CRUD-Operationen mit zusätzlichen Funktionen in Flask

Geschätzte benötigte Zeit: 15 Minuten

Überblick

Erstellen, Lesen, Aktualisieren und **Löschen** (CRUD) sind grundlegende Funktionen, die jede Anwendung mit einer Datenbank ausführen muss. Um die CRUD-Operationen effektiv umzusetzen, müssen Sie verschiedene HTTP-Methoden verwalten, wie z.B. `GET`- und `POST`-Anfragen. Eine `GET`-Anfrage wird allgemein verwendet, um Daten abzurufen oder zu lesen und wird oft verwendet, um ein Formular anzuzeigen. Eine `POST`-Anfrage wird häufig verwendet, um Daten zu senden, um Daten zu erstellen oder zu aktualisieren. Ein Beispiel für eine POST-Anfrage ist die Formularübermittlung.

Diese Lektüre soll Sie in zusätzliche Funktionen von Flask einführen, wobei der Schwerpunkt auf den CRUD-Funktionen liegt. Sie werden auch lernen, wie diese Funktionen miteinander verknüpft sind und wie sie die verschiedenen HTML-Dateien, dynamischen Routen und verschiedenen HTTP-Methoden nutzen.

Ziele

In dieser Lektüre werden Sie:

- **Formulardaten abrufen**, um Benutzereingaben mit `flask.request.form` in `POST`-Anfragen zu erfassen
- Die Benutzernavigation steuern: mithilfe der `redirect`-Funktion von Flask
- **URLs dynamisch generieren** mit `url_for`, um anpassungsfähige URLs in Ihrer Flask-Anwendung zu erstellen
- **Verschiedene HTTP-Anfragetypen verwalten**, um flexible Routen zu entwerfen, die auf verschiedene HTTP-Anfragetypen reagieren
- **CRUD-Operationen implementieren** für die Datenverwaltung in einer Flask-App

Hinweis: Jeder Abschnitt enthält relevante Codebeispiele und Erklärungen, um Ihr Verständnis der wesentlichen Flask-Funktionen zu vertiefen.

Zugriff auf Formulardaten mit flask.request.form

Sie können `flask.request.form` verwenden, um auf Formulardaten zuzugreifen, die ein Benutzer über eine `POST`-Anfrage gesendet hat. Diese Funktion kann beispielsweise verwendet werden, wenn Sie ein Anmeldeformular mit Benutzername und Passwortfeldern haben.

In Ihrer HTML-Datei könnten Sie ein Formular wie dieses haben:

```
<form method="POST" action="/login">
    <input type="text" name="username">
    <input type="password" name="password">
    <input type="submit" value="Submit">
</form>
```

Der Python-Code zum Zugriff auf den Benutzernamen und das Passwort wird wie folgt aussehen:

```
from flask import request
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    # process login here
```

Weiterleitung zu einer URL mit flask.redirect

Flask bietet eine Funktion namens flask.redirect, um Benutzer zu anderen Webseiten (oder Endpunkten) zu leiten. Die Funktion flask.redirect kann in mehreren Szenarien nützlich sein. Zum Beispiel können Sie die Funktion flask.redirect verwenden, um einen Benutzer zu einer **Anmeldeseite** weiterzuleiten, wenn er versucht, auf eine eingeschränkte **Admin**-Seite zuzugreifen.

Python-Code:

```
from flask import redirect
@app.route('/admin')
def admin():
    return redirect('/login')
```

Dynamische URLs mit flask.url_for generieren

Die `flask.url_for`-Funktion generiert dynamisch URLs für einen bestimmten Endpunkt. Das dynamische Generieren von URLs kann besonders nützlich sein, wenn die URL für eine Route geändert wird. Die `flask.url_for`-Funktion aktualisiert die URL automatisch in Ihren Vorlagen oder Ihrem Code, wodurch der manuelle Aufwand minimiert wird. Betrachten Sie zum Beispiel das Szenario, in dem ein Benutzer versucht, auf die **Admin**-Seite zuzugreifen und zur **Login**-Seite umgeleitet werden muss. In diesem Szenario wird `url_for('login')` die URL für die **Login**-Seite aus den bestehenden Routen abrufen.

Python-Code:

```
from flask import url_for
@app.route('/admin')
def admin():
    return redirect(url_for('login'))
@app.route('/login')
def login():
    return "<Login Page>"
```

Umgang mit verschiedenen HTTP-Anfragetypen

Flask ermöglicht es Ihnen, Routen zu definieren, um verschiedene Arten von HTTP-Anfragen zu verwalten. Sie können die Route sowohl mit den Zugriffsarten GET als auch POST definieren und in der Funktionsbeschreibung die Anwendungsfälle für beide Methoden festlegen.

Python-Code:

```
@app.route('/data', methods=['GET', 'POST'])
def data():
    if request.method == 'POST':
        # process POST request
    if request.method == 'GET':
        # process GET request
```

In der HTML-Datei fügen Sie ein Formular hinzu, das sowohl `GET`- als auch `POST`-Anfragen ermöglicht:

```
<!-- For POST -->
<form method="POST" action="/data">
    <!-- Your input fields here -->
    <input type="submit" value="Submit">
</form>
<!-- For GET -->
<a href="/data">Fetch data</a>
```

Im letzten Beispiel akzeptiert die `/data`-Route sowohl `GET`- als auch `POST`-Anfragen. Der Typ der Anfrage kann mit `flask.request.method` überprüft werden.

CRUD-Operationen

Die CRUD-Operationen repräsentieren die vier grundlegenden Funktionen, die Sie benötigen, um mit einem persistenten Speicher, wie einer Datenbank, zu interagieren. In der Webentwicklung entsprechen die CRUD-Operationen häufig den HTTP-Methoden.

Erstelloperation

Die Erstellung von Daten beinhaltet oft die Präsentation eines Formulars für den Benutzer, um die Informationen zu sammeln, die Sie als neuen Datensatz in der Datenbank speichern möchten. In Flask wird auf diese Daten über `flask.request.form` zugegriffen.

HTML-Formular zur Erstellung von Daten:

```
<form method="POST" action="/create">
    <input type="text" name="name">
    <input type="submit" value="Create">
</form>
```

```
# This is a sample Python code
def greet(name):
    return f"Hello, {name}!"
print(greet("World"))
```

```
@app.route('/create', methods=['GET', 'POST'])
def create():
    if request.method == 'POST':
        # Access form data
        name = request.form['name']
        # Create a new record with the name
        record = create_new_record(name)  # Assuming you have this function defined
        # Redirect user to the new record
        return redirect(url_for('read', id=record.id))
    # Render the form for GET request
    return render_template('create.html')
```

Lesevorgang

Das Lesen von Daten umfasst den Zugriff auf die Daten und deren Präsentation für den Benutzer. Um auf bestimmte Einträge zuzugreifen, muss die Anfrage mit spezifischen IDs erfolgen. Daher müssen Sie die ID als Argument an die Funktion übergeben. Das folgende Beispiel zeigt, dass die ID über die Route abgerufen werden kann.

Python-Code:

```
@app.route('/read/<int:id>', methods=['GET'])
def read(id):
    # Get the record by id
    record = get_record(id)  # Assuming you have this function defined
    # Render a template with the record
    return render_template('read.html', record=record)
```

Aktualisierungsoperation

Das Aktualisieren von Daten erfordert den Zugriff auf spezifische Einträge, ähnlich der **Lese**-Operation, und beinhaltet das Bereitstellen neuer Daten für den betreffenden Parameter, wie bei der **Erstellen**-Operation. Daher sollte der Pfad die ID zugreifen und beide Zugriffsarten enthalten.

Beispiel-HTML-Formular zum Aktualisieren von Daten:

```
<form method="POST" action="/update/{{record.id}}">
    <input type="text" name="name" value="{{record.name}}">
    <input type="submit" value="Update">
</form>
```

```
# This is a simple Python script
def greet(name):
    return f"Hello, {name}!"
print(greet("World"))
```

```
@app.route('/update/<int:id>', methods=['GET', 'POST'])
def update(id):
    if request.method == 'POST':
        # Access form data
        name = request.form['name']
        # Update the record with the new name
        update_record(id, name)  # Assuming you have this function defined
        # Redirect user to the updated record
        return redirect(url_for('read', id=id))

    # Render the form for GET request with current data
    record = get_record(id)  # Assuming you have this function defined
    return render_template('update.html', record=record)
```

Löschvorgang

Das Löschen von Daten beinhaltet das Entfernen eines Datensatzes basierend auf seiner ID. Der Löschvorgang erfordert typischerweise, dass die ID übergeben wird, wie sie von der HTML-Seite gemeldet wird, in Form eines Arguments an die Funktion.

Beispiel für ein HTML-Formular zum Löschen von Daten:

```
<form method="POST" action="/delete/{{record.id}}">
    <input type="submit" value="Delete">
</form>
```

```
def greet(name):
    print(f"Hello, {name}!")
greet("World")
```

```
@app.route('/delete/<int:id>', methods=['POST'])
def delete(id):
    # Delete the record
    delete_record(id)  # Assuming you have this function defined
    # Redirect user to the homepage
    return redirect(url_for('home'))
```

Autor(en)

Vicky Kuo

Zusätzlicher Mitwirkender

Abhishek Gagneja

Änderungsprotokoll

Datum	Version	Geändert von	Änderungsbeschreibung
2023-07-24	2.0	Steve Hord	QA-Durchgang mit Bearbeitungen
2023-07-20	1.0	Vicky Kuo	Erster Entwurf erstellt