FLASK HELLO WORLD

Flask bir python package. Kullanmak için ilk olarak pip ile flask'i yüklememiz gerek. Bunun yanında ubuntu'ya istediğimiz bir IDE'yi yüklüyoruz, ben VSCode yüklerim.

Şimdi istediğimiz bir folder içerisinde app.py dosyası oluşturuyoruz ilk flask web service'imizi bu file içerisinde yazacağız:

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello World!"

if __name__=="__main__":
    app.run()
```

Şimdilik bu kodu anlamamız gerekmiyor, ilk web service'imizi yaratmış olduk, ancak henüz bu service'i run etmedik.

İlgili web service'i run etmek için, terminalden app.py dosyasının klasörüne konumlandıktan sonra:

```
export FLASK_APP = app.py
flask run
```

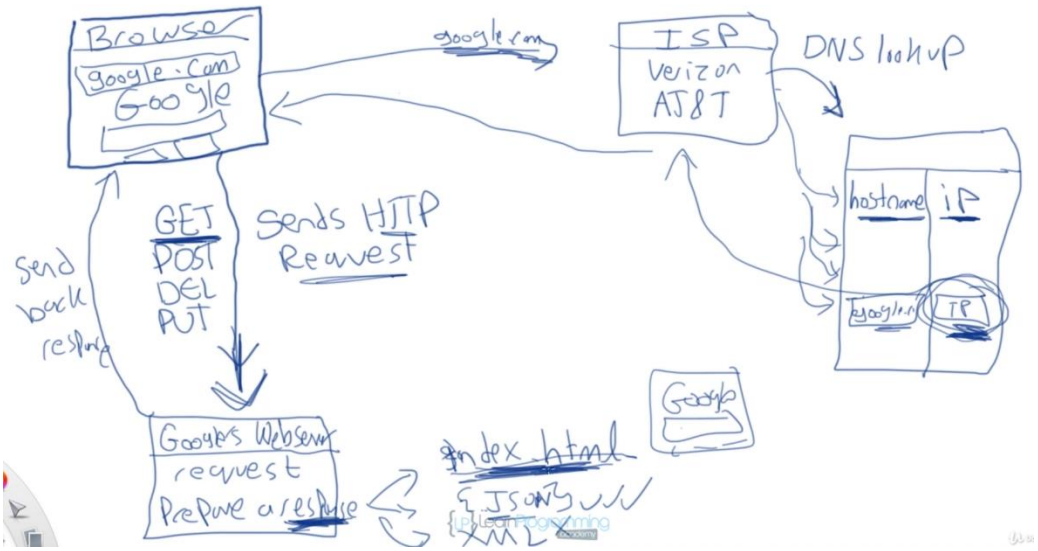
Diyoruz, sonuçta ilgili web service'i local host üzerinde çalıştırılmış olacak, service'in çalıştığı adres ve dinlediği port adresi terminalde görülecek, eğer bir tarayıcıdan bu adrese request gönderirsek, taryıcıda "Hello World!" yazdığını göreceğiz, ilk web service'imizi çalıştırmış olduk, detaylara yavaş yavaş gireceğiz.

UNDERSTANDING THE WEB CYCLE FOR DEVELOPMENT

Flask'a daha detaylı bir giriş yapmadan önce web nasıl çalışır bunun temellerini anlamaya çalışacağız.

Genel akış aşağıdaki gibi olacak, browser'a bir URL girdiğimizde (örneğin www.google.com) ilk olarak bu url internet service provider'a gönderilir. Bu ISP bir lookup yaparak ilgili URL'ye karşılık gelen IP address'i browser'a geri döndürür. Bazı browser'lar her seferinde ISP ile muhattap olmamak için URL'lere karşılık gelen IP addressleri cache ederler.

Sonuç olarak, ilgili URL'ye karşılık gelen IP adresi elde edilir ve daha sonrasında, browser ilgili IP adresine bir **HTTP Request** gönderir, bu IP adresi google'ın bir web server'ına denk gelir. Bu server üzerinde bir web service çalışır ve bu web service gelen request'e karşılık bir **response** hazırlar ve hazırlanan response'u browser'a geri yollar.



Brower'dan server'a GET, POST, DEL, PUT gibi farklı tipte request'ler gönderilebilir, bunların her birine karşılık, server üzerinde çalışan web service'in yapacağı işlemler farklıdır, bu işlemler sonucunda, web service ilgili browser'a bir hmtl sayfası, json veya xml verisi return edebilir.

Daha sonra ilgili browser return edilen response'a göre farklı işlemler yapar. Burada bizim ilgilenceğimiz kısım, request'in alındığı ve istenen response'un hazırlanıp return edildiği web service kısmı. Bu kısım back-end olarak bilinir.

Web service ile API interchangebly kullanılabiliyor.

Önceki başlıktaki örnekte, web service'i run ettiğimizde bize **127.0.0.1:5000/** gibi bir URL vermişti, aslında bu URL doğrudan bir IP adrese denk geliyor bu da bizim LOCAL HOST'umuzun, yani kendi bilgisayarımızın IP adresi. Bu localhost'a yalnızca kendi pc'mizden ulaşabileceğimizi de unutma.

Flask uygulamamızı run ettiğimizde, local host üzerinde çalışan bir web service yaratmış olduk ve dedik ki, bu web service'e "/" ile bir request gelirse, request'i gönderen browser'a "Hello world" return edilsin.

REVISITING HELLO WORLD PROGRAM

Şimdi ilk başlıkta oluşturduğumuz basit web service'i biraz daha detaylı inceleyeceğiz.

```
from flask import Flask
app = Flask (__name__)
```

Yukarıdaki ilk satır ile flask package'i içerisinden Flask objesini import ediyoruz. Bu Flask objesini bir flask application construct etmek için kullanacağız.

İkinci satırda bu objeyi kullanarak bir Flask application oluşturuyoruz, bu object içerisine istediğimiz bir application ismi verebiliriz, ancak yukarıdaki gibi __name__ variable'ı kullanmak convention'dır. Bunu kullandığımızda, ilgili dosya çalıştırılırsa __name__ "main"'e karşılık gelecektir.

Şimdi incelemeye devam edelim:

```
@app.route('/')
def hello_world():
    return "Hello World!"
```

Burada ilk satır şunu ifade ediyor, ne zaman ki ilgili flask app'i çalışır durumda ise yani web service çalışır durumdaysa, bu application her zaman '/' endpoint'i için bir dinleme yapıyor, her zaman birinin server'in ip adresini ve port'unu girip yanına da '/' koyarak bir request göndermesini bekliyor, ne zaman ki böyle bir request gönderildi, hellp_world() fonksiyonu request'i handle ediyor ve request'in geldiği browser'a bir response return ediyor.

Burada '/' yerine istediğimiz başka yapılar da kullanabiliriz:

```
@app.route('/hithere')
def heyFunc():
    return "I just hit /hithere"
```

Artık flask app hem '/' hem de '/hithere' ile gönderilecek requestler için dinlemede olacak, ve bunlardan herhangi biri gönderildiğinde yazılan fonksiyon ile request'i handle edip bir response return edecek.

return

Fonksiyonlar içerisinde mutlaka string return etmek zorunda değiliz, json veya page yapıları da return edebiliriz.

```
if __name__ == ”__main__”:
        app.run(host=”127.0.0.1”,port=80)
```

Burada yapılan şu, eğer bu file yani app.py dosyası çalışıtırılırsa, *__name__==”main”* olacaktır, yok import edilirse __name__==”app” olacaktır. O yüzden dedik ki, eğer bu app.py dosyası çalıştırılırsa, yukarıda tanımlanan flask application’u run edilsin.

Run içine girilen host ve port bilgisi girilmese de olur, zaten bunu yalnızca terminalden flask’ı run ettikten sonra görebiliriz, oradan alıp buraya kopyalanmış, sadece bilgi amaçlı sanıyorum.

Ancak app.run() satırında önemli bir nokta şu, debug = True olmalı, yani:

```
if __name__ == ”__main__”:
        app.run(debug=True)
```

Bu yapı önemli çünkü run ederken debug=True olduğu zaman yapılacak olası hataları, app’i run ettiğimizde terminalde görebiliriz.

Web Service(APIs) vs Web Apps

Web services (APIs) mostly return JSON, buna karşın Web Apps index.html gibi web page return ederler.

Web services yani APIs genelde client’dan request ile birlikte JSON veriler alabilir ve bunu işleyip karşılığında JSON bir response return edebilir.

JSON, EXAMPLES OF JSON & JSON IN FLAKS

JSON temelde bir data formattır. Bu format between browsers veya between servers and browsers arasında data interchange için sıklıkla kullanılır.

Burada önemli bir nokta var, internet üzerinde server/server, server/browser veya browser/browser tüm communications TEXT ile yapılır! Örneğin doğrudan images/videos yollayamayız, text representations of images/videos yollayabiliriz.

Bunun sebebi TCP internet protokolüdür, bu protokol yalnızca text transfer'e izin verir.

JSON da bir çeşit TEXT formatıdır, JSON formatı sayesinde server/server, server/browser veya browser/browser data interchange kolay ve anlaşılır şekikde yapılabilir.

JSON formatı içerisinde çok çeşitli yapıları (strings, numbers, booleans, null, arrays, arrays of different types, other json objects, array of json objects, array of arrays, …) value olarak gönderebilirim. Key olarak ise mutlaka string göndermek zorundayız! Aşağıda örnek bir json yapısı return ediliyor.

```python
from flask import Flask,jsonify
app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello World!"

@app.route('/hey')
def hey():
    return "Hey!"

@app.route('/json')
def json0():
    rjson = {
        'Name': 'Erdo',
        'Age': 25,
        'phones': [
            {
                'phoneName': "Xiaomi",
                'phoneNum': 555
            },
            {
                'phoneName': "Iphone",
                'phoneNum': 222
            }
        ]
    }
    return  jsonify(rjson)

if __name__=="__main__":
    app.run(debug=True)
```

Dikkat edersen burada **jsonify**'i import ettik. JSON return etmek, için python içerisinde oluşturulan dictionary'i **jsonify** objesi içerisine verip öyle return ediyoruz.

GET & POST REQUESTS + RESPONSE STRUCTURE

Şimdiye kadar, browser'ın server'a request yolladığından ve server'ın da bunun karşılığında bir response döndürdüğünden bahsettik. Burada biraz daha detaya inelim.

Browser request'i yollarken mutlaka spesifik bir **request method** ile yollar, bir çok farklı request method'dan söz edilebilir ancak şimdilik iki temel request method'dan söz edelim. **GET** ve **POST** requests.

GET

Şimdiye kadar gönderdiğimiz requests zaten GET method'u kullanıyordu, bu request'e bir endpoint belirtiyorduk, browser server'dan ilgili endpoint için yapılacak işlemin yapılmasını request eder, bu request sonucunda daha önce de bahsettiğimiz gibi farklı tipte response'lar return edilebilir.

Biraz evvelki örnekte /json endpoint'ine bir get request'i gönderdiğimizde terminalde şöyle bir çıktı görüyorduk:

```
"GET /json HTTP/1.1" -200
```

Burada görüyoruz ki /json endpoint'ine bir GET request gönderildi, bu request HTTP protokolünü kullandı.

POST

Bir diğer popüler request ise post request'tir. Bu request methodunu server'a bir veri gönderceğimiz zaman kullanırız. Örneğin server'a ilgili endpoint için bir JSON gönderip, web service'in bu veriye göre bazı işlemler yapıp bana return etmesini istiyor olabilirim.

Server gönderilen her tip veriyi kabul etmek zorunda değil, bunun için authentication da yapılacak, bunları daha sonra göreceğiz. Şimdilik gönderilen tüm verilerin iyi niyetli olduğunu varsayalım.

Özetle get request ilgili end point için sever'dan bir response beklerken, post request server'ın ilgili endpoint'ine bir veri yollar ve karşılığında yine bir response bekler!

RESPONSE

Server'a bir request geldiğinde, request'i gönderen browser server'dan bir response bekler, server'ın bir response göndermesi gerekir. Peki bu response'un yapısı nasıldır?

Status Line

Öncelikle response'un bir **Status line**'u olur, status line içerisinde bir **status code** içerir. Bu kod temelde, request'e ne olduğunu gösterir. Örneğin eğer **status code 200** ise bu **SUCCESS** demektir, yani ilglili request'in successfully handled edildiğini belirtir. Bir diğer status code ise **404** bu **NOT FOUND** anlamına gelir, yani request handle edilemedi! Bunun farklı sebepleri olabilir örneğin eğer ilgili endpoint'in server'da bir karşılığı yoksa bu kodu alırız.

Message Body

Response yapısının ikinci önemli parçası, message body'dir. Response body içerisinde bir JSON object return edilir.

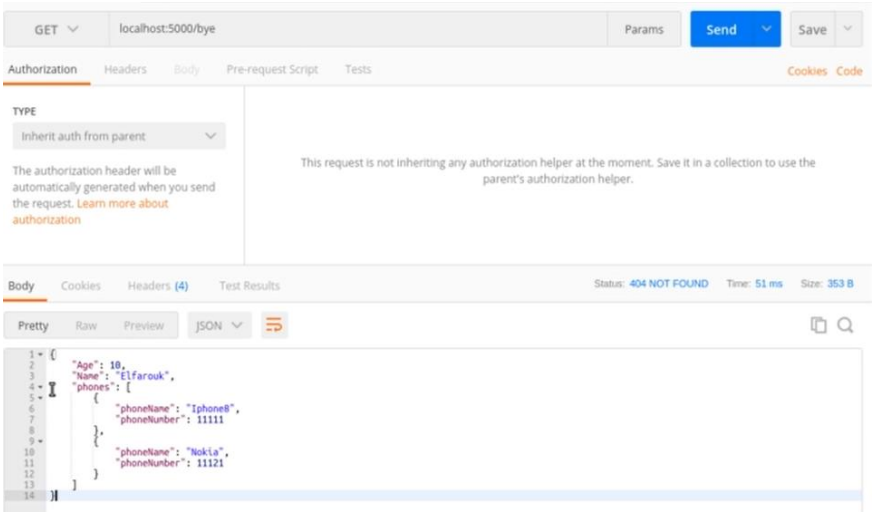
Message body reponse'un bir parçası olduğu gibi, post request'in de bir parçasıdır çünkü post request de karşı tarafa bir veri taşır.

POSTMAN

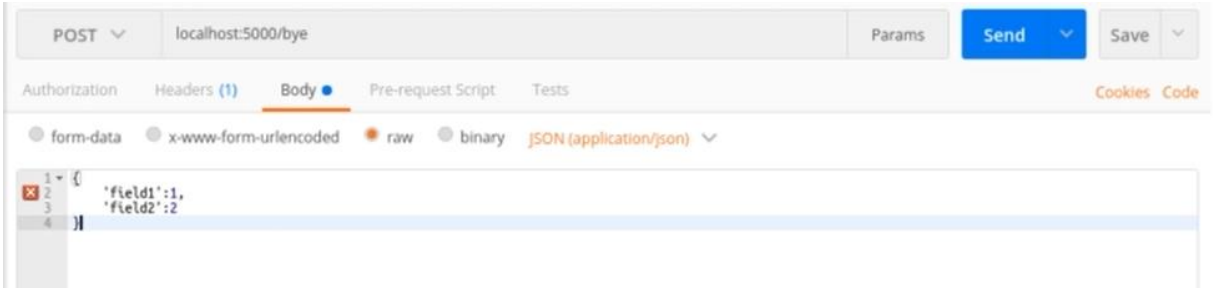
Get request'i browser üzerinden direkt server ip adresini ve endpoint'i yazarak gönderebiliyorduk, ancak post request'i bu şekilde gönderemiyoruz.

Bu sebeple, ubuntu sistemine POSTMAN programını install ediyoruz, bu programın sağladığı arayüz sayesinde, server'a istediğimiz tipte request'ler gönderebileceğiz.

GET



POST



Ancak şuan server post request'ine bir cevap veremeyecektir çünkü app'imizi bunun için hazırlamadık, bir sonraki başlıkta, post requests'i nasıl handle ederiz buna bakacağız.

HANDLING POST REQUESTS IN FLASK & BUILDING A MINI API

Şuanda web service'imiz yalnızca get request'lere cevap verecek şekilde hazırlandı, varolan endpoints post requests'i handle edemez.

Şimdi web service'imizi post request'i handle edebilecek şekilde manipüle edelim. Yapmak istediğimiz basit örnek şu: browser'dan server'a aşağıdaki gibi bir json objesi post request ile message body olarak gönderilsin

```
{
    "x":1,
    "y":2
}
```

API ise x ve y'yi toplasın ve geriye sonucu response olarak return etsin.

Bunun için ilk adım daha önce olduğu gibi bir end point tanımlamak, burada endpoint'in yanında methods da tanımladık.

```
@app.route('/add_two_nums', methods=["POST"])
```

Böylelikle, ilgili endpoint yalnızca POST requests kabul edecek, eğer bu endpoint'e başka tipte bir request gelirse, service bu request'i handle etmeyecek ve hata döndürecektir.

Daha önceki endpoints için herhangi bir method tanımlamadığımıza dikkat et, eğer bir method tanımlanmazsa default olarak "GET" methodu tanımlanmış olur, methods içerisine birden fazla method da yazılabilir, yukarıdaki örnek için sadece "POST" yazıldı bu sebeple bu end point yalnızca POST requests'e cevap verecektir.

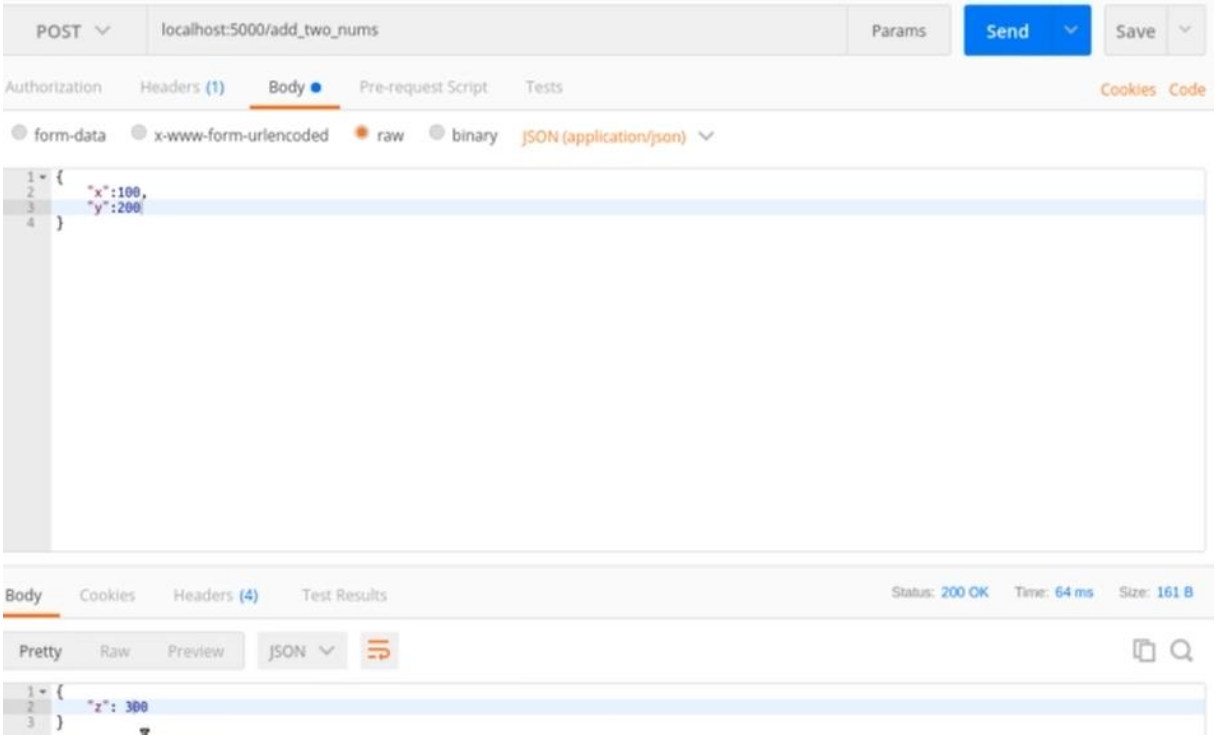
Şimdi sırada, bu endpoint için bir fonskiyon tanımlamak var, bu fonskiyon aşağıdaki gibi dört temel adımdan oluşacak, request objesi kullanılarak data alınacak, istenilen şekilde manipüle edildikten sonra jsonify ile response döndürlecek, ayrıca response kodu da return ediliyor:

```
@app.route('/add_two_nums', methods=["POST"])
def add_two_nums():

        #Get x,y from posted data
        dataDict = request.get_json()
        x = dataDict["x"]
        y = dataDict["y"]

        #Obtain z = x+y
        z = x+y

        #Prepare a JSON to send as a response.
        retJSON = {
                "z":z
        }

        #Return prepared json using jsonify.
        return jsonify(retJSON), 200
```

Önemli bir nokta şu ki, post request ile gönderilen message body'e ulaşabilmek için, request objesini import etmeyi unutmuyoruz!

```
from flask import request
```

Yukarıdaki flask app'i run ettiğimizde, ilgili endpoint'e gönderilen json için service sayıların toplamını veren bir başka json'ı response olarak gönderecektir:



Peki ya, bu endpoint'e gönderilen json yapısı beklediğimiz gibi değilse, örneğin x ve ye yerine sadece x içeren bir json gönderilirse? Bu durumda doğal olarak API görevini yerine getiremeyecek, ve hata return edecek.

Ancak bu hata, API çöktüğü için gönderiliyor, bunun olmasına izin vermeden önce gönderilen verinin istediğimiz kriterlere uyup uymadığını test etmek ve eğer uymuyorsa buna göre bir hata mesajı ve status code return etmek gereklidir.

Örneğin aşağıdaki gibi bir if yapısında, gönderilen request json'ı içerisinde y field'i yoksa, ERROR return edilsin ve 305 kodu gönderilsin dedik, böylece server'ımız çökmeden browser'a istediğimiz veriyi return etmiş olduk.

```
@app.route('/add_two_nums', methods=["POST"])
def add_two_nums():
    #Get x,y from the posted data
    dataDict = request.get_json()

    if "y" not in dataDict:
        return "ERROR", 305
    x = dataDict["x"]
    y = dataDict["y"]
```