LU06.L04 - Erstellen einer RESTful API für die ToDo-Liste mit Flask

main.py

main.py

```
from flask import Flask, jsonify, request
from todoItem import TodoItem  # Stellen Sie sicher, dass die TodoItem-Klasse importiert ist
from todoDao import TodoDao  # Stellen Sie sicher, dass die TodoDao-Klasse importiert ist

app = Flask(__name__)
dao = TodoDao('todo_example.db')
dao.create_table()


@app.route('/todos', methods=['POST'])
def add_todo():
    data = request.get_json()
    new_item = TodoItem(None, data['title'], data['is_completed'])
    dao.add_item(new_item)
    return jsonify({'message': 'Todo item created'}), 201


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


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


@app.route('/todos/<int:item_id>', methods=['PUT'])
def update_todo(item_id):
    data = request.get_json()
    updated_item = TodoItem(item_id, data['title'], data['is_completed'])
    if dao.update_item(updated_item):
```

```
        return jsonify({'message': 'Item updated'}), 200
    else:
        return jsonify({'message': 'Item not found or not updated'}),
404


@app.route('/todos/<int:item_id>', methods=['DELETE'])
def delete_todo(item_id):
    if dao.delete_item(item_id):
        return jsonify({'message': 'Item deleted'}), 200
    else:
        return jsonify({'message': 'Item not found or not deleted'}),
404


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

todoDao.py

todoDao.py

```
import sqlite3
from todoItem import TodoItem


# TODO: Implementiere die TodoDao-Klasse für CRUD-Operationen
class TodoDao:

    def __init__(self, db_file):
        self.conn = sqlite3.connect(db_file, check_same_thread=False)
        self.cursor = self.conn.cursor()

    def create_table(self):
        self.cursor.execute("""DROP TABLE IF EXISTS todo_items""")
        self.cursor.execute(
            """CREATE TABLE IF NOT EXISTS todo_items (item_id INTEGER
PRIMARY KEY, title TEXT, is_completed BOOLEAN)"""
        )
        self.conn.commit()

    def add_item(self, todo_item):
        self.cursor.execute(
            "INSERT INTO todo_items (title, is_completed) VALUES (?,
?)",
            (todo_item.title, todo_item.is_completed),
        )
```

```
        self.conn.commit()

    def get_item(self, item_id):
        self.cursor.execute("SELECT * FROM todo_items WHERE item_id =
?", (item_id,))
        row = self.cursor.fetchone()
        if row:
            return TodoItem(row[0], row[1], row[2])
        return None

    def get_all_items(self):
        self.cursor.execute("SELECT * FROM todo_items")
        rows = self.cursor.fetchall()
        todo_items = [TodoItem(row[0], row[1], row[2]) for row in rows]
        return todo_items

    def update_item(self, todo_item):
        self.cursor.execute(
            "UPDATE todo_items SET title = ?, is_completed = ? WHERE
item_id = ?",
            (todo_item.title, todo_item.is_completed,
todo_item.item_id),
        )
        if self.cursor.rowcount > 0:
            self.conn.commit()
            return True
        return False

    def delete_item(self, item_id):
        self.cursor.execute("DELETE FROM todo_items WHERE item_id = ?",
(item_id,))
        if self.cursor.rowcount > 0:
            self.conn.commit()
            return True
        return False

    def close(self):
        self.conn.close()
```

todoItem.py

todoItem.py

```
from dataclasses import dataclass


# TODO: Implementiere die TodoItem-Klasse mit @dataclass
@dataclass
```

```
class TodoItem:
    item_id: int
    title: str
    is_completed: bool
```