc.add_paragraph(resume_data["summary"] or "Summary Not
Provided")
    p.style.font.size = Pt(10)
```

- What it does: Adds a "Summary" section with your text.

(Similar blocks for Experience, Projects, etc.)

```
   buffer = BytesIO()
    doc.save(buffer)
    buffer.seek(0)
```

- What it does: Saves the doc to memory (not disk).

```
   if return_pdf:
        with tempfile.NamedTemporaryFile(delete=False,
suffix=".docx") as tmp:
            tmp.write(buffer.getvalue())
            tmp_path = tmp.name
        pdf_path = tmp_path.replace(".docx", ".pdf")
        docx2pdf.convert(tmp_path, pdf_path)
        ... (reads PDF into buffer, deletes temp files)
        return pdf_buffer
    return buffer
```

- What it does: If PDF is wanted, converts the Word doc to PDF and returns it; otherwise, returns the Word doc.

Intermediate Template (Word)

```
def generate_intermediate_word_resume(return_pdf=False):
```

- What it does: Similar to `generate_word_resume`, but styled for mid-career folks.

(Adds name, job title, contact info with different formatting, then sections like Skills, Work Experience, etc.)

Veteran Template (PDF)

```
def generate_veteran_pdf_resume():
```

- What it does: Makes a PDF directly for experienced users.

```
c = canvas.Canvas(buffer, pagesize=letter)
 c.setFillColor(HexColor("#0077b6"))
 c.rect(0, height - 50, width, 50, fill=1, stroke=0)
```

- What it does: Starts a PDF with a blue header bar.

(Adds sections with colors and formatting)

---

12. Generating Interview Questions

```
def generate_interview_questions():
```

- What it does: Creates 20 interview questions based on your resume.

```
resume_content = (
     f"Personal Info: {resume_data['personal_info']}\n"
     ... (combines all resume data)
 )
 prompt = f"""
 Based on the following resume data, generate 20 specific
interview questions...
 Resume Data:
 {resume_content}
 """
```

- What it does: Puts your resume into a message for the AI.

```
response = get_gemini_response(prompt)
```

- What it does: Gets the AI's list of questions.

---

13. Previewing and Downloading

```
def preview_pdf_scrollable(pdf_buffer):
```

- What it does: Shows the PDF in the app.

```
def create_download_link(file_path_or_buffer, file_name,
link_text):
```

- What it does: Makes a clickable link to download your resume.

---

14. Main App Logic

The rest uses `st.session_state["page"]` to show different screens:

- Login/Signup: Checks or creates your account.
- Welcome: Explains the app.
- Language Selection: Picks your language.
- Consent: Asks permission to record.
- Resume Template: Chooses your style.
- Questions: Asks questions, records/transcribes answers, edits resume live.
- Preview: Shows the resume, questions, and download options.

---

15. Styling and Footer

```
st.markdown("<style>...</style>", unsafe_allow_html=True)
```

- What it does: Adds CSS to make the app look nice.

```
st.markdown("© 2025 Resume Builder AI | Powered by CSSTUV",
unsafe_allow_html=True)
```

- What it does: Adds a footer with copyright info.

That's it! Every line builds this cool app that turns your voice into a polished resume. Let me know if you want to zoom in on anything!

How It All Ties Together

1. User Flow: Login → Welcome → Language → Consent → Template → Questions → Preview.
2. Data Flow: Voice/text → Gemini AI → Processed data → MySQL → Resume document.
3. Features: Multi-language support, voice input, real-time editing, and professional output.

This code creates a powerful, user-friendly tool for resume building with AI assistance! Let me know if you'd like a deeper dive into any part.