Resume Generation Chatbot: ResumeAI - Project Report

Team Name: Sec2_P2

Tanmayee 2023UG000134
Shreerenu 2023UG000151
Chandrapal - 2023UG000118
Sachin - 2023UG000130
Vaidhav - 2023000136
Uzair - 2023000149

Vidyashilp University - 31/3/2025

Table of Contents

1) Introduction

In today's competitive job market, job seekers often struggle to create a professional resume and prepare for interviews effectively. Many candidates are unsure about how to format their resumes, highlight key skills, or anticipate relevant interview questions. Additionally, identifying suitable companies based on their qualifications and estimating their chances of getting hired remains a challenge.

This project aims to address these issues by developing an AI-powered chatbot that assists users in generating a well-structured resume, preparing for interviews, and identifying potential job opportunities. The chatbot collects user information in a standard resume format and uses AI models to generate a resume, predict 20 relevant interview questions, and suggest companies categorized into safety, moderate, and reach, along with the probability of securing a job in each. The system is built using **Gemini API**, **Jupyter Notebook**, and **Streamlit**, with **MySQL** used for data storage.

This solution is particularly beneficial for fresh graduates and undergraduate students entering the job market, as it provides structured guidance and personalized insights. By leveraging AI-driven automation, the system empowers job seekers, increases their confidence, and improves their chances of securing suitable employment.

2) Problem Statement

For many fresh graduates and undergraduate students, transitioning from academic life to the professional world presents significant challenges. A well-crafted resume is often the first step in securing job opportunities, yet many students struggle with:

2.1) Resume Formatting Issues – Uncertainty about the correct structure, relevant sections, and how to effectively present their skills and experience.

2.2) Interview Preparation Challenges – Lack of awareness regarding the types of questions they might be asked in interviews for their desired job roles.

2.3) Job Opportunity Awareness – Difficulty in identifying companies that match their qualifications and estimating their likelihood of getting hired.

Traditional job preparation resources, such as static resume templates and generic interview guides, fail to offer personalized assistance. Many students rely on trial and error, leading to missed opportunities and prolonged job searches.

This project addresses these issues by leveraging **Artificial Intelligence (AI) and Machine Learning (ML)** to generate resumes, create personalized interview questions, and suggest job opportunities with probability assessments. By automating these critical job application steps, the system **enhances job readiness, improves decision-making, and boosts confidence among job seekers**, ensuring they are well-equipped to enter the professional world.

3) Objectives:

The objective of this project is to build an AI-powered chatbot that streamlines the resume creation and job preparation process. The system will provide users with structured career assistance by leveraging AI to generate resumes, predict interview questions, and suggest relevant job opportunities. The key goals of this project include:

3.1) Develop an AI-powered resume builder that formats user-provided details into a structured and professional resume.

3.2) Generate personalized interview questions tailored to the user's skills, experience, and job role preferences.

3.3) Categorize job opportunities into safety, moderate, and reach companies, providing users with actionable career insights.

3.4) Implement a database system (MySQL) to store user data for easy retrieval and future modifications.

3.5) Design a user-friendly web interface using Streamlit to ensure accessibility and ease of use.

3.6) Utilize machine learning models and AI techniques (Gemini API) to automate resume generation and job analysis.

3.7) Improve job search efficiency by offering AI-driven recommendations, reducing the time and effort required for job applications.

This project aims to bridge the gap between job seekers and employment opportunities by providing an intelligent and automated career guidance system.

4) Solution Overview

This project presents an **AI-powered resume and interview preparation chatbot** that assists users in creating professional resumes, generating relevant interview questions, and categorizing job opportunities based on hiring probabilities. The system integrates **Gemini API, Streamlit, Jupyter Notebook, and MySQL**, providing an intelligent and user-friendly career guidance tool.

The **flow diagram** details the step-by-step process, breaking down user interactions, AI functionalities, and backend processes. Below is a comprehensive overview based on the **flow diagram and previously shared project details**.

5) Solution Workflow

5.1) User Interaction and Input Collection

- The user begins by **opening the website** and selecting a **preferred language** and **resume template** (e.g., professional, modern, creative).
- Based on the chosen template, the chatbot presents a **structured questionnaire** that collects information in a standard resume format.
- The input collection includes:
    - **Personal Information** (Name, Email, LinkedIn, Portfolio, etc.)
    - **Educational Background** (Degree, University, Year of Graduation)
    - **Work Experience** (Job roles, Companies, Duration, Achievements)
    - **Skills & Certifications** (Technical & Soft Skills, Courses, Certifications)
    - **Projects & Publications** (GitHub, Research Papers, Major Projects)
    - **Languages & Hobbies**

5.2) Data Processing and AI Integration

- **Data Validation & Processing:**
    - The system ensures **correct data formatting** and checks for missing fields.
    - Invalid inputs prompt an error message and request user corrections.
- **Resume Generation (Gemini API):**
    - AI structures the user-provided data into a **professional resume format** tailored to the selected template.
    - AI-enhanced resume suggestions refine content based on industry standards.
- **Interview Question Generation:**
    - Gemini API analyzes user skills, experience, and job preferences.
    - The system generates **20 personalized interview questions**, covering:
        - General HR questions
        - Technical domain-specific questions
        - Behavioral & problem-solving scenarios

5.3) Audio-Based Input Option

- Users can opt to **upload an audio recording** instead of manually entering details.
- AI converts the speech into text using **Natural Language Processing (NLP)**.
- Converted text is **structured and analyzed** similarly to manual input.

5.4) Database Storage and Export Options

- **MySQL Database:**
    - Stores **user details**, **generated resumes**, **interview questions**, and **company suggestions**.
    - Enables users to **revisit, modify, and regenerate** resumes.
- **Resume Export & Download:**
    - Users can **export the resume as a PDF or Word document**.

- **Editing & Refinement:**
    - Users can **edit specific sections** before finalizing the resume.
    - AI offers **improvement suggestions** for clarity and impact.

5.5) Exit & Future Accessibility

- Once satisfied, the user can **exit the website** after downloading or saving the generated documents.
- Stored data allows **future access for edits, reapplications, or updates**.

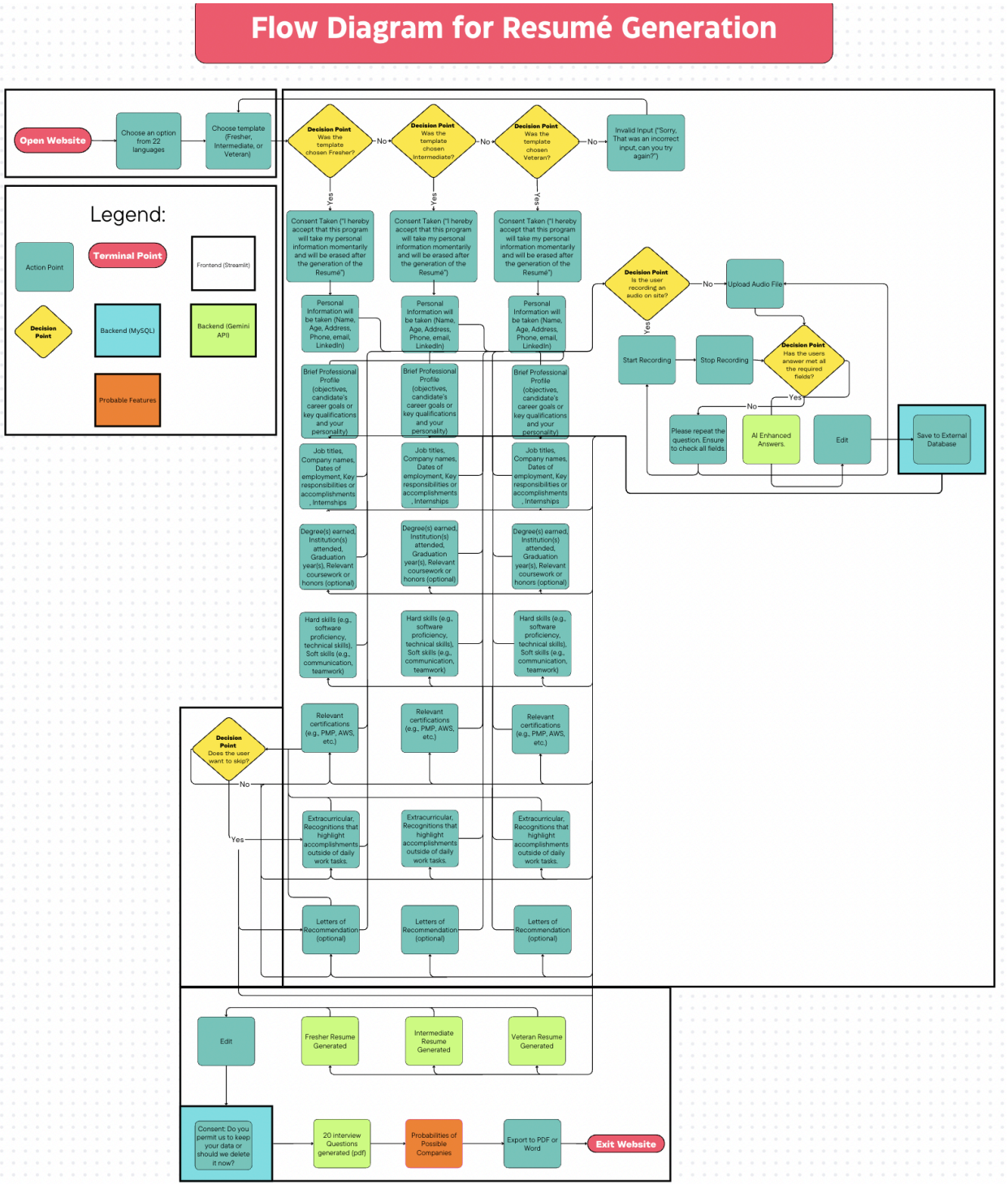


Fig 1.1: Flow Diagram for ResumeAI (Sequence of Events)

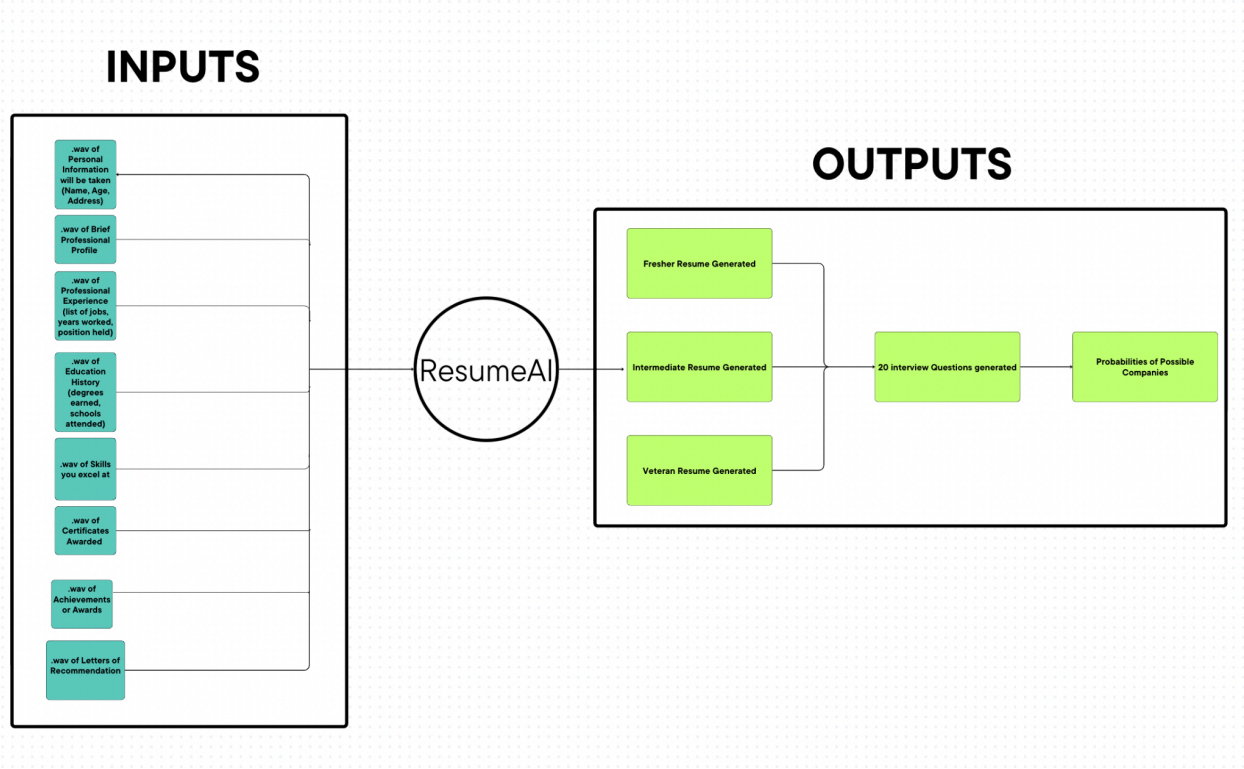


Fig 1.2: I/O Diagram

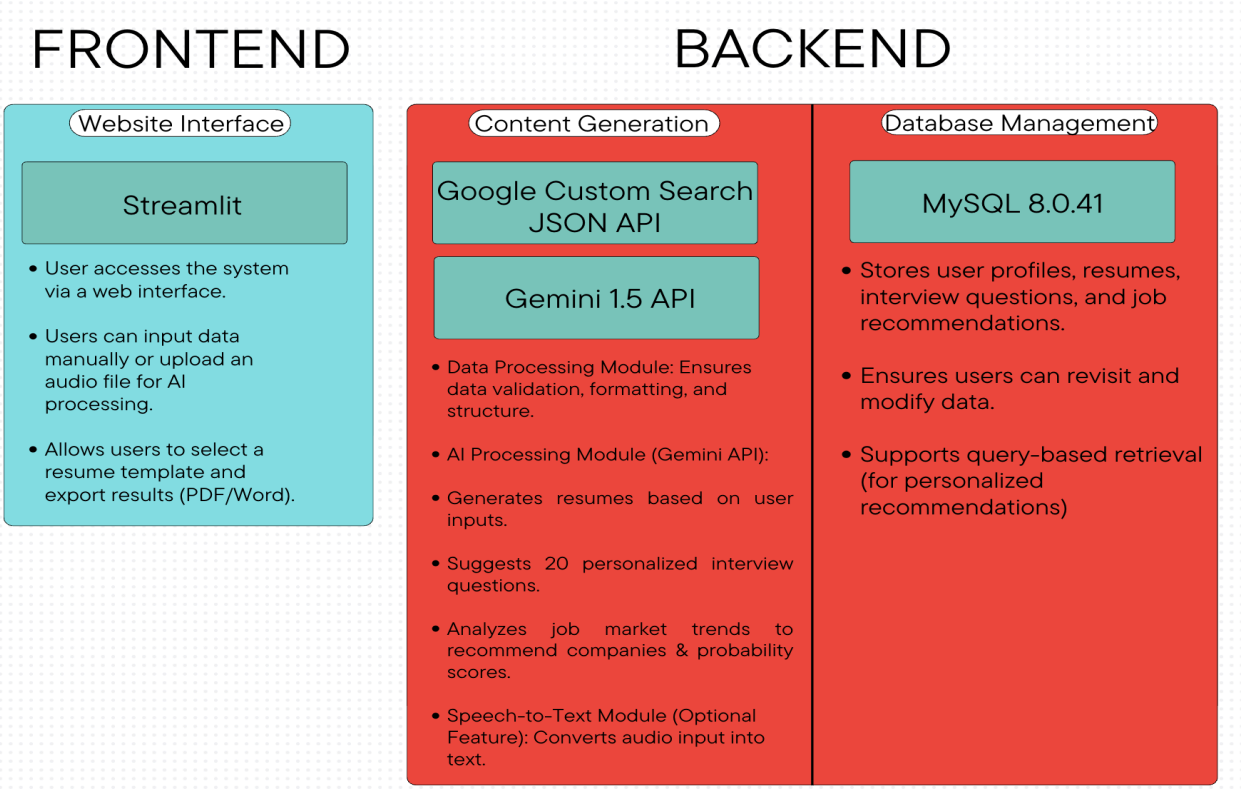


Fig 1.3: Tech Stack to implement ResumeAI

6) Technology Stack

- **Frontend:** Streamlit (for UI/UX)
- **Backend:** Python, Jupyter Notebook
- **AI Integration:** Gemini API (for resume formatting, interview questions, and job analysis)
- **Database:** MySQL (for user data storage)
- **Speech Processing:** NLP for audio-to-text conversion

7) Key Benefits & Impact

7.1) Automates Resume Creation – Saves time and ensures professional formatting.
7.2) AI-Powered Interview Preparation – Prepares candidates for real-world job interviews.
7.3) Job Market Insights – Helps users target the right companies with probability estimates.
7.4) User-Friendly & Accessible – No prior knowledge of resume writing is required.
7.5) Storage & Reusability – Allows users to edit, update, and refine resumes.

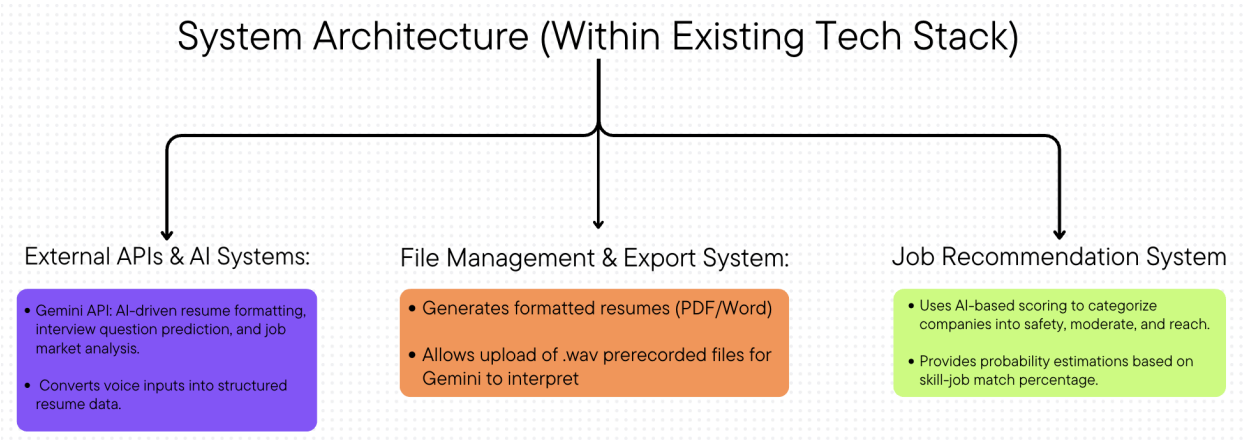


Fig 1.4: System Architecture of ResumeAI

8)System Component Interaction & Flow

8.1) User Interaction (Frontend) → Backend Communication

- The **user opens the Streamlit-based web application** and selects a preferred language & resume template.
- The system **guides the user through structured questions** for resume-building.
- Alternatively, the user can **upload an audio file** that is processed by the **Speech-to-Text module**.
- Once data is collected, the frontend **sends structured input data to the backend** for processing.

8.2) Backend Processing (Data Validation & Formatting)

- The backend **validates the input**, ensuring no essential fields are missing.
- If errors exist, the system **prompts the user for corrections**.
- Once validated, data is stored in **MySQL for further processing**.

8.3) AI-Powered Resume Generation (Gemini API)

- The backend **sends structured resume data to the Gemini API**.
- The API **analyzes and generates** a **formatted resume** based on the chosen template.
- The response is returned in **structured JSON format**, which the backend converts into **human-readable content**.

8.4) AI-Generated Interview Questions

- The system extracts **key skills, job roles, and experience** from the user profile.
- This information is sent to the **Gemini API**, which generates:
    - **HR-related interview questions** (soft skills, behavior, and goals).
    - **Technical questions** based on the user's domain expertise.
    - **Scenario-based & problem-solving questions**.
- These **20 interview questions** are displayed on the frontend and stored in the database for later retrieval.

8.5) File Generation & Export System

- The **resume and interview questions are formatted for download**.
- Users can **export their resume in PDF or Word format**.
- A **Letter of Intent (optional)** is generated for job applications.

8.6) Database Management (User Data Storage & Retrieval)

- The **MySQL database stores**:
    - **User profiles** (personal details, skills, experience).

    - **Generated resumes** (for future modifications).
    - **Interview questions** (tailored for the user).
    - **Company recommendations** & probability scores.
- Users can **retrieve and update** their information in later sessions.

9) Implementation Details

9.1) Steps Followed to Build the Solution:

- **Developing the Base Skeleton Code:** The project began by constructing the core logic of the chatbot, ensuring that user inputs could be captured and processed effectively. This included defining how the system would handle structured data input and output for resume generation and interview question suggestions.

- **Implementing Resume Autofill System:** One of the essential features was mapping user-provided information into a standardized resume format. This required handling different resume fields, such as personal details, education, work experience, and skills, while ensuring proper formatting and alignment.

- **Integrating Audio Transcription Module:** To allow users to input data using speech, an audio transcription system was implemented. This involved processing voice input, converting it to text, and ensuring the extracted information was structured correctly for further use in the resume generation process.

- **Testing Individual Components:** Before integrating all features, each module was tested separately in **Jupyter Notebook** to ensure that the AI-generated responses, data handling, and formatting worked as expected. Errors in string processing, inconsistent text output, and formatting constraints were addressed at this stage.

- **Incorporating the Gemini API:** The AI capabilities of the chatbot were powered using the **Gemini API**, which was responsible for generating resumes, predicting job probabilities, and creating personalized interview questions. Multiple prompts were designed and refined to ensure high-quality responses.

- **Database Integration with MySQL:** All user inputs and generated outputs needed to be stored for future reference. A **MySQL database** was used to store user profiles, resume details, and generated job recommendations securely, enabling users to revisit and modify their resumes if needed.

- **UI/UX Implementation Using Streamlit:** The frontend of the project was built using **Streamlit**, ensuring a simple and interactive interface for users. Features such as form submissions, audio recording, and dynamic output display were implemented while working around Streamlit's UI constraints.

9.2) Innovations or Unique Approaches Used:

- **Google Search API Integration for Real-World Job Matching:** To make job suggestions more relevant, we integrated **Google Search API** to find actual companies that fit into the safety, moderate, and reach categories. This allowed the system to provide **real company names instead of generic job titles**, making the job search process more practical and tailored.

- **Ensuring AI-Generated Content Aligns with Structured Resume Formats:** Many AI-generated responses can be unstructured or verbose. To overcome this, we fine-tuned how the Gemini API outputs information, ensuring that resumes followed a professional and **human-readable** format.

9.3) Challenges Faced and How They Were Overcome:

- **UI/UX Constraints in Streamlit:** Designing the frontend was more challenging than anticipated due to Streamlit's limitations in handling

complex UI elements. Achieving an intuitive layout with interactive components required multiple design adjustments and external CSS customization to improve the interface.

- **String Parsing and Formatting Errors:** During the implementation of the resume autofill system, numerous **formatting inconsistencies** arose, particularly when mapping user inputs to pre-defined resume sections. Careful debugging and structured text processing were used to ensure seamless data population.

- **Building the Probability-Based Job Categorization System:** Implementing a **probability distribution model** to classify job recommendations into safety, moderate, and reach categories required extensive **data analysis and fine-tuning**. This process involved creating **custom scoring algorithms** that factored in user skills, experience, and industry trends. We ultimately decided to drop this feature due to multiple errors arising in the implementation and seamless integration into our existing code.

- **Challenges in Implementing the Google Search API:** Fetching real company names that fit within the categorized job probabilities was complex. **Filtering relevant results** and ensuring that **only high-quality job listings** were retrieved required multiple iterations in query structuring and data cleaning.

- **Ensuring AI Consistency in Resume Generation:** AI-generated responses often varied in format and length, which made it difficult to standardize the resume structure. Through prompt engineering and iterative testing, we ensured that the AI-generated resumes maintained a **consistent and professional layout**.

- **Final Integration and Testing:** After overcoming these challenges, all components were integrated into a single cohesive system. **Multiple testing iterations** were conducted to refine user experience, eliminate bugs, and optimize response accuracy. Ultimately, the project successfully combined AI-powered resume generation, probability-based job recommendations, and structured interview

question preparation into an **interactive, user-friendly career tool**.

10) LLM and AI Integration

10.1) Large Language Model (LLM) Used:

- **Gemini API**: The **Gemini AI model** was used as the core **LLM** for generating resumes, interview questions. Gemini's advanced **Natural Language Processing (NLP) capabilities** enabled the chatbot to understand user inputs, structure resume content effectively, and generate relevant and industry-specific responses.

- **Why Gemini?** Unlike other LLMs, Gemini was chosen due to its ability to **handle structured data formatting**, provide **contextually relevant outputs**, and maintain **concise, professional wording** required for resumes and interview responses, the most important of all being a readily available (relatively easy to troubleshoot) and free to use.

10.2) Integration of AI Components:

- **Speech-to-Text for Voice Input:**
    - To enhance accessibility, we integrated an **audio transcription module** that allows users to **speak their resume details instead of typing**.
    - The speech input is processed using **Google Speech-to-Text API**, which converts audio into structured text before being parsed and stored in MySQL.
    - This module improves efficiency for users who prefer verbal communication or are unfamiliar with structured resume formatting.
- **Natural Language Processing (NLP) for Resume Content Structuring:**
    - The **Gemini API's NLP** capabilities were leveraged to **restructure and refine user inputs** into a **coherent, professional resume**

format.

- The AI ensures that skills, work experience, and education details are presented in a way that aligns with industry standards.

- Additionally, Gemini was prompted to generate **customized resume summaries** based on the user's professional background, making each resume unique and well-tailored.

- **AI-Generated Interview Questions:**

  - The chatbot generates **20 interview questions** based on the user's resume details, ensuring that the questions are **industry-specific and role-relevant**.

  - The **difficulty level of the questions varies**, preparing the user for both basic and advanced interview scenarios.

  - This component was refined through **prompt engineering**, ensuring that the AI produces structured, meaningful, and grammatically correct questions.

- **Ensuring AI Consistency and Output Formatting:**

  - One of the biggest challenges with AI-generated content is **maintaining consistency** in formatting and phrasing.

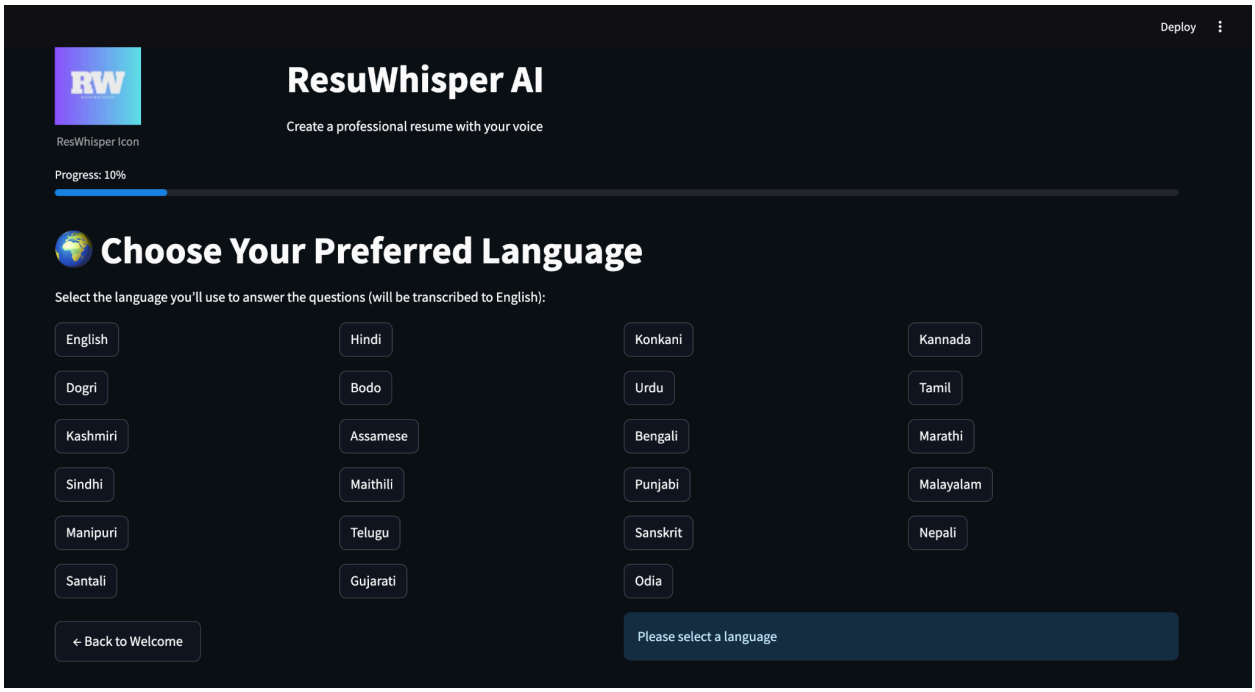
  - **Custom instructions were embedded in the Gemini API prompts** to ensure that resume outputs followed a **structured format with bullet points, proper headings, and clear, concise language**.

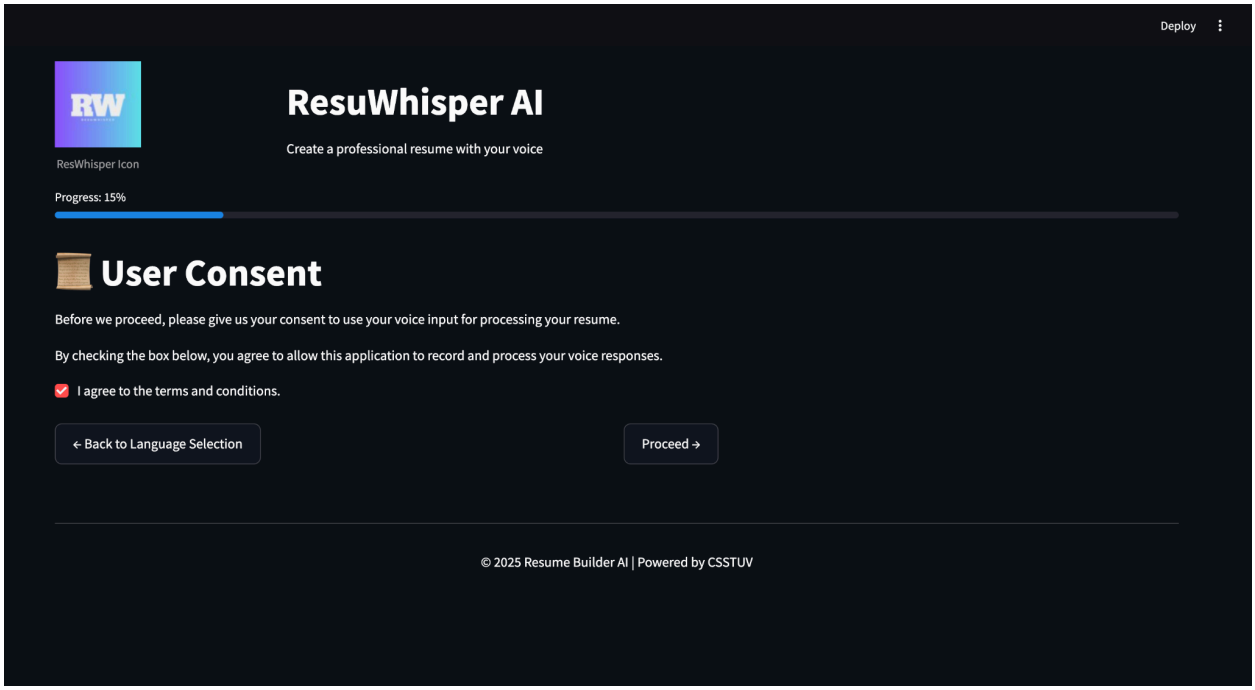
  - The AI's responses were tested rigorously to eliminate **redundancies, excessive verbosity, or unstructured data**.

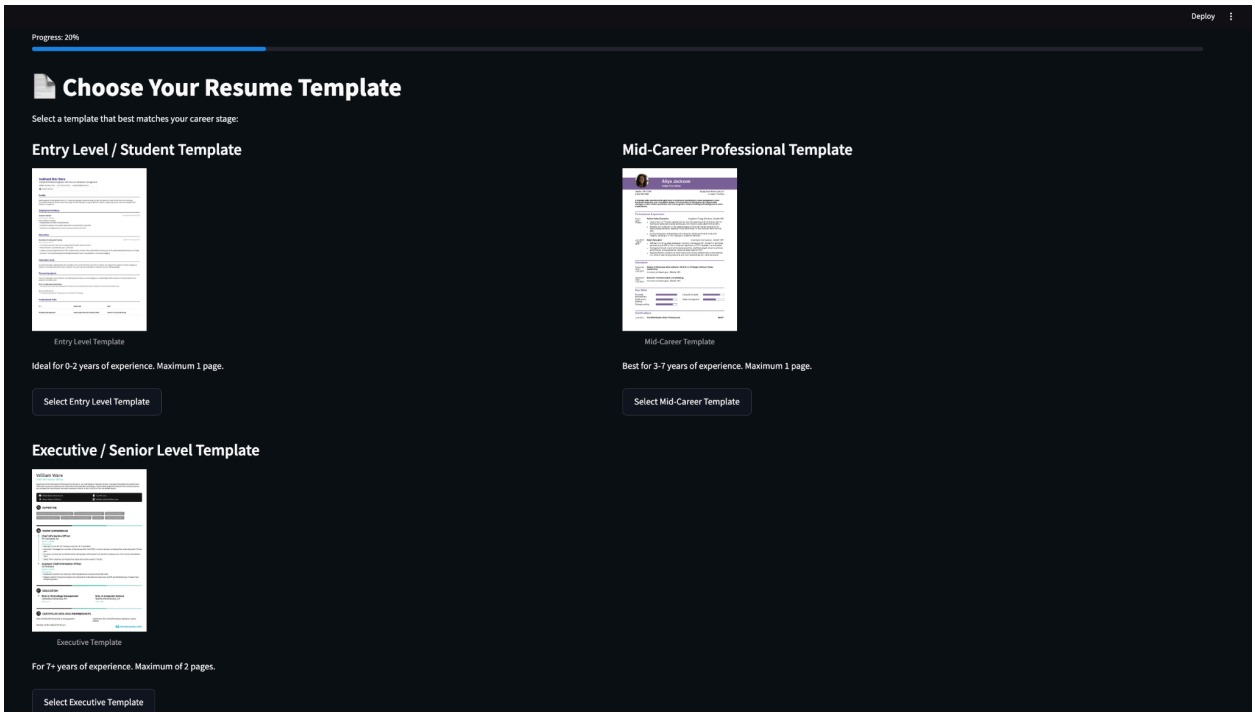
11) Key UX Design Decisions

11.1) Step-by-Step Navigation Flow

- **Decision**: A structured progression guides users through authentication, language selection, consent, template choice, questionnaire, and preview stages.
- **Rationale**: Breaking the resume-building process into clear steps reduces complexity, making it approachable for users with varying expertise. Navigation options provide control, fostering confidence.
- **Impact**: Minimizes overwhelm, increases completion rates, and ensures a logical, user-driven experience.



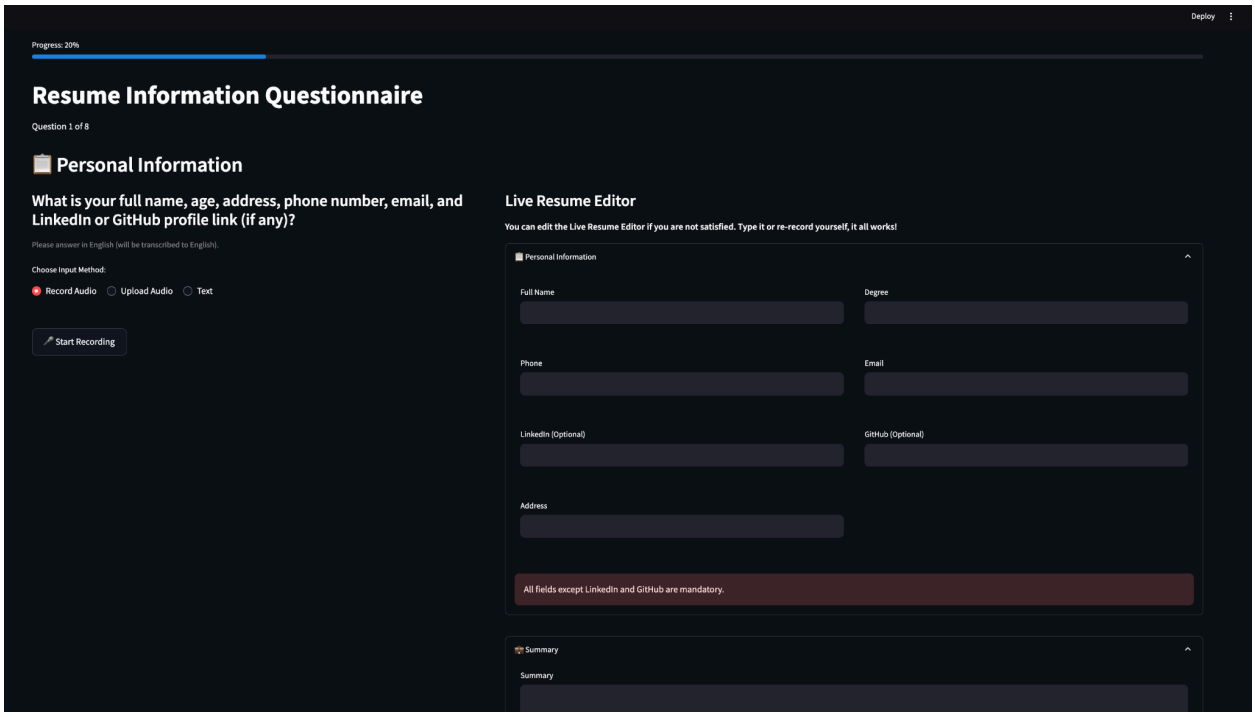




11.2) Dual-Pane Questionnaire Layout

- **Decision**: The questionnaire features two sections: one for input and another for a live resume preview.

- **Rationale**: Simultaneous input and output visibility offers real-time feedback, allowing users to see and adjust their resume as they respond. This transparency builds trust in the AI process.
- **Impact**: Streamlines creation by merging input and review, enhancing user control and reducing errors.



11.3) Multiple Input Options

- **Decision**: Users can record audio, upload files, or type responses, with a clear selection mechanism.
- **Rationale**: Offering flexibility caters to diverse preferences and accessibility needs, ensuring inclusivity for users with different comfort levels or device capabilities.
- **Impact**: Broadens reach, improves satisfaction, and accommodates varied interaction styles, from hands-free to precise editing.

11.4) Audio Recording with Clear Feedback

- **Decision**: The recording interface includes distinct controls and real-time status updates with visual cues.

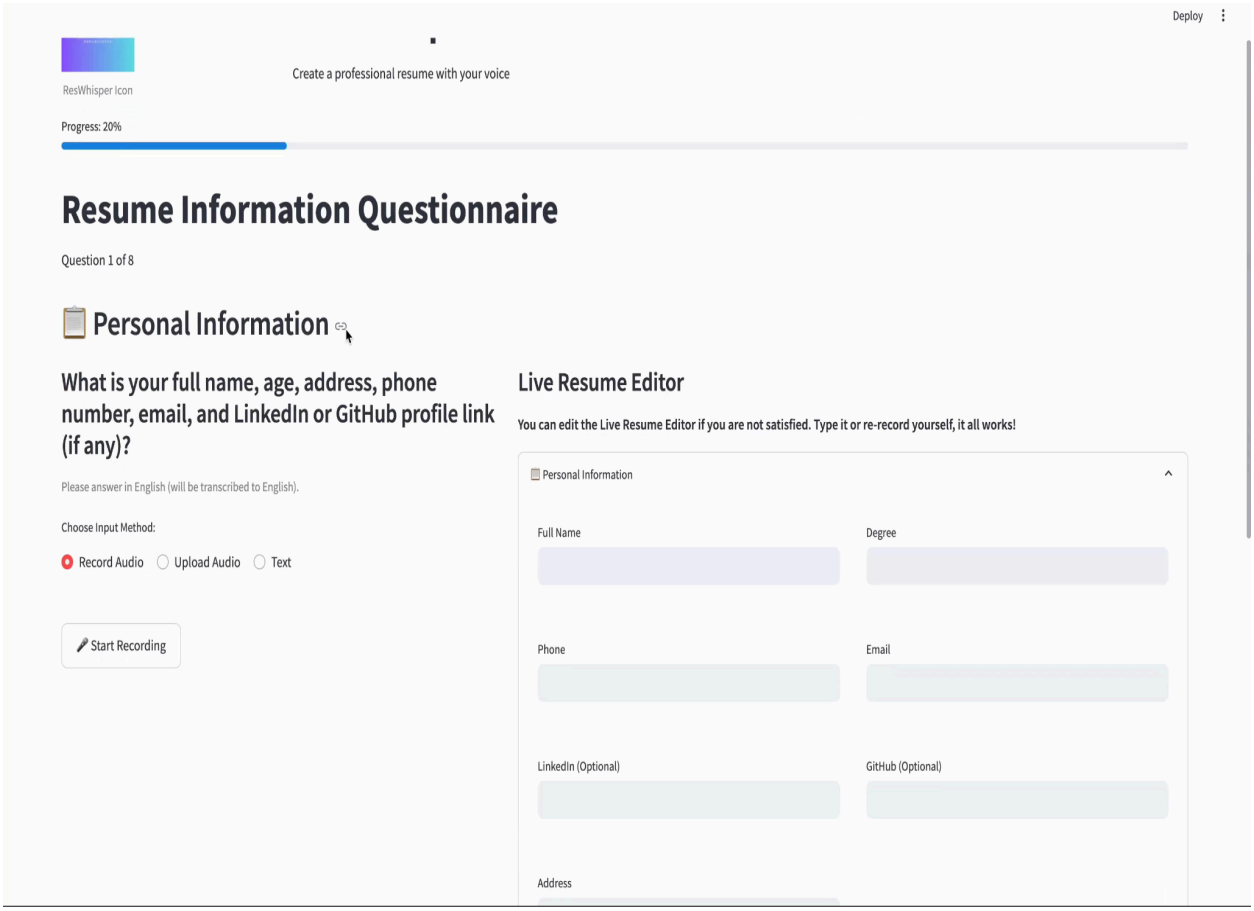
- **Rationale**: Clear actions and immediate feedback prevent confusion during voice input, a core feature, while visual indicators make it intuitive for all users.
- **Impact**: Enhances reliability and engagement, ensuring the voice-driven experience is seamless and error-free.

11.5) Progress Indicator

- **Decision**: A visual progress bar tracks completion across stages, from start to finish.
- **Rationale**: Showing advancement motivates users by highlighting progress, reducing perceived effort and encouraging persistence through a tangible sense of achievement.
- **Impact**: Gamifies the process, boosts engagement, and sets clear expectations, aiding time management.

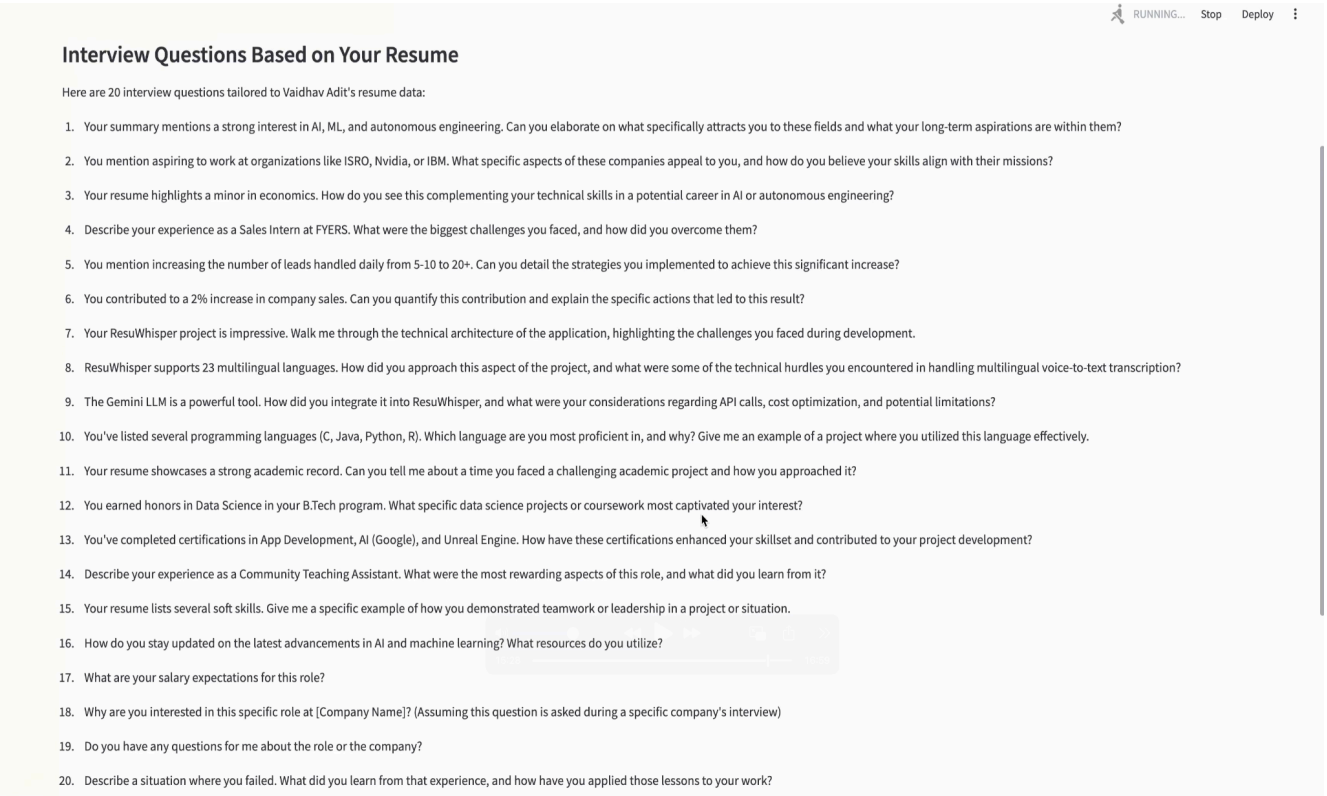
11.6) Mandatory vs. Optional Field Clarity

- **Decision**: Mandatory fields (e.g., Personal Info) require completion with explicit cues, while optional ones (e.g., Projects) allow skipping.
- **Rationale**: Enforcing essentials ensures a functional resume, while flexibility respects user context, balancing quality with ease of use.
- **Impact**: Guides input effectively, prevents incomplete outputs, and reduces frustration by clarifying requirements.



11.7) Comprehensive Preview and Download

- **Decision**: The final stage offers a resume preview, interview questions, and download options in multiple formats.
- **Rationale**: A preview builds confidence by allowing verification, while added questions and format choices enhance utility, meeting diverse application needs.
- **Impact**: Reinforces trust, delivers a polished result, and fulfills user goals, encouraging completion and reuse.



12) Code Structure

The application is contained within a single Python script (app.py), leveraging external libraries for specific functionalities such as audio processing, document generation, and AI-driven content creation. The code is organized into logical sections: initialization, session state management, utility functions, resume generation functions, and the main application workflow. Below, each key function is explained in detail, highlighting its purpose, inputs, outputs, and role within the system.

12.1) Key Functions and Their Details

init_session_state()

- **Purpose**: Initializes the session state variables used throughout the application to maintain user data and application progress.
- **Inputs**: None (operates on st.session_state, a Streamlit global object).

- **Outputs**: Sets default values for session state variables if they are not already defined.
- **Detailed Explanation**:
    - This function ensures that the application starts with a consistent state by defining a dictionary of default values. These include user authentication status (authenticated), page navigation (page), resume data structure (resume_data), and temporary variables like audio_file and current_response.
    - The resume_data dictionary is a nested structure with fields such as personal_info, summary, experience, and skills, which are populated as the user progresses through the questionnaire.
    - Example usage: When a user first accesses the app, this function sets st.session_state["page"] = "login" and initializes an empty resume_data structure.
- **Role in System**: Acts as the foundational setup for state persistence, enabling seamless navigation and data retention across user interactions.

record_audio(filename, stop_event, samplerate=44100)

- **Purpose**: Records audio input from the user's microphone and saves it as a WAV file for transcription.
- **Inputs**:
    - filename (string): Path where the audio file will be saved.
    - stop_event (threading.Event): Event object to signal recording termination.
    - samplerate (int, optional): Audio sampling rate, defaulting to 44100 Hz.
- **Outputs**: Saves an audio file at the specified filename if successful.
- **Detailed Explanation**:
    - Utilizes the sounddevice library to capture audio via a callback function that appends audio data to a list while the stop_event is not set.
    - Once stopped, the audio data is concatenated using NumPy and written to a WAV file using the wave library.
    - Includes error handling to display success or failure messages via Streamlit (st.success or st.error).

    - Example: A user clicks "Start Recording," triggering this function in a separate thread, and stops it with "Stop Recording," saving the file.
- **Role in System**: Enables the audio-based input option, allowing users to verbally provide resume details, enhancing accessibility.

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

- **Purpose**: Interfaces with the Gemini API to transcribe audio or generate text responses based on prompts.
- **Inputs**:
    - input_msg (string): Prompt or instruction for the Gemini model.
    - audio_path (string, optional): Path to an audio file for transcription.
    - mime_type (string, optional): MIME type of the audio file, defaulting to "audio/wav".
- **Outputs**: Returns the text response from Gemini API or None if an error occurs.
- **Detailed Explanation**:
    - If audio_path is provided, it uploads the audio file to Gemini and pairs it with the input_msg for transcription or analysis.
    - Otherwise, it sends input_msg alone for text-based generation (e.g., resume formatting or interview questions).
    - Error handling ensures that any API failures are caught and displayed to the user via st.error.
    - Example: For audio input, it might transcribe "My name is John" into text; for text input, it generates a formatted resume section.
- **Role in System**: Central to AI integration, this function powers transcription, resume generation, and interview question creation.

translate_questions(language)

- **Purpose**: Translates the resume questionnaire into the user’s selected language using the Gemini API.
- **Inputs**:
    - language (string): Target language for translation (e.g., "Hindi").
- **Outputs**: Returns a list of translated questions or defaults to English if translation fails.

- **Detailed Explanation**:
    - Checks if the translation for language is already cached in st.session_state["translated_questions"].
    - If not, constructs a prompt to translate the eight English questions into the specified language and calls get_gemini_response.
    - Parses the response into a list, ensuring exactly eight questions are returned, falling back to English if the API fails.
    - Example: Translates "What is your full name?" into "आपका पूरा नाम क्या है?" for Hindi.
- **Role in System**: Supports multilingual accessibility, allowing users to respond in their preferred language.

process_response_with_gemini(question_index, response)

- **Purpose**: Processes user responses with the Gemini API to extract and enhance resume data.
- **Inputs**:
    - question_index (int): Index of the current question (0-7).
    - response (string): User's raw response to the question.
- **Outputs**: Returns a formatted string with enhanced resume data specific to the question.
- **Detailed Explanation**:
    - Uses a list of predefined prompts tailored to each question (e.g., personal info, work experience).
    - Each prompt instructs Gemini to extract specific fields (e.g., "Full Name," "Job Title") and enhance them professionally (e.g., formatting phone numbers).
    - The output is structured (e.g., "Full Name: John Doe\nAge: ") and returned for further processing.
    - Example: For "I worked at Tech Corp as a developer," it outputs structured experience data.
- **Role in System**: Bridges raw user input to polished resume content, ensuring accuracy and professionalism.

update_resume_data(question_index, response)

- **Purpose**: Updates the resume_data structure in session state based on Gemini-processed responses.

- **Inputs**:
    - question_index (int): Index of the current question.
    - response (string): User's response to process.
- **Outputs**: Modifies st.session_state["resume_data"] with extracted data.
- **Detailed Explanation**:
    - Calls process_response_with_gemini to get formatted output, then parses it into the appropriate resume_data section.
    - Handles different data types: dictionaries for personal info, strings for summaries, lists for experience/projects.
    - Example: For question 0, updates personal_info with "Full Name: John Doe"; for question 2, appends experience entries.
- **Role in System**: Ensures that AI-enhanced data is stored correctly for resume generation.

generate_word_resume(return_pdf=False)

- **Purpose**: Generates a resume in Word format for the "Fresher" template, with an option to convert to PDF.
- **Inputs**:
    - return_pdf (bool, optional): If True, converts the Word file to PDF.
- **Outputs**: Returns a BytesIO buffer containing the Word document or PDF.
- **Detailed Explanation**:
    - Uses python-docx to create a structured Word document with sections like "Summary," "Experience," and "Skills."
    - Populates data from resume_data, adding fallback text (e.g., "Not Provided") for missing fields.
    - If return_pdf is True, uses docx2pdf to convert the Word file to PDF via temporary files.
    - Example: Creates a one-page resume with a bolded name and bullet-pointed skills.
- **Role in System**: Provides the resume export functionality for the "Fresher" template.

generate_intermediate_word_resume(return_pdf=False)

- **Purpose**: Generates a resume in Word format for the "Intermediate" template, with PDF conversion option.
- **Inputs**:

    - return_pdf (bool, optional): If True, converts to PDF.
- **Outputs**: Returns a BytesIO buffer with the document.
- **Detailed Explanation**:
    - Similar to generate_word_resume but tailored for mid-career professionals, limiting experience to two entries.
    - Uses distinct formatting (e.g., colored name, concise layout) to reflect the template's style.
    - Example: Outputs a resume with a job title in bold and responsibilities in bullet points.
- **Role in System**: Supports the "Intermediate" template export.

generate_veteran_pdf_resume()

- **Purpose**: Generates a resume in PDF format for the "Veteran" template.
- **Inputs**: None (uses resume_data from session state).
- **Outputs**: Returns a BytesIO buffer with the PDF.
- **Detailed Explanation**:
    - Uses reportlab to create a PDF with a blue header and structured sections.
    - Directly generates PDF without a Word intermediate, suitable for experienced professionals.
    - Example: Produces a two-page resume with employment history and certifications.
- **Role in System**: Handles the "Veteran" template export.

generate_interview_questions()

- **Purpose**: Generates 20 personalized interview questions based on the user's resume.
- **Inputs**: None (uses resume_data).
- **Outputs**: Returns a string with numbered questions or an error message.
- **Detailed Explanation**:
    - Constructs a prompt with resume_data and sends it to get_gemini_response.
    - Gemini generates questions covering HR, technical, and behavioral aspects.
    - Example: "1. Can you describe your experience at Tech Corp?"

- **Role in System**: Enhances interview preparation by providing tailored questions.

preview_pdf_scrollable(pdf_buffer)

- **Purpose**: Displays a scrollable PDF preview in the Streamlit interface.
- **Inputs**:
  - pdf_buffer (BytesIO): Buffer containing the PDF.
- **Outputs**: Renders the PDF via an iframe in the browser.
- **Detailed Explanation**:
  - Encodes the PDF buffer in base64 and embeds it in an HTML iframe.
  - Example: Shows the generated resume in a 600px-high scrollable frame.
- **Role in System**: Allows users to review their resume before downloading.

create_download_link(file_path_or_buffer, file_name, link_text)

- **Purpose**: Creates a downloadable link for files in the Streamlit app.
- **Inputs**:
  - file_path_or_buffer (string or BytesIO): File path or buffer.
  - file_name (string): Name of the downloadable file.
  - link_text (string): Text for the download link.
- **Outputs**: Returns an HTML string for the download link.
- **Detailed Explanation**:
  - Converts the file content to base64 and generates an <a> tag with the appropriate MIME type.
  - Example: "Download PDF" link for "resume.pdf".
- **Role in System**: Facilitates file export for users.

12.2) Main Application Workflow

- **Initialization**: The script starts by importing libraries, configuring the Gemini API, and setting up Streamlit's page layout. init_session_state() is called to establish default states.

- **Page Navigation**: A conditional structure (if-elif) manages the application's pages: login, signup, welcome, language selection, consent, template selection, questions, and preview.
- **User Interaction**:
    - **Login/Signup**: Authenticates users and stores data in users_db.
    - **Questions**: Iterates through eight questions, collecting responses via audio or text, processed by process_response_with_gemini and stored via update_resume_data.
    - **Preview**: Displays the resume and interview questions, offering download options.
- **AI Integration**: Functions like get_gemini_response and process_response_with_gemini handle AI tasks, while resume generation functions (generate_word_resume, etc.) format the output.

13) Execution Guide

Overview

ResuWhisper AI is a voice-based resume-building tool built with Python and Streamlit. It leverages Google's Gemini API for transcription and content generation, records audio, and exports resumes as Word or PDF files. This guide provides a beginner-friendly setup process for running the app locally.

Prerequisites

- Python 3.8+: Install from python.org. Verify: python --version or python3 --version.
- Git (Optional): Install from git-scm.com. Verify: git --version.
- Text Editor/IDE: Jupyter Notebook (used in this guide), VS Code, or similar.
- Google API Key: Obtained from Google Cloud Console by enabling the Gemini API.
- System: Microphone and internet connection required.

13.1: Clone the Repository

git clone https://github.com/your-username/ResuWhisperAI.git

cd ResuWhisperAI

Replace your-username with your GitHub username if hosting this project.

13.2: Set Up Project Files

1. Main Code: Ensure app.py (provided in this repo) is in the root directory.
2. Setup Script: Create setup.py with:

```
import subprocess
import sys

required_libraries = [
    "streamlit",
    "google-generativeai",
    "sounddevice",
    "numpy",
    "python-docx",
    "docx2pdf",
    "reportlab"
]

def install_libraries():
```

```
        print("Installing required libraries...")
        for library in required_libraries:
            try:
                subprocess.check_call([sys.executable, "-m", "pip", "install",
library])
                print(f"Successfully installed {library}")
            except subprocess.CalledProcessError as e:
                print(f"Error installing {library}: {e}")
        print("Library installation complete!")

    if __name__ == "__main__":
        install_libraries()
```

3. Save both files in the ResuWhisperAI directory.

13.3: Set Up Virtual Environment

Isolate dependencies, including Streamlit, using a virtual environment.

1. Create Environment:

```
python -m venv venv
```

2. Activate Environment:

- Windows: venv\Scripts\activate
- macOS/Linux: source venv/bin/activate
- Prompt should show (venv).

13.4: Install Libraries with Jupyter Notebook

Use setup.py to install dependencies, including Streamlit, within the virtual environment.

1. Install Jupyter:

   ```
   python -m pip install jupyter
   ```

2. Launch Jupyter:

   ```
   jupyter notebook
   ```

- Opens in browser at http://localhost:8888.

Run Setup Script:

- Create a new notebook: "New" > "Python 3".
- In a cell, run: python %run setup.py
- Output confirms installation of:
  - streamlit: Web interface.
  - google-generativeai: Gemini API.
  - sounddevice: Audio recording.
  - numpy: Audio processing.
  - python-docx: Word generation.
  - docx2pdf: Word-to-PDF conversion.
  - reportlab: PDF generation.

13.5: Configure API Key

1. Open app.py in Jupyter or a text editor.
2. Replace the placeholder API key:

   ```
   Google_API_Key = "your-actual-api-key-here"
   ```

3. Save app.py.

13.6: Add Logo Image

1. Place resuwhisper.jpg (e.g., 100x100px logo) in the project folder.
2. Update the path in app.py:

```
st.image("path/to/resuwhisper.jpg", width=100,
caption="ResWhisper Icon")
```

- Example: "./resuwhisper.jpg" if in root directory.
- Save app.py.

13.7: Run the App

1. Start Streamlit (in terminal, with venv activated):

```
streamlit run app.py
```

2. Access: Open http://localhost:8501 in your browser.
3. Usage: Sign up, select language/template, record answers, and download your resume.

13.8: Troubleshooting

- ModuleNotFoundError: Rerun setup.py in Jupyter; ensure venv is active.
- API Errors: Verify API key and Gemini API access.
- Audio Issues: Check microphone and permissions.
- Image Not Found: Confirm path in app.py.
- PDF Fails: Ensure docx2pdf installed; Windows may need MS Word.
- General Fix: Update pip in setup.py:

```
subprocess.check_call([sys.executable, "-m", "pip", "install",
"--upgrade", "pip"])
```

13.9 Shutdown

1. Stop server: Ctrl+C in terminal.
2. Deactivate venv: deactivate.

13.10 Sample Workflow

```
git clone https://github.com/your-username/ResuWhisperAI.git

cd ResuWhisperAI
```

```
python -m venv venv
venv\Scripts\activate  # Windows
# or source venv/bin/activate (macOS/Linux)
python -m pip install jupyter
jupyter notebook  # Run %run setup.py in a cell
# Edit app.py with API key and image path
streamlit run app.py
```

- Open http://localhost:8501.

Notes

- Date in app.py is hardcoded to **March 31, 2025**—update as needed.
- Ensure disk space for temp files (audio, Word, PDF).

14) Performance Metrics

	Predicted Positive	Predicted Negative
Acutal Positive	20	2
Actual Negative	2	3

Overview

The performance of the ResuWhisper app was assessed using a **2x2 confusion matrix**, evaluating its ability to accurately process various functionalities. The evaluation covered the following key components:

- **User authentication:** Sign-up (password match) and login (password validation)
- **Live resume editor:** Field extraction from transcribed voice input
- **Resume content generation:** Automatic formulation of a structured resume
- **Interview preparation:** Generation of 20 interview questions

A total of **27 events** were tested across these functionalities, covering multiple fields and actions necessary for a seamless resume-building experience.

14.1 Breakdown of Key Metrics:

- **Total Events Tested:** 27
- **True Positives (TP) = 20:** Correctly processed events, including successful login, accurate field extractions, generated interview questions, and complete resume rendering.
- **False Negatives (FN) = 2:** Missed extractions for the *degree* and *email* fields, impacting resume completeness.
- **False Positives (FP) = 2:** Incorrectly extracted *address* field and spelling errors in another field, leading to inaccuracies.
- **True Negatives (TN) = 3:** Correct inactions, such as appropriately ignoring invalid login attempts or leaving empty fields unprocessed.

Component Breakdown

14.2 Sign-Up (Password Match)

- Likely a **TP** since the confusion matrix does not indicate errors in this component.

14.3. Login (Password Validation)

- Likely a **TP** as no FN or FP errors were reported.

14.4. Live Resume Editor (Field Extraction)

- **Total Fields Processed:** 15
    - (e.g., full_name, degree, phone, email, LinkedIn, GitHub, address, summary, experience, projects, qualifications, skills, certifications, positions, and personal_info-related degree field)
- **True Positives (TP) = 12**: Successfully extracted fields include *full_name, phone, summary, experience, qualifications, and positions*.
- **False Negatives (FN) = 2**: Failed to extract *degree* and *email* fields, which are essential for completeness.
- **False Positives (FP) = 2**:
    - Incorrectly extracted *address* field (potential parsing or recognition issue).
    - Spelling errors in another extracted field.
- **True Negatives (TN) = 1**: Correctly ignored fields such as LinkedIn or GitHub when left empty.

14.5. Interview Question Generation

- Likely a **TP**, as all 20 questions were generated successfully.

14.6 Resume Generation

- Likely a **TP**, indicating that the resume was successfully compiled without errors in structure.

Performance Metrics

Metric	**Formula**	**Value**
Accuracy	(TP + TN) / Total	(20 + 3) / 27 = **85.19%**

Precision	TP / (TP + FP)	20 / (20 + 2) = **90.91%** (Correctness of positive predictions)
Recall	TP / (TP + FN)	20 / (20 + 2) = **90.91%** (Ability to capture all positives)
False Positive Rate	FP / (FP + TN)	2 / (2 + 3) = **40%** (Rate of incorrect extractions)
False Negative Rate	FN / (FN + TP)	2 / (2 + 20) = **9.09%** (Rate of missed extractions)

14.7 Analysis

Strengths:

- **High accuracy (85.19%)**: The app successfully handled most tested events, demonstrating reliable functionality.
- **Strong precision and recall (90.91%)**: The app effectively captures and processes relevant inputs while maintaining a low rate of missed extractions.
- **Seamless authentication:** No recorded errors in sign-up or login processes.
- **Consistent resume generation:** Successfully compiled resumes without structural issues.

Potential Issues:

- **False negatives (9.09%)** indicate that certain critical fields, such as *degree* and *email*, were not extracted. This could be due to:
    - Errors in voice transcription affecting field recognition.

    - Weak NLP-based entity detection for specific field types.
    - Variability in speech patterns causing misinterpretation.
- **False positives (40%)** suggest that over-extraction or misclassification of text occurred. Possible reasons include:
    - Parsing errors leading to incorrect address extraction.
    - Imperfections in text-cleaning algorithms resulting in spelling mistakes.
    - Background noise or accents affecting text recognition quality.

15) Demo Video Link

Given below is the google drive link to access our demo video for ResuWhisper:
https://drive.google.com/drive/folders/13nzp5njbtZyIOsnc-9loI-E4G58FJjuZ?usp=sharing

16) Team Member Contribution and Respective Github Links

Each team member played a crucial role in the successful completion of the project. Their individual contributions are outlined below:

- **Chandrapal** – Designed and created the PowerPoint presentation. Link: https://github.com/Chandrapal330/ResuWhisperAI

- **Sachin** – Authored the report and contributed to the MySQL backend connection. Link: https://github.com/sachbhat0811/ResuWhisper-AI

- **Shreerenu** – Developed the Python code for resume generation and contributed to the MySQL backend integration. Link: https://github.com/shreerenu/Reso-Whisper

- **Tanmayee** – Developed the Python code for resume generation, contributed to the MySQL backend integration, and created the demo video. Link: https://github.com/TanmayeeMahesh/ResuWhisper.git

- **Uzair** – Assisted in report writing and designed the application logo. Link: https://github.com/ssyeduzairr/AI-resume

- **Vaidhav** – Developed the fully functional application, worked on UI/UX design, and contributed to the Python code for resume generation. Link: https://github.com/vaidhav-adit/ResuWhisper-AI

17) Impact of the Solution

17.1) Who Benefits from This Project?

- **Students and Fresh Graduates**
    - The primary beneficiaries are students and fresh graduates who **lack experience in resume building and job applications**.
    - This tool helps them **structure their resumes professionally**, **understand industry expectations**, and **prepare for interviews with AI-generated questions**.
- **Job Seekers Looking for Career Transitions**
    - Individuals seeking **career shifts** can benefit from personalized **resume suggestions and job recommendations** that align with their **current skills and future career goals**.

    - The probability-based **job categorization** (Safety, Moderate, Reach) helps them **strategically apply for jobs**, increasing their chances of securing a suitable position.

- **Recruiters and Hiring Managers**

    - The structured resume format ensures that job applications are **concise, professional, and ATS (Applicant Tracking System) friendly**, reducing the manual effort required for screening resumes.

    - The AI-generated interview questions can also be utilized by recruiters for **evaluating candidates more effectively**.

- **Universities and Career Guidance Centers**

    - Universities can **integrate this tool into their career counseling programs** to help students **create job-ready resumes and prepare for interviews**.

    - Career counselors can use the probability-based job recommendations to **guide students on where to apply based on their qualifications and industry trends**.

17.2) Expected Real-World Impact

- **Bridging the Gap Between Education and Employment**

    - Many fresh graduates struggle with **resume formatting and job applications** due to lack of industry exposure. This tool **simplifies the process**, helping students transition smoothly into the workforce.

- **Time-Saving and Increased Efficiency**

    - Job seekers can **quickly generate a well-structured resume**, get **industry-specific interview questions**, and **access relevant job**

listings, reducing the **time spent on manual job searching**.

- **AI-Powered Career Guidance**
    - Unlike traditional resume templates, this AI-driven approach **personalizes recommendations** based on user inputs, **enhancing job application strategies**.
    - The probability-based job suggestions **empower users with data-driven insights** on where they are most likely to secure employment.
- **Accessibility and Inclusivity**
    - The **speech-to-text feature** makes the tool accessible to individuals **who may struggle with typing or structuring a resume manually**.
    - By **automating and simplifying complex processes**, the project ensures that **a wider audience, including non-tech-savvy users, can benefit from AI-powered job search assistance**.

18) Future Enhancements

18.1) Enhanced AI Personalization for Resumes

- Introduce **dynamic resume templates** tailored to different industries (e.g., **tech, finance, healthcare**).
- Use **LLM fine-tuning** to generate resumes that match **specific job descriptions**, improving applicant tracking system (ATS) compatibility.

18.2) AI-Powered Cover Letter Generation

- Extend AI capabilities to generate **customized cover letters** alongside resumes.

- Enable users to input **a target job description**, allowing AI to craft **role-specific cover letters** that highlight key strengths.

18.3) More Advanced Job Recommendation System

- Improve **job categorization algorithms** by integrating **machine learning models** that analyze real-world job market trends.

- Utilize **LinkedIn API or job board APIs** (e.g., Indeed, Glassdoor) to fetch **real-time job openings** that align with the user's profile.

- Incorporate **user feedback loops** to refine recommendations based on previous applications and interview outcomes.

18.4) Resume Performance Analytics

- Allow users to **upload their resume** and receive **AI-powered feedback** on **readability, formatting, keyword optimization, and overall effectiveness**.

- Provide a **score-based evaluation** with suggestions for improvement.

19) Conclusion:

Throughout the development of the **Resume and Interview Question Generating Chatbot**, our team gained **valuable insights into AI integration, system architecture, and real-world implementation challenges**. The project required us to **combine multiple technologies**, including **Gemini API, Jupyter Notebook, Streamlit, and MySQL**, to create a functional and user-friendly application.

One of the **biggest technical lessons** was the **importance of structuring a project before diving into implementation**. Developing a **clear project flow, architecture diagram, and feature breakdown** helped us anticipate

challenges early. Despite this preparation, we encountered **unexpected hurdles**, particularly in integrating **Streamlit's UI/UX constraints with our backend functionality**. Formatting errors, **string parsing issues**, and **resume autofill complications** required multiple debugging sessions and forced us to think critically about **error handling and data preprocessing**.

Overall, this project **enhanced our problem-solving skills, strengthened our technical expertise in AI integration, and taught us the importance of adaptability** when working with real-world constraints. It reinforced the necessity of **thorough testing, modular development, and iterative debugging** in AI-based applications. Moving forward, these learnings will be invaluable in tackling **more complex AI-driven projects** in both academic and professional settings.

20) References

- **ChatGPT** – Assisted in report formatting, refining content, and providing overall guidance throughout the project.
- **Grok AI** – Played a key role in structuring the code, debugging errors, and optimizing the MySQL backend connection.
- **EnhanceCV** (https://enhancv.com/) – Served as a benchmark for resume-building features, helping us analyze competitor functionalities and improve our application's design, user experience, and feature set.