Personality Test Project Documentation

Overview

This project is a web-based personality test application that uses a Flask backend to serve questions from Excel datasets and process user responses, including text and audio inputs. The frontend, built with HTML, CSS, and JavaScript, provides an interactive chat-like interface with a mode selection (Adult or Child), question display, and response submission options. The application supports both text input and voice recording, with transcription handled by the Whisper model for audio responses.

- **Purpose**: Conduct personality tests with adaptive modes (Adult and Child) and store user responses in an Excel file.
- **Technologies Used**:
  - Backend: Python, Flask, Pandas, Whisper (speech-to-text).
  - Frontend: HTML5, CSS3, JavaScript (with jQuery).
  - External Libraries: Google Fonts (Roboto), Font Awesome (icons).
- **Key Features**:
  - Mode selection (Adult/Child) with different question formats (text for adults, images for children).
  - Text and voice-based response submission with audio transcription.
  - Real-time question loading and response saving to an Excel file.
  - Responsive chat interface with animations.
  - Mute/unmute functionality for text-to-speech.

Project Structure

```
project_root/
├── templates/
│   └── index.html          # Frontend HTML file with inline CSS and JavaScript
├── images/                 # Directory for child mode image options (must exist,
even if empty)
├── Personality_Test.xlsx   # Excel file containing adult mode questions
├── Child_Mode_Questions.xlsx # Excel file containing child mode questions
├── User_Responses.xlsx     # Output file where user responses are saved (created
automatically)
└── main.py                 # Flask backend application (previously app.py)
```

Required Files

- **Personality_Test.xlsx**: Contains questions for adult mode. Expected format:
  - Column 1: Question (e.g., "How do you feel today?")
  - Columns 2-5: Options (e.g., "Happy", "Sad", "Angry", "Neutral").
- **Child_Mode_Questions.xlsx**: Contains questions for child mode. Expected format:
  - Column 1: Question (e.g., "Which animal do you like?")
  - Columns 2-5: Image filenames (e.g., "lion.jpg", "elephant.jpg", "giraffe.jpg", "bear.jpg").
- **images/**: Directory where image files (referenced in Child_Mode_Questions.xlsx) must be stored.
- **User_Responses.xlsx**: Automatically generated to store user responses with columns: Mode, Question, Options, User_Answer, Timestamp.

Prerequisites

To run this project, you need the following:

Software

- **Python 3.x**: Install Python (e.g., Python 3.9 or later) from python.org.
- **pip**: Python package manager (comes with Python).
- **FFmpeg**: Required for audio processing by Whisper.
  - Download from ffmpeg.org.
  - Add to system PATH (e.g., C:\ffmpeg\bin on Windows).
- **Web Browser**: A modern browser (e.g., Chrome, Firefox) with support for Web Speech API and MediaRecorder.

Python Dependencies

Install the following Python packages using pip:

```
pip install flask pandas flask-cors whisper openpyxl
```

- **flask**: Web framework for the backend.
- **pandas**: For reading/writing Excel files.
- **flask-cors**: For handling Cross-Origin Resource Sharing.
- **whisper**: For speech-to-text transcription of audio.
- **openpyxl**: For Excel file operations with Pandas.

Hardware

- **Microphone**: Required for audio recording.
- **Internet Connection**: Needed initially to download the Whisper model.

Setup Instructions

1. **Create Project Directory**:
   - Create a folder (e.g., personality-test) and place all project files inside it.
2. **Prepare Excel Files**:
   - Ensure Personality_Test.xlsx and Child_Mode_Questions.xlsx are in the project root with the correct format.
   - Create an images folder in the project root and place any images referenced in Child_Mode_Questions.xlsx.
3. **Install FFmpeg**:
   - Download FFmpeg from ffmpeg.org.
   - Extract it to a folder (e.g., C:\ffmpeg).
   - Add the bin folder to your system PATH:
     - On Windows: Edit environment variables, add C:\ffmpeg\bin to PATH.
     - On macOS/Linux: Add export PATH=$PATH:/path/to/ffmpeg/bin to your shell configuration (e.g., .bashrc).
4. **Install Python Dependencies**:
   - Open a terminal in the project directory.
   - Run the pip command above to install required packages.
5. **Run the Application**:
   - In the terminal, run: python main.py.
   - Open a browser and navigate to http://localhost:5000.

File Descriptions and Code Explanation

1. main.py (Flask Backend)

This file contains the Flask application logic, handling API endpoints for mode selection, question retrieval, answer submission, audio transcription, and static file serving.

Code Explanation

```python
from flask import Flask, request, jsonify, render_template, send_from_directory
import pandas as pd
from flask_cors import CORS
import os
import logging
import whisper
import tempfile
import time

# Configure paths
os.environ["PATH"] += os.pathsep + "C:\\ffmpeg\\bin"
app = Flask(__name__, template_folder="templates")
CORS(app, resources={r"/*": {"origins": "*"}})

# Configure logging
logging.basicConfig(level=logging.DEBUG)

# Load Whisper model
model = whisper.load_model("base")
```

- **Imports**: Libraries for Flask (web framework), Pandas (Excel handling), CORS (cross-origin requests), Whisper (speech-to-text), and other utilities.
- **Path Configuration**: Adds FFmpeg to the system PATH for audio processing.
- **Flask App Setup**: Initializes the Flask app with the templates folder for HTML files.
- **CORS**: Allows requests from any origin (useful for development).
- **Logging**: Sets up logging at DEBUG level for detailed output.
- **Whisper Model**: Loads the "base" Whisper model for audio transcription.

```
# Load datasets and initialize responses
try:
    adult_df = pd.read_excel("Personality_Test.xlsx")
    child_df = pd.read_excel("Child_Mode_Questions.xlsx")
    responses_file = "User_Responses.xlsx"
    if os.path.exists(responses_file):
        os.remove(responses_file)
        app.logger.debug(f"Deleted existing {responses_file}")
    responses_df = pd.DataFrame(columns=['Mode', 'Question', 'Options', 'User_Answer',
'Timestamp'])
except Exception as e:
    logging.error(f"Error during initialization: {str(e)}")
    raise
```

- **Data Loading**: Reads the adult and child question datasets into Pandas DataFrames.
- **Response File Initialization**: Deletes any existing User_Responses.xlsx and creates a new DataFrame with columns for saving user responses.

```
# Application state
current_mode = "adult"
user_responses = []
question_index = 0
```

State Variables:

- **current_mode**: Tracks whether the test is in "adult" or "child" mode.
- **user_responses**: List to store user answers temporarily.
- **question_index**: Tracks the current question index.

```python
def save_response_to_excel(mode, question, options, answer):
    global responses_df
    try:
        new_response = pd.DataFrame({
            'Mode': [mode],
            'Question': [question],
            'Options': [', '.join(str(opt) for opt in options)],
            'User_Answer': [answer],
            'Timestamp': [pd.Timestamp.now()]
        })
        responses_df = pd.concat([responses_df, new_response], ignore_index=True)
        responses_df.to_excel(responses_file, index=False)
        app.logger.debug(f"Saved response to {responses_file}")
    except Exception as e:
        app.logger.error(f"Error saving to Excel: {str(e)}")
```

- **Function**: save_response_to_excel
- **Purpose**: Saves a user's response to User_Responses.xlsx.
- **Explanation**:
    - Creates a new DataFrame row with the response details.
    - Concatenates it with the global responses_df.
    - Writes the updated DataFrame to Excel.
    - Logs success or errors.

```python
@app.route('/')
def home():
    return render_template('index.html')
```

- **Route**: /
- **Purpose**: Serves the main page.
- **Explanation**: Renders index.html to display the personality test interface.

```python
@app.route('/set_mode', methods=['POST'])
def set_mode():
    global current_mode, question_index, user_responses
    try:
        data = request.get_json()
        current_mode = "child" if data.get('child_mode', False) else "adult"
        question_index = 0
        user_responses = []
        app.logger.debug(f"Mode changed to: {current_mode}")
        return jsonify({"status": "success", "mode": current_mode})
    except Exception as e:
        app.logger.error(f"Mode change error: {str(e)}")
        return jsonify({"error": str(e)}), 500
```

- **Route**: /set_mode
- **Purpose**: Sets the test mode (adult or child).
- **Explanation**:
    - Parses JSON data to check if child_mode is true.
    - Updates current_mode, resets question_index and user_responses.
    - Returns a success JSON response or an error if something fails.

```python
@app.route('/get_question', methods=['GET'])
def get_question():
    global question_index
    try:
        df = child_df if current_mode == "child" else adult_df
        if question_index >= len(df):
            return jsonify({"status": "completed"})
        question = df.iloc[question_index]['Question']
        if current_mode == "child":
            image_filenames = df.iloc[question_index, 1:5].dropna().tolist()
            image_paths = [f"/images/{filename.strip()}" for filename in
image_filenames]
            response = {
                "question": question,
                "options": image_paths,
                "voice_options": [],
                "mode": current_mode,
                "index": question_index
            }
        else:
            options = df.iloc[question_index, 1:5].dropna().tolist()
            response = {
                "question": question,
                "options": options,
                "voice_options": options,
                "mode": current_mode,
                "index": question_index
            }
        app.logger.debug(f"Serving question {question_index}: {question}")
        return jsonify(response)
    except Exception as e:
        app.logger.error(f"Get question error: {str(e)}")
        return jsonify({"error": str(e)}), 500
```

Route: /get_question

- **Purpose**: Retrieves the next question.
- **Explanation**:

  - Selects the correct DataFrame based on mode.
  - If all questions are answered, returns a "completed" status.
  - For child mode, maps image filenames to paths; for adult mode, uses text options.
  - Returns a JSON object with the question, options, and mode.

```python
@app.route('/submit_answer', methods=['POST'])
def submit_answer():
    global question_index
    try:
        data = request.get_json()
        answer = data.get('answer')
        df = child_df if current_mode == "child" else adult_df
        question = df.iloc[question_index]['Question']
        options = df.iloc[question_index, 1:5].tolist()
        user_responses.append(answer)
        save_response_to_excel(current_mode, question, options, answer)
        question_index += 1
        return jsonify({"status": "success", "next": question_index < len(df)})
    except Exception as e:
        app.logger.error(f"Submit answer error: {str(e)}")
        return jsonify({"error": str(e)}), 500
```

- **Route**: /submit_answer
- **Purpose**: Processes a user's answer.
- **Explanation**:
    - Extracts the answer from JSON.
    - Retrieves the current question and options.
    - Saves the answer to Excel and increments question_index.
    - Returns whether more questions exist.

```python
@app.route('/transcribe', methods=['POST'])
def transcribe():
    try:
        if 'file' not in request.files:
            return jsonify({"error": "No audio file provided"}), 400
        audio_file = request.files['file']
        temp_file = tempfile.NamedTemporaryFile(delete=False, suffix='.webm')
        temp_path = temp_file.name
        audio_file.save(temp_path)
        temp_file.close()
        result = model.transcribe(temp_path)
        max_attempts = 5
        for attempt in range(max_attempts):
            try:
                os.unlink(temp_path)
                break
            except PermissionError:
                if attempt < max_attempts - 1:
                    time.sleep(0.1)
                    continue
                app.logger.warning(f"Could not delete temporary file: {temp_path}")
        app.logger.debug(f"Transcription result: {result['text']}")
        return jsonify({"text": result['text']})
    except Exception as e:
        app.logger.error(f"Transcription error: {str(e)}")
        return jsonify({"error": str(e)}), 500
```

- **Route**: /transcribe
- **Purpose**: Transcribes an audio file.
- **Explanation**:
    - Validates the presence of an audio file.
    - Saves it to a temporary file, transcribes it with Whisper, and deletes the file.
    - Uses retry logic for file deletion to handle PermissionError.
    - Returns the transcribed text.

```python
@app.route('/images/<path:filename>')
def serve_static(filename):
    return send_from_directory('images', filename)
```

- **Route**: /images/<path:filename>
- **Purpose**: Serves static image files.
- **Explanation**: Delivers images from the images folder for child mode questions

```python
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

- **Purpose**: Runs the Flask app.
- **Explanation**: Starts the server in debug mode, accessible on all network interfaces at port 5000.

2. index.html (Frontend)

This file provides the user interface for the personality test, with inline CSS and JavaScript for styling and interactivity.

Summary of Frontend Code

The index.html file is a standard HTML5 document that integrates a responsive, interactive frontend for the personality test. Below is a high-level summary of its components and functionality:

- **Structure and Dependencies**:
  - A basic HTML5 document with a UTF-8 charset and title "Personality Test".
  - Imports external resources: Google Fonts (Roboto), Font Awesome (icons), and jQuery (for AJAX and DOM manipulation).
- **Styling (Inline CSS)**:
  - Applies global styles with a gradient background, Roboto font, and flexbox centering.
  - Designs a start screen modal with a toggle switch for Adult/Child mode selection, styled with animations and hover effects.
  - Creates a chat box layout with a header, scrollable message area, options grid (for text or images), and input section, using flexbox for responsiveness.
  - Styles message bubbles (blue for bot, gray for user) and option cards with hover effects.

- **HTML Layout**:
  - **Start Screen**: A full-screen modal with a title, mode toggle (Adult/Child), and a "Start the Test" button. Initially visible, hides when the test begins.
  - **Chat Box**: Hidden initially, shown after starting the test. Includes:
    - A header with a mute/unmute button.

        - A message area for bot and user messages.
        - An options grid for displaying question options (text for adults, images for children).
        - An input section with a text field, record button, and audio preview controls (play, delete, send).
- **JavaScript Functionality (Inline)**:
    - Initializes variables for audio recording, mode tracking (Adult/Child), and mute state.
    - Starts the test by setting the mode via an AJAX call to /set_mode, hiding the start screen, and loading the first question.
    - Loads questions dynamically via /get_question, displaying them as bot messages and showing options (text or images) in a clickable grid.
    - Submits answers via /submit_answer, either by clicking options, typing in the text field, or recording audio and transcribing it via /transcribe.
    - Implements text-to-speech using the Web Speech API to read questions and options aloud, with a mute/unmute toggle.
    - Handles audio recording using the MediaRecorder API, allowing users to record, preview, and send audio for transcription.
    - Ends the test with a thank-you message when all questions are answered.
- **Additional Script**:
    - Contains a Cloudflare security script (likely for bot protection), which can be removed or configured based on deployment needs.

Running the Application

- Run python main.py in the terminal.
- Open http://localhost:5000 in a browser.
- Select the mode (Adult/Child) and start the test.
- Answer questions via text, option clicks, or voice recording.
- Responses are saved to User_Responses.xlsx.

Testing

- **Mode Switching**: Toggle between Adult and Child modes to verify question types (text vs. images).
- **Audio Recording**: Test recording and transcription with a microphone.
- **Excel Output**: Check User_Responses.xlsx for saved responses.
- **Mute/Unmute**: Test the mute button to ensure speech toggles correctly.

Potential Improvements

- **Authentication**: Add user login to restrict access.
- **Database**: Replace Excel with a database (e.g., SQLite) for scalability.
- **Error Handling**: Improve error messages and retry logic.
- **Accessibility**: Add ARIA labels and improve contrast.

- **Styling**: Extract inline CSS to a separate file for better maintainability.

Troubleshooting

- **Audio Issues**: Ensure microphone permissions are granted and FFmpeg is in PATH.
- **Excel Errors**: Verify file paths and formats.
- **CORS Errors**: Adjust CORS settings if deploying to a different origin.