WHAT IS A RESTful API

Bir restful api build etmeden önce ne olduğunu kabaca anlamaya çalışacağız.

It is basically an architectural style of building API

An API is basically code that allows you to take a request, does some process and prepares a response and returns this response. Daha önceki kısımlarda çok basit bir mini-api yaratmıştık, bu api'ya post request ile istediğimiz sayıları yolladığımız zaman bize toplamı veriyordu, bildiğimiz bir web service.

Bunun dışında fastai'ı hatırlarsan, datablock API deniyordu, aslında libraries de APIs olarak düşünülebilir, yani örneğin ben numpy'ı import ettikten sonra np.array() diyerek aslında numpy API'ını kullanıyorum ve bu API'ya bir takım request'ler gönderim sonucunda bir takım response'lar alıyorum gibi düşünebilirsin, yani library'lerin programmer'lara exposed edilmiş olan kısımlarını API olarak düşünmek mümkün. Burada arayüz tanımı önemli, library dediğimzide tüm kodlar aklımıza gelir, yani numpy içerisinde arkaplanda ne işlemler yapıldığı vesaire library bunları kapsar. API dediğimizde ise, numpy kütüphanesinin kullanıcılarına sunduğu kısım aklımıza gelir, kullanıcılar bu API sayesinde farklı fonksiyonarla (bunları endpoints gibi düşün) farklı işlemleri request edebilir bu işlemler arkaplanda yapılır ve kullanıcıya istediği sonuç return edilir.

Web APIs iletişim yolu olarak, http protokolünü kullanırlar, yani API bir http request alır ve gerekli işlemleri yaptıktan sonra yine http protokolü ile bir response döndürür.

RESTful API ise specific bir API dizaynıdır, API build etmenin bundan başka yolları da vardır. Anladığım kadarıyla http iletişimi kullanıldığında Web API oluyor, bu web api spesifik dizayn kritilerlerini yerine getiriyorsa da RESTful API oluyor.

RESTful APIs sayesinde bir çok complicated işlemi (heavy computations) browser üzerinde veya web sunucusunda veya bir mobile app ise mobile device üzerinde yapmak yerine, bu işlemi yapacak API'ya bir request yolluyoruz ve bu complicated işlem server üzerinde yapılıp cevap mobile app'e veya browser'a return ediliyor.

Örneğin bir image recognition uygulaması yapıyoruz, browser veya mobile app üzerine yüklenen fotoğraf http request ile API'ya yollanacak ve API server'ı üzerinde image recognition uygulaması çalışacak, daha sonra API sonucu response olarak web veya mobile uygulamasına gönderecek, bu şekilde mobile üzerinde hiçbir local image recognition işlemi yapılmamış olacak zira bunu yapmak zahmetli olurdu.

APIs sayesinde, uygulamaları ligtweight tasarlayabiliriz, hiçbir complex hesaplama localde (mobile veya browser üzerinde) yapılmaz, bu işlemleri yapmak için bir cloud üzerinde bir sürü bağımsız API oluşturabiliriz ve işlerimizi bu APIs kullanarak halledebiliriz. Bunlar gerektiğinde uygulamalar tarafından http aracılığı ile kullanılabilecek web services!

RESTFUL API RESOURCE METHOD CHART

Şimdi oluşturmak istediğimiz restful api'yı kodlamadan önce sürece yardımcı olacak bir chart oluşturacağız. Bu chart kodlama süresince bize yardımcı olacak.

Örneğin diyelim ki amacımız iki number alan ve bu numbers üzerinde dört işlemi yapıp (+,-,*,/) sonucu return eden bir restful api oluşturmak olsun.

Resources: API'ın offer ettiği assetleridir, user'ın API'dan beklediği/istediği şeydir, örneğin yukarıda örnekte API bize operations offer ediyor, resources operations olmuş oluyor.

GET API'dan resource istemek için kullanılır, POST ile API'ya veri yollarız ve sonucunda yine bir response bekleriz. Bunların yanında PUT ve DELETE de vardır. PUT'u update olarak düşünebiliriz, post'a benzer ancak post genelde yeni bir kayıt oluşturmak için kullanılırken, put ise var olan kaydı update etmek için kullanılır. Ancak put ve delete pratikte çok kullanılmaz çünkü bunların yaptığı işlemleri istersek post ile de yapabiliriz.

Şimdi yukarıda bahsedilen API'ı kodlamadan önce bir resource-method chart yapacağız.

Resource	**Method**	**Path**	**Used For**	**Param**	**Error Code**
+	POST	/add	adding 2 nums	x:int y:int	200:OK, 301: Missing param.
-	POST	/subtract	subtracting 2 nums	x:int y:int	200:OK 301: Missing param.
/	POST	/divide	dividing 2 nums	x:int y:int	200 OK 301: Missing param. 302: y is missing
*	POST	/multiply	multiplying 2 nums	x:int y:int	200 OK 301: Missing param.

PROJECT 1, BUILDING RESTFUL API PART 1

Flask ile restful api's oluşturmak için ilk olarak flask üzerinde çalışan restful library'sini yüklememiz gerek:

```
pip3 install flask-restful
```

denemek için terminalde python3 dedikten sonra,

```
import flask_restful as fr
```

diyerek flask_restful'u import ederiz eğer hata almıyorsak paket başarıyla yüklenmiş demektir artık **exit()** diyerek python arayüzünden çıkabiliriz.

Bu paketi install ettikten sonra artık bir önceki kısımda chart'ını oluşturduğumuz api'yı implement etmeye başlayabiliriz.

Buradaki API yapısı daha önce olduğu gibi app.route() fonksiyonu üzerinden dönmeyecek, onun yerine **flask_restful** içerisinden import edilen **Api** ve **Resource** fonksiyonlarından yararlanılacak, temel bir yapı kurulacak.

Her bir resource için, **Resource** class'ından türeyen ayrı bir class oluşturulacak, bunun altında da **get**, **post**, **put**, **delete** gibi methodlar oluşturalabilecek, örneğin Add class'ı altındaki post fonksiyonu bu resource'un endpoint'ine gelen post request'leri handle edecek. Hangi resource class'ının hangi endpoint ile ilişkili olduğunu ise **add_resource()** methodu ile en sonda belirteceğiz.

Şimdi koda geçelim kod üzerinde tekrar açıklamalarda bulunuruz, bu başlık altında sadece add resource'u için işlemleri gerçekleştireceğiz

İlk olarak aşağıdaki importları gerçekleştiriyoruz ve daha sonrasında, flask app'ini oluşturup, bu app'i wrap eden bir Api objesi yaratıyoruz.

```
from flask import Flask, jsonify, request
from flask_restful import Api, Resource

app = Flask(__name__)
api = Api(app)
```

Daha sonra, post edilen verinin valid olup olmadığını kontrol edip sonucunda bir kod return eden checkPostedData isimli bir fonksiyon yaratıyoruz. Unutma bu fonksiyon şuan sadece add resource'u için işlevsel, kalanları daha sonra ekleyecez mantık aynı.

```
#Function to check validity of the posted data:
def checkPostedData(postedData, funcName):

    #Validity check for add function.
    #If x or y is missing then return code 301 else return code 200
    if(funcName == "add"):
        if "x" not in postedData or "y" not in postedData:
            return 301
        else:
            return 200
```

Sıradaki adımda, add resource'unu bir class olarak tanımlıyoruz, bunun için Resource class'ından inherit etmemiz gerekiyor. Her bir resource için, ayrı bir Resource class'ı tanımlayacağımızı unutma şimdilik sadece Add için bu işlemi yapıyoruz.

Bu resource'a farklı şekillerde ulaşılabilir (get, post, put, delete) bu request methodlarının herbirini ayrı ayrı handle edebiliriz bunun için tek yapmamız gereken method ismi ile bir class methodu tanımlamak. Bizim uygulamamızda sadece post requests handle edileceği için post methodu tanımlanmış, bundan başka methodlu request'ler not allowed hatası alacaktır.

```
class Add(Resource):

    def post(self):
        #If the resource Add is requested using POST method:

        #Get posted data:
        postedData = request.get_json()

        #Check the validity of the posted data:
        checkCode = checkPostedData(postedData,"add")


        #If posted data is invalid, return error json
        if checkCode!=200:
            retJson = {
                'Message': 'Value missing',
                'Status Code': checkCode
            }
            return jsonify(retJson)


        #If posted data is valid, continue and execute the required operations:
        x = postedData["x"]
        y = postedData["y"]
        #Convert x and y to int just in case:
        x = int(x)
        y = int(y)
        #Prepare the json to return:
        sum = x+y
        retMap = {
            'Message':sum,
            'Status Code': 200
        }
        return jsonify(retMap)
```

Yukarıdaki kısım zaten self-explainotary ancak ben yine de kısaca özet geçeyim, daha önce olduğu gibi request objesi üzerinden posted data alınıyor, checkPostedData methodu ile ilgili data'nın validity'si kontrol ediliyor, eğer 200 return edilmemişse data valid değil demektir o halde bir hata json'ı return ediyoruz.

Yok eğer data valid ise, add resource'unun gerektirdiği process'i execute ediyoruz ve sonucunda succes json'ını return ediyoruz.

API'ımızın diğer resource'ları için de yukarıdakine benzer işlemler yapacağız, bunlar şimdilik boş, bir sonraki başlıkta altını dolduracağız ancak mantık aynı.

```
class Subtract(Resource):
    pass

class Multiply(Resource):
    pass

class Divide(Resource):
    pass

api.add_resource(Add,"/add")

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

Peki resources ile endpoints'i nerede birbirine bağlıyoruz? Bunu da yukarıda görülen **api.add_resource()** satırında yapıyoruz bu fonksiyonun içerisine verilen resource class'ı ile endpoint artık eşlenmiş oluyoru, server'ın /add endpoint'ine gelecek request'leri artık, Add class'ının methodları handle edecek!

Son olarak da daha önce olduğu gibi, bu app.py dosyası çalıştırıldığında flask app run, debug=True parametresi ile run edilsin dedik.

PROJECT 1, BUILDING RESTFUL API PART 2

Bir önceki başlıkta oluşturmaya çalıştığımız restful api'ın Add resource'unu implement etmiştik, zaten bunu implement ederken, request methodları nasıl handle edebileceğimizi, endpoints ile resource'ları nasıl bağlayabileceğimizi vesaire anlamıştık, yani temel mantığı kavramıştık.

Şimdi diğer üç resource'u da implement edeceğiz, zaten temeli anladığımız için direkt implemantations'ı vereceğim, eğer farklı bir şey olursa not düşeceğim.

İlk değişiklik validity check eden fonksiyonumuzda olacak, bu fonksiyon add, subtract ve multiply resource'ları için aynı işlemi yapacak buna karşın divison için ekstra olarak, y'nin sıfır olup olmadığını check edecek, bu yüzden aşağıdaki gibi şekillenecek.

```python
#Function to check validity of the posted data:
def checkPostedData(postedData, funcName):

    #Validity check for add, subtract and multiply functions
    #If x or y is missing then return code 301 else return code 200
    if(funcName in ["add", "subtract", "multiply"]):
        if "x" not in postedData or "y" not in postedData:
            return 301
        else:
            return 200

    #Validity check for divide function
    #If x or y is missing return 301, if x and y exists but y is zero return 302
    if(funcName in ["divide"]):
        if "x" not in postedData or "y" not in postedData:
            return 301
        elif postedData["y"] == 0:
            return 302
        else:
            return 200
```

Bunun dışında, subtract fonksiyonu aşağıdaki gibi, add'e çok benziyor çok küçük farklar var:

```python
class Subtract(Resource):
    def post(self):
        #Get posted data:
        postedData = request.get_json()

        #Check the validity of the posted data:
        checkCode = checkPostedData(postedData,"subtract")

        #If posted data is invalid, return error json
        if checkCode!=200:
            retJson = {
                'Message': 'Value missing',
                'Status Code': checkCode
            }
            return jsonify(retJson)

        #If posted data is valid, continue and execute the required operations:
        x = postedData["x"]
        y = postedData["y"]
        #Convert x and y to int just in case:
        x = int(x)
        y = int(y)
        #Prepare the json to return:
        diff = x-y
        retMap = {
            'Message':diff,
            'Status Code': 200
        }
        return jsonify(retMap)
```

Multiply için de aynı şey söylenebilir:

```python
class Multiply(Resource):
    def post(self):
        #Get posted data:
        postedData = request.get_json()

        #Check the validity of the posted data:
        checkCode = checkPostedData(postedData,"multiply")

        #If posted data is invalid, return error json
        if checkCode!=200:
            retJson = {
                'Message': 'Value missing',
                'Status Code': checkCode
            }
            return jsonify(retJson)

        #If posted data is valid, continue and execute the required operations:
        x = postedData["x"]
        y = postedData["y"]
        #Convert x and y to int just in case:
        x = int(x)
        y = int(y)
        #Prepare the json to return:
        mult = x*y
        retMap = {
            'Message':mult,
            'Status Code': 200
        }
        return jsonify(retMap)
```

Divide resource'u için daha fazla değişiklik söz konusu bunun temel sebebi, bu resource için y=0 durumunun ayrı bir hata olarak ele alınması gerekmesi, bunun dışında yapı diğer resources ile birebir aynı.

```python
class Divide(Resource):
    def post(self):
        #Get posted data:
        postedData = request.get_json()

        #Check the validity of the posted data:
        checkCode = checkPostedData(postedData,"divide")

        #If missing data return missing data error:
        if checkCode==301:
            retJson = {
                'Message': 'Value missing',
                'Status Code': checkCode
            }
            return jsonify(retJson)

        #If y = 0, return division with 0 not allowed error.
        if checkCode==302:
            retJson = {
                'Message': 'Division with zero is not allowed',
                'Status Code': checkCode
            }
            return jsonify(retJson)

        #If posted data is valid, continue and execute the required operations:
        x = postedData["x"]
        y = postedData["y"]
        #Convert x and y to int just in case:
        x = int(x)
        y = int(y)
        #Prepare the json to return:
        div = x/y
        retMap = {
            'Message':div,
            'Status Code': 200
        }
        return jsonify(retMap)
```

Son olarak, resources ile endpoints'i daha önce yaptığımız gibi set ediyoruz:

```
api.add_resource(Add,"/add")
api.add_resource(Subtract,"/subtract")
api.add_resource(Multiply,"/multiply")
api.add_resource(Divide,"/divide")
```

Böylece dört işlemi yapabilen temel bir restful api'ı implement etmiş olduk.