Università degli Studi di Messina

MANAGING GOOGLE KEEP SERVICE USING GOOGLE APIS

Professor: Lorenzo Carnevale

STUDENT: HAMIDY ADEEBULLAH

CONTENTS

Introductions

In this project, the objective is to establish communication between a server and client to interact with Google Keep APIs, enabling seamless integration with the Google Keep service. Recognizing the scope of the project, Flask, a lightweight web framework suitable for smaller-scale applications, has been employed to set up the server. Additionally, the GKEEPAPI library has been incorporated to facilitate interactions with the Google Keep API within the Flask environment.

On the server side, custom APIs have been implemented using Flask, providing an intuitive interface for users to interact with Google Keep functionalities. These APIs include creating a new note, retrieving all notes, deleting a specific note, and modifying the content of an existing note. The server, built on Flask, ensures a robust foundation for handling HTTP requests and managing the communication flow between the client and the Google Keep service.

Meanwhile, on the client side, a versatile console-based application has been developed to handle four fundamental types of HTTP requests: POST, GET, DELETE, and PUT. The client application offers a user-friendly experience, featuring loops

that allow users to seamlessly perform actions from the console. This design choice enhances the overall usability of the client, enabling users to intuitively interact with the server and, consequently, with the Google Keep service.

This project combines the simplicity of Flask for server-side development, the power of GKEEPAPI for Google Keep integration, and a well-crafted console-based client to create a cohesive system for managing and manipulating notes in the Google Keep application. Through this approach, the project aims to provide a practical demonstration of server-client communication for interacting with external APIs.

Creating Proxy Server Using Flask Library:

In the initial phase of our project, we established a robust server using Flask, a Python web framework used for its simplicity and flexibility. The Flask library facilitated the fast development of our server-side application, providing a base for handling HTTP requests and responses seamlessly.

Here we have initiated the flask application with the standard procedure:

```
from flask import Flask, jsonify, request
import gkeepapi

app = Flask(__name__)
```

the Flask application was executed with the app.run(debug=True) statement, allowing the server to run in debug mode for easier development and debugging.

The Flask server, coupled with Google Keep integration, forms the backbone of our project, providing a responsive and user-friendly interface for interacting with Google Keep services:

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

Providing Endpoints:

/create_note Endpoint (HTTP POST):

```
@app.route('/create_note', methods=['POST'])
def create_note():
    title = request.json.get('title')
    text = request.json.get('text')
    gnote = keep.createNote(title, text)
    keep.sync()
    return jsonify({'note_id': gnote.id})
```

This endpoint handles HTTP POST requests and is designed for creating a new note on Google Keep.

It expects a JSON payload in the request body with 'title' and 'text' fields.

The 'title' and 'text' values are extracted from the JSON payload.

The keep.createNote method is then called with the extracted title and text to create a new note on Google Keep.

The keep.sync() function is used to synchronize changes with the Google Keep service.

Finally, it returns a JSON response containing the ID of the newly created note.

/show_all_notes Endpoint (HTTP GET):

```python
@app.route('/show_all_notes', methods=['GET'])
def show_all_notes():
    notes = keep.all()
    all_notes = []

    for note in notes:
        text = note.text.replace("☐ ", "").replace("☑ ", "").replace("Item 2",""
        all_notes.append({"id": note.id, "title": note.title, "text": text})

    return jsonify(all_notes)
```

This endpoint handles HTTP GET requests and is responsible for retrieving all notes from Google Keep.

It uses the keep.all() method to retrieve a list of all notes.

The text of each note is processed to remove specific characters ('☐ ', '☑ ', 'Item 2') for improved presentation.

The details of each note, including ID, title, and processed text, are collected into a list of dictionaries.

The endpoint returns a JSON response containing details of all the notes.

/delete_note/<note_id> Endpoint (HTTP DELETE):

```python
@app.route('/delete_note/<note_id>', methods=['DELETE'])
def delete_note(note_id):
    note = keep.get(note_id)
    if note:
        note.delete()
        keep.sync()
        return jsonify({'message': 'Note deleted successfully.'})
    else:
        return jsonify({'error': 'Note not found.'})
```

This endpoint handles HTTP DELETE requests and is intended for deleting a specific note on Google Keep.

It takes a dynamic parameter, note_id, which represents the ID of the note to be deleted.

The keep.get(note_id) method is used to retrieve the note with the specified ID.

If the note is found, it is deleted using note.delete(), and changes are synchronized with keep.sync().

The endpoint returns a JSON response indicating whether the note was deleted successfully or not found.

/modify_note/<note_id> Endpoint (HTTP PUT):

```python
@app.route('/modify_note/<note_id>', methods=['PUT'])
def modify_note(note_id):
    note = keep.get(note_id)
    if note:
        title = request.json.get('title')
        text = request.json.get('text')
        note.title = title
        note.text = text
        keep.sync()
        return jsonify({'message': 'Note modified successfully.'})
    else:
        return jsonify({'error': 'Note not found.'})
```

This endpoint handles HTTP PUT requests and is designed for modifying the title and text of a specific note on Google Keep.

It takes a dynamic parameter, note_id, representing the ID of the note to be modified.

The keep.get(note_id) method is used to retrieve the specified note.

If the note is found, the 'title' and 'text' values are extracted from the JSON payload in the request body.

The note's title and text are updated with the extracted values, and changes are synchronized with keep.sync().

The endpoint returns a JSON response indicating whether the note was modified successfully or not found.

These Flask endpoints collectively provide a comprehensive set of functionalities for interacting with Google Keep through basic CRUD operations: creating, reading, updating, and deleting notes. This server-side functionality serves as the backend infrastructure for the overall project, interfacing with the Google Keep API.

Creating Client-Side Interface:

The client-side program is meticulously crafted to offer users a seamless and user-friendly console interface, facilitating straightforward interactions with the server. This server, built with Flask, acts as an intermediary, orchestrating communication with the Google Keep API. Leveraging the simplicity and effectiveness of the requests library, the client-side program proficiently sends HTTP requests to the meticulously defined endpoints on the server.

By utilizing the Flask server as an abstraction layer, the client program encapsulates the intricacies of communicating with the Google Keep API, providing users with a streamlined experience. This abstraction allows users to focus on essential actions, such as creating, viewing, modifying, and deleting notes, without being directly exposed to the complexities of the underlying Google Keep service.

The console-based interface not only enhances accessibility but also ensures a user-friendly environment where individuals can effortlessly manage their notes. The modularity of the program, coupled with the server's well-defined endpoints, contributes to a cohesive and scalable system. This system not only meets the project's requirements but also serves as a versatile model

for client-server interactions, illustrating the potential for extending similar architectures to other web-based services.

In summary, the client-side program acts as the user's gateway to the broader system, encapsulating intricacies and promoting a user-centric experience while seamlessly interacting with the Flask server and, consequently, the Google Keep API.

The ‘’Request’’ Library:

The requests library is a popular Python library designed to simplify the process of sending HTTP requests and handling responses. It abstracts the complexities of making HTTP requests, providing a higher-level API for interacting with web services.

Implementation in the Project:

In my project, the requests library is employed to communicate with the Flask server. Each function in client-side program corresponds to a different HTTP method (POST, GET, DELETE, PUT), and the requests library facilitates the execution of these methods.

We import the request library using the following command as we did in the code:

```
import requests

base_url = "http://localhost:5000"
```

In addition, that is to say that we used the base url: http://localhost:5000 because the default address for the flask proxy server is the above mentioned url with the port 5000.

Defining Functions:

1. Create a Note (create_note function):

```python
def create_note():
    title = input("Enter the title of the note: ")
    text = input("Enter the content of the note: ")
    url = f"{base_url}/create_note"
    data = {"title": title, "text": text}
    response = requests.post(url, json=data)

    if response.status_code == 200:
        return response.json()
    else:
        return {"error": "Failed to create note."}
```

Takes user input for the title and content of a new note.

Sends a POST request to the /create_note endpoint with the provided data.

Receives and prints the server's response, indicating whether the note creation was successful.

2. Fetch Notes (fetch_notes function):

```
def fetch_notes():
    url = f"{base_url}/show_all_notes"
    response = requests.get(url)

    if response.status_code == 200:
        return response.json()
    else:
        return {"error": "Failed to fetch notes."}
```

Sends a GET request to the /show_all_notes endpoint to retrieve all notes from the server.

Prints the details of each note, including ID, title, and processed text, to the console.

Used in both delete_note and modify_note functions to display a list of existing notes.

3. Delete a Note (delete_note function):

```
def delete_note():
    notes = fetch_notes()
    for note in notes:
        print(f"ID: {note['id']}, Title: {note['title']}, Text: {note['text']}")

    note_id = input("Enter the ID of the note you want to delete: ")
    url = f"{base_url}/delete_note/{note_id}"
    response = requests.delete(url)
    return response.json()
```

Calls fetch_notes to display the existing notes along with their details.

Takes user input for the ID of the note to be deleted.

Sends a DELETE request to the /delete_note/<note_id> endpoint with the provided note ID.
Receives and prints the server's response, indicating whether the note deletion was successful.

4. Modify a Note (modify_note function):

```
def modify_note():
    notes = fetch_notes()
    for note in notes:
        print(f"ID: {note['id']}, Title: {note['title']}, Text: {note['text']}")

    note_id = input("Enter the ID of the note you want to modify: ")
    new_title = input("Enter the new title of the note: ")
    new_text = input("Enter the new content of the note: ")

    url = f"{base_url}/modify_note/{note_id}"
    data = {"title": new_title, "text": new_text}
    response = requests.put(url, json=data)
    return response.json()
```

Calls fetch_notes to display the existing notes along with their details.
Takes user input for the ID of the note to be modified and the new title and content.
Sends a PUT request to the /modify_note/<note_id> endpoint with the provided note ID and modification data.
Receives and prints the server's response, indicating whether the note modification was successful.

5. Main Loop (while True loop):

```
while True:
    print("\nMenu:")
    print("1. See all notes")
    print("2. Create a note")
    print("3. Modify a note")
    print("4. Delete a note")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        notes = fetch_notes()
        for note in notes:
            print(f"ID: {note['id']}, Title: {note['title']}, Text: {note['text']}")

    elif choice == '2':
        create_response = create_note()
        print(create_response)

    elif choice == '3':
        modify_response = modify_note()
        print(modify_response)
```

```
    elif choice == '4':
        delete_response = delete_note()
        print(delete_response)

    elif choice == '5':
        break

    else:
        print("Invalid choice. Please try again.")
```

Displays a menu to the user with options to see all notes, create a note, modify a note, delete a note, or exit the program.

Takes user input for the desired action.

Executes the corresponding function based on the user's choice and prints the result.
Continues looping until the user chooses to exit (choice == '5').

Conclusion

In the culmination of this project, a comprehensive system for interacting with Google Keep services has been successfully developed. The project embraces a client-server architecture, utilizing Flask for the server-side and the requests library on the client-side to communicate with the Google Keep API. The server, designed to be lightweight and scalable, leverages Flask's simplicity, while the client-side program offers an intuitive console-based interface for users.

Server-Side Implementation

The Flask server acts as a crucial intermediary, managing endpoints for creating, retrieving, modifying, and deleting notes on Google Keep. The integration with the GKEEPAPI library facilitates seamless communication with Google Keep services, abstracting the complexities of the underlying API for a user-friendly experience. The server's endpoints encapsulate the essential functionalities, providing a solid foundation for note management.

Client-Side Program

The client-side program enhances user interaction by offering a straightforward console interface. Utilizing the requests library, it communicates with the Flask server, which, in turn, interacts with the Google Keep API. The modular design allows users to effortlessly create, view, modify, and delete notes, providing a practical demonstration of client-server communication.

Overall Reflection

This project not only achieves its primary goal of managing notes on Google Keep but also serves as a valuable educational resource for understanding the dynamics of client-server architecture. The Flask framework and the requests library, being pivotal components, showcase their effectiveness in simplifying web development tasks.

Moving forward, this project provides a foundation for expanding functionalities and adapting to diverse web-based services. The modular design allows for future enhancements, making it a versatile model for similar applications.

In conclusion, the successful implementation of the Google Keep interaction system, coupled with the insightful integration of Flask and requests, underscores the project's achievement in providing a user-friendly,

console-based interface for managing notes. This project not only fulfills its practical objectives but also stands as a testament to the potential of leveraging web frameworks and libraries to create efficient and scalable applications.

Links Related to the Project

GKEEP library
Google Keep Service
Google Keep Api Documentation