1. Introduction: General Information about the Project

This project is a web application that provides text analysis tools. Users can paste their texts into the Streamlit interface and obtain various statistical features of the text, such as character count, word count, sentence count, unique words, paragraph count, etc. The application uses **Streamlit** for the frontend and **Flask** for the backend.

- **Frontend**: Streamlit (Python)
- **Backend**: Flask (Python REST API)
- **Communication**: HTTP JSON-based

---

2. Streamlit (Frontend) Code Explanations

2.1. Importing Required Libraries

```python
import streamlit as st
import requests
```

- **Introduction**: In this section, the `streamlit` and `requests` libraries are imported into the project.
- **Output**: `streamlit` will be used to build the user interface, and `requests` will be used to make HTTP requests to the Flask API.

2.2. Defining the API URL

```python
API_URL = "http://localhost:5000/count"
```

- **Introduction**: The URL of the Flask API is defined. The API will be hosted on `localhost:5000`.
- **Output**: The requests to the API will be made through this URL.

2.3. Starting the Streamlit Application and Displaying Project Information

```python
def main():
    st.title("□ Text Analysis Tool")

    with st.expander("□ About the Project", expanded=False):
        st.markdown("""
                        **□ Text Analysis Tool** is a web application that
analyzes statistical features of your texts.
                        ...
        """)
```

- **Introduction**: This section displays the title of the app and provides information about the project, including the key features and technical details.
- **Output**: Users will understand what the application does and how to use it.

2.4. Taking Text Input from the User

```python
st.write("Paste your text below and view statistics!")

if 'text_input' not in st.session_state:
    st.session_state.text_input = ""
```

- **Introduction**: This section displays an instruction for the user to paste their text. If the `text_input` session state does not exist, it will be initialized as an empty string.
- **Output**: The user can input text, which will be saved in the session state.

2.5. Text Update Function

```python
def update_text():
    st.session_state.text_input = st.session_state.widget_text
```

- **Introduction**: This function ensures that any changes made in the text area are reflected in the session state.
- **Output**: The text entered by the user is saved in the session state as `st.session_state.text_input`.

2.6. Text Area and Buttons

```python
text_input = st.text_area(
    "Text Input",
    height=200,
    value=st.session_state.text_input,
    key="widget_text",
    on_change=update_text
)

col1, col2 = st.columns(2)
with col1:
    analyze_button = st.button("Analyze")
with col2:
    clear_button = st.button("Clear")
```

- **Introduction**: A text area (`text_area`) is created for the user to input text. Two buttons (Analyze and Clear) are also added.
- **Output**: The user can enter text and click "Analyze" to start the analysis or "Clear" to reset the text input.

2.7. Clear Button Functionality

```
    if clear_button:
        st.session_state.text_input = ""
        st.rerun()
```

- **Introduction**: When the "Clear" button is clicked, the text input is cleared.
- **Output**: The page will reload, and the text input will be cleared.

2.8. Analyze Button Functionality

```
    if analyze_button:
        if st.session_state.text_input.strip():
            try:
                response = requests.post(API_URL, json={"text": 
st.session_state.text_input})

                if response.status_code == 200:
                    results = response.json()
                    col1, col2, col3, col4, col5 = st.columns(5)
                    col1.metric("Characters", results['characters'])
                    col2.metric("Words", results['words'])
                    col3.metric("Sentences", results['sentences'])
                    col4.metric("Unique", results['unique_words'])
                    col5.metric("Paragraphs", results['paragraphs'])
                else:
                    st.error(f"API Error ({response.status_code}): 
{response.text}")
            except requests.exceptions.ConnectionError:
                st.error("Could not connect to server. Please ensure 
Flask server is running.")
        else:
            st.warning("Please enter text for analysis!")
```

- **Introduction**: When the "Analyze" button is clicked:
  - If the input text is not empty, the text is sent to the Flask API for analysis.
  - If the API responds with a successful status code (200), the results (such as character count, word count, sentence count, etc.) are displayed.
- **Output**: The analysis results are presented in a user-friendly format. In case of errors, appropriate error messages are shown.

2.9. Running the Main Function

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

- **Introduction**: This ensures that the `main()` function is called when the application is run.
- **Output**: The Streamlit app starts and runs the code defined in `main()`.

---

3. Flask (Backend) Code Explanations

3.1. Importing Required Libraries

```
from flask import Flask, jsonify, request
from flask_cors import CORS
import re
```

- **Introduction**: The `Flask`, `CORS`, and `re` libraries are imported into the project. Flask is used to create the backend API, CORS is used to handle cross-origin requests, and `re` (regular expressions) will be used for text analysis.
- **Output**: The necessary libraries are included for building the API and processing the text.

3.2. Starting the Flask Application and Enabling CORS

```
app = Flask(__name__)
CORS(app)
```

- **Introduction**: The Flask application is initialized, and CORS is enabled to allow cross-origin requests.
- **Output**: The API can handle requests from different domains.

3.3. Helper Functions: Sentence and Paragraph Count

```
def count_sentences(text: str) -> int:
    sentences = re.split(r'[.!?]+[\s\n]*', text)
    return len([s for s in sentences if s.strip()])

def count_paragraphs(text: str) -> int:
    paragraphs = re.split(r'\n\s*\n', text)
    return len([p for p in paragraphs if p.strip()])
```

- **Introduction**: These functions use regular expressions to count sentences and paragraphs in the given text.
- **Output**: The number of sentences and paragraphs in the text is returned.

3.4. API Route for Text Statistics

```
@app.route('/count', methods=['POST'])
def count_text_stats():
    data = request.get_json()
    raw_text = data.get('text', '')
    stripped_text = raw_text.strip()  # Trim whitespace
```

- **Introduction**: The Flask API listens for `POST` requests on the `/count` endpoint. The request JSON body is processed, and the text is extracted.
- **Output**: The API will process the text and calculate the statistics.

3.5. Calculating and Returning Text Statistics

```
    if not stripped_text:
        return jsonify({
            "characters": 0,
            "words": 0,
            "sentences": 0,
            "unique_words": 0,
            "paragraphs": 0
        })

    words = raw_text.split()
    unique_words = len(set(words)) if words else 0

    return jsonify({
        "characters": len(raw_text),
        "words": len(words),
        "sentences": count_sentences(raw_text),
        "unique_words": unique_words,
        "paragraphs": count_paragraphs(raw_text)
    })
```

- **Introduction**: If the input text is empty, the API returns all statistics as 0. If the text is valid, the statistics (character count, word count, sentence count, unique word count, and paragraph count) are calculated and returned.
- **Output**: The API responds with a JSON object containing the calculated statistics.

3.6. Running the Flask Application

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

- **Introduction**: The Flask application is started, running on port 5000 in debug mode.
- **Output**: The Flask API becomes active and starts listening for requests.

---

4. Summary of the Project

This project consists of a frontend built with **Streamlit** and a backend built with **Flask**. The **Streamlit frontend** allows users to input text, which is then sent to the **Flask backend** for analysis. The backend calculates various statistics about the text, such as character count, word count, sentence count, unique words, and paragraph count, and sends the results back to the frontend. The frontend then displays these results in a clean, user-friendly interface.

- **Frontend (Streamlit)**: Provides a simple user interface for text input and displays the analysis results.
- **Backend (Flask)**: Handles the text analysis and returns the calculated statistics in JSON format.
- **Communication**: The frontend and backend communicate using HTTP requests and JSON data.

In summary, this project enables real-time text analysis with a clean and interactive interface, making it an effective tool for analyzing and understanding various aspects of a given text.