**# Setting Up and Running the Unified Accessibility Assistant**

This guide provides detailed instructions for setting up the environment and running the Unified Accessibility Assistant, a Flask-based web application designed to enhance accessibility for individuals with visual and hearing impairments. The application includes features like text-to-Indian Sign Language (ISL) conversion, text-to-speech with emotion detection, speech-to-text translation, and an AI voice assistant.

**## Prerequisites**

Before setting up the environment, ensure you have the following installed on your system:

- **\*\*Python 3.8 or higher\*\***: The application uses Python for the backend. Download and install Python from [python.org](https://www.python.org/downloads/). Verify the installation by running:

  ```bash

  python --version

  ```

- **\*\*Node.js (optional)\*\***: Only required if you need to manage front-end dependencies or use npm for additional tools. Download from [nodejs.org](https://nodejs.org/).

- **\*\*Git\*\***: For cloning repositories or managing version control. Install from [git-scm.com](https://git-scm.com/downloads).

- **\*\*A modern web browser\*\***: Chrome is recommended for speech recognition support.

- **\*\*A code editor\*\***: VS Code, PyCharm, or any preferred editor for editing files.

- **\*\*Internet access\*\***: Required for downloading dependencies and accessing APIs.

**## Step 1: Clone or Set Up the Project Directory**

1. **\*\*Create a project directory\*\***:

   Create a directory for the project, e.g., `unified-accessibility-assistant`.

   ```bash

   mkdir unified-accessibility-assistant

   cd unified-accessibility-assistant

   ```

2. **\*\*Copy the provided files\*\***:

   Place the provided files (`emotion\_detection.py`, `app.py`, `requirements.txt`, `index.html`, `style.css`, `script.js`, and `ISL\_Dictionary\_words.xlsx`) in the project directory. Ensure the directory structure is as follows:

   ```

   unified-accessibility-assistant/

   ├── static/

   │   ├── audio/            # Will be created automatically for audio output

   │   ├── videos/           # Directory for ISL video files

   │   └── css/

   │       └── style.css

   ├── templates/

   │   └── index.html

   ├── emotion\_detection.py

   ├── app.py

   ├── requirements.txt

   ├── ISL\_Dictionary\_words.xlsx

   └── script.js

   ```

3. **\*\*Set up the `static/videos/` directory\*\***:

   The application relies on video files for ISL conversion (e.g., `A.mp4`, `B.mp4`, etc., as listed in `ISL\_Dictionary\_words.xlsx`). Ensure these video files are placed in the `static/videos/` directory. If you don’t have these videos, you can:

   - Obtain them from the source specified in `ISL\_Dictionary\_words.xlsx`.

   - Create placeholder videos for testing (e.g., short MP4 files named `A.mp4`, `B.mp4`, etc.).

   - Note: Without these videos, the text-to-ISL feature will fall back to letter-by-letter conversion, which may still work if individual letter videos are present.

   Create the `static/videos/` directory:

   ```bash

   mkdir -p static/videos

   ```

4. **\*\*Set up the `templates/` directory\*\***:

   Flask uses the `templates/` directory to store HTML files. Create it and move `index.html` there:

   ```bash

   mkdir templates

   mv index.html templates/

   ```

5. **\*\*Set up the `static/css/` directory\*\***:

   Create the `static/css/` directory and move `style.css` there:

   ```bash

   mkdir -p static/css

   mv style.css static/css/

   ```

6. **\*\*Move `script.js`\*\***:

   Place `script.js` in the `static/` directory:

   ```bash

   mv script.js static/

   ```

**## Step 2: Set Up the Python Environment**

1. **\*\*Create a virtual environment\*\***:

   To avoid conflicts with system-wide Python packages, create a virtual environment:

   ```bash

   python -m venv venv

   ```

2. **\*\*Activate the virtual environment\*\***:

   - On Windows:

     ```bash

     venv\Scripts\activate

     ```

   - On macOS/Linux:

     ```bash

     source venv/bin/activate

     ```

   After activation, your terminal prompt should indicate the virtual environment (e.g., `(venv)`).

3. **\*\*Install dependencies\*\***:

   The `requirements.txt` file lists all required Python packages. Install them using pip:

   ```bash

   pip install -r requirements.txt

   ```

   This installs the following packages:

   - `flask`: Web framework for the backend.

   - `flask\_cors`: Enables Cross-Origin Resource Sharing.

   - `speechrecognition`: For speech-to-text functionality.

   - `googletrans==4.0.0-rc1`: For text translation.

   - `gTTS`: For text-to-speech conversion.

   - `python-dotenv`: For loading environment variables.

   - `pywhatkit`: For YouTube playback.

   - `wikipedia`: For fetching Wikipedia summaries.

   - `pyjokes`: For generating jokes.

   - `cohere`: For AI-powered responses.

   - `requests`: For making HTTP requests (e.g., weather API).

   - `transformers`: For emotion detection using a DistilBERT model.

   - `torch==2.4.1`: PyTorch, required for the transformers library.

   If you encounter issues with `torch`, ensure your system is compatible (e.g., CPU or GPU support). For CPU-only systems, you can install a CPU-specific version:

   ```bash

   pip install torch==2.4.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

   ```

**## Step 3: Configure Environment Variables**

The application requires API keys for Cohere and OpenWeatherMap, stored in a `.env` file.

1. **\*\*Create a `.env` file\*\***:

   In the project root, create a file named `.env`:

   ```bash

   touch .env

   ```

2. **\*\*Add API keys\*\***:

   Obtain the following API keys:

   - **\*\*Cohere API Key\*\***: Sign up at [cohere.ai](https://cohere.ai/) to get an API key for AI-powered responses.

   - **\*\*OpenWeatherMap API Key\*\***: Sign up at [openweathermap.org](https://openweathermap.org/) to get an API key for weather data.

   Add these keys to the `.env` file:

   ```

   COHERE\_API\_KEY=your\_cohere\_api\_key\_here

   WEATHER\_API\_KEY=your\_openweathermap\_api\_key\_here

   ```

   Replace `your\_cohere\_api\_key\_here` and `your\_openweathermap\_api\_key\_here` with your actual API keys.

3. **\*\*Verify the `.env` file\*\***:

   Ensure the `.env` file is in the project root and not committed to version control (add `.env` to `.gitignore` if using Git).

**## Step 4: Verify Directory Structure**

Before running the application, confirm the directory structure:

```

unified-accessibility-assistant/

├── venv/                    # Virtual environment

├── static/

│   ├── audio/              # Created automatically for audio output

│   ├── videos/             # ISL video files (A.mp4, B.mp4, etc.)

│   ├── css/

│   │   └── style.css

│   └── script.js

├── templates/

│   └── index.html

├── emotion\_detection.py

├── app.py

├── requirements.txt

├── ISL\_Dictionary\_words.xlsx

└── .env

```

**## Step 5: Run the Application**

1. **\*\*Activate the virtual environment\*\*** (if not already active):

   ```bash

   source venv/bin/activate  # macOS/Linux

   venv\Scripts\activate     # Windows

   ```

2. **\*\*Run the Flask application\*\***:

   From the project root, run:

   ```bash

   python app.py

   ```

   This starts the Flask development server on `http://localhost:5500`. You should see output like:

   ```

    \* Serving Flask app 'app'

    \* Debug mode: on

    \* Running on http://127.0.0.1:5500

   ```

3. **\*\*Access the application\*\***:

   Open a web browser and navigate to `http://localhost:5500`. The homepage (`index.html`) should load, displaying the Unified Accessibility Assistant interface.

**Till now if everything is ok then app should run**

**## Step 6: Test the Application Features**

The application provides several accessibility features. Test them as follows:

1. **\*\*Text-to-ISL Converter\*\***:

   - Enter text (e.g., “STAY HOME”) in the “Universal Text Input” field.

   - Click “Convert to ISL”.

   - Ensure the `static/videos/` directory contains the required MP4 files (e.g., `STAY.mp4` or individual letter videos like `S.mp4`, `T.mp4`, etc.).

   - The output should display videos for each word or letter in sequence.

2. **\*\*Text-to-Audio with Emotion Detection\*\***:

   - Enter text in the “Universal Text Input” field (e.g., “I am so happy today!”).

   - Click “Convert to Audio”.

   - The application detects emotions using the DistilBERT model and adjusts speech parameters (rate and volume) accordingly. Check the “Detected Emotions” and “Speech Parameters” outputs, and listen to the generated audio.

3. **\*\*Speech-to-Text Translation with TTS\*\***:

   - Enter text in the “Universal Text Input” field.

   - Select a target language (e.g., Kannada) from the dropdown.

   - Click “Translate & Speak”.

   - Verify the translated text and listen to the audio output in the selected language.

4. **\*\*AI Voice Assistant\*\***:

   - Enter a query (e.g., “What is the weather?” or “Play a song”) in the “AI Voice Assistant” input field.

   - Click “Speak” or use the microphone for speech input (Chrome recommended).

   - Check the response text and audio output.

5. **\*\*Speech Recognition\*\***:

   - Click the microphone icon next to the “Universal Text Input” or “AI Voice Assistant” input.

   - Speak clearly (e.g., “Hello world”).

   - The input field should populate with the transcribed text. For the AI Voice Assistant, it will also trigger a response.

**## Step 7: Troubleshooting Common Issues**

- **\*\*ModuleNotFoundError\*\***:

  Ensure all dependencies are installed (`pip install -r requirements.txt`). If a specific module fails, install it individually (e.g., `pip install flask`).

- **\*\*API Key Errors\*\***:

  Verify that the `.env` file contains valid `COHERE\_API\_KEY` and `WEATHER\_API\_KEY`. Check for typos or expired keys.

- **\*\*Video Files Missing\*\***:

  If ISL videos are missing, the converter will attempt to display letter-by-letter videos. Ensure `static/videos/` contains the necessary MP4 files or create placeholders for testing.

- **\*\*Speech Recognition Not Working\*\***:

  Use Google Chrome, as it supports the Web Speech API. Ensure your microphone is enabled and permissions are granted.

- **\*\*gTTS Language Errors\*\***:

  The `gTTS` library supports specific language codes (e.g., `en` for English, `kn` for Kannada). Check supported languages using:

  ```python

  from gtts.lang import tts\_langs

  print(tts\_langs())

  ```

- **\*\*Port Conflicts\*\***:

  If port 5500 is in use, change the port in `app.py` by modifying the `app.run` line, e.g.:

  ```python

  app.run(debug=True, port=5000)

  ```

- **\*\*CORS Issues\*\***:

  The application uses `flask\_cors` to allow cross-origin requests. If you encounter CORS errors, ensure `CORS(app)` is correctly configured in `app.py`.

**## Step 8: Additional Notes**

- **\*\*Performance Considerations\*\***:

  - The DistilBERT model (`transformers`) may require significant memory. For low-memory systems, consider running on a machine with at least 4GB RAM.

  - Ensure a stable internet connection for API calls (Cohere, OpenWeatherMap).

- **\*\*Security\*\***:

  - Do not expose API keys in public repositories. Keep the `.env` file secure.

  - The Flask debug mode (`debug=True`) is enabled for development. Disable it in production (`debug=False`) to prevent exposing sensitive information.

- **\*\*Customization\*\***:

  - Modify `CITY\_NAME` in `app.py` to change the default weather location (currently set to “Pune”).

  - Adjust `EMOTION\_PARAMS` in `app.py` to fine-tune speech parameters for different emotions.

  - Update `index.html` and `style.css` to customize the UI or add new features.

- **\*\*Extending the Application\*\***:

  - Add more ISL videos to `static/videos/` to support additional words.

  - Integrate additional APIs (e.g., for news or calendar events) in `app.py`.

  - Enhance `script.js` to add more interactive features, such as real-time feedback or animations.

**## Step 9: Running in Production**

For production deployment, consider the following:

- Use a production-ready WSGI server like Gunicorn:

  ```bash

  pip install gunicorn

  gunicorn --bind 0.0.0.0:5500 app:app

  ```

- Deploy on a cloud platform (e.g., Heroku, AWS, or Render).

- Serve static files using a web server like Nginx or Apache.

- Secure the application with HTTPS and proper CORS policies.