LU06.L02 - Blog-Verwaltung mit Flask und JSON

```
from flask import Flask, jsonify, request
app = Flask(__name__)
# Eine einfache Datenstruktur zur Speicherung der Blog-Beiträge
posts = [
    {'id': 1, 'title': 'Erster Beitrag', 'content': 'Dies ist der Inhalt des
ersten Beitrags.'},
    {'id': 2, 'title': 'Zweiter Beitrag', 'content': 'Dies ist der Inhalt
des zweiten Beitrags.'}
@app.route('/posts', methods=['GET'])
def blog overview():
    """Gibt eine Übersicht aller Blog-Beiträge als JSON zurück."""
    return jsonify(posts)
@app.route('/posts/<int:post_id>', methods=['GET'])
def blog detail(post id):
    """Gibt die Details eines bestimmten Blog-Beitrags als JSON zurück."""
    post = next(filter(lambda p: p['id'] == post id, posts), None) # mit
    # post = next((post for post in posts if post['id'] == post id), None) #
mit Generator
    # Beide geben einen Iterator mit nur einem Element zurück, dieses kann
mit next() geholt werden.
    if post:
        return jsonify(post)
    return jsonify({'message': 'Post nicht gefunden'}), 404
@app.route('/posts', methods=['POST'])
def add post():
    """Fügt einen neuen Blog-Beitrag hinzu und gibt diesen als JSON
zurück."""
    data = request.get_json()
    new post = {
        'id': len(posts) + 1,
        'title': data['title'],
        'content': data['content']
    posts.append(new_post)
    return jsonify(new post), 201
@app.route('/posts/<int:post_id>', methods=['DELETE'])
```

```
def delete_post(post_id):
    """Löscht einen bestimmten Blog-Beitrag und gibt eine
Bestätigungsnachricht zurück."""
   global posts
   posts = [post for post in posts if post['id'] != post_id]
   return jsonify({'message': 'Post erfolgreich gelöscht'}), 200
if name == ' main ':
   app.run()
```

1. Initialisierung und Datenstruktur:

- Wir beginnen mit dem Importieren der notwendigen Module und dem Erstellen einer Flask-App-Instanz.
- Eine einfache Datenstruktur (posts-Liste) wird erstellt, um die Blog-Beiträge zu speichern. Jeder Beitrag ist ein Wörterbuch mit einem eindeutigen id, einem title und einem content.

2. Blog-Übersicht (blog_overview):

- Mit dem Dekorator @app.route('/posts', methods=['GET']) wird eine Route für den Pfad /posts definiert, die nur GET-Anfragen akzeptiert.
- Die Funktion gibt eine JSON-Übersicht aller Blog-Beiträge zurück.

3. Blog-Details (blog detail):

- o Mit dem Dekorator @app.route('/posts/<int:post_id>', methods=['GET']) wird eine Route für den Pfad /posts/<post id> definiert, wobei <post id> eine Variable ist, die die ID des gewünschten Beitrags darstellt.
- Die Funktion sucht nach dem Beitrag mit der angegebenen ID und gibt dessen Details als JSON zurück. Wenn der Beitrag nicht gefunden wird, wird eine 404-Nachricht zurückgegeben.

4. Beitrag hinzufügen (add post):

- Mit dem Dekorator @app.route('/posts', methods=['POST']) wird eine Route für den Pfad /posts definiert, die nur POST-Anfragen akzeptiert.
- Die Funktion nimmt die Daten aus dem Request-Body, erstellt einen neuen Beitrag und fügt ihn der posts-Liste hinzu. Der neue Beitrag wird dann als JSON zurückgegeben.

5. Beitrag löschen (delete post):

- Mit dem Dekorator @app.route('/posts/<int:post id>', methods=['DELETE']) wird eine Route für den Pfad /posts/<post id> definiert, die nur DELETE-Anfragen akzeptiert.
- Die Funktion entfernt den Beitrag mit der angegebenen ID aus der posts-Liste und gibt eine Bestätigungsnachricht zurück.

6. App ausführen:

 Am Ende des Skripts wird die Flask-App mit app. run () gestartet, wenn das Skript direkt ausgeführt wird.

From:

https://wiki.bzz.ch/ - BZZ - Modulwiki

Permanent link:

https://wiki.bzz.ch/modul/m323/learningunits/lu06/loesungen/routingmitjson

Last update: 2024/03/28 14:07



https://wiki.bzz.ch/ Printed on 2025/11/27 10:59