LU06.A02 - Blog-Verwaltung mit Flask und JSON

In dieser Aufgabe wirst du ein einfaches Blog-System erstellen, bei dem Benutzer Blog-Beiträge über JSON-Endpunkte lesen, hinzufügen und löschen können. Du wirst mehrere Routen verwenden müssen, um die verschiedenen Funktionen des Blogs zu implementieren.

Routing

Blog-Übersicht

- Erstelle eine Route `/posts`, die alle Blog-Beiträge als JSON zurückgibt.
- Die Ausgabe sollte eine Liste von Blog-Beiträgen sein.

Blog-Details

- Erstelle eine Route `/posts/<int:post_id>`, die einen bestimmten Blog-Beitrag als JSON zurückgibt.
- Wenn der Beitrag nicht existiert, sollte eine entsprechende Fehlermeldung zurückgegeben werden.

Blog-Beitrag hinzufügen

- Erstelle eine Route `/posts`, auf der Benutzer über eine POST-Anfrage mit JSON-Daten einen neuen Blog-Beitrag hinzufügen können.
- Die Anfrage sollte Titel und Inhalt des Beitrags enthalten.

Blog-Beitrag löschen

- Erstelle eine Route `/posts/<int:post_id>`, die über eine DELETE-Anfrage einen Blog-Beitrag löscht.
- Nach dem Löschen sollte eine Bestätigungsnachricht zurückgegeben werden.

Code-Vorlage

main.py

```python
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.',
    },
]


# TODO: Implementiere eine Route für '/posts', die bei einem GET-Request eine Übersicht aller Blog-Beiträge als JSON zurückgibt.
def blog_overview():
    """Gibt eine Übersicht aller Blog-Beiträge als JSON zurück."""
    pass


# TODO: Implementiere eine Route für '/posts/<int:post_id>', die bei einem GET-Request die Details eines bestimmten Blog-Beitrags als JSON zurückgibt.
def blog_detail(post_id):
    """Gibt die Details eines bestimmten Blog-Beitrags als JSON zurück."""
    pass


# TODO: Implementiere eine Route für '/posts', die bei einem POST-Request einen neuen Blog-Beitrag hinzufügt und diesen als JSON zurückgibt.
def add_post():
    """Fügt einen neuen Blog-Beitrag hinzu und gibt diesen als JSON zurück."""
    pass


# TODO: Implementiere eine Route für '/posts/<int:post_id>', die bei einem DELETE-Request einen bestimmten Blog-Beitrag löscht und eine Bestätigungsnachricht zurückgibt.
def delete_post(post_id):
    """Löscht einen bestimmten Blog-Beitrag und gibt eine
```

```
Bestätigungsnachricht zurück."""
    pass


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

Vorgehen

1. Akzeptiere das GitHub Classroom Assignment
2. Klone dein persönliches Repository in die Entwicklungsumgebung
3. Löse die Aufgabenstellung

API-Testing

Via `API-Testing.http`-File im Workspace

oder via: Run in Postman

Abgabe

Die Abgabe der Lösung erfolgt als Push in das persönliche GitHub-Repository.

Hinweis zur Verwendung von "Pure Functions" und "Immutable Values"

In idealen Szenarien, insbesondere wenn es um funktionale Programmierung oder Testbarkeit geht, streben wir oft die Verwendung von „pure functions“ an. „Pure functions“ sind Funktionen, die immer die gleiche Ausgabe für die gleiche Eingabe liefern und keine Seiteneffekte haben. Das bedeutet, dass sie nicht auf externe Daten zugreifen oder diese verändern und somit vorhersehbar und leicht zu testen sind.

In dieser Aufgabenstellung verwenden wir jedoch eine vereinfachte Implementierung, die direkt auf eine globale Datenstruktur zugreift und diese modifiziert. Dies dient dazu, den Umfang und die Komplexität der Aufgabe zu reduzieren und den Fokus auf das Hauptthema, nämlich Flask-Routing und Datenmanipulation, zu legen.

Es ist wichtig zu verstehen, dass in einer echten, produktionsreifen Anwendung der Zugriff auf Daten und deren Manipulation in der Regel über spezialisierte Datenbanken und Datenzugriffsschichten erfolgt, die von der Hauptanwendungslogik getrennt sind. In

solchen Szenarien wäre es wahrscheinlicher und wünschenswert, „pure functions“ oder zumindest funktionale Ansätze zu verwenden, um die Testbarkeit und Wartbarkeit des Codes zu verbessern.

In der Programmierung wird die Unveränderlichkeit (Immutability) oft geschätzt, da sie dazu beiträgt, den Code sicherer, vorhersehbarer und einfacher zu verstehen zu machen. Wenn Daten unveränderlich sind, können sie nicht von anderen Teilen des Programms oder von anderen Threads unerwartet geändert werden. Dies reduziert die Wahrscheinlichkeit von Fehlern und macht den Code oft einfacher zu verfolgen und zu testen.

In der gegebenen Aufgabenstellung mit Flask und der Blog-Verwaltung haben wir uns jedoch bewusst dafür entschieden, auf die strikte Einhaltung der Unveränderlichkeit zu verzichten. Es gibt dafür mehrere Gründe:

- Einfachheit und Verständlichkeit: Für Lernende, die gerade erst mit Flask und Webentwicklung beginnen, kann die Einführung von Immutability zusätzliche Komplexität bedeuten. Das Hauptziel dieser Aufgabe ist es, die Grundlagen von Flask, Routing und Datenmanipulation zu vermitteln.
- Performance: Das Erstellen neuer Datenstrukturen anstelle der Modifikation bestehender kann in einigen Fällen zu Performance-Einbußen führen, insbesondere wenn die Datenstrukturen groß sind.
- Natur der Daten: In vielen realen Webanwendungen, insbesondere solchen, die mit Datenbanken interagieren, werden Daten häufig geändert. Während Immutability in bestimmten Szenarien (z.B. bei funktionaler Programmierung oder in bestimmten Datenstrukturen) wertvoll ist, ist es nicht immer praktikabel oder notwendig, sie in allen Aspekten einer Anwendung durchzusetzen.

Es ist wichtig zu betonen, dass der Verzicht auf Immutability in dieser Aufgabe nicht bedeutet, dass sie in der realen Softwareentwicklung nicht wichtig oder wertvoll ist. Es handelt sich vielmehr um eine didaktische Entscheidung, um den Schwerpunkt auf bestimmte Lernziele zu legen und den Einstieg für Anfänger zu erleichtern.

Für den Zweck dieser Übung und um die Dinge einfach und verständlich zu halten, verzichten wir auf solche Ansätze und konzentrieren uns auf die Grundlagen von Flask.

⇒ *GitHub Repo für externe Besucher*

GitHub Repository https://github.com/templates-python/m323-lu06-a02-routingjson

Lernende am BZZ müssen den Link zum GitHub Classroom Assignment verwenden