Praktisches Labor: CRUD-Anwendungsdesign mit zusätzlichen Funktionen in Flask

Geschätzte Zeit: **60** Minuten

Übersicht

CRUD, was für Erstellen, Lesen, Aktualisieren und Löschen steht, sind grundlegende Funktionen, die jede auf einer Datenbank basierende Anwendung besitzen muss. Die Entwicklung dieser Funktionen erfordert zusätzliches Wissen im Umgang mit Routen und Anfragen. Sie benötigen auch mehrere HTML-Schnittstellen für Endpunkte, um unterschiedliche Anfragen zu bedienen. Das Ziel dieses Labors ist es daher, Ihnen zusätzliche Übung im Umgang mit Flask zu geben und eine voll funktionsfähige Webanwendung zu entwickeln, die CRUD-Operationen durchführen kann.

Für dieses Labor werden Sie ein System zur Aufzeichnung finanzieller Transaktionen entwickeln. Das System muss in der Lage sein, **eine** neue Eingabe zu **erstellen**, **bestehende** Eingaben zu **lesen**, **bestehende** Eingaben zu **aktualisieren** und **bestehende** Eingaben zu **löschen**.

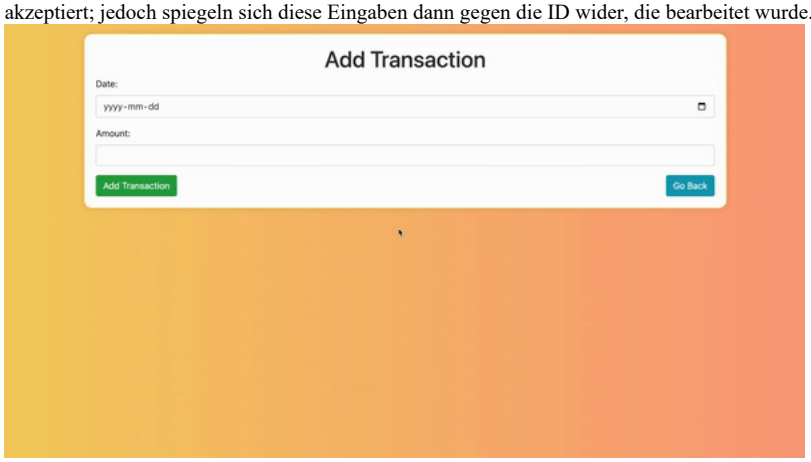
Ziele

Nach Abschluss dieses Labors werden Sie in der Lage sein:

- Die „Erstellen“-Operation zur Hinzufügung von Transaktionseinträgen zu implementieren
- Die „Lesen“-Operation zum Zugriff auf die Liste der Transaktionseinträge zu implementieren
- Die „Aktualisieren“-Operation zur Aktualisierung der Details eines bestimmten Transaktionseintrags zu implementieren
- Die „Löschen“-Operation zum Löschen eines Transaktionseintrags zu implementieren.

Nachdem Sie die Anwendung entwickelt haben, wird sie wie in der Animation angezeigt funktionieren.

Die Anwendung hat drei verschiedene Webseiten. Die erste zeigt alle aufgezeichneten Transaktionen an. Diese Seite heißt Transaktionsaufzeichnungen und zeigt alle Transaktionseinträge an, die im System erstellt wurden. Diese Seite bietet auch die Möglichkeit, die verfügbaren Einträge zu bearbeiten und zu löschen. Die Option zum Hinzufügen eines Eintrags ist ebenfalls auf dieser Seite verfügbar. Die zweite Seite ist „Transaktion hinzufügen“, die verwendet wird, wenn der Benutzer auf der vorherigen Seite einen Eintrag hinzufügen möchte. Der Benutzer fügt die Werte für Datum und Betrag des neuen Eintrags hinzu. Die dritte Seite ist „Transaktion bearbeiten“, zu der der Benutzer navigiert, wenn er die Option zum Bearbeiten des Eintrags auswählt. Auch auf dieser Seite werden Datum und Betrag als Eingaben akzeptiert; jedoch spiegeln sich diese Eingaben dann gegen die ID wider, die bearbeitet wurde.



Hinweis: Diese Plattform ist nicht persistent. Es wird empfohlen, eine Kopie Ihres Codes auf Ihren lokalen Maschinen zu behalten und von Zeit zu Zeit Änderungen zu speichern. Falls Sie das Labor erneut besuchen, müssen Sie die Dateien in dieser Laborumgebung mit den gespeicherten Kopien von Ihren Maschinen neu erstellen.

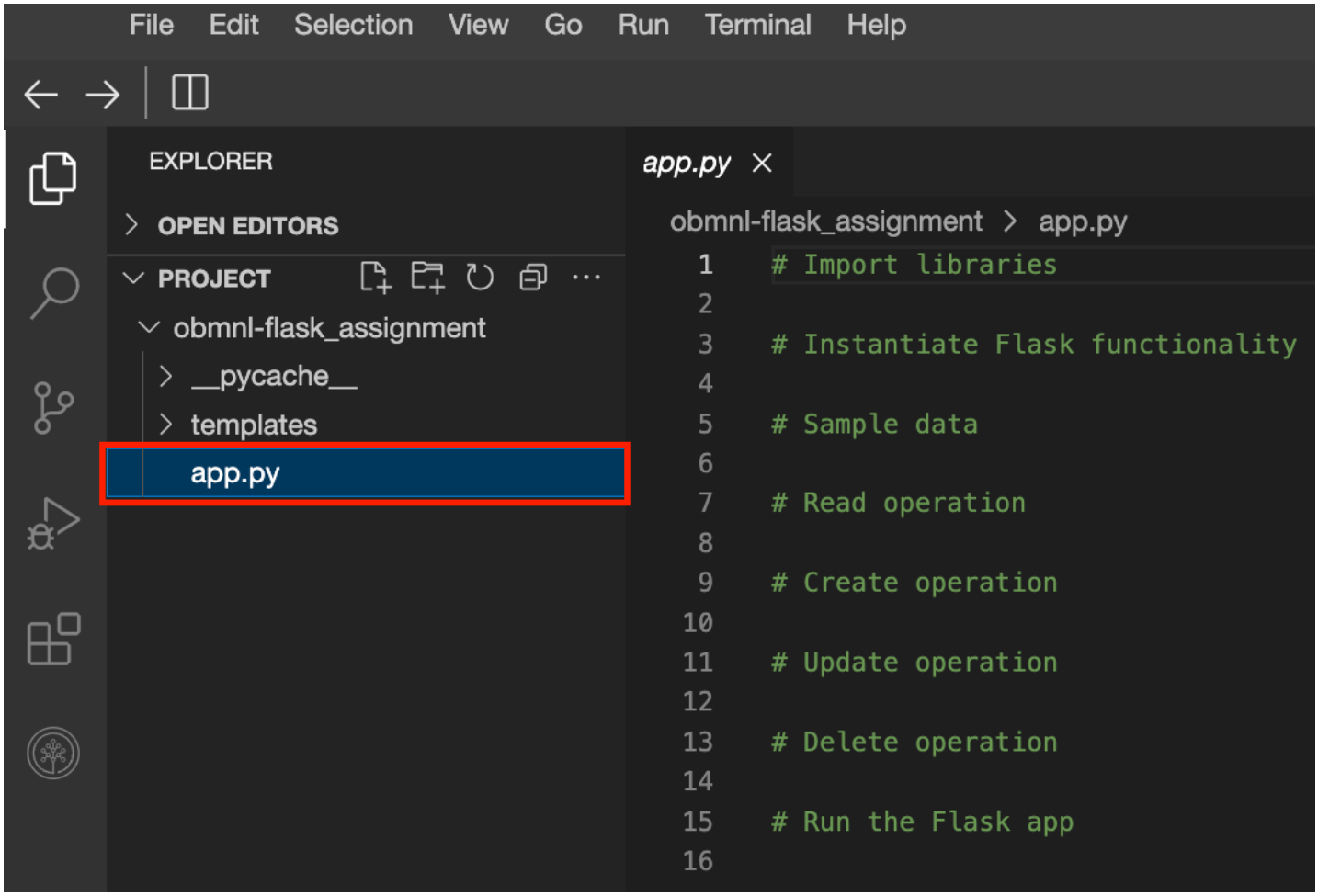
Lassen Sie uns anfangen!

Klonen des Projekt-Repositorys

Dieses Labor erfordert mehrere HTML-Schnittstellendateien, die bereits für Sie erstellt wurden. Sie müssen die Ordnerstruktur mit dem folgenden Befehl in die IDE-Schnittstelle klonen.

```
git clone https://github.com/ibm-developer-skills-network/obmnl-flask_assignment.git
```

Wenn der Befehl erfolgreich ausgeführt wird, muss der Projekt-Tab die Ordnerstruktur wie im Bild gezeigt haben. Der Stammordner, `obmnl-flask_assignment`, sollte den Ordner `templates` und eine Datei `app.py` enthalten. Der Ordner `templates` hat alle erforderlichen HTML-Dateien: `edit.html`, `form.html` und `transactions.html`. Im Laufe dieses Labors werden Sie die erforderlichen Funktionen in `app.py` implementieren.



Erste Einrichtung

In der `app.py`-Datei müssen Sie die erforderlichen Module von Flask importieren und die Flask-Anwendung instanziieren.
Für dieses Labor müssen Sie die folgenden Funktionen aus der ***flask***-Bibliothek importieren.

- Flask - um die Anwendung zu instanziieren
- request - um die `GET`- und `POST`-Anfragen zu verarbeiten
- url_for - um die URL für eine gegebene Funktion mithilfe ihres Decorators zuzugreifen
- redirect - um Zugriffsanforderungen gemäß den Anforderungen umzuleiten
- render_template - um die HTML-Seite zu rendern

Nachdem Sie die Funktionen importiert haben, instanziieren Sie die Anwendung in einer Variablen `app`.

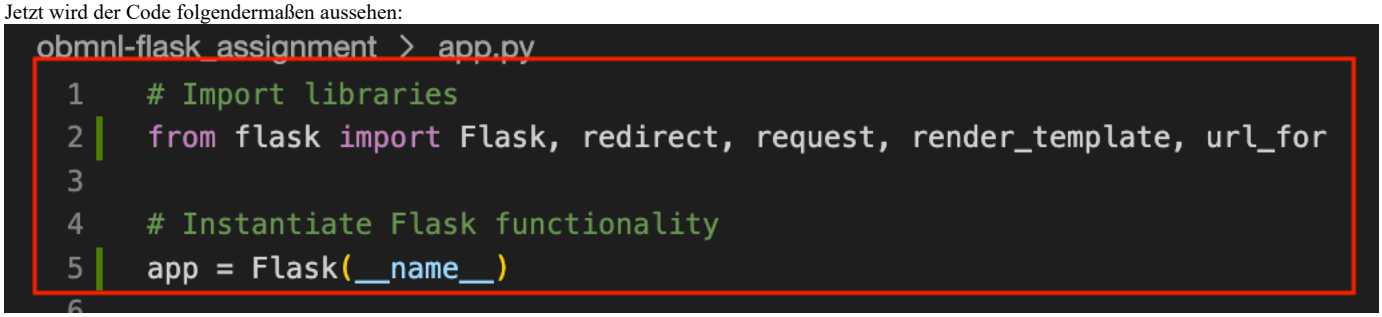
▼ *Hier klicken für einen Hinweis*

```
from flask import <functions>
```

▼ *Klicken Sie hier für die Lösung*

```
# Import libraries
from flask import Flask, redirect, request, render_template, url_for
# Instantiate Flask functionality
app = Flask(__name__)
```

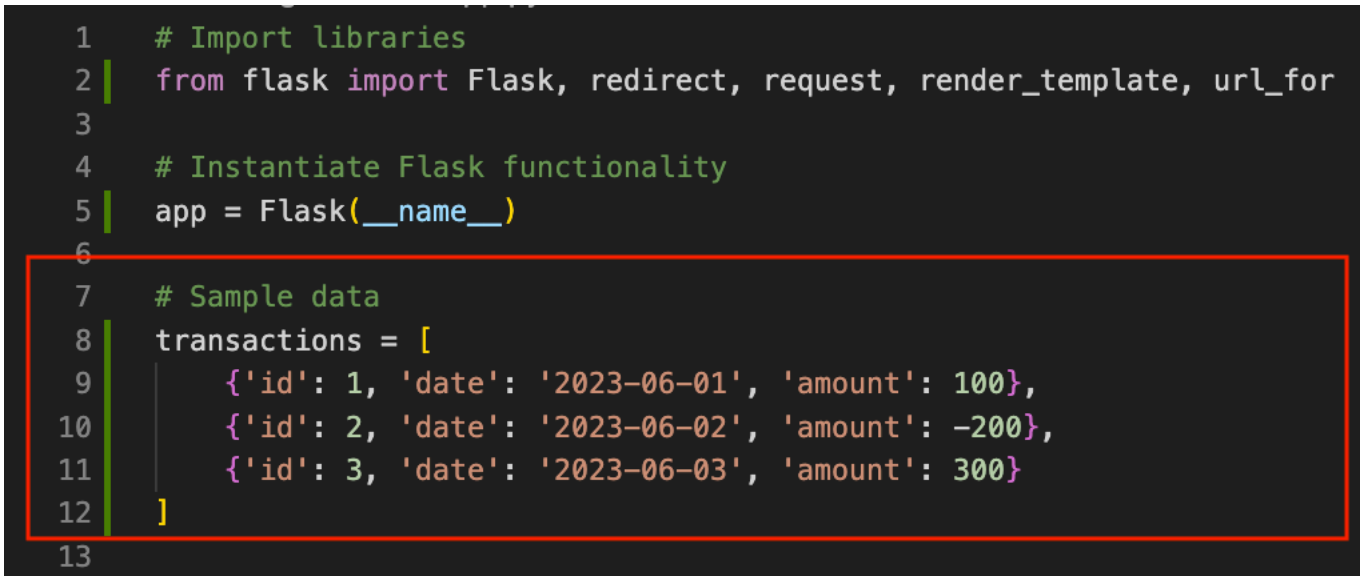
Jetzt wird der Code folgendermaßen aussehen:



Als Nächstes erstellen wir eine Liste von Beispieltransaktionen zu Testzwecken. Sie können davon ausgehen, dass die Transaktionen bereits in der Benutzeroberfläche vorhanden sind, wenn sie zum ersten Mal ausgeführt wird. Bitte beachten Sie, dass dieser Schritt völlig optional ist und die Funktionalität, die Sie in diesem Labor entwickeln werden, nicht beeinflusst. Fügen Sie den Code-Snippet wie unten gezeigt zu `app.py` hinzu.

```
# Sample data
transactions = [
    {'id': 1, 'date': '2023-06-01', 'amount': 100},
    {'id': 2, 'date': '2023-06-02', 'amount': -200},
    {'id': 3, 'date': '2023-06-03', 'amount': 300}
]
```

▼ *Hier klicken für die Lösung*



Die Reihenfolge, in der Sie die Funktionen entwickeln werden, ist wie folgt:
1. Lesen
2. Erstellen
3. Aktualisieren
4. Löschen

Der Grund, die Funktion **Lesen** vor den anderen Funktionen zu implementieren, besteht darin, dass man bei jeder Erstellung, Aktualisierung oder Löschung einer neuen Transaktion auf die Seite mit allen Transaktionen umleiten kann. Daher muss die Funktion zum Lesen der bestehenden Transaktionen existieren, bevor die anderen implementiert werden.

Lesevorgang

Um den **Lesevorgang** zu implementieren, müssen Sie eine Route erstellen, die eine Liste aller Transaktionen anzeigt. Diese Route verarbeitet `GET`-Anfragen, die verwendet werden, um Daten in `app.py` abzurufen und anzuzeigen.

Die wichtigsten Schritte zur Implementierung des Lesevorgangs sind wie folgt:

1. Erstellen Sie eine Funktion namens `get_transactions`, die `render_template` verwendet, um eine HTML-Vorlage namens `transactions.html` zurückzugeben. Diese Funktion sollte die Transaktionen an die Vorlage zur Anzeige übergeben.

2. Verwenden Sie den Flask-Decorator `@app.route`, um diese Funktion der Wurzel-(`/`) URL zuzuordnen. Das bedeutet, dass Flask die Funktion `get_transactions` ausführt und deren Ergebnis zurückgibt, wenn ein Benutzer die Basis-URL Ihrer Anwendung besucht.

▼ *Klicken Sie hier für einen Hinweis*
Diese Funktion ist eine grundlegende render_template-Funktion, wie sie in den vorherigen Laboren implementiert wurde.
▼ *Klicken Sie hier für die Lösung*

```
# Read operation: List all transactions
```

```
@app.route("/")
def get_transactions():
    return render_template("transactions.html", transactions=transactions)
```

Jetzt wird der Code so aussehen:

```
# Import libraries
from flask import Flask, redirect, request, render_template, url_for

# Instantiate Flask functionality
app = Flask(__name__)

# Sample data
transactions = [
    {'id': 1, 'date': '2023-06-01', 'amount': 100},
    {'id': 2, 'date': '2023-06-02', 'amount': -200},
    {'id': 3, 'date': '2023-06-03', 'amount': 300}
]

# Read operation: List all transactions
@app.route("/")
def get_transactions():
    return render_template("transactions.html", transactions=transactions)
```

Erstellen Operation

Für die **Erstellen**-Operation werden Sie eine Route implementieren, die es Benutzern ermöglicht, neue Transaktionen hinzuzufügen. Dies umfasst die Verarbeitung sowohl von `GET`- als auch von `POST`-HTTP-Anfragen - `GET`, um das Formular dem Benutzer anzuzeigen, und `POST`, um die vom Benutzer gesendeten Formulardaten zu verarbeiten.

Hier ist die Liste der Schritte zur Implementierung der Erstellen-Operation.

1. Erstellen Sie eine Funktion mit dem Namen `add_transaction`.
2. Verwenden Sie `add` als Dekorator für diese Funktion. Stellen Sie sicher, dass Sie sowohl `GET` als auch `POST` als mögliche Methoden übergeben.
3. Wenn die Anfragemethode `GET` ist, verwenden Sie die Funktion `render_template`, um ein HTML-Formular mit einer Vorlage namens `form.html` anzuzeigen. Dieses Formular ermöglicht es Benutzern, Daten für eine neue Transaktion einzugeben.
4. Wenn die Anfragemethode `POST` ist, verwenden Sie `request.form`, um die Formulardaten zu extrahieren, eine neue Transaktion zu erstellen, sie zur Transaktionsliste hinzuzufügen und dann `redirect` und `url_for` zu verwenden, um den Benutzer zurück zur Liste der Transaktionen zu senden.
5. Die neue Transaktion wird in folgendem Format an die Lese-Funktion übergeben.

```
transation = {
             'id': len(transactions)+1
             'date': request.form['date']
             'amount': float(request.form['amount'])
            }
```

Hier analysiert die Funktion request.form die Informationen, die aus dem im Formular vorgenommenen Eintrag empfangen wurden.

▼ *Klicken Sie hier für einen Hinweis*
Der Inhalt der Funktion add_transaction benötigt die folgenden Implementierungen.

Für die `POST`-Methode erstellen Sie die neue Transaktion wie oben gezeigt, fügen Sie sie der bestehenden Liste von Transaktionen hinzu und leiten Sie zur URL für die **Lese**-Operation weiter.

Für die `GET`-Methode rendern Sie die Seite form.html, die die Informationen von der Benutzeroberfläche akzeptiert.
▼ *Klicken Sie hier für die Lösung*

```
# Create operation: Display add transaction form
# Route to handle the creation of a new transaction
@app.route("/add", methods=["GET", "POST"])
def add_transaction():
    # Check if the request method is POST (form submission)
    if request.method == 'POST':
        # Create a new transaction object using form field values
        transaction = {
            'id': len(transactions) + 1,            # Generate a new ID based on the current length of the transactions list
            'date': request.form['date'],           # Get the 'date' field value from the form
            'amount': float(request.form['amount']) # Get the 'amount' field value from the form and convert it to a float
        }
        # Append the new transaction to the transactions list
        transactions.append(transaction)
        # Redirect to the transactions list page after adding the new transaction
        return redirect(url_for("get_transactions"))

    # If the request method is GET, render the form template to display the add transaction form
    return render_template("form.html")
```

Jetzt wird der Code so aussehen:

```
14  # Read operation: List all transactions
15  @app.route("/")
16  def get_transactions():
17      return render_template("transactions.html", transactions=transactions)
18
19  # Create operation: Display add transaction form
20  @app.route("/add", methods=["GET", "POST"])
21  def add_transaction():
22      if request.method == 'POST':
23          # Create a new transaction object using form field values
24          transaction = {
25              'id': len(transactions) + 1,
26              'date': request.form['date'],
27              'amount': float(request.form['amount'])
28          }
29          # Append the new transaction to the list
30          transactions.append(transaction)
31
32          # Redirect to the transactions list page
33          return redirect(url_for("get_transactions"))
34
35      # Render the form template to display the add transaction form
36      return render_template("form.html")
```

Hinweis: Die Anweisungen außerhalb des `if`-Falls sind standardmäßig der `else`-Fall. Die Anweisungen im `if`-Fall enden mit einer Rückgabewertung; daher wird immer nur einer der beiden Fälle zur gleichen Zeit ausgeführt.

::page{title=”Aktualisierungsoperation”}

Für die **Aktualisierungs**operation müssen Sie eine Route implementieren, die es Benutzern ermöglicht, vorhandene Transaktionen zu aktualisieren. Sie werden erneut sowohl `GET`- als auch `POST`-HTTP-Anfragen behandeln - `GET`, um die aktuellen Transaktionsdaten in einem Formular anzuzeigen, und `POST`, um die aktualisierten Daten zu verarbeiten, die vom Benutzer gesendet werden.

Vervollständigen Sie die folgenden Schritte, um die Aktualisierungsoperation zu implementieren.

1. Erstellen Sie eine Funktion namens `edit_transaction`, die sowohl `GET`- als auch `POST`-Anfragen behandelt. Diese Funktion sollte einen Parameter `transaction_id` akzeptieren.

2. Dekorieren Sie die Funktion mit `@app.route` und verwenden Sie die Routenzeichenfolge `/edit/<int:transaction_id>`. Der Teil `<int:transaction_id>` in der URL ist ein Platzhalter für jede ganze Zahl. Flask wird diese ganze Zahl Ihrer Funktion als Argument `transaction_id` übergeben.

3. Wenn die Anfragemethode `GET` ist, suchen Sie die Transaktion mit der ID, die mit `transaction_id` übereinstimmt, und verwenden Sie `render_template`, um ein Formular anzuzeigen, das mit den aktuellen Daten der Transaktion vorab ausgefüllt ist, unter Verwendung einer Vorlage namens `edit.html`.

4. Wenn die Anfragemethode `POST` ist, verwenden Sie request.form, um die aktualisierten Daten zu erhalten, suchen Sie die Transaktion mit der ID, die mit `transaction_id` übereinstimmt, und ändern Sie deren Daten, und leiten Sie den Benutzer dann zurück zur Liste der Transaktionen.

▼ *Klicken Sie hier für die Lösung*

```
# Update operation: Display edit transaction form
# Route to handle the editing of an existing transaction
@app.route("/edit/<int:transaction_id>", methods=["GET", "POST"])
def edit_transaction(transaction_id):
    # Check if the request method is POST (form submission)
    if request.method == 'POST':
        # Extract the updated values from the form fields
        date = request.form['date']           # Get the 'date' field value from the form
        amount = float(request.form['amount'])# Get the 'amount' field value from the form and convert it to a float
        # Find the transaction with the matching ID and update its values
        for transaction in transactions:
            if transaction['id'] == transaction_id:
                transaction['date'] = date       # Update the 'date' field of the transaction
                transaction['amount'] = amount   # Update the 'amount' field of the transaction
                break                            # Exit the loop once the transaction is found and updated
        # Redirect to the transactions list page after updating the transaction
        return redirect(url_for("get_transactions"))

    # If the request method is GET, find the transaction with the matching ID and render the edit form
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            # Render the edit form template and pass the transaction to be edited
            return render_template("edit.html", transaction=transaction)
    # If the transaction with the specified ID is not found, handle this case (optional)
    return {"message": "Transaction not found"}, 404
```

Jetzt wird der Code so aussehen:

```
38  # Update operation: Display edit transaction form
39  @app.route("/edit/<int:transaction_id>", methods=["GET", "POST"])
40  def edit_transaction(transaction_id):
41      if request.method == 'POST':
42          # Extract the updated values from the form fields
43          date = request.form['date']
44          amount = float(request.form['amount'])
45
46          # Find the transaction with the matching ID and update its values
47          for transaction in transactions:
48              if transaction['id'] == transaction_id:
49                  transaction['date'] = date
50                  transaction['amount'] = amount
51                  break
52
53          # Redirect to the transactions list page
54          return redirect(url_for("get_transactions"))
55
56      # Find the transaction with the matching ID and render the edit form
57      for transaction in transactions:
58          if transaction['id'] == transaction_id:
59              return render_template("edit.html", transaction=transaction)
60
```

Hinweis: Es kann mehrere Möglichkeiten geben, das gleiche Ergebnis zu erzielen. Bitte verwenden Sie die oben angegebene Lösung nur als Referenz.

::page{title="Löschoperation"}

Schließlich müssen Sie eine Route implementieren, die es Benutzern ermöglicht, vorhandene Transaktionen zu löschen.

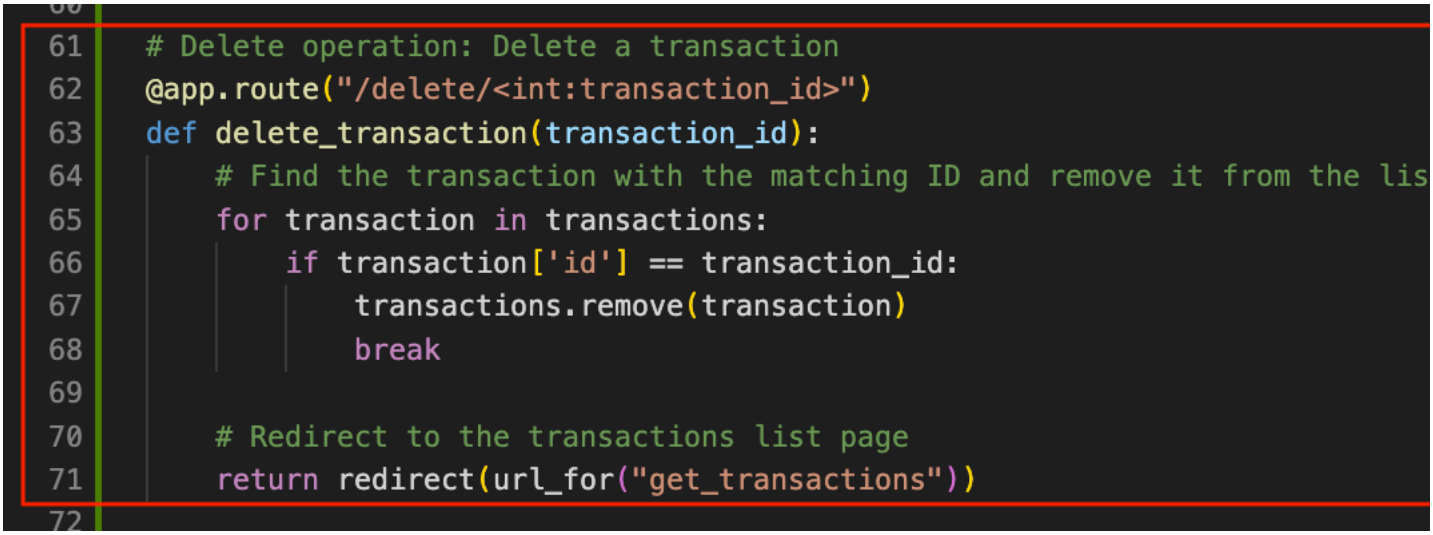
Vervollständigen Sie die folgenden Schritte, um die Löschoperation zu implementieren.

1. Erstellen Sie eine Funktion mit dem Namen `delete_transaction`, die einen Parameter `transaction_id` entgegennimmt.
2. Dekorieren Sie die Funktion mit `@app.route` und verwenden Sie den Routen-String `/delete/<int:transaction_id>`. Der Teil `<int:transaction_id>` in der URL ist ein Platzhalter für jede Ganzzahl. Flask übergibt diese Ganzzahl als Argument `transaction_id` an Ihre Funktion.
3. Im Funktionskörper suchen Sie die Transaktion mit der ID, die mit `transaction_id` übereinstimmt, und entfernen Sie sie aus der Liste der Transaktionen, und `redirect` den Benutzer zurück zur Liste der Transaktionen.

▼ *Hier klicken für die Lösung*

```
# Delete operation: Delete a transaction
# Route to handle the deletion of an existing transaction
@app.route("/delete/<int:transaction_id>")
def delete_transaction(transaction_id):
    # Find the transaction with the matching ID and remove it from the list
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            transactions.remove(transaction)  # Remove the transaction from the transactions list
            break  # Exit the loop once the transaction is found and removed
    # Redirect to the transactions list page after deleting the transaction
    return redirect(url_for("get_transactions"))
```

Jetzt wird der Code so aussehen:



::page{title="Abschluss Schritte und Ausführen der Anwendung"}

Überprüfen Sie, ob das aktuelle Skript das Hauptprogramm ist (das heißt, es wurde nicht aus einem anderen Skript importiert) mit der Bedingung `if __name__ == "__main__":`.

Wenn die Bedingung wahr ist, rufen Sie `app.run(debug=True)` auf, um den Flask-Entwicklungsserver mit aktiviertem Debug-Modus zu starten. Dadurch können Sie detaillierte Fehlermeldungen in Ihrem Browser anzeigen, falls etwas schiefgeht.

```
# Run the Flask app
if __name__ == "__main__":
    app.run(debug=True)
```

Jetzt sieht der Code so aus:

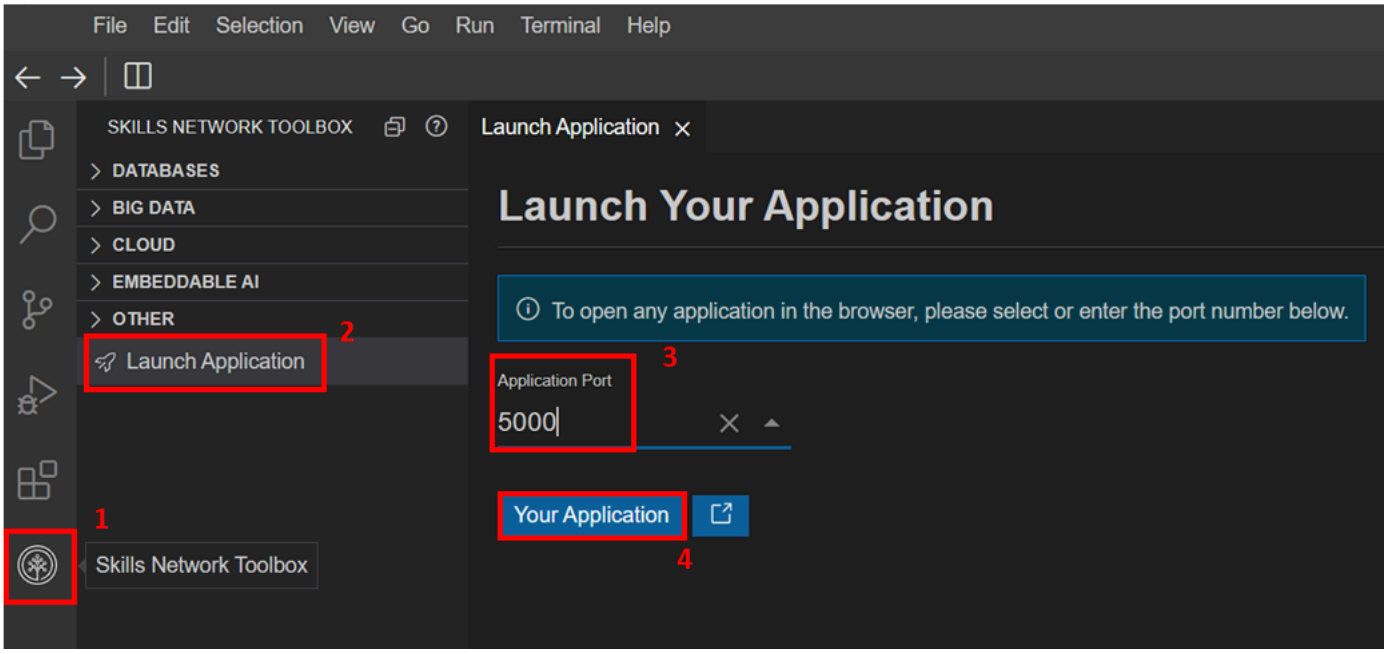
```python
# Delete operation: Delete a transaction
@app.route("/delete/<int:transaction_id>")
def delete_transaction(transaction_id):
    # Find the transaction with the matching ID and remove it from the lis
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            transactions.remove(transaction)
            break

    # Redirect to the transactions list page
    return redirect(url_for("get_transactions"))


# Run the Flask app
if __name__ == "__main__":
    app.run(debug=True)
```

Der Code ist jetzt vollständig. Führen Sie die Datei `app.py` aus einem Terminal mit dem Befehl aus:

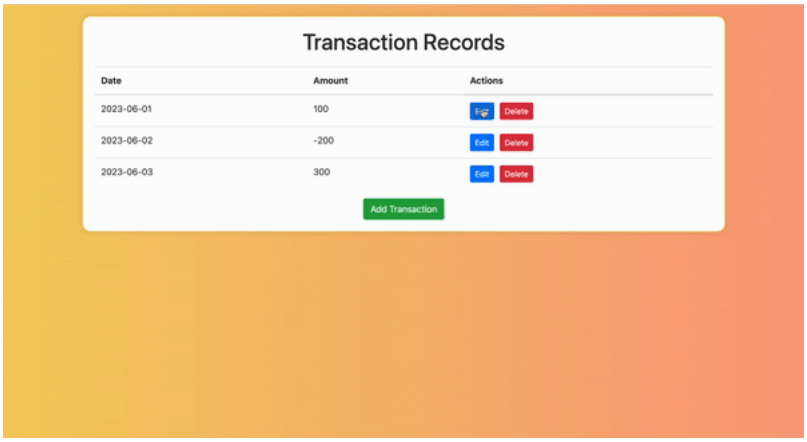
```
python3.11 app.py
```

Standardmäßig startet Flask die Anwendung auf LocalHost:5000. Wie im Bild dargestellt,

1. Starten Sie die Anwendung, indem Sie zur Skills Network Library gehen und auf `Launch Application` klicken.
2. Geben Sie `5000` in die Portnummer ein und starten Sie das Anwendungsfenster.



Die endgültige Anwendung sieht so aus.



::page{title="Laborhilfe"}

Falls Sie während der Durchführung aller Schritte auf einen Fehler stoßen, wird hier der endgültige Code für `app.py` als Referenz bereitgestellt. Bitte beachten Sie, dass dies nur als letzte Möglichkeit verwendet werden sollte, um sicherzustellen, dass Sie das beabsichtigte Lernen durch dieses Labor erreichen.

▼ *Endgültiger Code für app.py*

```
# Notwendige Bibliotheken von Flask importieren
from flask import Flask, redirect, request, render_template, url_for
# Flask-Anwendung instanziieren
app = Flask(__name__)
# Beispieldaten, die Transaktionen darstellen
transactions = [
    {'id': 1, 'date': '2023-06-01', 'amount': 100},
    {'id': 2, 'date': '2023-06-02', 'amount': -200},
    {'id': 3, 'date': '2023-06-03', 'amount': 300}
]
# Leseoperation: Route zur Auflistung aller Transaktionen
@app.route("/")
def get_transactions():
    # Template für die Transaktionsliste rendern und die Transaktionsdaten übergeben
    return render_template("transactions.html", transactions=transactions)
# Erstellungsoperation: Route zur Anzeige und Verarbeitung des Formulars zur Hinzufügung von Transaktionen
@app.route("/add", methods=["GET", "POST"])
def add_transaction():
    if request.method == 'POST':
        # Formulardaten extrahieren, um ein neues Transaktionsobjekt zu erstellen
        transaction = {
            'id': len(transactions) + 1,         # Eine neue ID basierend auf der aktuellen Länge der Transaktionsliste generieren
            'date': request.form['date'],        # Den Wert des Feldes 'date' aus dem Formular abrufen
            'amount': float(request.form['amount']) # Den Wert des Feldes 'amount' aus dem Formular abrufen und in einen Float umwandeln
        }
        # Die neue Transaktion zur Transaktionsliste hinzufügen
        transactions.append(transaction)
        # Nach dem Hinzufügen der neuen Transaktion zur Transaktionsliste umleiten
        return redirect(url_for("get_transactions"))
    # Das Formular-Template rendern, um das Formular zur Hinzufügung von Transaktionen anzuzeigen, wenn die Anforderungsmethode GET ist
    return render_template("form.html")
# Aktualisierungsoperation: Route zur Anzeige und Verarbeitung des Formulars zur Bearbeitung von Transaktionen
@app.route("/edit/<int:transaction_id>", methods=["GET", "POST"])
def edit_transaction(transaction_id):
    if request.method == 'POST':
        # Die aktualisierten Werte aus den Formularfeldern extrahieren
        date = request.form['date']
        amount = float(request.form['amount'])
        # Die Transaktion mit der übereinstimmenden ID finden und ihre Werte aktualisieren
        for transaction in transactions:
            if transaction['id'] == transaction_id:
                transaction['date'] = date       # Das Feld 'date' der Transaktion aktualisieren
                transaction['amount'] = amount   # Das Feld 'amount' der Transaktion aktualisieren
                break                            # Die Schleife verlassen, sobald die Transaktion gefunden und aktualisiert wurde
        # Nach der Aktualisierung der Transaktion zur Transaktionsliste umleiten
        return redirect(url_for("get_transactions"))
    # Die Transaktion mit der übereinstimmenden ID finden und das Bearbeitungsformular rendern, wenn die Anforderungsmethode GET ist
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            # Das Bearbeitungsformular-Template rendern und die zu bearbeitende Transaktion übergeben
            return render_template("edit.html", transaction=transaction)
# Löschoperation: Route zum Löschen einer Transaktion
@app.route("/delete/<int:transaction_id>")
def delete_transaction(transaction_id):
    # Die Transaktion mit der übereinstimmenden ID finden und aus der Liste entfernen
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            transactions.remove(transaction)  # Die Transaktion aus der Transaktionsliste entfernen
            break                             # Die Schleife verlassen, sobald die Transaktion gefunden und entfernt wurde
    # Nach dem Löschen der Transaktion zur Transaktionsliste umleiten
    return redirect(url_for("get_transactions"))
# Die Flask-Anwendung ausführen
if __name__ == "__main__":
    app.run(debug=True)
```

::page{title="Testen der Benutzeroberfläche"}

Sobald Ihre Anwendung bereit ist, versuchen Sie die CRUD-Operationen in der gestarteten Anwendung. Mögliche Aufgaben zum Testen der Anwendung könnten sein:

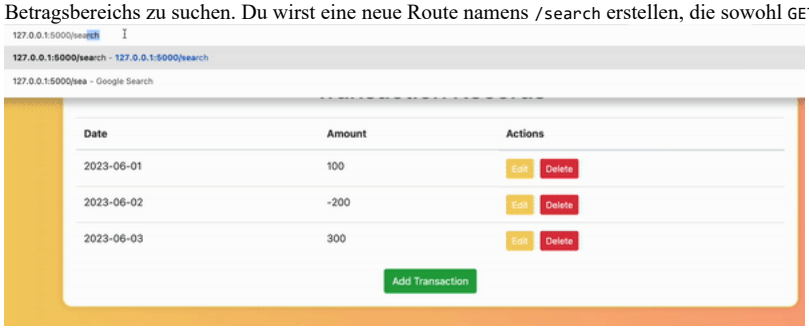
1. Klicken Sie auf die Schaltfläche "Hinzufügen", um das Formular zu öffnen und eine neue Transaktion hinzuzufügen.
2. Klicken Sie auf die Schaltfläche "Bearbeiten" für eine beliebige Transaktion und aktualisieren Sie die Informationen (Datum und Betrag) für die Transaktion.
3. Klicken Sie auf die Schaltfläche "Löschen" für eine beliebige Transaktion, um sie aus der Liste zu entfernen.
4. Überprüfen Sie, ob die Transaktionen korrekt angezeigt werden.

::page{title="Übungsaufgaben"}

Die folgenden sind einige Übungsaufgaben für interessierte Lernende. Wir stellen keine Lösungen für diese Aufgaben zur Verfügung, um die Lernenden zu ermutigen, es selbst zu versuchen. Bitte zögere nicht, das Diskussionsforum des Kurses zu nutzen, um deine Meinungen zur Lösung mit anderen interessierten Lernenden zu teilen.

Übung 1: Transaktionen suchen

In dieser Übung wirst du eine neue Funktion zur Anwendung hinzufügen, die es den Nutzern ermöglicht, nach Transaktionen innerhalb eines bestimmten Betragsbereichs zu suchen. Du wirst eine neue Route namens `/search` erstellen, die sowohl `GET`- als auch `POST`-Anfragen in `app.py` verarbeitet.



Anweisungen:

1. Erstelle eine neue Funktion namens `search_transactions` und verwende den `@app.route`-Dekorator, um sie der URL `/search` zuzuordnen.
2. Überprüfe innerhalb der Funktion, ob die Anfragemethode `POST` ist. Wenn ja, rufe die Mindest- und Höchstbetragswerte aus den vom Benutzer übermittelten Formulardaten ab. Konvertiere diese Werte in Gleitkommazahlen.
3. Filtere die Transaktionsliste basierend auf dem vom Benutzer angegebenen Betragsbereich. Erstelle eine neue Liste, `filtered_transactions`, die nur die Transaktionen enthält, deren Betrag innerhalb des angegebenen Bereichs liegt. Du kannst dafür eine Listenkomprehension verwenden.
4. Übergebe die Liste `filtered_transactions` an die `transactions.html`-Vorlage, indem du die Funktion `render_template` verwendest. In dieser Vorlage zeige die Transaktionen ähnlich der bestehenden `transactions.html`-Vorlage an.
5. Wenn die Anfragemethode `GET` ist, rendere eine neue Vorlage namens `search.html`. Diese Vorlage sollte ein Formular enthalten, das es den Nutzern ermöglicht, die Mindest- und Höchstbetragswerte für die Suche einzugeben.

Übung 2: Gesamtsaldo

In dieser Übung wirst du eine neue Funktion hinzufügen, die den Gesamtsaldo aller Transaktionen berechnet und anzeigt. Du wirst die Route in `app.py` erstellen.

Anweisungen:

1. Erstelle eine neue Funktion namens `total_balance` und verwende den `@app.route`-Dekorator, um sie der URL `/balance` zuzuordnen.
2. Berechne innerhalb der Funktion den Gesamtsaldo, indem du die Betragwerte aller Transaktionen in der Transaktionsliste summierst.
3. Gib den Gesamtsaldo als Zeichenkette im Format "Total Balance: {balance}" zurück.
4. Um den Gesamtsaldo anzuzeigen, musst du keine neue Vorlage erstellen. Stattdessen wirst du die `transactions.html`-Vorlage ändern, um den Gesamtsaldo am Ende der Tabelle anzuzeigen.
5. Füge nach der Anzeige der Transaktionsliste in der `transactions.html`-Vorlage eine neue Zeile hinzu, um den Gesamtsaldo anzuzeigen. Du kannst die gleiche Funktion `render_template` wie zuvor verwenden und sowohl die Transaktionsliste als auch den Gesamtsaldo übergeben.

::page{title="Fazit"}

Herzlichen Glückwunsch zum Abschluss dieses Labors.

In diesem Labor haben Sie gelernt, wie man:

- CRUD-Funktionalität in einer Datenbankanwendung implementiert.
- Zusätzliche Funktionen aus der Flask-Bibliothek für fortgeschrittenes Routing und Anfrageverwaltung verwendet.
- Routing zwischen mehreren HTML-Dateien gemäß den Anforderungen verwaltet.

Author(s)

Vicky Kuo

Additional Contributor

Abhishek Gagneja

Changelog

Datum	Version	Geändert von	Änderungsbeschreibung
2023-07-24	2.0	Steve Hord	QA-Durchgang mit Bearbeitungen
2023-07-15	1.0	Vicky Kuo	Erste Version erstellt