Uygulamalı Laboratuvar: Flask'ta Ek Özellikler Kullanarak CRUD Uygulama Tasarımı

Tahmini süre: **60** dakika

Genel Bakış

CRUD, Yaratma, Okuma, Güncelleme, Silme anlamına gelir ve bir veritabanına dayanan herhangi bir uygulamanın sahip olması gereken temel işlevlerdir. Bu özelliklerin geliştirilmesi, yolları ve istekleri yönetme konusunda ek bilgi gerektirir. Ayrıca, farklı istekleri karşılamak için birden fazla uç nokta HTML arayüzüne ihtiyacınız vardır. Bu laboratuvarın amacı, Flask kullanımında size ek pratik yapma imkanı sunmak ve tamamen işlevsel, CRUD işlemi yapabilen bir web uygulaması geliştirmektir.

Bu laboratuvar için bir finansal işlem kayıt sistemi geliştireceksiniz. Sistem, **Yeni** bir giriş **Oluşturma**, mevcut girişleri **Okuma**, mevcut girişleri **Güncelleme** ve mevcut girişleri **Silme** yeteneğine sahip olmalıdır.

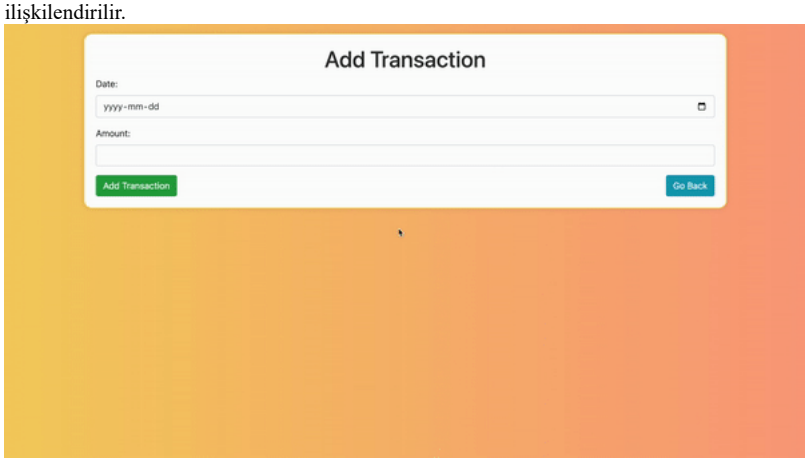
Hedefler

Bu laboratuvarı tamamladıktan sonra, şunları yapabileceksiniz:

- İşlem girişi eklemek için “Oluştur” işlemini uygulamak
- İşlem girişleri listesini erişmek için “Oku” işlemini uygulamak
- Verilen bir işlem girişinin detaylarını güncellemek için “Güncelle” işlemini uygulamak
- Bir işlem girişini silmek için “Sil” işlemini uygulamak.

Uygulamayı geliştirmeyi tamamladıktan sonra, animasyonda gösterildiği gibi çalışacaktır.

Uygulamanın üç farklı web sayfası vardır. İlk sayfa, kaydedilen tüm işlemleri gösterir. Bu sayfaya İşlem Kayıtları denir ve sistemde oluşturulan tüm işlem girişlerini görüntüler. Bu sayfa ayrıca mevcut girişleri Düzenleme ve Silme seçeneği sunar. Bir giriş ekleme seçeneği de bu sayfada mevcuttur. İkinci sayfa, önceki sayfada giriş eklemek için kullanıcı tarafından seçilen Ekle İşlem sayfasıdır. Kullanıcı, yeni giriş için Tarih ve Miktar değerlerini ekler. Üçüncü sayfa, düzenleme giriş seçeneğine tıkladığında kullanıcı tarafından gidilen Düzenle İşlem sayfasıdır. Bu sayfada da tarih ve miktar giriş olarak kabul edilir; ancak bu girişler, düzenlenen ID ile ilişkilendirilir.



Not: Bu platform kalıcı değildir. Kodunuzun bir kopyasını yerel makinelerinizde saklamanız ve zaman zaman değişiklikleri kaydetmeniz önerilir. Laboratuvara yeniden döndüğünüzde, bu laboratuvar ortamında dosyaları, makinelerinizden kaydedilen kopyalarla yeniden oluşturmanız gerekecektir.

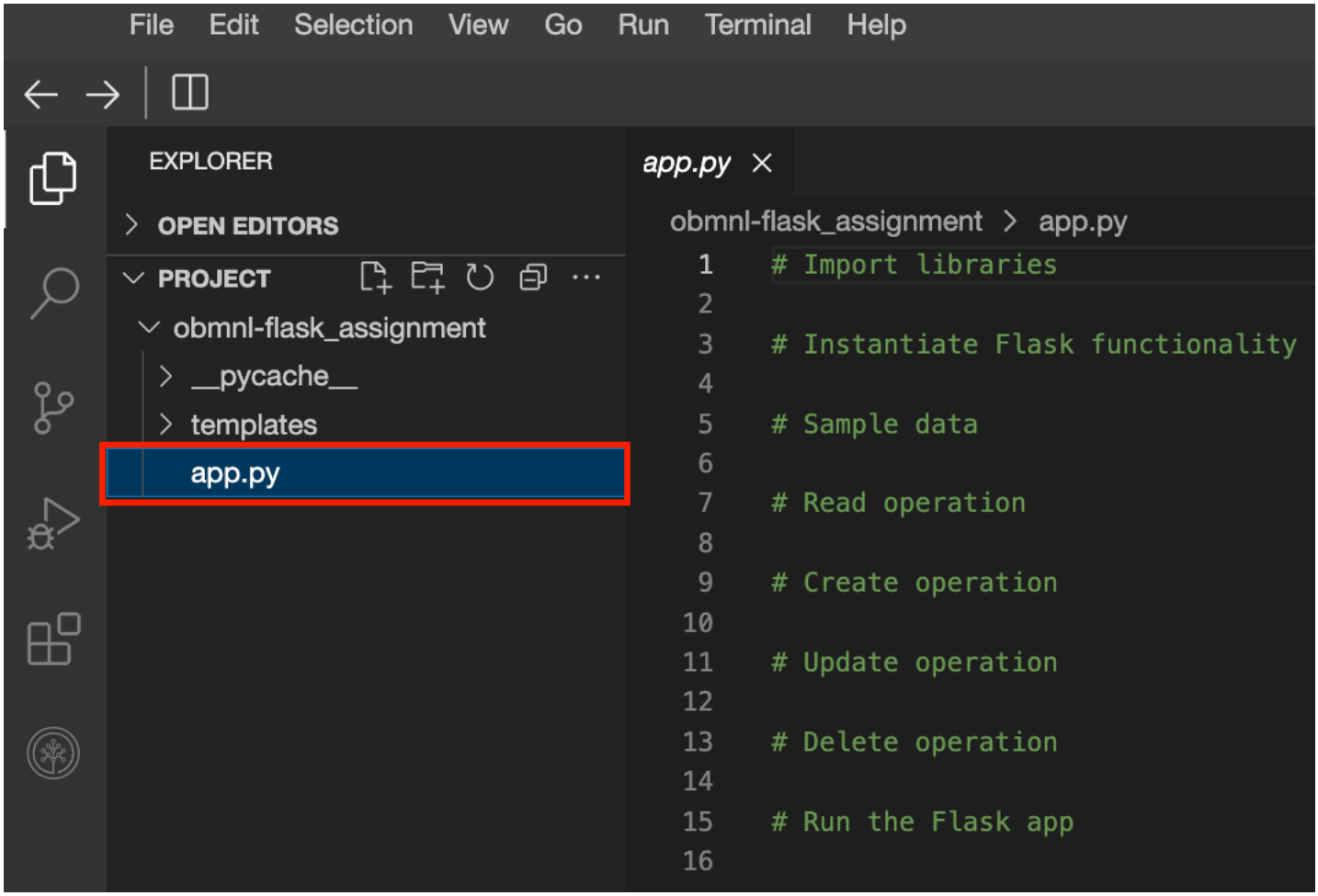
Hadi başlayalım!

Proje Deposu Klonlama

Bu laboratuvar, sizin için önceden oluşturulmuş birden fazla HTML arayüz dosyası gerektirir. Terminal kabuğunda aşağıdaki komutu kullanarak klasör yapısını IDE arayüzüne klonlamanız gerekecek.

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

Komut başarıyla çalıştırıldığında, Proje sekmesinde resimde gösterilen klasör yapısı olmalıdır. Kök klasör, `obmnl-flask_assignment` `templates` klasörüne ve `app.py` dosyasına sahip olmalıdır. `templates` klasörü, gerekli tüm HTML dosyalarını, `edit.html`, `form.html` ve `transactions.html` içermektedir. Bu laboratuvar boyunca, `app.py` dosyasında gerekli fonksiyonları uygulayacaksınız.



İlk Kurulum

`app.py` dosyasında, Flask'tan gerekli modülleri içe aktarmanız ve Flask uygulamasını başlatmanız gerekiyor. Bu laboratuvar için, ***flask*** kütüphanesinden aşağıdaki fonksiyonları içe aktarmanız gerekecek.

- Flask - uygulamayı başlatmak için
- request - `GET` ve `POST` isteklerini işlemek için
- url_for - belirli bir fonksiyonun dekoratörünü kullanarak url'ye erişmek için
- redirect - erişim isteklerini gereksinimlere göre yönlendirmek için
- render_template - html sayfasını oluşturmak için

Fonksiyonları içe aktardıktan sonra, uygulamayı `app` adlı bir değişkene atayın.

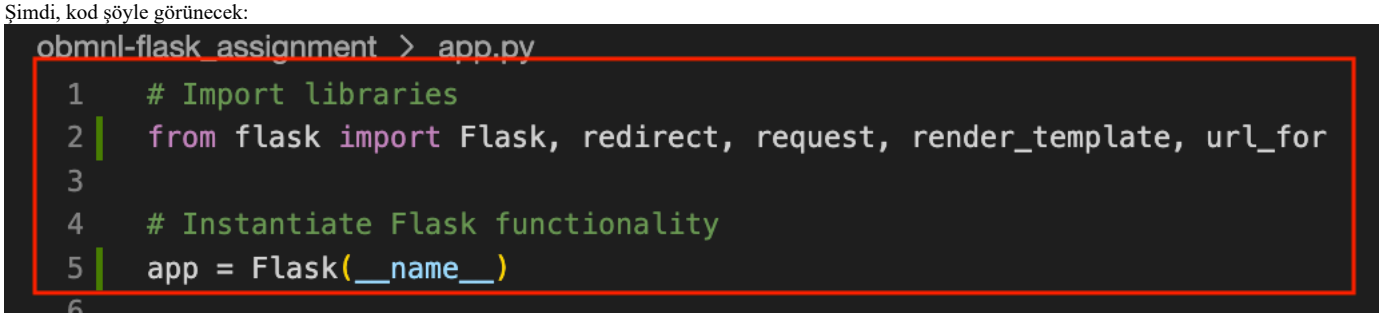
▼ *İpucu için buraya tıklayın*

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

▼ *Çözüm için buraya tıklayın*

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

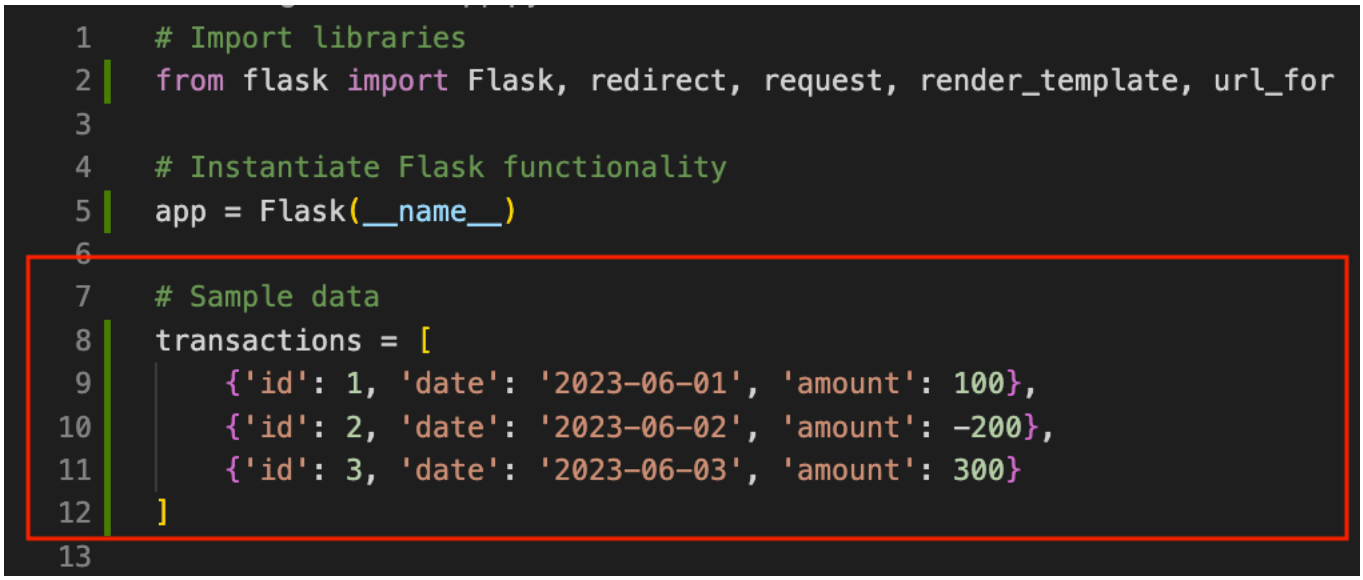
Şimdi, kod şöyle görünecek:



Sonraki adımda, test amaçlı örnek işlemler listesini oluşturalım. İşlemlerin, ilk kez çalıştırıldığında arayüzde zaten var olduğunu varsayabilirsiniz. Lütfen bu adımın tamamen isteğe bağlı olduğunu ve bu laboratuvarın geliştireceğiniz işlevselliği etkilemediğini unutmayın. Aşağıda gösterildiği gibi `app.py` dosyasına kod parçasını ekleyin.

```
# 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}
]
```

▼ *Çözüm için buraya tıklayın*



Fonksiyonları geliştireceğiniz sıra şu şekildedir:
1. Okuma
2. Oluşturma
3. Güncelleme
4. Silme

Okuma fonksiyonunu diğerlerinden önce uygulamanın nedeni, her yeni işlem oluşturulduğunda, güncellendiğinde veya silindiğinde tüm işlemlerin bulunduğu sayfaya yönlendirme yapabilmektir. Bu nedenle, mevcut işlemleri okumak için fonksiyonun diğerleri uygulanmadan önce var olması gerekmektedir.

Okuma İşlemi

Okuma işlemini uygulamak için, tüm işlemlerin bir listesini görüntüleyen bir rota oluşturmanız gerekir. Bu rota, `app.py` içinde verileri almak ve görüntülemek için kullanılan `GET` isteklerini yönetecektir.

Okuma işlemini uygulamak için ana adımlar şunlardır:

1. `get_transactions` adında bir fonksiyon oluşturun; bu fonksiyon, `render_template` kullanarak `transactions.html` adlı bir HTML şablonu döndürmelidir. Bu fonksiyon, işlemleri görüntülenmek üzere şablona iletmelidir.

2. Flask `@app.route` dekoratörünü kullanarak bu fonksiyonu kök (`/`) URL'sine eşleyin. Bu, bir kullanıcı uygulamanızın temel URL'sini ziyaret ettiğinde Flask'ın `get_transactions` fonksiyonunu çalıştıracağı ve sonucunu döndüreceği anlamına gelir.

▼ *İpucu için buraya tıklayın*
Bu fonksiyon, önceki laboratuvarlarda uygulanan temel bir render_template fonksiyonudur.
▼ *Çözüm için buraya tıklayın*

```
# Read operation: List all transactions
@app.route("/")
```

```
def get_transactions():
    return render_template("transactions.html", transactions=transactions)
```

Şimdi, kod şöyle görünecek:

```python
# 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)
```

Oluşturma İşlemi

Oluşturma işlemi için, kullanıcıların yeni işlemler eklemesine olanak tanıyan bir rota uygulayacaksınız. Bu, hem `GET` hem de `POST` HTTP isteklerini ele almayı içerecektir - `GET`, kullanıcıya formu göstermek için ve `POST`, kullanıcının gönderdiği form verilerini işlemek için.

İşte Oluşturma işlemini uygulamak için adımların listesi.

1. `add_transaction` adında bir fonksiyon oluşturun.
2. Bu fonksiyon için dekoratör olarak `add` kullanın. Hem `GET` hem de `POST` yöntemlerini mümkün olan yöntemler olarak geçirdiğinizden emin olun.
3. Eğer istek yöntemi `GET` ise, bir HTML formunu `form.html` adlı bir şablon kullanarak göstermek için `render_template` fonksiyonunu kullanın. Bu form, kullanıcıların yeni bir işlem için veri girmesine olanak tanıyacaktır.
4. Eğer istek yöntemi `POST` ise, form verilerini çıkarmak için `request.form` kullanın, yeni bir işlem oluşturun, bunu işlemler listesine ekleyin ve ardından kullanıcıyı işlemler listesinin olduğu sayfaya geri göndermek için `redirect` ve `url_for` kullanın.
5. Yeni işlem, okuma fonksiyonuna aşağıdaki formatta iletilir.

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

Burada, request.form fonksiyonu formda yapılan girişten alınan bilgileri ayrıştırır.

▼ *İpucu için buraya tıklayın*
add_transaction fonksiyonu içeriği aşağıdaki uygulamaları gerektirir.

`POST` yöntemi için, yukarıda gösterildiği gibi yeni bir işlem oluşturun, mevcut işlem listesinin sonuna ekleyin ve **Okuma** işlemi için URL'ye yönlendirin.

`GET` yöntemi için, arayüzden bilgi kabul eden form.html sayfasını render edin.

▼ *Çözüm için buraya tıklayın*

```
# 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")
```

Şimdi, kod şu şekilde görünecek:

```
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")
```

Not: `if` durumunun dışındaki ifadeler, varsayılan olarak `else` durumudur. `if` durumundaki ifadeler bir return ifadesi ile sona erer; bu nedenle, her seferinde yalnızca iki durumdan biri çalışacaktır.

::page{title="Güncelleme İşlemi"}

Güncelleme işlemi için, kullanıcıların mevcut işlemleri güncelleyebilmeleri için bir rota oluşturmanız gerekiyor. Yine `GET` ve `POST` HTTP isteklerini ele alacaksınız - `GET`, mevcut işlem verilerini bir formda görüntülemek için, `POST` ise kullanıcının gönderdiği güncellenmiş verileri işlemek için.

Güncelleme işlemini uygulamak için aşağıdaki adımları tamamlayın.

1. `edit_transaction` adında, hem `GET` hem de `POST` isteklerini işleyen bir fonksiyon oluşturun. Bu fonksiyon bir `transaction_id` parametresi almalıdır.
2. Fonksiyonu `@app.route` ile süsleyin ve rota dizesini `/edit/<int:transaction_id>` olarak kullanın. URL'deki `<int:transaction_id>` kısmı, herhangi bir tam sayı için bir yer tutucudur. Flask, bu tam sayıyı fonksiyonunuza `transaction_id` argümanı olarak iletecektir.

3. Eğer istek yöntemi `GET` ise, `transaction_id` ile eşleşen ID'ye sahip işlemi bulun ve `render_template` kullanarak, `edit.html` adında bir şablonla mevcut işlem verileriyle önceden doldurulmuş bir formu görüntüleyin.

4. Eğer istek yöntemi `POST` ise, güncellenmiş verileri almak için request.form'ı kullanın, `transaction_id` ile eşleşen ID'ye sahip işlemi bulun ve verilerini değiştirin, ardından kullanıcıyı işlemler listesinin bulunduğu sayfaya yönlendirin.

▼ *Çözüm için buraya tıklayın*

```
# 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
```

Şimdi, kod şöyle görünecek:

```
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
```

Not: Aynı sonuca ulaşmanın birden fazla yolu olabilir. Lütfen yukarıda verilen çözümü yalnızca bir referans olarak kullanın.

::page{title="Silme İşlemi"}

Son olarak, kullanıcıların mevcut işlemleri silmesine olanak tanıyan bir rota uygulamanız gerekiyor.

Silme işlemini uygulamak için aşağıdaki adımları tamamlayın.

1. `transaction_id` adında bir parametre alan `delete_transaction` adlı bir fonksiyon oluşturun.
2. Fonksiyonu `@app.route` ile süsleyin ve rota dizesi olarak `/delete/<int:transaction_id>` kullanın. URL'deki `<int:transaction_id>` kısmı, herhangi bir tam sayı için bir yer tutucudur. Flask, bu tam sayıyı fonksiyonunuza `transaction_id` argümanı olarak iletecektir.
3. Fonksiyon gövdesinde, `transaction_id` ile eşleşen ID'ye sahip işlemi bulun ve işlemler listesinden çıkarın, ardından kullanıcıyı işlemler listesinin olduğu sayfaya `redirect` edin.

▼ *Çözüm için buraya tıklayın*

```
# 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"))
```

Şimdi, kod şöyle görünecek:

```
61  # Delete operation: Delete a transaction
62  @app.route("/delete/<int:transaction_id>")
63  def delete_transaction(transaction_id):
64      # Find the transaction with the matching ID and remove it from the lis
65      for transaction in transactions:
66          if transaction['id'] == transaction_id:
67              transactions.remove(transaction)
68              break
69
70      # Redirect to the transactions list page
71      return redirect(url_for("get_transactions"))
72
```

::page{title="Tamamlama Adımları ve Uygulamayı Çalıştırma"}

Mevcut script'in ana program olup olmadığını (yani başka bir script'ten içe aktarılmadığını) `if __name__ == "__main__":` koşuluyla kontrol edin.

Koşul doğruysa, `app.run(debug=True)` çağrısını yaparak Flask geliştirme sunucusunu hata ayıklama modu etkin şekilde başlatın. Bu, bir şeyler ters gittiğinde tarayıcınızda ayrıntılı hata mesajlarını görmenizi sağlar.

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

Şimdi kod şöyle görünecek:

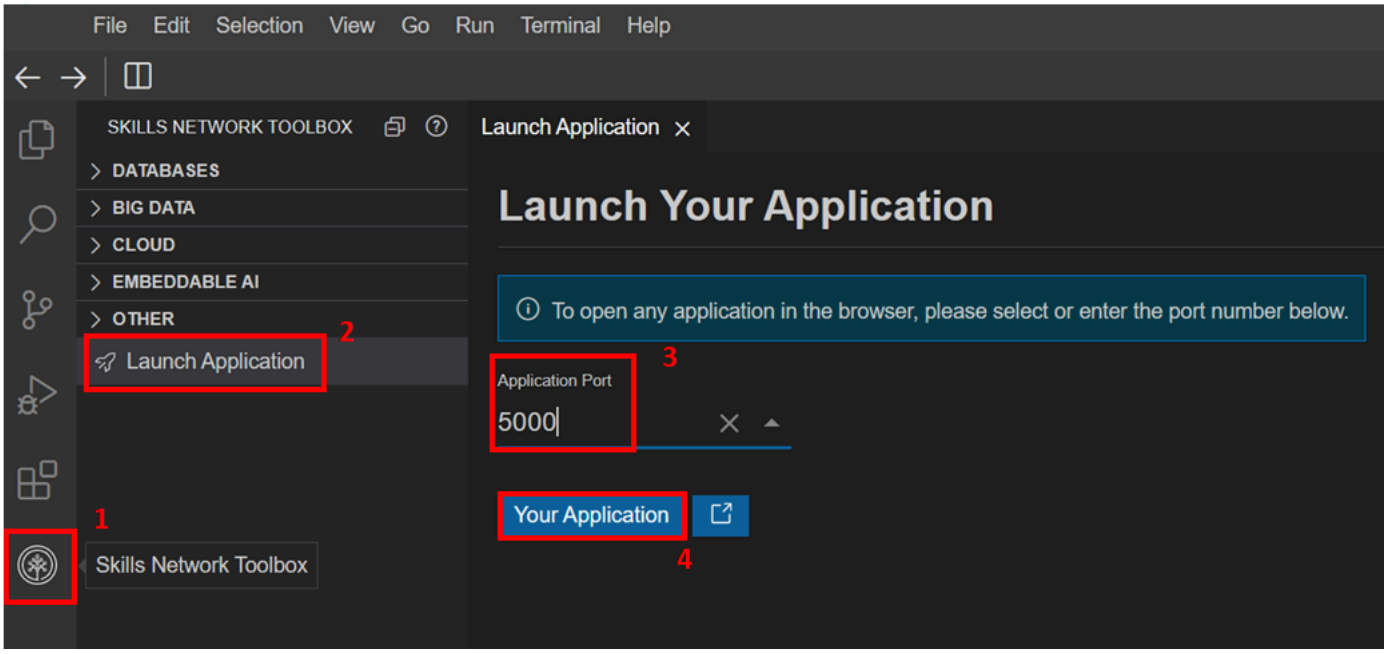
```
# Delete operation: Delete a transaction
@app.route("/delete/<int:transaction_id>")
def delete_transaction(transaction_id):
    # Find the transaction with the matching ID and remove it from the lis
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            transactions.remove(transaction)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

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


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

Kod artık tamamlandı. Terminal kabuğundan aşağıdaki komutu kullanarak `app.py` dosyasını çalıştırın:

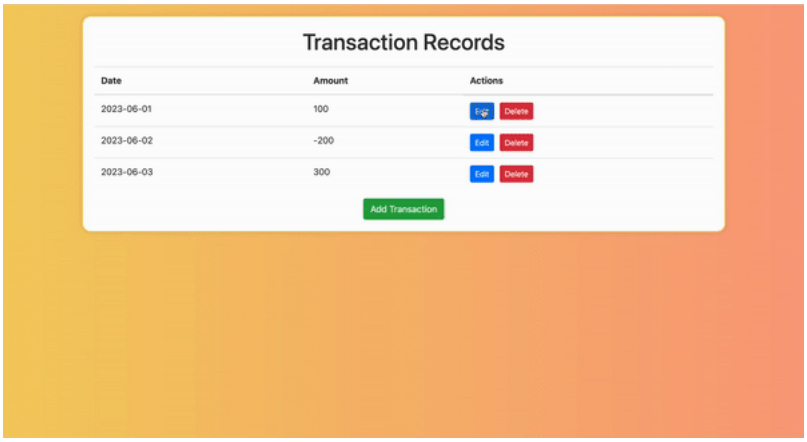
```
python3.11 app.py
```

Varsayılan olarak, Flask uygulamayı LocalHost:5000 üzerinde başlatır. Görselde gösterildiği gibi,

1. Uygulamayı başlatmak için Beceriler Ağı Kütüphanesi'ne gidin, `Uygulamayı Başlat` seçeneğine tıklayın.
2. Port numarasına `5000` girin ve uygulama penceresini başlatın.



Son uygulama bu şekilde görünmektedir.



::page{title="Laboratuvar Yardımı"}

Herhangi bir hata ile karşılaşırsanız, `app.py` için son kod burada hazır bir referans olarak paylaşılmaktadır. Lütfen bunun sadece son çare olarak kullanılmasını ve bu laboratuvarın amaçladığı öğrenimi elde etmenizi sağlamak için kullanılmasını unutmayın.

▼ *app.py için son kod*

```
# Flask'tan gerekli kütüphaneleri içe aktar
from flask import Flask, redirect, request, render_template, url_for
# Flask uygulamasını oluştur
app = Flask(__name__)
# İşlemleri temsil eden örnek veri
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}
]
# Okuma işlemi: Tüm işlemleri listelemek için rota
@app.route("/")
def get_transactions():
    # İşlemler listesi şablonunu render et ve işlemler verisini geçir
    return render_template("transactions.html", transactions=transactions)
# Oluşturma işlemi: İşlem ekleme formunu görüntülemek ve işlemek için rota
@app.route("/add", methods=["GET", "POST"])
def add_transaction():
    if request.method == 'POST':
        # Yeni bir işlem nesnesi oluşturmak için form verilerini çıkar
        transaction = {
            'id': len(transactions) + 1,         # İşlemler listesinin mevcut uzunluğuna göre yeni bir ID oluştur
            'date': request.form['date'],        # Formdan 'date' alanının değerini al
            'amount': float(request.form['amount']) # Formdan 'amount' alanının değerini al ve float'a dönüştür
        }
        # Yeni işlemi işlemler listesine ekle
        transactions.append(transaction)
        # Yeni işlemi ekledikten sonra işlemler listesi sayfasına yönlendir
        return redirect(url_for("get_transactions"))
    # İstek yöntemi GET ise işlem ekleme formunu görüntülemek için form şablonunu render et
    return render_template("form.html")
# Güncelleme işlemi: İşlem düzenleme formunu görüntülemek ve işlemek için rota
@app.route("/edit/<int:transaction_id>", methods=["GET", "POST"])
def edit_transaction(transaction_id):
    if request.method == 'POST':
        # Form alanlarından güncellenmiş değerleri çıkar
        date = request.form['date']
        amount = float(request.form['amount'])
        # Eşleşen ID'ye sahip işlemi bul ve değerlerini güncelle
        for transaction in transactions:
            if transaction['id'] == transaction_id:
                transaction['date'] = date       # İşlemin 'date' alanını güncelle
                transaction['amount'] = amount   # İşlemin 'amount' alanını güncelle
                break                            # İşlem bulunduğunda ve güncellendiğinde döngüden çık
        # İşlemi güncelledikten sonra işlemler listesi sayfasına yönlendir
        return redirect(url_for("get_transactions"))
    # İstek yöntemi GET ise eşleşen ID'ye sahip işlemi bul ve düzenleme formunu render et
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            # Düzenleme formu şablonunu render et ve düzenlenecek işlemi geçir
            return render_template("edit.html", transaction=transaction)
# Silme işlemi: Bir işlemi silmek için rota
@app.route("/delete/<int:transaction_id>")
def delete_transaction(transaction_id):
    # Eşleşen ID'ye sahip işlemi bul ve listeden çıkar
    for transaction in transactions:
        if transaction['id'] == transaction_id:
            transactions.remove(transaction)  # İşlemi işlemler listesinden çıkar
            break                             # İşlem bulunduğunda ve çıkarıldığında döngüden çık
    # İşlemi sildikten sonra işlemler listesi sayfasına yönlendir
    return redirect(url_for("get_transactions"))
# Flask uygulamasını çalıştır
if __name__ == "__main__":
    app.run(debug=True)
```

::page{title="Arayüzü Test Etme"}

Uygulamanız hazır olduğunda, başlatılan uygulama üzerinde CRUD işlemlerini deneyin. Uygulamayı test etmek için olası görevler şunlar olabilir:

1. Formu açmak ve yeni bir işlem eklemek için "Ekle" butonuna tıklayın.
2. Herhangi bir işlem için bilgileri (tarih ve tutar) güncellemek üzere "Düzenle" butonuna tıklayın.
3. Listeden silmek için herhangi bir işlem için "Sil" butonuna tıklayın.
4. İşlemlerin doğru bir şekilde görüntülendiğini doğrulayın.

::page{title="Pratik Alıştırmalar"}

Aşağıda, ilgili öğrenenler için bazı pratik alıştırmalar bulunmaktadır. Bu alıştırmaların çözümlerini sağlamıyoruz, böylece öğrenenlerin kendi başlarına denemeleri teşvik edilsin. Çözüm hakkında diğer ilgili öğrenenlerle görüşlerinizi paylaşmak için kurs tartışma forumunu kullanmaktan çekinmeyin.

Alıştırma 1: İşlemleri Ara

Bu alıştırmada, kullanıcıların belirli bir miktar aralığında işlemleri aramasını sağlayan yeni bir özellik ekleyeceksiniz. `app.py` dosyasında `GET` ve `POST` isteklerini yöneten `/search` adında yeni bir rota oluşturacaksınız.

Talimatlar:

1. `search_transactions` adında yeni bir fonksiyon oluşturun ve bunu `/search` URL'sine eşlemek için `@app.route` dekoratörünü kullanın.
2. Fonksiyonun içinde, istek yönteminin `POST` olup olmadığını kontrol edin. Eğer öyleyse, kullanıcının gönderdiği form verilerinden minimum ve maksimum miktar değerlerini alın. Bu değerleri ondalık sayılara dönüştürün.
3. Kullanıcının belirttiği miktar aralığına göre işlemler listesini filtreleyin. Sadece belirtilen aralıkta kalan işlemleri içeren `filtered_transactions` adında yeni bir liste oluşturun. Bunun için bir liste anlama yöntemi kullanabilirsiniz.
4. `filtered_transactions` listesini `render_template` fonksiyonu aracılığıyla `transactions.html` şablonuna geçirin. Bu şablonda, işlemleri mevcut `transactions.html` şablonuna benzer şekilde görüntüleyin.
5. Eğer istek yöntemi `GET` ise, `search.html` adında yeni bir şablon oluşturun. Bu şablon, kullanıcıların arama için minimum ve maksimum miktar değerlerini girmesine olanak tanıyan bir form içermelidir.

Alıştırma 2: Toplam Bakiye

Bu alıştırmada, tüm işlemlerin toplam bakiyesini hesaplayan ve görüntüleyen yeni bir özellik ekleyeceksiniz. Rotayı `app.py` dosyasında oluşturacaksınız.

Talimatlar:

1. `total_balance` adında yeni bir fonksiyon oluşturun ve bunu `/balance` URL'sine eşlemek için `@app.route` dekoratörünü kullanın.
2. Fonksiyonun içinde, işlemler listesindeki tüm işlemlerin miktar değerlerini toplayarak toplam bakiyeyi hesaplayın.
3. Toplam bakiyeyi "Toplam Bakiye: {balance}" formatında bir string olarak döndürün.
4. Toplam bakiyeyi görüntülemek için yeni bir şablon oluşturmanıza gerek yoktur. Bunun yerine, `transactions.html` şablonunu toplam bakiye değerini tablonun altına ekleyecek şekilde değiştireceksiniz.
5. `transactions.html` şablonunda işlemler listesini görüntüledikten sonra, toplam bakiyeyi göstermek için yeni bir satır ekleyin. Daha önce olduğu gibi, hem işlemler listesini hem de toplam bakiye değerini geçerek aynı `render_template` fonksiyonunu kullanabilirsiniz.

::page{title="Sonuç"}

Bu laboratuvarı tamamladığınız için tebrikler.

Bu laboratuvar sırasında şunları öğrendiniz:

- Bir veritabanı uygulamasında CRUD işlevselliğini uygulamak.
- Gelişmiş yönlendirme ve istek yönetimi için Flask kütüphanesinden ek işlevler kullanmak.
- Gereksinimlere göre birden fazla HTML dosyası arasında yönlendirmeyi yönetmek.

Author(s)

Vicky Kuo

Additional Contributor

Abhishek Gagneja

Changelog

Date	Version	Changed by	Change Description
2023-07-24	2.0	Steve Hord	QA geçişi ile düzenlemeler
2023-07-15	1.0	Vicky Kuo	İlk sürüm oluşturuldu