Flask’ta Ek Özelliklerle CRUD İşlemleri

Gerekli tahmini süre: 15 dakika

Genel Bakış

Oluşturma, Okuma, Güncelleme ve **Silme** (CRUD), bir veritabanına sahip herhangi bir uygulamanın gerçekleştirmesi gereken temel işlevlerdir. CRUD işlemlerini etkili bir şekilde uygulamak için, `GET` ve `POST` istekleri gibi farklı HTTP yöntemlerini yönetmeniz gerekecek. `GET` isteği genellikle verileri almak veya okumak için kullanılır ve sıklıkla bir formu görüntülemek için tercih edilir. `POST` isteği ise verileri göndermek, oluşturmak veya güncellemek için yaygın olarak kullanılır. Bir POST isteği örneği, form gönderimidir.

Bu okuma, sizi Flask’ın ek özellikleriyle tanıştırmayı amaçlamakta olup, CRUD işlevlerine odaklanmaktadır. Bu işlevlerin nasıl birbiriyle ilişkili olduğunu ve farklı HTML dosyalarını, dinamik yolları ve çeşitli HTTP yöntemlerini nasıl kullandıklarını da öğreneceksiniz.

Hedefler

Bu okumada şunları yapacaksınız:

- **Kullanıcı girdilerini yakalamak için form verilerine erişin** `flask.request.form` ile `POST` isteklerinde
- Kullanıcı navigasyonunu kontrol edin: Flask’ın `redirect` fonksiyonunu kullanarak
- **Dinamik URL’ler oluşturun** `url_for` ile Flask uygulamanızda uyumlu URL’ler oluşturmak için
- **Farklı HTTP istek türlerini yönetin** çeşitli HTTP istek türlerine yanıt veren esnek rotalar tasarlamak için
- **CRUD işlemlerini uygulayın** bir Flask uygulamasında veri yönetimi için

Not: Her bölüm, Flask’ın önemli özelliklerini anlamanızı artırmak için ilgili kod örnekleri ve açıklamalar içermektedir.

Flask.request.form ile Form Verilerine Erişim

Kullanıcının `POST` isteği ile gönderdiği form verilerine erişmek için `flask.request.form` kullanabilirsiniz. Örneğin, kullanıcı adı ve şifre alanlarına sahip bir giriş formunuz varsa bu özelliği kullanabilirsiniz.

HTML dosyanızda şöyle bir formunuz olabilir:

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

Python kodu kullanıcı adı ve şifreye erişmek için aşağıdaki gibi olacaktır:

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

flask.redirect ile Bir URL'ye Yönlendirme

Flask, kullanıcıları farklı web sayfalarına (veya uç noktalarına) yönlendirmek için flask.redirect adlı bir fonksiyon sağlar. flask.redirect fonksiyonu, çeşitli senaryolarda faydalı olabilir. Örneğin, bir kullanıcı kısıtlı bir **admin** sayfasına erişmeye çalıştığında, onu **giriş** sayfasına yönlendirmek için flask.redirect fonksiyonunu kullanabilirsiniz.

Python kodu:

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

Flask.url_for ile Dinamik URL'ler Üretme

`flask.url_for` fonksiyonu, belirli bir uç nokta için dinamik olarak URL'ler üretir. Dinamik olarak URL'ler üretmek, bir rota için URL değiştiğinde özellikle faydalı olabilir. `flask.url_for` fonksiyonu, URL'yi şablonlarınızda veya kodunuzda otomatik olarak güncelleyerek manuel çalışmayı en aza indirir. Örneğin, bir kullanıcının **admin** sayfasına erişmeye çalıştığı ve **giriş** sayfasına yönlendirilmesi gerektiği senaryoyu düşünün. Bu durumda, `url_for('login')` mevcut rotalardan **giriş** sayfasının URL'sini alır.

Python kodu:

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

```
    return "<Login Page>"
```

Farklı HTTP istek türlerini yönetme

Flask, farklı türdeki HTTP isteklerini yönetmek için rotalar tanımlamanıza olanak tanır. Hem GET hem de POST erişim yöntemleri ile rotayı tanımlayabilir ve fonksiyon açıklamasında her iki yöntem için kullanım durumlarını belirleyebilirsiniz.

Python kodu:

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

HTML dosyasına hem `GET` hem de `POST` isteklerine izin veren bir form ekleyeceksiniz:

```
<!-- 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>
```

Son örnekte, `/data` rotası hem `GET` hem de `POST` isteklerini kabul eder. İsteğin türü `flask.request.method` kullanılarak kontrol edilebilir.

CRUD işlemleri

CRUD işlemleri, herhangi bir kalıcı depolama ile etkileşimde bulunmak için gereken dört temel işlevi temsil eder; bu, bir veritabanı gibi. Web geliştirmede, CRUD işlemleri genellikle HTTP yöntemleriyle karşılık gelir.

Oluşturma işlemi

Veri oluşturma genellikle kullanıcıya bir form sunmayı içerir; bu form, veritabanında yeni bir kayıt olarak saklamak istediğiniz bilgileri toplamak için kullanılır. Flask'ta bu verilere `flask.request.form` kullanılarak erişilir.

Veri oluşturmak için HTML formu:

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

Python kodu:

```
@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')
```

Okuma işlemi

Veri okumak, verilere erişimi ve bunları kullanıcıya sunmayı içerir. Belirli girişlere erişmek için, isteğin belirli kimliklerle birlikte gitmesi gerekir. Bu nedenle, kimliği fonksiyona bir argüman olarak geçirmeniz gerekecektir. Aşağıdaki örnek, kimliğin rotadan erişilebileceğini göstermektedir.

Python kodu:

```
@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)
```

Güncelleme işlemi

Verileri güncellemek, **Okuma** işlemi gibi belirli kayıtlara erişim sürecini gerektirir ve **Oluşturma** işlemi gibi ilgili parametreye yeni veriler vermeyi içerir. Bu nedenle, yolun ID'ye erişmesi ve her iki erişim yöntemini de içermesi gerekir.

Verileri güncellemek için örnek HTML formu:

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

```
# Python kodu:
print("Merhaba, dünya!")
```

```
@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)
```

Silme işlemi

Veri silme, bir kaydı ID'sine göre kaldırmayı içerir. Silme işlemi genellikle, HTML sayfası tarafından bildirilen ID'nin, işlevin bir argümanı olarak geçirilmesini gerektirir.

Veri silmek için örnek HTML formu:

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

```
# Python kodu:
print("Merhaba, Dünya!")
```

```
@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'))
```

Yazar(lar)

Vicky Kuo

Ek Katkı Sağlayıcı

Abhishek Gagneja

Değişiklik Günlüğü

Tarih	Versiyon	Değiştiren	Değişiklik Açıklaması
2023-07-24	2.0	Steve Hord	Düzenlemeler ile QA geçişi
2023-07-20	1.0	Vicky Kuo	İlk versiyon oluşturuldu