LU06h - Erstellen einer RESTful API mit Flask und DAO-Klassen

Nachdem wir unsere DAO-Klassen und Datenmodelle definiert haben, können wir diese nun in einer RESTful API in Flask nutzen. Eine RESTful API ermöglicht es Clients, über HTTP-Methoden mit dem Server zu interagieren. In diesem Beispiel werden wir uns auf die CRUD-Operationen für die `ShoppingItem`-Ressource konzentrieren.

HTTP-Methoden und ihre Entsprechung in CRUD

Die folgenden HTTP-Methoden werden verwendet, um CRUD-Operationen auszuführen:

- **GET**: Lesen (Read)
- **POST**: Erstellen (Create)
- **PUT**: Aktualisieren (Update)
- **DELETE**: Löschen (Delete)

Beispiel für eine Flask-API

Nachfolgend finden Sie ein einfaches Beispiel, wie die `ShoppingItemDao`-Klasse in einer Flask-API verwendet werden kann:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/shoppingItems', methods=['GET'])
def get_items():
    items = [vars(item) for item in dao.get_all_items()]
    return jsonify(items), 200

@app.route('/shoppingItems/<int:item_id>', methods=['GET'])
def get_item(item_id):
    item = dao.get_item(item_id)
    if item:
        return jsonify(vars(item)), 200
    return jsonify({'error': 'Item not found'}), 404

@app.route('/shoppingItems', methods=['POST'])
def add_item():
    data = request.get_json()
    new_item = ShoppingItem(None, data['item_name'], data['quantity'])
    dao.add_item(new_item)
    return jsonify(vars(new_item)), 201
```

```python
@app.route('/shoppingItems/<int:item_id>', methods=['PUT'])
def update_item(item_id):
    data = request.get_json()
    item = dao.get_item(item_id)
    if item:
        item.item_name = data['item_name']
        item.quantity = data['quantity']
        dao.update_item(item)
        return jsonify(vars(item)), 200
    return jsonify({'error': 'Item not found'}), 404

@app.route('/shoppingItems/<int:item_id>', methods=['DELETE'])
def delete_item(item_id):
    if dao.delete_item(item_id):
        return jsonify({'success': True}), 200
    return jsonify({'error': 'Item not found'}), 404

if __name__ == '__main__':
    dao = ShoppingItemDao('example.db')
    app.run()
```

HTTP-Statuscodes

In obigem Beispiel werden HTTP-Statuscodes verwendet, um den Erfolg oder Misserfolg einer Operation anzugeben:

- **200 OK**: Die Anfrage war erfolgreich (für GET, PUT und DELETE Methoden).
- **201 Created**: Ein neuer Eintrag wurde erfolgreich erstellt (für POST Methode).
- **404 Not Found**: Die angeforderte Ressource konnte nicht gefunden werden.

Datenaustausch über JSON

In der API verwenden wir JSON (JavaScript Object Notation) für den Datenaustausch. Mit der `jsonify`-Funktion aus dem Flask-Paket können wir Python-Objekte einfach in JSON umwandeln und als HTTP-Antwort zurückgeben.

In ähnlicher Weise verwenden wir `request.get_json()` in Flask, um die JSON-Daten aus dem HTTP-Request-Body zu extrahieren.

Dies ermöglicht eine einfache und standardisierte Kommunikation zwischen dem Client und dem Server.

M323-LU06