Uygulamalı Laboratuvar: Flask ile API Oluşturma: Rota Oluşturma, Hata Yönetimi ve HTTP İstekleri

Gerekli tahmini süre: 45 dakika

Flask laboratuvarının 2. bölümüne hoş geldiniz. Bu laboratuvar, rotalar ve HTTP istekleri ile çalışacaksınız. Küçük bir RESTful API oluşturma pratiği yapacaksınız. Son olarak, 404 NOT FOUND ve 500 INTERNAL SERVER ERROR gibi yaygın hatalar için uygulama düzeyinde hata yöneticileri ile çalışacaksınız.

Bu laboratuvar için gereken tüm kavramları önceki video setinden bilmelisiniz. Bir görevi nasıl yerine getireceğinizden emin değilseniz veya daha fazla bilgiye ihtiyacınız varsa, laboratuvarı duraklatmaktan ve modülü gözden geçirmekten çekinmeyin.

Öğrenme Hedefleri

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

- Belirli URL'lerde Flask sunucusuna gelen istekleri işlemek için rotalar yazmak
- URL'lere gönderilen parametreleri ve argümanları işlemek
- Sunucu ve kullanıcı hataları için hata yöneticileri yazmak

Skills Network Cloud IDE Hakkında

Skills Network Cloud IDE (Theia ve Docker tabanlı) kurs ve proje ile ilgili laboratuvarlar için pratik bir ortam sağlar. Theia, masaüstünde veya bulutta çalışan açık kaynaklı bir IDE'dir (Entegre Geliştirme Ortamı). Bu laboratuvarı tamamlamak için, Theia ve Docker konteynerinde çalışan MongoDB tabanlı Cloud IDE'yi kullanacaksınız.

Bu laboratuvar ortamı hakkında önemli bir not

Lütfen bu laboratuvar ortamında oturumların kalıcı olmadığını unutmayın. Bu laboratuvara her bağlandığınızda, sizin için yeni bir ortam oluşturulur. Önceki oturumlarda kaydedilen veriler kaybolacaktır. Verilerinizi kaybetmemek için bu laboratuvarları tek bir oturumda tamamlamayı planlayın.

Laboratuvar Ortamını Kurma

Laboratuvara başlamadan önce bazı gerekli ön hazırlıklar vardır.

Terminali Açın

Editördeki menüyü kullanarak bir terminal penceresi açın: **Terminal** > **Yeni Terminal**.

Terminalde, eğer `/home/project` klasöründe değilseniz, şimdi proje klasörünüze geçin.

```
cd /home/project
```

Laboratuvar dizinini oluşturun

Laboratuvarın 1. bölümünden bir laboratuvar dizininiz olmalıdır. Eğer dizininiz yoksa, şimdi oluşturun.

```
mkdir lab
```

`lab` dizinine geçin:

```
cd lab
```

Laboratuvarın 1. bölümünde lab dizininde bir `server.py` dosyası oluşturdunuz. Dosya mevcut değilse oluşturun ve aşağıdaki başlangıç kodunu ekleyin.

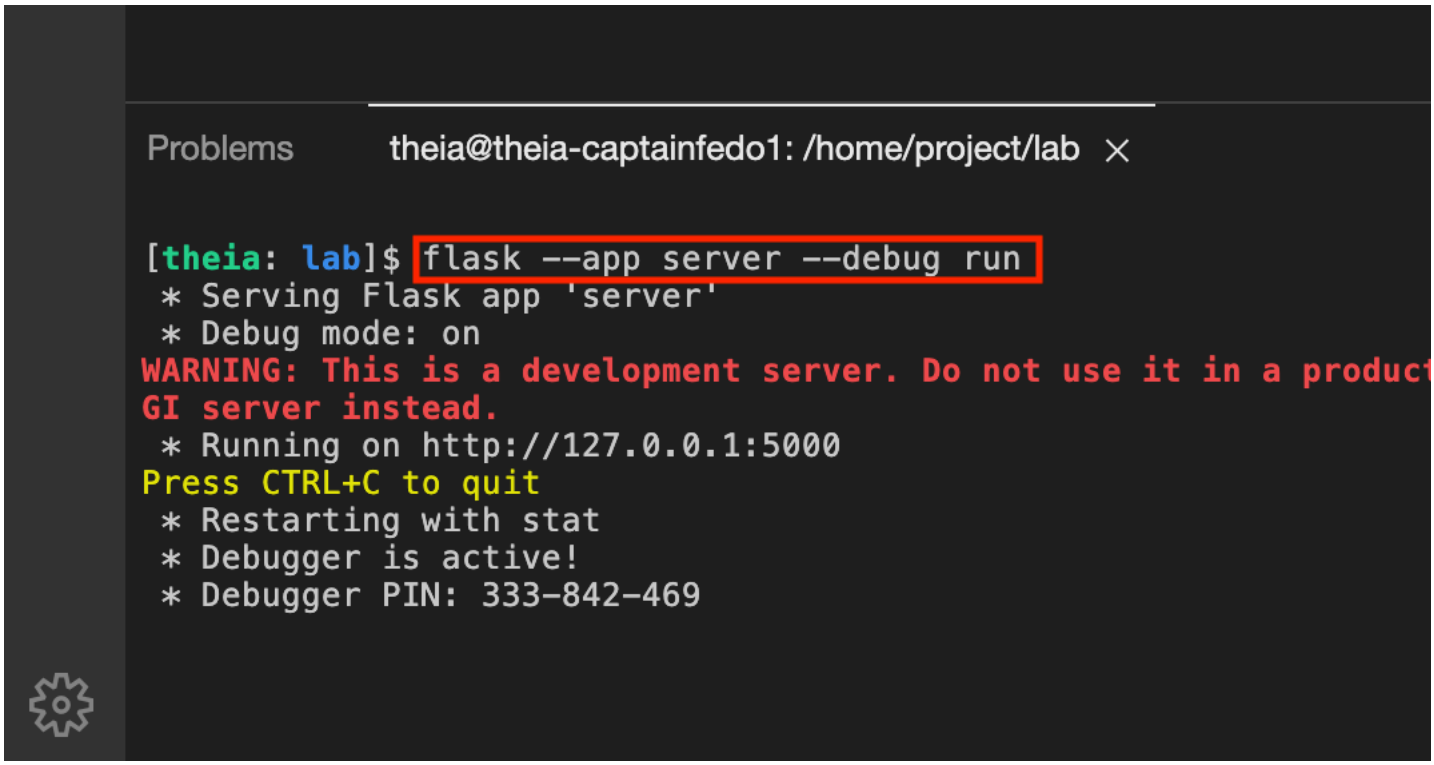
```
# Import the Flask class from the flask module
from flask import Flask
# Create an instance of the Flask class, passing in the name of the current module
app = Flask(__name__)
# Define a route for the root URL ("/")
@app.route("/")
def index():
    # Function that handles requests to the root URL
    # Return a plain text response
    return "hello world"
```

Not: Tüm kodun `IndentationError`'dan kaçınmak için 4 boşluk girintisi kullandığından emin olun.

Yukarıdaki kodun bir Flask sunucusu oluşturduğunu ve **hello world** dizesini döndüren bir ana uç nokta "/" eklediğini hatırlayın. Bu laboratuvarda bu dosyaya daha fazla kod ekleyeceksiniz.

Hatırlatma olarak, sunucuyu terminalden çalıştırmak için aşağıdaki komutu kullanın:

```
flask --app server --debug run
```



Not: Eğer `5000` numaralı port kullanılıyorsa, `5001` gibi farklı bir port deneyin. Sunucu başlatılamazsa, `server.py` dosyasını sözdizimi hataları veya eksik importlar için kontrol edin.

Artık `localhost:5000/` ile CURL komutunu kullanmalısınız. Terminalin sunucuyu çalıştırdığını unutmayın. Terminali bölmek için `Split Terminal` butonunu kullanarak ikinci sekmede aşağıdaki komutu çalıştırabilirsiniz.

```
curl -X GET -i -w '\n' localhost:5000
```

Opsiyonel

Eğer terminalde çalışmak zorlaşırsa çünkü komut istemi uzunsa, aşağıdaki komutu kullanarak istemi kısaltabilirsiniz:

```
export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

Adım 1: Yanıt durum kodunu ayarlama

Son bölümde, Flask'ın bir mesaj geri gönderdiğinizde otomatik olarak `HTTP 200 OK` başarılı yanıtını gönderdiğini gördünüz. Ancak, geri dönen durumu açıkça da ayarlayabilirsiniz. Bunu yapmanın iki yolu olduğunu videoda tartışmıştık:

1. Mesaj ile birlikte bir demet göndermek
2. Durumu ayarlamak için **make_response()** yöntemini kullanarak özel bir yanıt oluşturmak

Görevleriniz

1. Bir demet ile özel HTTP kodu geri gönderin.

   Geçen bölümde üzerinde çalıştığınız `server.py` dosyasını yeniden kullanacaksınız. `/no_content` URL'si ve `@app.route` dekoratörü ile `no_content` adında yeni bir yöntem oluşturun. Yöntemin herhangi bir argümanı yoktur. `No content found` JSON mesajı ile bir demet döndürün.

▶ İpucu için buraya tıklayın.

Aşağıdaki CURL komutuyla uç noktayı test edebilirsiniz:

```
curl -X GET -i -w '\n' localhost:5000/no_content
```

Aşağıdaki gibi bir çıktı görmelisiniz. `204` durumunu ve `application/json` içeriğini not edin. Bir JSON mesajı döndürmüş olsanız bile, bu durum `204` olarak istemciye geri gönderilmez. Varsayılan olarak, hiçbir şey döndürülmez.

```
HTTP/1.1 204 NO CONTENT
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Wed, 28 Dec 2022 19:49:18 GMT
Content-Type: application/json
Connection: close
```

Neden 204 Kullanılır?

204 No Content durumu, bir isteğin başarılı olduğu ancak geri dönecek veri olmadığı durumlarda REST API'lerinde kullanılır. Örneğin:

- Bir kaynağı (örneğin, bir kullanıcıyı) sildikten sonra, sunucu silinen veriyi göndermeden başarıyı onaylar.
- Ek bilgi olmadan bir eylemi (örneğin, ayarları güncelleme) onaylarken.
- Burada bir JSON mesajı döndürüyoruz. Ancak, 204 durum kodu ile istemciye içerik döndürülmeyeceğini belirttiğimiz için, çoğu HTTP istemcisi gövdeyi göz ardı edecektir. Eğer bir mesaj göstermeniz gerekiyorsa, 200 durum kodunu kullanın.

2. `make_response()` yöntemi ile özel bir HTTP kodu geri gönderin. Flask'taki `make_response()` yöntemi, bir HTTP yanıt nesnesini manuel olarak oluşturmak için kullanılır ve yanıt üzerinde yalnızca bir değer veya bir demet döndürmekten daha fazla kontrol sağlar.

   `@app.route` dekoratörü ile `/exp` URL'sine sahip `index_explicit` adında ikinci bir yöntem oluşturun. Yöntemin hiçbir argümanı yoktur. Yeni bir yanıt oluşturmak için `make_response()` yöntemini kullanın. Durumu 200 olarak ayarlayın.

▼ İpucu için buraya tıklayın.

Flask modülünden Flask sınıfını içe aktarın.
Flask modülünden make_response yöntemini içe aktarın.

```
{insert @app decorator}
def {insert method name here}:
    """return 'Hello World' message with a status code of 200
    Returns:
        string: Hello World
        status code: 200
    """
    resp = make_response({insert ditionary here})
    resp.status_code = {insert status code here}
    return resp
```

Aşağıdaki CURL komutuyla uç noktayı test edebilirsiniz:

```
curl -X GET -i -w '\n' localhost:5000/exp
```

Aşağıda verilenle benzer bir çıktı görmelisiniz. `200` durumunu, `application/json` içerik türünü ve `{"message": "Hello World"}` JSON çıktısını not edin:

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Wed, 28 Dec 2022 19:55:46 GMT
Content-Type: application/json
Content-Length: 31
Connection: close
{
  "message": "Hello World"
}
```

Çözüm

Yaptığınız işin aşağıdaki çözümle eşleştiğinden emin olun.

▼ Cevap için buraya tıklayın.

```
from flask import Flask, make_response
# Create an instance of the Flask class, passing in the name of the current module
app = Flask(__name__)
# Define a route for the root URL ("/")
@app.route("/")
def index():
    # Function that handles requests to the root URL
    # Return a plain text response
    return "hello world"
# Define a route for the "/no_content" URL
@app.route("/no_content")
def no_content():
    """Return 'no content found' with a status of 204.
    Returns:
        tuple: A tuple containing a dictionary and a status code.
    """
```

```
    # Create a dictionary with a message and return it with a 204 No Content status code
    return ({"message": "No content found"}, 204)
# Define a route for the "/exp" URL
@app.route("/exp")
def index_explicit():
    """Return 'Hello World' message with a status code of 200.
    Returns:
        response: A response object containing the message and status code 200.
    """
    # Create a response object with the message "Hello World"
    resp = make_response({"message": "Hello World"})
    # Set the status code of the response to 200
    resp.status_code = 200
    # Return the response object
    return resp
```

Adım 2: Girdi argümanlarını işleme

Müşterilerin URL'de argümanlar geçirmesi yaygındır. Bu laboratuvar, argümanların nasıl işleneceğini öğretecek. Laboratuvar, bir nesne içinde id, ad, soyad ve adres bilgileri ile birlikte bir liste insan sağlar. Normalde bu bilgiler bir veritabanında saklanır, ancak basit kullanım durumunuz için sabit kodlanmış bir liste kullanıyorsunuz. Bu veri Mockaroo ile oluşturulmuştur.

Müşteri, `http://localhost:5000?q=first_name` biçiminde istekler gönderecektir. Bir first_name girdi olarak kabul edecek ve bu adı taşıyan bir kişiyi döndürecek bir yöntem oluşturacaksınız.

▼ Veriyi dosyaya kopyalamak için buraya tıklayın.

Aşağıdaki listeyi `server.py` dosyasına kopyalayın:

```
from flask import Flask, make_response
app = Flask(__name__)
data = [
    {
        "id": "3b58aade-8415-49dd-88db-8d7bce14932a",
        "first_name": "Tanya",
        "last_name": "Slad",
        "graduation_year": 1996,
        "address": "043 Heath Hill",
        "city": "Dayton",
        "zip": "45426",
        "country": "United States",
        "avatar": "http://dummyimage.com/139x100.png/cc0000/ffffff",
    },
    {
        "id": "d64efd92-ca8e-40da-b234-47e6403eb167",
        "first_name": "Ferdy",
        "last_name": "Garrow",
        "graduation_year": 1970,
        "address": "10 Wayridge Terrace",
        "city": "North Little Rock",
        "zip": "72199",
        "country": "United States",
        "avatar": "http://dummyimage.com/148x100.png/dddddd/000000",
    },
    {
        "id": "66c09925-589a-43b6-9a5d-d1601cf53287",
        "first_name": "Lilla",
        "last_name": "Aupol",
        "graduation_year": 1985,
        "address": "637 Carey Pass",
        "city": "Gainesville",
        "zip": "32627",
        "country": "United States",
        "avatar": "http://dummyimage.com/174x100.png/ff4444/ffffff",
    },
    {
        "id": "0dd63e57-0b5f-44bc-94ae-5c1b4947cb49",
        "first_name": "Abdel",
        "last_name": "Duke",
        "graduation_year": 1995,
        "address": "2 Lake View Point",
        "city": "Shreveport",
        "zip": "71105",
        "country": "United States",
        "avatar": "http://dummyimage.com/145x100.png/dddddd/000000",
    },
    {
        "id": "a3d8adba-4c20-495f-b4c4-f7de8b9cfb15",
        "first_name": "Corby",
        "last_name": "Tettley",
        "graduation_year": 1984,
        "address": "90329 Amoth Drive",
        "city": "Boulder",
        "zip": "80305",
        "country": "United States",
        "avatar": "http://dummyimage.com/198x100.png/cc0000/ffffff",
    }
]
```

Verilerin dosyaya kopyalandığını onaylayalım. Aşağıdaki kodu **server.py** dosyasına kopyalayarak, kişinin verilerini istemciye JSON formatında döndüren bir uç nokta oluşturun.

```
@app.route("/data")
def get_data():
    try:
        # Check if 'data' exists and has a length greater than 0
        if data and len(data) > 0:
            # Return a JSON response with a message indicating the length of the data
            return {"message": f"Data of length {len(data)} found"}
        else:
            # If 'data' is empty, return a JSON response with a 500 Internal Server Error status code
            return {"message": "Data is empty"}, 500
    except NameError:
        # Handle the case where 'data' is not defined
        # Return a JSON response with a 404 Not Found status code
        return {"message": "Data not found"}, 404
```

Yukarıdaki kod, `data` değişkeninin var olup olmadığını kontrol eder. Eğer yoksa, `NameError` hatası fırlatılır ve bir `HTTP 404` döner. Eğer veri varsa ve boşsa, bir `HTTP 500` döner. Eğer veri varsa ve boş değilse, bir `HTTP 200` başarı mesajı döner.

Başarı mesajını aldığınızı doğrulamak için bir CURL komutu çalıştırın:

```
curl -X GET -i -w '\n' localhost:5000/data
```

Beklenen sonuç:

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Wed, 28 Dec 2022 20:51:56 GMT
Content-Type: application/json
Content-Length: 42
Connection: close
{
  "message": "Data of length 5 found"
}
```

Görevleriniz

`@app.route` dekoratörü ile `name_search` adında bir yöntem oluşturun. Bu yöntem, bir istemcinin `/name_search` URL'sini talep ettiğinde çağrılmalıdır. Yöntem herhangi bir parametre almayacak, ancak gelen istek URL'sinde `q` argümanını arayacaktır. Bu argüman, istemcinin aradığı `first_name` değerini tutar. Yöntem şunları döndürmelidir:

- Eğer `first_name` verilerde bulunursa `HTTP 200` durumu ile kişi bilgilerini
- Eğer `q` argümanı istekten eksikse `HTTP 400` durumu ile `Geçersiz giriş parametresi` mesajını
- Eğer kişi verilerde bulunamazsa `HTTP 404` durum kodu ile `Kişi bulunamadı` mesajını

İpucu

Flask'tan `request` modülünü içe aktardığınızdan emin olun. HTTP isteğinden ilk adı almak için bunu kullanacaksınız.

```
from flask import request
```

Aşağıdaki kodu başlangıç noktası olarak kullanabilirsiniz:

▼ İpucu için buraya tıklayın.

```
@app.route("/name_search")
def name_search():
    """Find a person in the database.
    Returns:
        json: Person if found, with status of 200
        404: If not found
        400: If argument 'q' is missing
        422: If argument 'q' is present but invalid
    """
    # Get the argument 'q' from the query parameters of the request
    query = request.args.get('q')
    # Check if the query parameter 'q' is missing
    if query is None:
        return {"message": "Query parameter 'q' is missing"}, 400
    # Check if the query parameter is present but invalid (e.g., empty or numeric)
    if query.strip() == "" or query.isdigit():
        return {"message": "Invalid input parameter"}, 422
    # Iterate through the 'data' list to look for the person whose first name matches the query
    for person in data:
        if query.lower() in person["first_name"].lower():
            # If a match is found, return the person as a JSON response with a 200 OK status code
            return person, 200
    # If no match is found, return a JSON response with a message indicating the person was not found and a 404 Not Found status code
    return {"message": "Person not found"}, 404
```

Aşağıdaki CURL komutuyla uç noktayı test edebilirsiniz. Sunucunun, önceki adımlardaki gibi terminalde çalıştığından emin olun.

```
curl -X GET -i -w '\n' "localhost:5000/name_search?q=Abdel"
```

Aşağıda verilen çıktıya benzer bir çıktı görmelisiniz. `200` durumunu, `application/json` içerik türünü ve ilk adı **Abdel** olan kişinin JSON çıktısını not edin:

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Wed, 28 Dec 2022 21:14:31 GMT
Content-Type: application/json
Content-Length: 295
Connection: close
{
  "address": "2 Lake View Point",
  "avatar": "http://dummyimage.com/145x100.png/dddddd/000000",
  "city": "Shreveport",
  "country": "United States",
  "first_name": "Abdel",
  "graduation_year": 1995,
  "id": "0dd63e57-0b5f-44bc-94ae-5c1b4947cb49",
  "last_name": "Duke",
  "zip": "71105"
}
```

Sonra, `q` argümanı geçersizse metodun `HTTP 422` döndüğünü test edin:

```
curl -X GET -i -w '\n' "localhost:5000/name_search?q=123"
```

ve

```
curl -X GET -i -w '\n' "localhost:5000/name_search?q="
```

> **Not:** Gerekli sorgu parametresi eksik olduğunda `HTTP 400` (Geçersiz İstek) kullanın. Parametre mevcut ancak geçersiz veya kabul edilemez bir içeriğe sahipse yalnızca `HTTP 422` (İşlenemez Varlık) kullanın.

Aşağıda verilen çıktıya benzer bir çıktı görmelisiniz. `422` durumunu, `application/json` içerik türünü ve `Geçersiz giriş parametresi` JSON çıktısını not edin:

```
HTTP/1.1 422 UNPROCESSABLE ENTITY
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Wed, 28 Dec 2022 21:16:07 GMT
Content-Type: application/json
Content-Length: 43
Connection: close
{
  "message": "Invalid input parameter"
}
```

Sonunda, first_name'in listemizdeki kişiler arasında bulunmadığı durumu test edelim:

```
curl -X GET -i -w '\n' "localhost:5000/name_search?q=qwerty"
```

Aşağıda verilenle benzer bir çıktı görmelisiniz. `404` durumunu, `application/json` içerik türünü ve `Person not found` JSON çıktısını not edin:

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Wed, 28 Dec 2022 21:17:28 GMT
Content-Type: application/json
Content-Length: 36
Connection: close
{
  "message": "Person not found"
}
```

Çözüm

Yaptığınız işin aşağıdaki çözüme uygun olduğundan emin olun. Bu çözümü uygulamanın başka yolları da vardır.

▼ Cevap için buraya tıklayın.

```
@app.route("/name_search")
def name_search():
    """Find a person in the database based on the provided query parameter.
    Returns:
        json: Person if found, with status of 200
        404: If not found
        400: If the argument 'q' is missing
        422: If the argument 'q' is present but invalid (e.g., empty or numeric)
    """
    # Get the 'q' query parameter from the request URL
    query = request.args.get("q")
    # Check if the query parameter 'q' is missing
    if query is None:
        return {"message": "Query parameter 'q' is missing"}, 400
    # Check if the query parameter is present but invalid (empty or numeric)
    if query.strip() == "" or query.isdigit():
        return {"message": "Invalid input parameter"}, 422
    # Iterate through the 'data' list to search for a matching person
    for person in data:
        # Check if the query string is present in the person's first name (case-insensitive)
        if query.lower() in person["first_name"].lower():
            # Return the matching person as a JSON response with a 200 OK status code
            return person, 200
 # If no matching person is found, return a JSON response with a message and a 404 Not Found
    return {"message": "Person not found"}, 404
```

Adım 3: Dinamik URL'ler Ekleyin

Arka uç programlamanın önemli bir parçası API'ler oluşturmaktır. API, bir sağlayıcı ile bir kullanıcı arasındaki bir sözleşmedir. Genellikle, ön uç veya diğer istemciler tarafından çağrılabilen RESTful API'ler oluşturmak yaygındır. REST tabanlı bir API'de, kaynak bilgileri istek URL'sinin bir parçası olarak gönderilir. Örneğin, kaynaklarınız veya kişileriniz ile istemci aşağıdaki isteği gönderebilir:

```
GET http://localhost/person/unique_identifier
```

Bu istekte benzersiz bir tanımlayıcıya sahip bir kişi talep edilmektedir. Bir diğer örnek:

```
DELETE http://localhost/person/unique_identifier
```

Bu durumda, istemci bu benzersiz tanımlayıcıya sahip kişiyi silmek için istekte bulunuyor.

Görevleriniz

Bu alıştırmada bu iki uç noktayı uygulamanız isteniyor. Ayrıca, `data` listesindeki toplam kişi sayısını döndüren bir `count` metodunu da uygulayacaksınız. Bu, GET ve DELETE yöntemlerinin gerektiği gibi çalıştığını doğrulamaya yardımcı olacaktır.

Görev 1: GET /count uç noktasını oluşturun

1. **/count** uç noktasını oluşturun.

   `/count` URL'si için `@app.get()` dekoratörünü ekleyin. `data` listesindeki öğelerin sayısını döndüren count fonksiyonunu tanımlayın.

   ▼ İpucu için buraya tıklayın.

   len() metodunu kullanarak **data** listesindeki öğelerin sayısını döndürün.

   ```
   @app.route("/count")
   def count():
       try:
           # 'data' listesindeki öğelerin sayısını döndürmeyi deneyin
           # {insert code to find length of data} kısmını len(data) ile değiştirin
           return {"data count": len(data)}, 200
       except NameError:
           # 'data' tanımlı değilse ve NameError hatası verirse
   ```

```
        # Bir mesaj ile birlikte JSON yanıtı döndürün ve 500 Internal Server Error durum kodu verin
        return {"message": "data tanımlı değil"}, 500
```

count metodunu çağırarak test edin.

```
curl -X GET -i -w '\n' "localhost:5000/count"
```

Veri listesinde bulunan öğelerin sayısını gösteren bir çıktı almalısınız.

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 22:41:35 GMT
Content-Type: application/json
Content-Length: 22
Connection: close
{
  "data count": 5
}
```

Görev 2: GET /person/id uç noktasını oluşturun

1. Bir kişiyi id ile sormak için **GET** uç noktasını uygulayın.

   `http://localhost/person/benzersiz_tanımlayıcı` için yeni bir uç nokta oluşturun. Metodun adı `find_by_uuid` olmalıdır. UUID türünde bir argüman almalı ve kişi JSON'sini döndürmelidir. Eğer kişi bulunamazsa, metod **kişi bulunamadı** mesajı ile birlikte 404 döndürmelidir. Son olarak, istemci (curl) bu metodu yalnızca geçerli bir UUID türünde id geçirerek çağırabilmelidir.

   ▼ İpucu için buraya tıklayın.

   - geçerli bir UUID geçişine yalnızca çağrıcıların izin vermesi için type:name sözdizimini kullanın.
   - uuid ile string karşılaştırması False döner. Kişinin id niteliği ile karşılaştırırken UUID'yi str'ye dönüştürdüğünüzden emin olun.

```
@app.route("/person/<var_name>")
def find_by_uuid(var_name):
    # 'data' listesini döngüye alarak eşleşen ID'ye sahip bir kişiyi arayın
    for person in data:
        # Kişinin 'id' alanının 'var_name' parametresiyle eşleşip eşleşmediğini kontrol edin
        if person["id"] == str(var_name):
            # Eşleşme bulunursa kişiyi JSON yanıtı olarak döndürün
            return person
    # Eşleşen kişi bulunamazsa, bir mesaj ve 404 Not Found durum kodu ile JSON yanıtı döndürün
    return {"message": "Kişi bulunamadı"}, 404
```

/person/uuid URL'sini çağırarak test edin.

```
curl -X GET -i localhost:5000/person/66c09925-589a-43b6-9a5d-d1601cf53287
```

200 HTTP kodu ile birlikte kişiyi gösteren bir çıktı almalısınız.

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 22:48:32 GMT
Content-Type: application/json
Content-Length: 294
Connection: close
{
  "address": "637 Carey Pass",
  "avatar": "http://dummyimage.com/174x100.png/ff4444/ffffff",
  "city": "Gainesville",
  "country": "United States",
  "first_name": "Lilla",
  "graduation_year": 1985,
  "id": "66c09925-589a-43b6-9a5d-d1601cf53287",
  "last_name": "Aupol",
  "zip": "32627"
}
```

Geçersiz bir UUID geçerseniz, sunucu 404 mesajı döndürmelidir.

```
curl -X GET -i localhost:5000/person/not-a-valid-uuid
```

Çıktıda 404 kodu ile bir hata görmelisiniz. Flask otomatik olarak HTML döndürür, bu kısmı bir sonraki bölümde JSON döndürmesi için değiştireceksiniz.

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 22:50:52 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Connection: close
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>İstenen URL sunucuda bulunamadı. URL'yi manuel olarak girdiyseniz, lütfen yazımınızı kontrol edin ve tekrar deneyin.</p>
```

Son olarak, veri listesinde mevcut olmayan geçerli bir UUID geçin. Metod, **kişi bulunamadı** mesajı ile birlikte 404 döndürmelidir.

```
curl -X GET -i localhost:5000/person/11111111-589a-43b6-9a5d-d1601cf51111
```

HTTP kodu 404 ve **kişi bulunamadı** mesajı içeren bir JSON yanıtı görmelisiniz.

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 22:52:24 GMT
Content-Type: application/json
Content-Length: 36
Connection: close
{
  "message": "kişi bulunamadı"
}
```

Görev 3: DELETE /person/id uç noktasını oluşturun

1. Bir kişi kaynağını silmek için **DELETE** uç noktasını uygulayın.

   DELETE `http://localhost/person/benzersiz_tanımlayıcı` için yeni bir uç nokta oluşturun. Metodun adı `delete_by_uuid` olmalıdır. UUID türünde bir argüman almalı ve o id'ye sahip kişiyi **data** listesinden silmelidir. Kişi bulunamazsa, metod **kişi bulunamadı** mesajı ile birlikte 404 döndürmelidir. Son olarak, istemci (curl) bu metodu yalnızca geçerli bir UUID türünde id geçirerek çağırmalıdır.

   ▼ İpucu için buraya tıklayın.

   - geçerli bir UUID geçişine yalnızca çağrıcıların izin vermesi için type:name sözdizimini kullanın.
   - uuid ile string karşılaştırması False döner. Kişinin id niteliği ile karşılaştırırken UUID'yi str'ye dönüştürdüğünüzden emin olun.
   - @app dekoratörüne ikinci argüman olarak **DELETE** metod türünü geçin veya @app.delete() dekoratörünü kullanın.

```
@app.route("/person/<uuid:id>", methods=['DELETE'])
def delete_person(var_name):
    for person in data:
        if person["id"] == str(var_name):
            # Kişiyi data listesinden çıkarın
            data.remove(person)
            # Silme işlemiyle ilgili bir mesaj ve HTTP durum kodu 200 (OK) ile JSON yanıtı döndürün
            return {"message": "ID'si silinen kişi"}, 200
    # Verilen ID ile eşleşen bir kişi bulunamazsa, bir mesaj ve HTTP durum kodu 404 (Not Found) ile JSON yanıtı döndürün
    return {"message": "Kişi bulunamadı"}, 404
```

   DELETE **/person/uuid** URL'sini çağırarak test edin.

```
curl -X DELETE -i localhost:5000/person/66c09925-589a-43b6-9a5d-d1601cf53287
```

   Silinen kişinin id'si ile birlikte 200 durum kodu içeren bir çıktı almalısınız.

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 23:00:17 GMT
Content-Type: application/json
Content-Length: 56
Connection: close
{
  "message": "ID'si 66c09925-589a-43b6-9a5d-d1601cf53287 silindi"
}
```

   Önceki eklediğiniz **count** uç noktasını kullanarak kişilerin sayısının bir azaldığını test edebilirsiniz.

```
curl -X GET -i localhost:5000/count
```

Sayı bir azalmış olarak döndürülmelidir.

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 23:06:55 GMT
Content-Type: application/json
Content-Length: 22
Connection: close
{
  "data count": 4
}
```

Eğer geçersiz bir UUID geçerseniz, sunucu 404 mesajı döndürmelidir.

```
curl -X DELETE -i localhost:5000/person/not-a-valid-uuid
```

Çıktıda 404 kodu ile bir hata görmelisiniz. Flask otomatik olarak HTML döndürür, bunu bir sonraki bölümde JSON döndürmesi için değiştireceğiz.

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 23:05:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Connection: close
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>İstenen URL sunucuda bulunamadı. URL'yi manuel olarak girdiyseniz, lütfen yazımınızı kontrol edin ve tekrar deneyin.</p>
```

Son olarak, veri listesinde mevcut olmayan geçerli bir UUID geçin. Metod, **kişi bulunamadı** mesajı ile birlikte 404 döndürmelidir.

```
curl -X DELETE -i localhost:5000/person/11111111-589a-43b6-9a5d-d1601cf51111
```

HTTP kodu 404 ve **kişi bulunamadı** mesajı içeren bir JSON yanıtı görmelisiniz.

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sat, 31 Dec 2022 23:05:43 GMT
Content-Type: application/json
Content-Length: 36
Connection: close
{
  "message": "kişi bulunamadı"
}
```

Çözüm

Yaptığınız işin aşağıdaki çözüme eşit olduğundan emin olun.

▼ Yanıt için buraya tıklayın.

```
@app.route("/count")
def count():
    try:
        # 'data' listesindeki öğelerin sayısını JSON yanıtı olarak döndürmeyi deneyin
        return {"data count": len(data)}, 200
    except NameError:
        # 'data' tanımlı değilse durumu ele alın
        # Bir mesaj ile birlikte JSON yanıtı döndürün ve 500 Internal Server Error durum kodu verin
        return {"message": "data tanımlı değil"}, 500
@app.route("/person/<uuid:id>")
def find_by_uuid(id):
    # 'data' listesini döngüye alarak eşleşen ID'ye sahip bir kişiyi arayın
    for person in data:
        # Kişinin 'id' alanının 'id' parametresiyle eşleşip eşleşmediğini kontrol edin
        if person["id"] == str(id):
            # Eşleşen kişiyi 200 OK durum kodu ile JSON yanıtı olarak döndürün
            return person
    # Eşleşen kişi bulunamazsa, bir mesaj ve 404 Not Found durum kodu ile JSON yanıtı döndürün
    return {"message": "kişi bulunamadı"}, 404
@app.route("/person/<uuid:id>", methods=['DELETE'])
def delete_by_uuid(id):
    # 'data' listesini döngüye alarak eşleşen ID'ye sahip bir kişiyi arayın
    for person in data:
        # Kişinin 'id' alanının 'id' parametresiyle eşleşip eşleşmediğini kontrol edin
        if person["id"] == str(id):
            # Kişiyi 'data' listesinden çıkarın
            data.remove(person)
            # Silme işlemiyle ilgili bir mesaj ve 200 OK durum kodu ile JSON yanıtı döndürün
            return {"message": f"ID'si {id} silindi"}, 200
    # Eşleşen kişi bulunamazsa, bir mesaj ve 404 Not Found durum kodu ile JSON yanıtı döndürün
    return {"message": "kişi bulunamadı"}, 404
```

Adım 4: İstek gövdesinden JSON Ayrıştırma

Başka bir RESTful API oluşturalım. İstemci, kişi detaylarını içeren JSON'u gövde olarak `http://localhost:5000/person` adresine `POST` isteği ile gönderebilir. Sunucu, isteği gövde için ayrıştırmalı ve ardından bu detayla yeni bir kişi oluşturmalıdır. Sizin durumunuzda, kişiyi oluşturmak için `data` listesine ekleyin.

Görevleriniz

`@app.route` dekoratörü ile `add_by_uuid` adında bir yöntem oluşturun. Bu yöntem, bir istemci `/person` URL'si için `POST` yöntemi ile istek yaptığında çağrılmalıdır. Yöntem herhangi bir parametre almayacak, ancak POST isteğinin JSON gövdesinden kişi detaylarını alacaktır. Yöntem şunları döndürmelidir:

- Kişi başarıyla `data` listesine eklendiğinde HTTP 200 durum kodu ile kişinin `id`si.
- JSON gövdesi eksik veya boşsa HTTP 422 durum kodu ile `"Geçersiz giriş parametresi"` mesajı.

İpucu

Flask'tan `request` modülünü içe aktardığınızdan emin olun. Bunu, HTTP isteğinden adını almak için kullanacaksınız.

```
from flask import request
```

Aşağıdaki kodu başlangıç noktası olarak kullanabilirsiniz. Üretim kodunda, gelen JSON'u doğrulamak için bazı mantıklar ekleyeceksiniz. Bir istemciden gelen rastgele verileri depolamak istemezsiniz. Basit kullanım durumunuz için bu doğrulamayı atlayabilirsiniz.

▼ İpucu için buraya tıklayın.

```
@app.route("/person", methods=['POST'])
def create_person():
    # Get the JSON data from the incoming request
    new_person = request.get_json()
    # Check if the JSON data is empty or None
    if not new_person:
        # Return a JSON response indicating that the request data is invalid
        # with a status code of 422 (Unprocessable Entity)
        return {"message": "Invalid input, no data provided"}, 422
    # Proceed with further processing of 'new_person', such as adding it to a database
    # or validating its contents before saving it
    # Assuming the processing is successful, return a success message with status code 200 (Created)
    return {"message": "Person created successfully"}, 200
```

Aşağıdaki CURL komutunu kullanarak uç noktayı test edebilirsiniz. Sunucunun, önceki adımlardaki gibi terminalde çalıştığından emin olun.

```
curl -X POST -i -w '\n' \
  --url http://localhost:5000/person \
  --header 'Content-Type: application/json' \
  --data '{
        "id": "4e1e61b4-8a27-11ed-a1eb-0242ac120002",
        "first_name": "John",
        "last_name": "Horne",
        "graduation_year": 2001,
        "address": "1 hill drive",
        "city": "Atlanta",
        "zip": "30339",
        "country": "United States",
        "avatar": "http://dummyimage.com/139x100.png/cc0000/ffffff"
}'
```

Aşağıda verilenle benzer bir çıktı görmelisiniz. `200` durumunu, `application/json` içerik türünü ve ilk adı **Abdel** olan kişinin JSON çıktısını not edin:

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sun, 01 Jan 2023 23:14:34 GMT
Content-Type: application/json
Content-Length: 56
Connection: close
{
  "message": "4e1e61b4-8a27-11ed-a1eb-0242ac120002"
}
```

Aşağıdaki komutu kullanarak uç noktaya boş bir JSON gönderdiğiniz durumu da test edebilirsiniz:

```
curl -X POST -i -w '\n' \
  --url http://localhost:5000/person \
  --header 'Content-Type: application/json' \
  --data '{}'
```

Sunucu, `Invalid input parameter` mesajı ile birlikte 422 kodu döndürmelidir.

```
HTTP/1.1 422 UNPROCESSABLE ENTITY
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sun, 01 Jan 2023 23:15:54 GMT
Content-Type: application/json
Content-Length: 43
Connection: close
{
  "message": "Invalid input parameter"
}
```

Çözüm

Yaptığınız işin aşağıdaki çözümle eşleştiğini iki kez kontrol edin. Bu çözümü uygulamanın birden fazla yolu vardır. Ayrıca, `data` listesinin çözümde mevcut olup olmadığını kontrol ettiğinizi ve mevcut değilse 500 döndürdüğünüzü unutmayın.

▼ Buraya tıklayın cevabı görmek için.

```
@app.route("/person", methods=['POST'])
def add_by_uuid():
    new_person = request.json
    if not new_person:
        return {"message": "Invalid input parameter"}, 422
    # code to validate new_person ommited
    try:
        data.append(new_person)
    except NameError:
        return {"message": "data not defined"}, 500
    return {"message": f"{new_person['id']}"}, 200
```

Adım 5: Hata işleyicileri ekleyin

Laboratuvarın bu son kısmında, 404 (bulunamadı) ve 500 (sunucu hatası) gibi hataları işlemek için uygulama düzeyinde global işleyiciler ekleyeceksiniz. Videodan hatırlarsanız, Flask bu global hata işleyicilerini errorhandler() dekoratörü ile kolayca yönetmenizi sağlar.

Şimdi sunucuya geçersiz bir istek yaparsanız, Flask 404 hatası ile birlikte bir HTML sayfası döndürecektir. Böyle bir şey:

```
curl -X POST -i -w '\n' http://localhost:5000/notvalid
```

I'm here to assist you with translations. Please provide the markdown document snippet you would like me to translate.

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sun, 01 Jan 2023 23:21:54 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Connection: close
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually, please check your spelling and try again.</p>
```

Bu harika, ancak geçersiz istekler için bir JSON yanıtı döndürmek istiyorsunuz.

Görevleriniz

`@app.errorhandler` dekoratörü ile `api_not_found` adında bir yöntem oluşturun. Bu yöntem, istemci sunucu tarafından tanımlanan herhangi bir uç noktaya yönlendirmeyen bir URL talep ettiğinde, `API not found` mesajını `404` HTTP durum kodu ile döndürecektir.

İpucu

`@app.errorhandler` dekoratörünü kullanın ve `404` HTTP kodunu geçirin.

Aşağıdaki kodu başlangıç noktanız olarak kullanabilirsiniz:

▼ Buraya tıklayın bir ipucu için.

```
{insert errorhandler decorator here}({insert error code here})
def {insert method name here}(error):
    return {"message": "{insert error message here}"}, {insert error code here}
```

Aşağıdaki CURL komutuyla uç noktayı test edebilirsiniz. Sunucunun, önceki adımlarda olduğu gibi terminalde çalıştığından emin olun.

```
curl -X POST -i -w '\n' http://localhost:5000/notvalid
```

Aşağıdakine benzer bir çıktı görmelisiniz. `404` durumunu, `application/json` içerik türünü ve **API bulunamadı** JSON çıktı mesajını not edin:

```
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Sun, 01 Jan 2023 23:25:35 GMT
Content-Type: application/json
Content-Length: 33
Connection: close
{
  "message": "API not found"
}
```

Not edin ki sunucu artık HTML değil, gereken şekilde JSON döndürüyor.

Çözüm

Yaptığınızın aşağıdaki çözümle eşleştiğinden emin olun. Bu çözümü uygulamanın birden fazla yolu vardır.

▼ Cevap için buraya tıklayın.

```
@app.errorhandler(404)
def api_not_found(error):
    # This function is a custom error handler for 404 Not Found errors
    # It is triggered whenever a 404 error occurs within the Flask application
    return {"message": "API not found"}, 404
```

Benzer şekilde, `500 (iç sunucu hatası)` için de bir genel Hata İşleyici ekleyebilirsiniz.

Flask'ta herhangi bir işlenmemiş istisna için genel bir hata işleyici kaydetmek için:

```
@app.errorhandler(Exception)
def handle_exception(e):
    return {"message": str(e)}, 500
```

Bu, Flask'a uygulamanızda herhangi bir yerde meydana gelen işlenmemiş bir istisnayı yakalayıp bu işleyiciye yönlendirmesini ve hata mesajıyla birlikte 500 İç Sunucu Hatası yanıtı döndürmesini söyler.

Küresel işleyiciyi test etmek için kasıtlı olarak bir istisna yükseltebilirsiniz. Hata işleyicilerinizden önce aşağıdaki rotayı ekleyin:

```
@app.route("/test500")
def test500():
    raise Exception("Forced exception for testing")
```

Sonra bu curl komutunu terminalinizde çalıştırın:

```
curl http://localhost:5000/test500
```

Bir yanıt görmelisiniz:

```
{
  "message": "Forced exception for testing"
}
```

Yazar(lar)

CF