## January 20

## Establish the directory of the python interpreter

In VSCode

1. **View** -> **Command Palette**
2. *Python: Select Interpreter*
3. **Enter interpreter path**
4. **Browse your file system to find a Python interpreter**
5. Browse to: *C:\Users\andrew.dilley\development\chatbot3\venv\Scripts\python.exe*

Note: can use this Python interpreter across multiple development directory structures

## Creating a new chatbot development environment

### Establish the directory structure

1. Copy the last *chatbot* directory and paste it into the *development* directory
2. Rename it to *chatbot<next number in the sequence>*
3. Delete the *venv* directory

### Create and activate the environment

1. Open a command prompt (cmd)
2. Go to the newly created development directory
3. Create the environment with python -m venv venv
4. deactivate
5. Activate the environment with *venv\Scripts\activate*

hello

In Flask I would like to develop an app which has two panes separated by a button labelled Translate.

The idea is that the end user copies and pastes text into the left hand pane (or uploads a Word document which has content that they have written). Either ay, they can see their content in this pane.

When they click translate, the app rewrites their content based on the "Our Voice Guide - January 2023.PDF"

The translated content then appears in the right hand pane.

There is a copy icon which will copy this content to their notepad, ready for pasting into another document.

Could you please suggest how this may be achieved?

thanks!

To build the described app in Flask, here’s a suggested implementation approach:

**Key Features**

1. **User Interface (UI):**
   * Two text panes (left and right).
   * A "Translate" button between the panes.
   * File upload capability for .docx documents.
   * A copy-to-clipboard icon for the translated text.
2. **Backend Logic:**
   * Parse the uploaded document or use pasted content.
   * Implement text rewriting based on the rules in the *Our Voice Guide - January 2023.PDF*.
   * Use GPT-4 for the rewrite logic while adhering to the guide’s rules.
3. **Steps to Build the App:**

**1. Setup Flask**

Create a new Flask project:

pip install flask flask-wtf flask-cors python-docx PyPDF2

**2. Frontend (HTML/JS)**

Create an index.html with the following layout:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Translate App</title>

<style>

body { font-family: Arial, sans-serif; margin: 20px; }

textarea, button, input { margin: 10px 0; width: 100%; }

textarea { height: 200px; }

.container { display: flex; gap: 10px; }

.pane { flex: 1; display: flex; flex-direction: column; }

button { height: 40px; }

</style>

</head>

<body>

<h1>Translate Your Content</h1>

<div class="container">

<div class="pane">

<textarea id="input-text" placeholder="Paste your text here or upload a file"></textarea>

<input type="file" id="file-upload" accept=".docx">

</div>

<div>

<button id="translate-btn">Translate</button>

</div>

<div class="pane">

<textarea id="output-text" placeholder="Translated content will appear here"></textarea>

<button id="copy-btn">Copy to Clipboard</button>

</div>

</div>

<script>

document.getElementById("copy-btn").addEventListener("click", () => {

navigator.clipboard.writeText(document.getElementById("output-text").value);

alert("Content copied to clipboard!");

});

</script>

</body>

</html>

**3. Backend (Flask Logic)**

from flask import Flask, request, jsonify, render\_template

from werkzeug.utils import secure\_filename

import os

import docx

app = Flask(\_\_name\_\_)

app.config['UPLOAD\_FOLDER'] = 'uploads'

def rewrite\_text(text):

# Implement the rewrite logic based on Our Voice Guide

# Placeholder for GPT logic

return text # Replace this with GPT-4-generated output

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/translate', methods=['POST'])

def translate():

if 'file' in request.files:

file = request.files['file']

if file.filename.endswith('.docx'):

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], secure\_filename(file.filename))

file.save(filepath)

doc = docx.Document(filepath)

text = "\n".join([para.text for para in doc.paragraphs])

else:

return jsonify({"error": "Unsupported file format"}), 400

else:

text = request.form.get('text')

if text:

translated\_text = rewrite\_text(text)

return jsonify({"translated": translated\_text})

return jsonify({"error": "No text provided"}), 400

if \_\_name\_\_ == '\_\_main\_\_':

os.makedirs(app.config['UPLOAD\_FOLDER'], exist\_ok=True)

app.run(debug=True)

**4. Rewriting Logic**

Extract the rules from the *Our Voice Guide* into programmatic transformations:

* Replace complex words with simpler ones.
* Maintain tone guidelines (informative, neighborly, dynamic).
* Adhere to format rules (active voice, inclusive language).

**5. Testing**

* Test the app locally by running the Flask server (python app.py).
* Use GPT APIs for rewriting while integrating the rules from the PDF.

Would you like a specific part of this implementation detailed further?

Here’s a refined and actionable GPT-4 prompt tailored to Wannon Water’s *Our Voice Guide - January 2023*. This can be used to ensure GPT-4 rewrites content in line with the brand's guidelines:

**GPT-4 Prompt:**

You are an expert copy editor and tone specialist working for Wannon Water. Your task is to rewrite the provided text to align with Wannon Water's *Our Voice Guide - January 2023*. Adhere to the following guidelines strictly:

**1. Tone and Style**

* **Informative:** Write clearly and concisely, ensuring the message is easy to understand. Avoid jargon and technical language; if technical terms are necessary, explain them simply.
* **Neighborly:** Use warm, inclusive language that feels genuine and relatable. Avoid overly formal or colloquial expressions.
* **Local Leader:** Be dynamic and confident. Use active voice and short, engaging sentences to convey leadership and expertise.

**2. Language Rules**

* Replace complex or formal phrases with simpler alternatives. For example:
  + Replace "commence" with "start" or "begin."
  + Replace "assist" with "help."
* Use inclusive language:
  + Refer to "people with disabilities" instead of "disabled people."
  + Use gender-neutral pronouns like "they" unless a specific gender is relevant.
* Avoid overly complex sentence structures. Aim for an average sentence length of 18 words.

**3. Formatting**

* Use sentence case for headings and subheadings.
* Break content into logical paragraphs with the most important information at the top.
* Use bulleted lists to organize content where appropriate.

**4. Examples**

* **Before:** "The majority of respondents indicated their willingness to participate in the upcoming initiative."
  + **After:** "Most respondents said they’re willing to join the upcoming initiative."
* **Before:** "The community was informed that their water supply would undergo a temporary disruption."
  + **After:** "We let the community know there’ll be a temporary water supply disruption."

**5. Rewrite This Text:**

[Insert User Text Here]

Ensure the rewritten text:

1. Aligns with Wannon Water's tone of being informative, neighborly, and authoritative.
2. Uses simpler and more accessible language.
3. Follows all formatting and language guidelines.

**Steps for Integration in Flask**

**Backend Code Snippet** (modifying the rewrite\_text function):

import openai

def rewrite\_text(input\_text):

prompt = f"""

You are an expert copy editor and tone specialist working for Wannon Water. Your task is to rewrite the provided text to align with Wannon Water's "Our Voice Guide - January 2023". Follow these guidelines:

1. Tone and Style:

- Informative: Write clearly and concisely.

- Neighborly: Use warm and inclusive language.

- Local Leader: Be dynamic and confident.

2. Language Rules:

- Replace complex or formal phrases with simpler alternatives.

- Use inclusive language (e.g., "person with disability" instead of "disabled person").

- Use active voice and avoid jargon.

3. Formatting:

- Use sentence case for headings.

- Break content into logical paragraphs.

- Use bulleted lists where necessary.

Rewrite the following text using these principles:

{input\_text}

"""

response = openai.Completion.create(

engine="text-davinci-003", # or the GPT-4 equivalent

prompt=prompt,

max\_tokens=1000,

temperature=0.7

)

return response["choices"][0]["text"].strip()

This will take user input and generate rewritten content adhering to the voice guide.

Let me know if you’d like further refinements or help setting this up!

## Using GPT-4o Mini engine

If you’re using the GPT-4o Mini engine, you can adjust the implementation to accommodate this model. Below is how you can structure and integrate it for your app.

**Revised Backend Code with GPT-4o Mini**

Here’s how you can modify the rewrite\_text function and integrate it into your Flask app:

**Code Snippet**

import openai

# Function to rewrite text based on Wannon Water's Voice Guide

def rewrite\_text(input\_text):

prompt = f"""

You are an expert copy editor and tone specialist for Wannon Water. Your task is to rewrite the provided text to align with Wannon Water's "Our Voice Guide - January 2023". Follow these guidelines strictly:

### Guidelines:

1. \*\*Tone and Style\*\*

- Informative: Write clearly and concisely, avoiding jargon. If technical terms are required, explain them simply.

- Neighborly: Use warm, inclusive language that feels genuine and approachable.

- Local Leader: Be dynamic, confident, and reassuring. Use active voice and short, engaging sentences.

2. \*\*Language Rules\*\*

- Replace complex or formal phrases with simpler alternatives. For example:

- "Commence" → "Start"

- "Assist" → "Help"

- Use inclusive language. For example:

- "Person with disability" instead of "Disabled person."

- Avoid overly complex sentence structures. Aim for concise sentences (around 18 words).

3. \*\*Formatting\*\*

- Use sentence case for headings and subheadings.

- Break content into logical paragraphs, prioritizing the most important information at the start.

- Use bulleted lists to organize content where appropriate.

### Examples:

- \*\*Before:\*\* "The majority of respondents indicated their willingness to participate in the upcoming initiative."

- \*\*After:\*\* "Most respondents said they’re willing to join the upcoming initiative."

- \*\*Before:\*\* "The community was informed that their water supply would undergo a temporary disruption."

- \*\*After:\*\* "We let the community know there’ll be a temporary water supply disruption."

### Rewrite the following text:

{input\_text}

Ensure the rewritten text aligns with Wannon Water's tone, simplifies language, and adheres to inclusive and clear communication practices.

"""

try:

# Send request to GPT-4o Mini engine

response = openai.ChatCompletion.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "You are a helpful assistant."},

{"role": "user", "content": prompt}

],

max\_tokens=1000,

temperature=0.7

)

# Extract and return the response content

return response['choices'][0]['message']['content'].strip()

except Exception as e:

return f"Error: {str(e)}"

**Key Adjustments for GPT-4o Mini**

1. **API Endpoint**: Ensure your API endpoint uses the openai.ChatCompletion.create method, as the mini engine supports the Chat Completions API.
2. **Model Name**: Use "gpt-4o-mini" in the model parameter.
3. **Prompt Structure**: Adapt the prompt to be conversational by using a messages list that includes a system and user message.

**Flask Route for Translation**

Update your Flask route to process user input and call the rewrite\_text function:

@app.route('/translate', methods=['POST'])

def translate():

try:

if 'file' in request.files:

file = request.files['file']

if file.filename.endswith('.docx'):

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], secure\_filename(file.filename))

file.save(filepath)

doc = docx.Document(filepath)

text = "\n".join([para.text for para in doc.paragraphs])

else:

return jsonify({"error": "Unsupported file format"}), 400

else:

text = request.form.get('text')

if text:

translated\_text = rewrite\_text(text)

return jsonify({"translated": translated\_text})

return jsonify({"error": "No text provided"}), 400

except Exception as e:

return jsonify({"error": str(e)}), 500

**Testing and Deployment**

1. **Run Locally**: Start the app with python app.py and test locally by providing text or uploading .docx files.
2. **Validate Outputs**: Verify that the rewritten text aligns with the tone and guidelines in the *Our Voice Guide - January 2023*.
3. **Optimize for Performance**: Test latency to ensure the GPT-4o Mini engine meets your application's performance needs.

Would you like guidance on setting up API keys, error handling, or deploying this app?

### Testing

Cd *C:\Users\andrew.dilley\development\voice\voice\_app1\*

**In VSCode**

1. **View** -> **Command Palette**
2. *Python: Select Interpreter*
3. **Enter interpreter path**
4. **Browse your file system to find a Python interpreter**
5. Browse to: *C:\Users\andrew.dilley\development\voice\voice\_app1\venv\Scripts\python.exe*

Note: can use this Python interpreter across multiple development directory structures

**Creating a new chatbot development environment**

**Establish the directory structure**

1. Copy the last *chatbot* directory and paste it into the *development* directory
2. Rename it to *chatbot<next number in the sequence>*
3. Delete the *venv* directory

**Create and activate the environment**

1. Open a command prompt (cmd)
2. Go to the newly created development directory
3. Create the environment with python -m venv venv
4. deactivate
5. Activate the environment with *venv\Scripts\activate*

## Version control

### New remote repository

(created online in Github)

<https://github.com/AndrewDilley/OurVoiceGuideTranslator.git>

### initiate a local repository

git init

### Link the local and online repositories

git remote add origin <https://github.com/AndrewDilley/OurVoiceGuideTranslator.git>

git remote -v

origin https://github.com/AndrewDilley/OurVoiceGuideTranslator.git (fetch)

origin https://github.com/AndrewDilley/OurVoiceGuideTranslator.git (push)

### check the status of the local repository

git status

On branch master

No comments yet

### Stage All Changes:

git add .

### check the status of the local repository

git status

On branch master

No comments yet

List of all files that are new

### Committing Changes

git commit -m "Benchmark application"

15 files changed, 267 insertions(+)

create mode 100644 .gitignore

create mode 100644 All samples and responses.docx

create mode 100644 Sample 1.docx

create mode 100644 app.py

create mode 100644 documentation.docx

create mode 100644 guide/DummyVoice.docx

create mode 100644 guide/DummyVoice.pdf

create mode 100644 guide/Wannon Water - Our Voice Guide - January 2023.pdf

create mode 100644 requirements.txt

create mode 100644 static/css/style.css

create mode 100644 static/js/script.js

create mode 100644 templates/index.html

create mode 100644 uploads/Bad\_sample\_1.docx

create mode 100644 uploads/Sample\_communications\_that\_are\_not\_aligned\_with\_the\_WW\_guidelines.docx

### Push Changes to Remote:

git push origin master

Enumerating objects: 22, done.

Counting objects: 100% (22/22), done.

Delta compression using up to 8 threads

Compressing objects: 100% (19/19), done.

Writing objects: 100% (22/22), 1.01 MiB | 1.45 MiB/s, done.

Total 22 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)

remote: Resolving deltas: 100% (1/1), done.

To https://github.com/AndrewDilley/OurVoiceGuideTranslator.git

\* [new branch] master -> master

### Final check

Make sure that these files are now in the online repository

## Jan 21

### Deployment

Here are the detailed steps to create and test a Docker image for your Flask-based AI application:

**Prerequisites**

1. Ensure Docker is installed on your system.
2. Verify that Python and all required dependencies are already tested locally.

**Step 1: Create a Dockerfile**

Create a Dockerfile in the same directory as your application files.

Add the following content:

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory in the container

WORKDIR /app

# Copy requirements.txt to the working directory

COPY requirements.txt .

# Install Python dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code to the working directory

COPY . .

# Expose the port that the app runs on

EXPOSE 5000

# Set environment variables

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

# Run the Flask app

CMD ["flask", "run"]

****

**Step 2: Create a .dockerignore File**

This file will exclude unnecessary files from the Docker build context. Example content:

\_\_pycache\_\_/

\*.pyc

\*.pyo

.env

uploads/

****

**Step 3: Build the Docker Image**

Run the following command to build the Docker image. Replace flask-ai-app with your desired image name:

docker build -t voice-app .

****

**Step 4: Run the Docker Container**

Start a container from the image:

docker run -d -p 5000:5000 --env-file .env voice-app

Notes

* -p 5000:5000: Maps port 5000 on your machine to port 5000 in the container.
* --env-file .env: Passes the environment variables from your .env file to the container.

List all containers (stopped and running)

docker ps -a

****

**Step 5: Test the Application**

1. Open a browser and navigate to http://localhost:5000 to access the application.
2. Test the upload and text rewriting functionalities to ensure everything works as expected.

****

* Except for the ability to upload files
* Rather than debug this I will remove it because it makes the UI less intuitive

**Step 6: Stop and Remove Containers (Cleanup)**

To stop and remove the container:

docker stop c707307cdce5

docker rm c707307cdce5

### remove the upload document function

### First step - version control

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Docker application with document upload function"

### *Push Changes to Remote:*

git push origin master

### Then changes to the application files to remove the document upload function

given the uploaded files, please remove the functionality that allows for a user to upload a document

**ChatGPT said:**

import os

from flask import Flask, request, jsonify, render\_template

from openai import OpenAI

import docx

from dotenv import load\_dotenv

# Load environment variables

load\_dotenv()

# Initialize Flask app

app = Flask(\_\_name\_\_)

app.secret\_key = os.getenv("FLASK\_SECRET\_KEY")

# Set OpenAI API key

client = OpenAI(

api\_key=os.getenv("OPENAI\_API\_KEY"),

)

# Function to rewrite text using GPT-4o Mini

def rewrite\_text(input\_text):

prompt = f"""

You are an expert copy editor for Wannon Water. Rewrite the provided text to align with the "Our Voice Guide - January 2023" following these rules:

- Informative: Clear and concise, avoiding jargon. Explain technical terms simply.

- Neighborly: Warm and inclusive language. Avoid overly formal expressions.

- Local Leader: Active voice and short, dynamic sentences.

Example Rewrite:

Input: "The community will be informed of a water outage."

Output: "We'll let the community know about a water outage."

Ensure capitalization follows these rules:

- Seasons like "summer" and "winter" should always be lowercase unless part of a title or proper noun.

- Proper nouns like "Wannon Water" should always be capitalized.

- Job titles, departments, and organizational names (e.g., "Operations Manager," "Digital Services Department") should be capitalized.

Example Rewrite:

Input: "Winter is coming, and the operations manager is ready to manage the seasonal changes."

Output: "winter is coming, and the Operations Manager is ready to manage the seasonal changes."

Ensure that all dates are formatted as "date month year" (e.g., 10 March 1996). Avoid using "10th of March, 1996" or formats like "03/10/1996."

Example Rewrite:

Input: "The project began on the 10th of March, 1996, and ended on 1996-03-15."

Output: "The project began on 10 March 1996 and ended on 15 March 1996."

Ensure that all times are formatted as follows:

- Use lowercase "am" or "pm."

- Do not include a space between the number and "am" or "pm" (e.g., "4pm" instead of "4 pm").

- For times on the hour, do not include ":00" (e.g., "4pm" instead of "4:00pm").

Example Rewrite:

Input: "The meeting is scheduled for 4:00 PM and will end at 6:30 PM."

Output: "The meeting is scheduled for 4pm and will end at 6:30pm."

Ensure numbers 0–9 are written as numerals (e.g., 3).

Write numbers 10 and above in words, except for measurements (e.g., length, money, temperature). For measurements, retain numerals (e.g., 12 km, $45 million, 25°C).

Inclusive language principles:

- Avoid gendered terms unless specifically required. Use gender-neutral alternatives (e.g., "they/them" instead of "he/she").

- Use respectful terminology for groups or communities (e.g., "person with disability" instead of "disabled person").

- Avoid terms with negative connotations (e.g., "victim" should be replaced with "survivor").

- Use simple, clear terms to accommodate readers with varying literacy levels.

- Avoid referring to someone's age unless it is relevant to the context.

- Use culturally appropriate terms and avoid stereotypes (e.g., "First Nations Australians," or "Aboriginal and Torres Strait Islander peoples" are appropriate).

- Use gender-neutral terms like "chairperson" instead of "chairman" or "chairwoman."

- Avoid asking for Christian name; use "first name" instead.

- Avoid asking for surname; use "last name" instead.

- Avoid using "maiden name"; use "birth name" instead.

- Avoid using "husband" or "wife"; use "spouse" instead.

- Avoid using "mankind"; use "humanity" instead.

- Avoid using "manpower"; use "workforce" instead.

- Avoid using "ladies and gentlemen"; use "everyone" instead.

Examples of inclusive rewriting:

- Input: "Each employee should submit his timesheet."

Output: "All employees should submit their timesheets."

- Input: "The disabled will benefit from this project."

Output: "People with disabilities will benefit from this project."

Ensure the text includes subheadings where appropriate to break the content into logical sections. Subheadings should:

- Be concise and descriptive.

- Use sentence case (capitalize only the first word and proper nouns).

- Provide a clear structure to the content.

Example Rewrite:

Input:

"Our project involves several phases. First, we will upgrade the main pipeline. Then, we will expand the water treatment plant capacity. Lastly, we will implement new monitoring systems."

Output:

"Our project involves several phases:

Upgrading the main pipeline

We will start by upgrading the main pipeline to improve water flow and reliability.

Expanding water treatment capacity

Next, we will expand the water treatment plant capacity to meet future demand.

Implementing new monitoring systems

Lastly, we will implement advanced monitoring systems to ensure ongoing operational efficiency."

Rewrite this: {input\_text}

"""

response = client.chat.completions.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "You are a helpful assistant."},

{"role": "user", "content": prompt}

],

temperature=0.7,

)

return response.choices[0].message.content.strip()

# Routes

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/translate', methods=['POST'])

def translate():

try:

text = request.form.get('text')

if text:

translated\_text = rewrite\_text(text)

return jsonify({"translated": translated\_text})

return jsonify({"error": "No text provided"}), 400

except Exception as e:

return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

I have removed the functionality for users to upload a document from your Flask application. The revised code ensures that only direct text input is accepted for processing. Let me know if there are additional modifications or enhancements you'd like!

**You said:**

does it need a change to the html file also?

**ChatGPT said:**

ChatGPT

Yes, the **HTML file** will also need to be updated to remove the file upload functionality since it is no longer supported. Specifically, you should:

1. Remove the <input> field for file uploads.
2. Adjust the layout to ensure the UI still looks clean and consistent.

Here’s the updated index.html file:

**Updated index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text Translator</title>

<link rel="stylesheet" href="/static/css/style.css">

</head>

<body>

<div class="container">

<div class="pane">

<textarea id="input-text" placeholder="Paste your text here"></textarea>

</div>

<button id="translate-btn">Translate</button>

<div class="pane">

<textarea id="output-text" placeholder="Translated content will appear here"></textarea>

<button id="copy-btn">Copy to Clipboard</button>

</div>

</div>

<script src="/static/js/script.js"></script>

</body>

</html>

**Changes Made:**

1. **Removed the File Upload Input**:

<input type="file" id="file-upload" accept=".docx">

This input field was deleted.

1. **Retained the Text Area for Input**:
   * Users can now only paste their text into the textarea.
2. **Maintained the Translate and Copy Buttons**:
   * The core functionality remains intact, allowing users to translate text and copy the output.

Let me know if you need additional adjustments or testing!

**You said:**

and the js file?

**ChatGPT said:**

ChatGPT

Yes, the JavaScript file (script.js) will also need to be updated to remove references to file upload functionality. Here’s the updated version:

**Updated script.js**

document.getElementById("translate-btn").addEventListener("click", async () => {

const inputText = document.getElementById("input-text").value;

if (!inputText.trim()) {

alert("Please enter some text to translate.");

return;

}

let formData = new FormData();

formData.append("text", inputText);

try {

const response = await fetch("/translate", {

method: "POST",

body: formData,

});

const data = await response.json();

if (data.translated) {

document.getElementById("output-text").value = data.translated;

} else {

alert(data.error || "An error occurred.");

}

} catch (error) {

console.error("Error translating text:", error);

alert("Failed to communicate with the server.");

}

});

document.getElementById("copy-btn").addEventListener("click", async () => {

const outputText = document.getElementById("output-text").value;

if (outputText.trim() === "") {

alert("There's no text to copy!");

return;

}

try {

await navigator.clipboard.writeText(outputText);

alert("Text copied to clipboard!");

} catch (error) {

console.error("Failed to copy text:", error);

alert("Failed to copy text. Please copy manually.");

}

});

**Changes Made:**

1. **Removed File Upload Handling**:
   * Deleted references to file-upload input field.
   * Removed logic that appends a file to the FormData.
2. **Text Input Validation**:
   * Ensures the user provides text in the textarea.
3. **Simplified Logic**:
   * Focused solely on handling text translation.
4. **Error Handling**:
   * Retains error handling for cases where the server fails or the input is invalid.

This updated JavaScript file aligns with the removal of the file upload feature and ensures clean, streamlined functionality for text translation. Let me know if you need further refinements!

**You said:**

and the css?

**ChatGPT said:**

ChatGPT

The CSS (style.css) likely doesn't need significant changes unless there are specific styles related to the now-removed file upload input that need to be removed. However, to ensure the code is clean, you can verify and simplify the CSS by removing any unused styles.

Here’s the updated style.css file:

**Updated style.css**

body {

font-family: Arial, sans-serif;

margin: 20px;

}

.container {

display: flex;

gap: 20px;

}

.pane {

flex: 1;

display: flex;

flex-direction: column;

}

textarea {

margin: 10px 0;

width: 400px; /\* Adjust the width \*/

height: 500px; /\* Adjust the height \*/

resize: none; /\* Prevent manual resizing \*/

}

button {

margin: 10px 0;

width: 100%;

height: 40px;

cursor: pointer;

}

**Changes Made:**

1. **Removed File Upload-Specific Styles**:
   * If there were any styles targeting input[type="file"] or similar, those are removed.
2. **Simplified Layout**:
   * Ensured the layout is consistent for just the text areas and buttons.
3. **Retained General Styles**:
   * Text areas (textarea) and buttons (button) retain their styling to maintain UI consistency.

If the original CSS had additional file-upload-specific styles that you think were not addressed, feel free to share those details, and I can refine further!

### First step - version control

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Docker application without document upload function"

### *Push Changes to Remote:*

git push origin master

### then recreate the container

**Step 1: Build the Docker Image**

Run the following command to build the Docker image. Replace flask-ai-app with your desired image name:

docker build -t voice-app .

****

**Step 4: Run the Docker Container**

Start a container from the image:

docker run -d -p 5000:5000 --env-file .env voice-app

Notes

* -p 5000:5000: Maps port 5000 on your machine to port 5000 in the container.
* --env-file .env: Passes the environment variables from your .env file to the container.

List all containers (stopped and running)

docker ps -a

****

**Step 5: Test the Application**

1. Open a browser and navigate to http://localhost:5000 to access the application.
2. Test the upload and text rewriting functionalities to ensure everything works as expected.

****

* Except for the ability to upload files
* Rather than debug this I will remove it because it makes the UI less intuitive

### then deploy the container to the test server

**Step 1: Prepare Your Docker Image**

docker images

Output

REPOSITORY TAG IMAGE ID CREATED SIZE

voice-app latest 1d7ae8816569 14 minutes ago 332MB

**Tag the Docker Image**:

* + Tag your local image to prepare it for transfer. Replace <image-name> and <tag> with your image name and version:

docker tag voice-app:latest TSSVR23/voice-app:latest

**Check the Image**:

* + Verify that the image is tagged correctly:

docker images

output

REPOSITORY TAG IMAGE ID CREATED SIZE

TSSVR23/voice-app latest 1d7ae8816569 18 minutes ago 332MB

**Step 2: Export the Docker Image**

1. **Save the Docker Image**:
   * Save the image as a .tar file:

docker save -o voice-app.tar TSSVR23/voice-app:latest

1. **Transfer the .tar File**:

From

Local machine

C:\Users\andrew.dilley\development\voice\voice\_app1\voice-app.tar

To

G:\development\voice\voice-app.tar

To

TSSVR23

C:\Users\andrew.dilley\development\voice\voice-app.tar

**Step 3: Load the Image on the Test Server**

1. **Load the Docker Image**:
   * Load the transferred .tar file into Docker:

docker load -i flask-ai-app.tar

Output

ad0c94ffca1e: Loading layer [==================================================>] 93B/93B

0b096ce3dbb5: Loading layer [==================================================>] 192B/192B

a329e8e5ebf2: Loading layer [==================================================>] 14.63MB/14.63MB

2c8fe911a5d3: Loading layer [==================================================>] 18.72MB/18.72MB

Loaded image: TSSVR23/voice-app:latest

1. **Verify the Image**:
   * Confirm the image is now available on the test server:

docker images

Output

**Step 4: Run the Container on the Test Server**

1. **Run the Container**:
   * Start the container with the appropriate port mappings and environment variables:

docker run -d -p 5001:5000 --env-file .env voice-app:latest

note: port is 5001 used to avoid conflicts

**Explanation**

* **5001:5000**:
  + 5001: Host machine port (accessible from the browser or API clients as http://<server-ip>:5001).
  + 5000: Port inside the container where the Flask app is running.

**Explanation**

docker run -d -p 5001:5000 --env-file .env voice-app:latest

**-p (Publish Ports)**

The -p flag maps a port on the host machine to a port inside the Docker container. It follows the syntax:

-p <host-port>:<container-port>

* **5001**: The port on the host machine that will be exposed to the outside world.
* **5000**: The port inside the container where the application is running (e.g., your Flask app listens on port 5000).

**Effect of -p:**

* Allows external access to the application running inside the container.
* Example:
  + If your app runs on http://localhost:5000 inside the container, mapping it with -p 5001:5000 means it will be accessible from the host as http://localhost:5001.

**-d (Detached Mode)**

The -d flag runs the container in **detached mode**, which means:

* The container runs in the background.
* The terminal is not blocked by the container's output.

**Effect of -d:**

* You don't see logs or interact with the container directly in the terminal.
* To view logs or interact with the container later, you use Docker commands like:
  + View logs: docker logs <container-id>
  + Attach to container: docker attach <container-id>

**Summary of Command**

docker run -d -p 5001:5000 --env-file .env voice-app:latest

* **-d**: Runs the container in the background.
* **-p 5001:5000**: Maps port 5001 on the host to port 5000 in the container.
* **--env-file .env**: Passes environment variables from the .env file into the container.
* **voice-app:latest**: Specifies the image to use for the container.

Error: PS C:\Users\andrew.dilley\development\voice> docker run -d -p 5001:5000 --env-file .env voice-app:latest docker: open .env: The system cannot find the file specified.

Solution: Had to copy the .env file into C:\Users\andrew.dilley\development\voice>

Error: PS C:\Users\andrew.dilley\development\voice> docker run -d -p 5001:5000 --env-file .env voice-app:latest

Unable to find image 'voice-app:latest' locally

docker: Error response from daemon: pull access denied for voice-app, repository does not exist or may require 'docker login': denied: requested access to the resource is denied.

Solution:

Correct command:  
  
docker run -d -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

1. **Verify the Container**:
   * Check if the container is running:

docker ps

1. **Test the Application**:
   * Open a browser and navigate to http://TSSVR23:5001 to verify the app is running correctly.

**Optional: Push the Image to a Docker Registry**

**Note: maybe try this next time**

If the test server and your local machine can access a Docker registry, you can use it to simplify deployment:

1. **Log In to Docker Hub or a Private Registry**:

docker login

1. **Push the Image**:

docker push <your-dockerhub-username>/flask-ai-app:latest

1. **Pull the Image on the Test Server**:

docker pull <your-dockerhub-username>/flask-ai-app:latest

1. **Run the Container**:

docker run -d -p 5000:5000 --env-file .env <your-dockerhub-username>/flask-ai-app:latest

**Additional Notes**

* Ensure the test server has Docker installed and running.
* Use firewalls or security groups to allow access to port 5000 on the test server if needed.
* Regularly clean up unused images and containers to save space on the server.

Let me know if you run into any issues! 😊

## Jan 22

## Redacting input PII and hiding the copy button

#### Delete the tar file

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Redaction of PII and hiding copy button"

### *Push Changes to Remote:*

git push origin master

### *Create the Container*

#### Check if any containers are running

docker ps

#### Build the Docker Image

Run the following command to build the Docker image. Replace flask-ai-app with your desired image name:

docker build -t voice-app .

#### Run the Docker Container

Start a container from the image:

docker run -d -p 5000:5000 --env-file .env voice-app

#### Test the container

Open a browser and navigate to http://localhost:5000 to verify the app is running correctly.

#### View the image

docker images

#### Stop and remove all containers using the TSSVR23/voice-app image

docker ps -a

#### Stop Containers:

docker stop <container-id>

#### Stop Containers:

docker rm <container-id>

#### Remove the existing TSSVR23/voice-app image

docker rmi TSSVR23/voice-app

#### Tag the Docker Image:

Tag your local image to prepare it for transfer. Replace <image-name> and <tag> with your image name and version:

docker tag voice-app:latest TSSVR23/voice-app:latest

**Check the Image**:

* + Verify that the image is tagged correctly:

docker images

**Step 2: Export the Docker Image**

**Save the Docker Image**:

Save the image as a .tar file:

docker save -o voice-app.tar TSSVR23/voice-app:latest

**Transfer the .tar File**:

From

Local machine

C:\Users\andrew.dilley\development\voice\voice\_app1\voice-app.tar

To

G:\development\voice\voice-app.tar

To

TSSVR23

C:\Users\andrew.dilley\development\voice\voice-app.tar

**Step 3: Load the Image on the Test Server**

#### Stop and remove all containers using the TSSVR23/voice-app image

docker ps -a

#### Stop Containers:

docker stop <container-id>

#### Stop Containers:

docker rm <container-id>

#### Remove the existing TSSVR23/voice-app image

docker rmi TSSVR23/voice-app

**Load the Docker Image**:

Load the transferred .tar file into Docker:

docker load -i voice-app.tar

**Verify the Image**:

Confirm the image is now available on the test server:

docker images

#### View the image

docker images

**Step 4: Run the Container on the Test Server**

**Run the Container**:

* Start the container with the appropriate port mappings and environment variables:

docker run -d -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

**Verify the Container**:

* + Check if the container is running:

docker ps

**Test the Application**:

Open a browser and navigate to http://TSSVR23:5001 to verify the app is running correctly.

## January 23

## Redacting complete – addresses and names

#### Stage All Changes:

git add .

### *check the status of the local repository*

git status

### *Committing Changes*

git commit -m "Redaction complete"

### *Push Changes to Remote:*

git push origin master

## Deploy the updated code

### *On the local machine*

### *Create the Container*

#### Check if any containers are running

docker ps

#### Build the Docker Image

Run the following command to build the Docker image.

docker build -t voice-app .

note: voice-app is the image name

#### Run the Docker Container

Start a container from the image:

docker run -d -p 5001:5000 --env-file .env voice-app

#### Test the container

Open a browser and navigate to http://localhost:5001 to verify the app is running correctly.

#### View the image

docker images

### *On the test server*

#### Stop and remove all containers using the TSSVR23/voice-app image

docker ps -a

#### Stop Containers:

docker stop <container-id>

#### remove Containers:

docker rm <container-id>

#### View the image

docker images

#### Remove the existing TSSVR23/voice-app image

docker rmi TSSVR23/voice-app

#### View the image

docker images

### *On the local machine*

#### Tag the Docker Image:

Tag your local image to prepare it for transfer. Replace <image-name> and <tag> with your image name and version:

docker tag voice-app:latest TSSVR23/voice-app:latest

**Check the Image**:

* + Verify that the image is tagged correctly:

docker images

**Step 2: Export the Docker Image**

**Save the Docker Image**:

Save the image as a .tar file:

docker save -o voice-app.tar TSSVR23/voice-app:latest

**Transfer the .tar File**:

From

### *On the local machine*

C:\Users\andrew.dilley\development\voice\voice\_app1\voice-app.tar

To

G:\development\voice\voice-app.tar

To

### *On the test server*

C:\Users\andrew.dilley\development\voice\voice-app.tar

**Step 3: Load the Image on the Test Server**

#### Check that all containers using the TSSVR23/voice-app image are stopped and removed (see previous step)

docker ps -a

#### if needed…

#### Stop Containers:

docker stop <container-id>

#### Stop Containers:

docker rm <container-id>

#### Remove the existing TSSVR23/voice-app image

docker rmi TSSVR23/voice-app

**Load the Docker Image**:

Load the transferred .tar file into Docker:

docker load -i voice-app.tar

**Verify the Image**:

Confirm the image is now available on the test server:

docker images

**Step 4: Run the Container on the Test Server**

**Run the Container**:

* Start the container with the appropriate port mappings and environment variables:

docker run -d -p 5001:5000 --env-file .env TSSVR23/voice-app:latest

**Verify the Container**:

* + Check if the container is running:

docker ps

**Test the Application**:

### *On the local machine*

Open a browser and navigate to http://TSSVR23:5001 to verify the app is running correctly.

# January 31

## Data security considerations

If your main concern is users accidentally posting sensitive data through the API, apart from **redacting data in the app**, here are several additional precautions you can take:

**1. Implement Input Validation & Filtering**

* Use **regular expressions** or **pattern matching** to detect and block common sensitive data types before sending requests to the API.
  + Example: Mask **email addresses, credit card numbers, or personal identifiers** before submission.
  + Consider integrating **DLP (Data Loss Prevention)** tools for real-time scanning.

**2. Use the OpenAI Moderation API**

* The **Moderation API** can help detect potentially sensitive content before it is sent.
* It provides confidence scores for categories like **PII, violence, and self-harm**.
* If a response is flagged, **block the request or notify the user**.

**3. Use a Proxy or API Gateway for Pre-processing**

* Set up an **intermediate API gateway** to analyze and sanitize data before forwarding it to OpenAI.
* This allows:
  + **Logging and monitoring** (for auditing and compliance).
  + **Automated redaction of sensitive terms** (e.g., replacing full names with placeholders).
  + **Rate limiting** to prevent excessive or unexpected data flow.

**4. Enable OpenAI's "Zero Data Retention" (ZDR) Option**

* **Apply for ZDR** to prevent OpenAI from logging requests for even short-term retention.
* With ZDR enabled, OpenAI does **not persist request or response data** beyond the immediate API interaction.

**5. Educate Users & Provide Warnings**

* Before submitting data, **display a clear warning** to users about the risks of sharing sensitive information.
* Example: **"Do not enter personal data, passwords, or confidential business information."**
* Consider implementing **tooltips, modals, or confirmation popups** to reinforce this message.

**6. Monitor API Usage & Log Anomalies**

* Set up **logging and alerts** to monitor API requests for unusual patterns.
  + Example: If a request contains 16-digit sequences (credit card numbers) or specific keywords (e.g., "SSN", "DOB"), trigger an **alert or block the request**.
* Use an **SIEM (Security Information and Event Management) tool** to detect anomalies in real-time.

**7. Implement Role-Based Access Control (RBAC)**

* Restrict API access based on **user roles**:
  + General users: **Low permissions**, strict monitoring.
  + Admin users: **Higher permissions**, with **additional security** like MFA.
* **Mask sensitive responses** for certain users.

**8. Use Encryption & Secure Data Transmission**

* Ensure all API requests are sent over **TLS (HTTPS)**.
* If you store logs, encrypt them with **AES-256** and limit access.

**9. Data Masking in Responses**

* Even if a user mistakenly submits sensitive data, ensure your application **does not expose it back** in the response.
* Example: Replace detected sensitive data in API responses with "\*\*\*REDACTED\*\*\*".

**10. Define an Incident Response Plan**

* If **sensitive data is leaked**, have a clear **containment and mitigation** process:
  + **Immediate alerting** to security teams.
  + **Request data deletion** from OpenAI (if within retention period).
  + **Audit affected systems** for further leaks.

Would you like a structured **playbook** for handling accidental PII submission incidents? I can draft one based on these steps.