Thesis Assistant Platform Implementation Guide

This guide provides step-by-step instructions for implementing the **Thesis Assistant Platform** using **Streamlit**, **Groq API**, and modular Python files. The platform supports students throughout their thesis journey by offering AI-driven assistance for tasks like topic refinement, literature review, writing assistance, and more.

---

1. Project Structure

The project is organized into the following structure:

```
thesis_assistant/
│
├── main.py                # Main Streamlit app
├── utils/                 # Utility functions
│   ├── topic_refinement.py
│   ├── literature_review.py
│   ├── methodology_guidance.py
│   ├── data_analysis.py
│   ├── thesis_structure.py
│   ├── writing_assistance.py
│   ├── reference_management.py
│   ├── time_management.py
│   ├── presentation_preparation.py
│   └── proofreading.py
├── data/                  # Store datasets or user progress
│   └── user_progress.json
└── requirements.txt        # List of dependencies
```

---

2. Prerequisites

Before starting, ensure you have the following: - Python 3.8 or higher installed. - A Groq API key (sign up at Groq). - Basic knowledge of Python and Streamlit.

---

3. Installation

1. **Clone the Repository**:

```
git clone https://github.com/your-repo/thesis_assistant.git
cd thesis_assistant
```

2. **Install Dependencies**: Create a `requirements.txt` file with the following content:

   ```
   streamlit
   requests
   pandas
   numpy
   spacy
   ```

   Install the dependencies:

   ```
   pip install -r requirements.txt
   ```

3. **Download Spacy Language Model**: For proofreading and language enhancement:

   ```
   python -m spacy download en_core_web_sm
   ```

4. **Set Up Groq API Key**: Add your Groq API key to an environment variable or directly in the `utils/groq_api.py` file.

---

4. File Descriptions

4.1 Main Streamlit App (`main.py`)

The main entry point for the Streamlit app. It provides a sleek interface with buttons for each feature.

```
import streamlit as st
from utils.topic_refinement import refine_topic,
        generate_research_question
from utils.literature_review import search_literature,
        summarize_paper
from utils.writing_assistance import improve_writing,
        check_logical_flow
from utils.proofreading import proofread

# Sidebar Navigation
st.sidebar.title("Thesis Assistant")
option = st.sidebar.selectbox(
    "Choose a Feature",
    [
        "Topic Refinement",
        "Literature Review",
```

```
        "Writing Assistance",
        "Proofreading",
        "More Features..."
    ]
)

# Main Page Content
st.title("Thesis Assistant Platform")

if option == "Topic Refinement":
    st.header("Topic Refinement & Research Question Development")
    topic = st.text_input("Enter your thesis topic:")
    if st.button("Refine Topic"):
        st.write(refine_topic(topic))
    if st.button("Generate Research Question"):
        st.write(generate_research_question(topic))

elif option == "Literature Review":
    st.header("Literature Review Assistance")
    query = st.text_input("Enter a keyword for literature
        search:")
    if st.button("Search Literature"):
        results = search_literature(query)
        st.write("Relevant Papers:", results)

    paper_title = st.text_input("Enter a paper title to
        summarize:")
    if st.button("Summarize Paper"):
        summary = summarize_paper(paper_title)
        st.write("Summary:", summary)

elif option == "Writing Assistance":
    st.header("Writing Assistance & Draft Review")
    text = st.text_area("Enter text for writing assistance:")
    if st.button("Improve Writing"):
        improved_text = improve_writing(text)
        st.write("Improved Text:", improved_text)
    if st.button("Check Logical Flow"):
        feedback = check_logical_flow(text)
        st.write("Feedback:", feedback)

elif option == "Proofreading":
    st.header("Proofreading & Language Enhancement")
    text = st.text_area("Enter text for proofreading:")
```

```python
    if st.button("Proofread Text"):
        errors = proofread(text)
        st.write("Suggestions:", errors)

else:
    st.write("More features coming soon!")
```

4.2 Utility Functions

`utils/topic_refinement.py`

Handles topic refinement and research question generation using Groq API.

```python
from utils.groq_api import call_groq

def refine_topic(topic):
    prompt = f"Help me refine this thesis topic:
        {topic}. Provide suggestions for narrowing the scope."
    return call_groq(prompt)

def generate_research_question(topic):
    prompt = f"Generate a clear research question or hypothesis
        for this topic: {topic}."
    return call_groq(prompt)
```

`utils/literature_review.py`

Provides literature search and summarization.

```python
from utils.groq_api import call_groq

def search_literature(query):
    prompt =
        f"Recommend relevant academic papers and sources for this
        query: {query}."
    return call_groq(prompt)

def summarize_paper(paper_title):
    prompt = f"Summarize the key findings of this paper:
        {paper_title}."
    return call_groq(prompt)
```

`utils/writing_assistance.py`

Improves writing clarity and checks logical flow.

```python
from utils.groq_api import call_groq

def improve_writing(text):
    prompt = f"Improve the clarity, coherence, and academic tone of this text: {text}."
    return call_groq(prompt)

def check_logical_flow(text):
    prompt = f"Check the logical flow and argument strength of this text: {text}. Provide suggestions for improvement."
    return call_groq(prompt)
```

utils/proofreading.py

Detects grammar, spelling, and punctuation errors.

```python
from utils.groq_api import call_groq

def proofread(text):
    prompt = f"Proofread this text for grammar, spelling, and punctuation errors: {text}. Suggest improvements."
    return call_groq(prompt)
```

utils/groq_api.py

Handles Groq API requests.

```python
import requests
import os

GROQ_API_KEY = os.getenv("GROQ_API_KEY")  # Replace with your Groq API key
GROQ_API_URL = "https://api.groq.com/v1/completions"  # Replace with the actual Groq endpoint

def call_groq(prompt, max_tokens=100, temperature=0.7):
    headers = {
        "Authorization": f"Bearer {GROQ_API_KEY}",
        "Content-Type": "application/json"
    }
    data = {
        "prompt": prompt,
        "max_tokens": max_tokens,
        "temperature": temperature,
        "model": "llama2"  # Replace with the model you want to use
    }
    response = requests.post(GROQ_API_URL, headers=headers,
        json=data)
    if response.status_code == 200:
        return response.json().get("choices", [{}])[0].get("text",
        "")
    else:
        return f"Error: {response.status_code}, {response.text}"
```

4.3 Data Storage (`data/user_progress.json`)

Stores user progress or preferences in JSON format.

Example:

```json
{
    "user_1": {
        "topic": "AI in Education",
        "progress": ["Topic Refinement", "Literature Review"]
    }
}
```

4.4 Requirements File (`requirements.txt`)

Lists all dependencies:

```
streamlit
requests
pandas
numpy
spacy
```

---

5. Deployment

1. Run the app locally:

   ```
   streamlit run main.py
   ```

2. Deploy on **Streamlit Community Cloud**:

   - Push your code to a GitHub repository.
   - Create a new app on Streamlit Cloud and link it to your repository.

---

6. Conclusion

This implementation guide provides a comprehensive framework for building the Thesis Assistant Platform. By leveraging Groq API and modular Python files, the platform offers robust support for students throughout their thesis journey.