API tushunchasi. Python dasturlash tilida sodda API yaratamiz.

Mundarija:

Ushbu maqolada sodda va ba'zi misollar tariqasida **API** haqida maksimal darajada tushuncha bermoqchimiz. Maqola so'ngida esa python dasturlash tilida ma'lum bir vazifani bajaruvchi sodda **API** yaratamiz.

Nega bu maqolani yozayapmiz? *Dasturlashni o'rganayotganlar* **API** *haqida tushunchaga ega bo'lishlari va ularga biroz bo'lsada foydam tegishi uchun. Hozirgi vaqtda* **API** *haqida ma'lumotga ega bo'lish uchun o'zbek tilida yozilgan tushunarli maqola topish mushkulroq, shuning uchun o'zbek tilidagi foydali maqolalar sonini oshirish uchun ham yozayapmiz.*

API haqida bilib olishdan oldin **HTTP** nima ekanligini yodga olsak:

HTTP haqida

Siz brauzeringizda `google.com` bo'yicha so'rov yuborasiz , bu so'rov berilgan **URL** manzil bo'yicha serverga yuboriladi(`request`). Undan so'ng server bu so'rovni tahlil qilgan holda sizga javob qaytaradi (`response`). So'rov hamda kelgan javobning formati muhim biz uchun. Bu formatlar esa **HTTP**(**Hyper Text Transfer Protocol**) yordamida aniqlanadi.

Yuqorida aytib o'tganimdek siz **URL** bo'yicha so'rov yuborganingizda, siz tomondan serverga **GET - HTTP metodi orqali ma'lum bir resursga ega bo'lish uchun so'rov** yuboriladi. So'ngra server sizga javob qaytaradi ya'ni **HTML** ko'rinishidagi ma'lumot formatida (ma'lumotlar formati har xil bo'lishi mumkin , ko'p hollarda **HTML** formatda bo'ladi). Brauzer esa javob tariqasida yetib kelgan **HTML** faylni ekraningizga namoyish etadi va natijada sodda ko'rinishdagi veb sahifa yuzaga keladi (**P.S** siz menimcha **HTML** haqida yaxshi bilsangiz kerak).

Misol tariqasida bir holat , siz `uzbekcoders.uz`dan ro'yxatdan o'tayotgan vaqtingizda forma to'ldirasiz:

- *Ism ,familiya va shunga o'xshash ma'lumotlar.*
- *Email pochta*
- *O'qish manzili va hokazolar.*

"Ro'yxatdan o'tish" tugmasi bosilganda siz uzbekcoders.uz saytining serveriga `HTTP - POST metodi orqali` so'rov yuborgan bo'lasiz va siz yuborgan ma'lumotlar server tomonidan tahlil qilinadi. Agar xatolik bo'lmasa ro'yxatdan o'tish muvafaqiyatli amalga oshiriladi. Aksincha bo'lsa , sizga xatolik haqidagi xabari qayd etilad (`response`).

Resurs tushunchasi - Media fayllar (rasm ,video , musiqa va hk.) , Postlar (communitydagi maqolalarni ham misol keltirish mumkin) , Foydalanuvchi haqidagi ma'lumot , Postda qoldirilgan izohlar va shunga o'xshash narsalar bir so'z bilan **resurs** deb yuritiladi.

HTTP Metodlari

HTTP so'rovini amalga oshirish vaqtida **metod**lardan foydalanadi. Qisqa qilib aytganda **metod**lar siz so'rovingiz davomida nimani bajarmoqchi ekanligingizni anglatadi. Quyida eng ko'p qo'llaniladigan **metod**larni keltirib o'taman:

	Metod	Vazifasi
1.	**GET**	*Mavjud resurs haqida ma'lumotga ega bo'lish uchun qo'llaniladi*
2.	**POST**	*Yangi resurs yaratish uchun qo'llaniladi.*
3.	**PUT/PATCH**	*Mavjud resurs haqidagi ma'lumotni yangilash uchun qo'llaniladi.*
4.	**DELETE**	*Mavjud resursni o'chirishda qo'llaniladi*

Community.uzbekcoders.uz *misolida:*

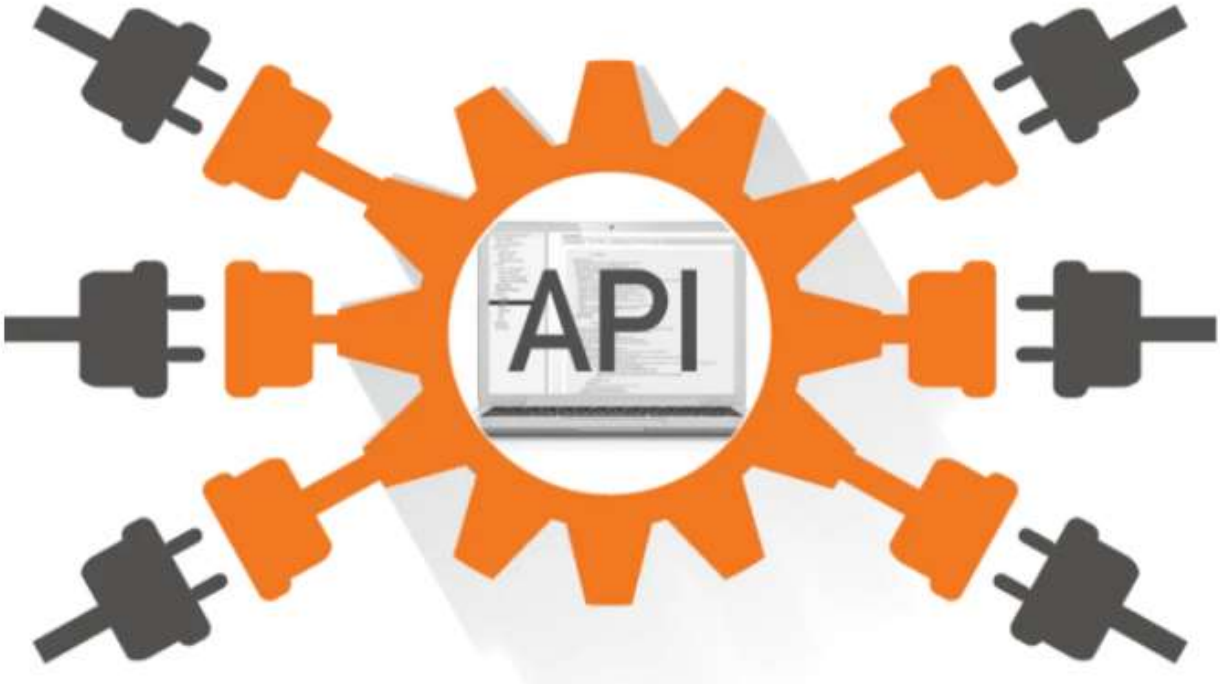
- ***GET:*** *biror-bir maqolani o'qimoqchi bo'lsangiz bu so'rovni yuborasiz. Server sizga mavjud bo'lgan resursni taqdim etiladi.*
- ***POST:*** *Maqola yozmoqchi bo'lsangiz "chop etish" tugmasini bosasiladi shu orqali serverga yangi resurs yaratish uchun so'rov yuborasiz.*
- ***PUT(+PATCH):*** *Maqoladagi ba'zi kamchiliklarni to'g'irlash uchun tahrirlaysiz va serverda mavjud bo'lgan resurs(maqola)ni yangilab qo'yasiz.*
- ***DELETE:*** *Community talablariga javob bermaydigan maqolalar administratorlar tomonidan o'chiriladi hamda bu vaqtida ular tomonidan* ***DELETE*** *metodi orqali*

serverga so'rov yuboriladi yoki bu so'rov maqola muallifi tomonidan yo'llanishi ham mumkin. Natijada ma'lum bir resurs(maqola) sayt serveridan o'chib ketadi.

***HTTP** status kodlari haqida qisqacha avvalgi maqolamda keltirib o'tganman.*

Yuqoridagi sodda misollar tariqasida `HTTP` qanday ishlashi hamda metodlarning vazifasi haqida ma'lumotlarni keltirib o'tdim , endi esa maqolamning asosiy qismiga o'tsam.

API o'zi nima? u o'zi nega kerak?



API *(a`pplication` p`rogramming` i`nterface`)-* *boshqa biror bir ilova ikkinchisi bilan to'g'ridan-to'g'ri muloqot qilishi uchun yaratilgan protseduralar, funksiyalar va klasslardan tashkil topgan katta to'plam.*

To'g'risi **API** nima ekanligini bir qoida bilan tushuntirish biroz mushkul. Tushunishingiz oson bo'lishi uchun ajoyib misol topib qo'yganman.

***API**ga misol qilib - rozetkani olsak.Uning vazifasi elektr tokini boshqa qurilmalarga yetkazish.Unga siz istalgan qurilmani ulab ishlatishingiz mumkin:`Elektr Choynak`,`muzlatgich`,`dazmol` va hokazolar .*

*Yuqoridagi misolda rozetka o'rnida - **API** , Dazmol o'rniga - Mobil ilova , Muzlatgich o'rniga - veb ilova , elektr choynak o'rniga esa - desktop ilovani bemalol misol qilishingiz mumkin va o'z o'zidan **ular ishlashi uchun rozetkaga ya'ni APIga murojaat qilishi lozimligi ma'lum bo'ladi.***

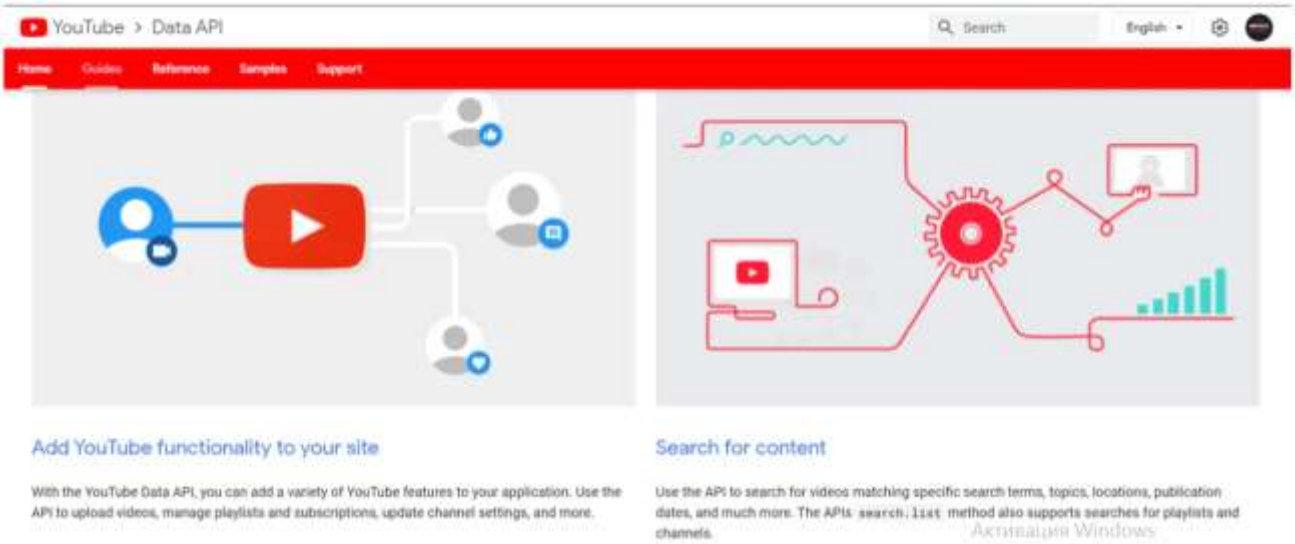
*Endi **API** nima ekanligi biroz bo'lsada ma'lum bo'ldi. Kattaroq loyihalarda **API**ning o'rni o'ta muhim.U `xoh veb ilova , mobil ilova , desktop ilova` bo'lsin - ular bir **API** bilan ma'lumotlar almashib ishlaydi.*

API ilovaning boshqa platformalarda ishlab chiqish jarayonini sezilarli darajada tezlashtiradi.

*Agar **API** mavjud bo'lmasa sizning ilovangiz funksionali cheklangan bo'ladi. Uni boshqa platformalarda ishlab chiqish jarayoni qiyinlashib ketadi.*

*Umuman olganda **API** bu backend dasturlashning asosi. Zamonaviy dasturlashni **API**siz tasavvur qilish o'ta mushkul.*

Har bir katta loyihaning o'z **API**si bo'ladi. Ajoyib funksionalga ega loyihalarda bir emas bir-nechta **API**dan foydalanilganligiga guvoh bo'lamiz. Bu esa ilovaning samaradorligi hamda funksionalining oshishiga katta hissa qo'shadi. Biz bilgan **API**larning ko'pgina qismi `public`**API**hisoblanadi ya'ni bu **API**dan barcha foydalanishi mumkin. Katta kompaniyalar, ijtimoiy tarmoqlar taqdim etgan APIni ham public API deb atashimiz mumkin. Sizning loyihangiz `spotify` bilan ma'lumot almashib ishlasa yoki uning xizmatlaridan foydalansa siz unga o'xshash servisni qaytadan qurishingiz shart emas. Shunchaki uning **API**sidan foydalanasiz. Saytingizga `youtube`ning funksionalini qo'shmoqchi bo'lsangiz , shunchaki uning **API**sidan foydalaning:



https://developers.google.com/youtube/v3

Covid19 bo'yicha statistik ma'lumot kerak bo'lsa buning uchun alohida kuch sarflash shart emas(statistik ma'lumotlarni yetkazib beruvchi loyihani ishga tushurish va hk.) **Covid19** statistikasini yetkazib beruvchi maxsus **API**lar mavjud. Ular yordamida ishingiz ancha yengil bitadi **:-) ,** qolaversa python dasturlash tilida talaygina kutubxonalar ham mavjud.

Foydali havola: https://pypi.org/project/COVID19Py

*Agarda telegram messengeri bizga o'z **API**sini taqdim etmaganida , telegram botlarni yozish bunchalik qulay bo'lmas edi.*

Mutaxassislarning fikricha, loyihalarga funksional qo'shish davomida **API**dan foydalanish eng ma'qul yechimdir.

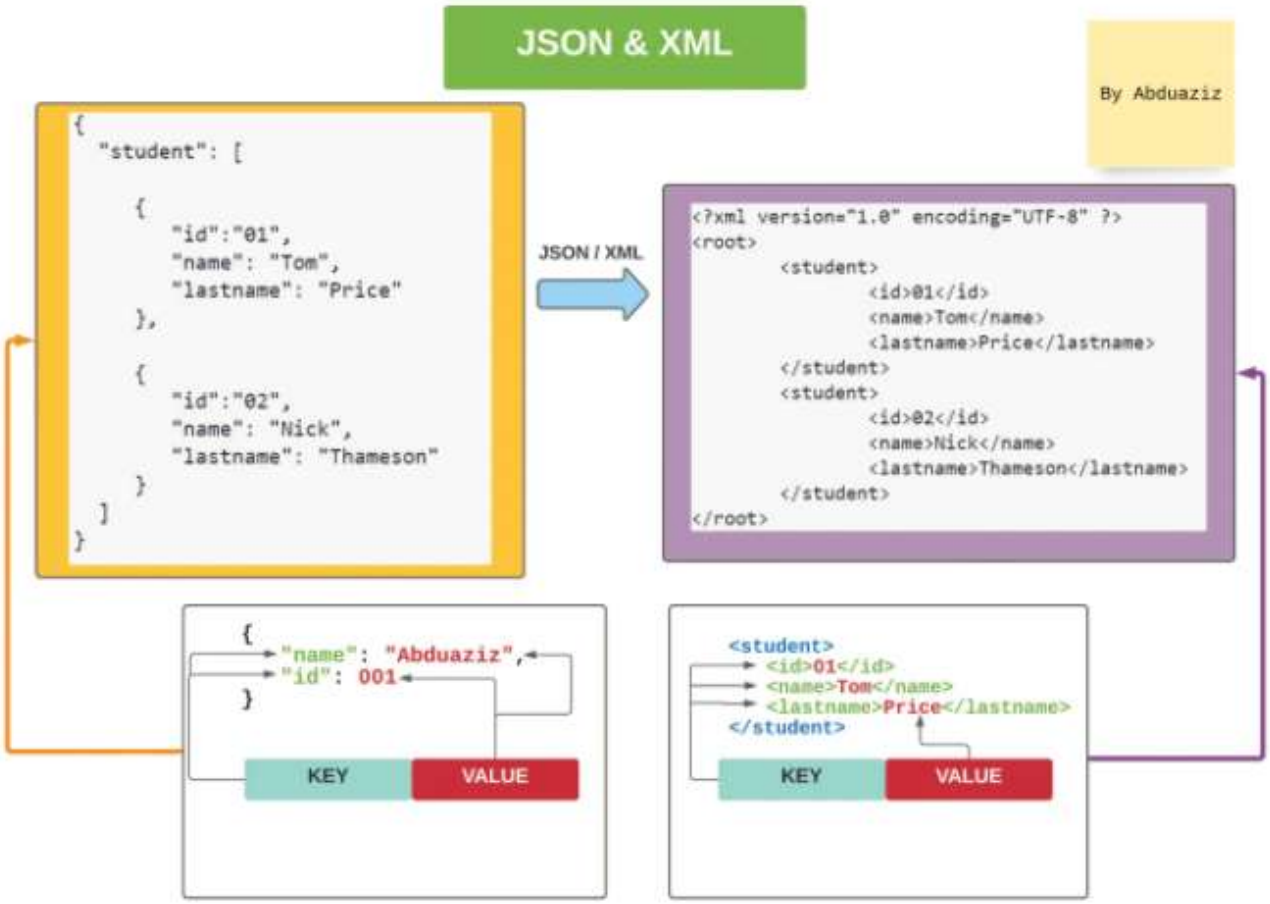
Ma'lumotlar formati haqida qisqacha

Veb servis **API**larida ma'lumotlar formatining asosan ikki turidan foydalanishadi: **Json** & **XML.** Bular ma'lumotlarni qabul qilish hamda yuborish uchun umumiy format ya'ni ma'lumot formati deb yuritiladi. Qisqa qilib aytganda serverlar o'rtasida ma'lumot almashish uchun juda ham qo'l keladi.

JSON (Java**S**cript **O**bject **N**otation**)** bu yuqorida aytib o'tganimdek ma'lumotlar almashinishi uchun ishlatiladigan ma'lumot formati bo'lib, javascript uchun yaratilgan va aynan shu tilda boshqalarga nisbatan kengroq foydalanadi. Ammo boshqa tillarda ham faol ravishda qo'llanilib kelinmoqda. Sintaksisini bir qarashdayoq tushunib olish mumkin.

XML - E**x**tensible **M**arkup **L**anguage (**HTML** bilan o'xshash tarzda yozilgan). Vazifasi xuddi **json** kabi ma'lumotlar tashish, tuzilishini saqlash va ta'riflashdan iborat. Sodda qilib aytganda ular oddiy ma'lumot formati.

Umuman olganda birning kamchiligi ikkinchisining yutug'idir!



Ma'lumotlar formati haqida yaxshiroq tushunishingiz uchun yuqoridagi sxemani taqdim etaman. Unda siz **json** hamda **xml**ni taqqoslashingiz ham mumkin.

Foydali havolalar:

https://www.w3schools.com/xml/default.asp ,
https://www.w3schools.com/js/js_json_intro.asp

REST API tushunchasi.

REST to'liq holatda **RE**`presentational` **S**`tate` **Tr**`ansfer` deb aytiladi. **REST HTTP** protokoli yordamida ilova yoki veb saytni server bilan muloqotga kirishishiga xizmat qiluvchi umumiy prinsipdir (**SOAP** kabi protokol emas). REST termining vujudga kelishida HTTP protokoli asoschilaridan biri Roy Fieldingning hissasi katta (bu termin 2000-yilda vujudga kelgan). REST **RPC**ning alternativi hisoblanadi. Maqolada keltirgan HTTP metodlarni "Rest So'rovlar" deb ham atash mumkin. **REST** `"stili"` **HTTP 1.1** bilan parallel ravishda o'sib kelmoqda. Chunki ikkalasining ham asosini **HTTP 1.0** tashkil etadi.

REST yordamida veb servislar yaratish davomida boshqa termin ya'ni **RESTful** qo'llaniladi.

RESTning afzalliklari:

- Har bir resurs **URL** bo'yicha aniqlanadi. Bu esa **URL** har bir resurs uchun kalit vazifasini o'tashini anglatadi.
- Resurs ustidan boshqaruv yoki ularni nazorati to'liq ravishda ma'lumotlarni uzatish protokoli yordamida amalga oshiriladi. Eng keng tarqalgan protokollardan biri bu albatta - **HTTP.**
- Komponentlarning portativligi.
- Yangilanish yoki yangi zamon talablariga mos ravishda tez va oson o'zgarish xususiyatiga ega.
- O'zgartirish kiritish o'ta yengil.
- Interfeys soddaligi.

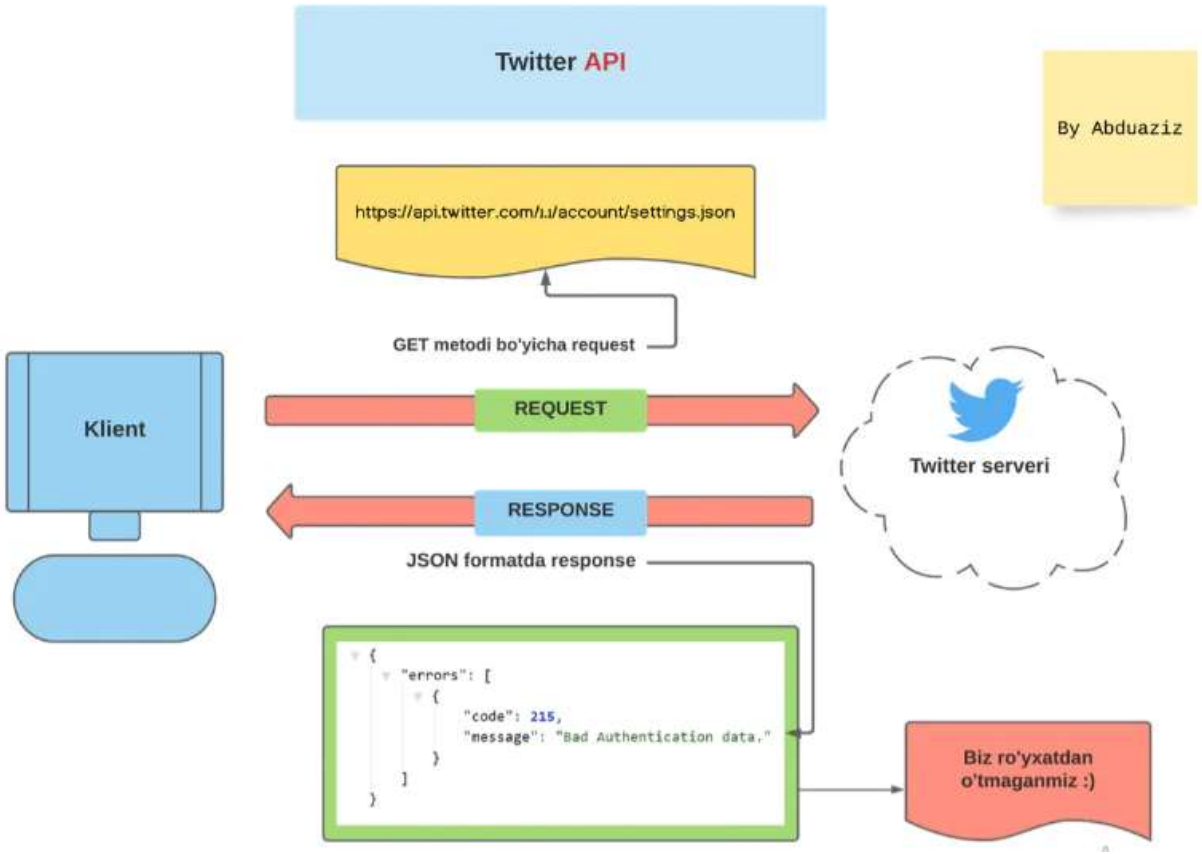
RESTda request & response Json formatda yetkaziladi.

```
GET: https://api.twitter.com/1.1/account/settings.json
{
  "errors":[
      {
          "code":215,
          "message":"Bad Authentication data."
      }
  ]
}
```

*`"https://api.twitter.com/1.1/account/settings.json"` bo'yicha so'rov yuborganimizda server yuqoridagi **response**ni qaytardi. Ya'ni biz **twitter**ga avtorizatsiya bo'lmaganmiz **:)** Shuning uchun server bizga **settings**ga oid ma'lumotni qaytara olmaydi.*

STATUS`bu HTTP status kodi. (200-OK , 404-Not Found ...)`

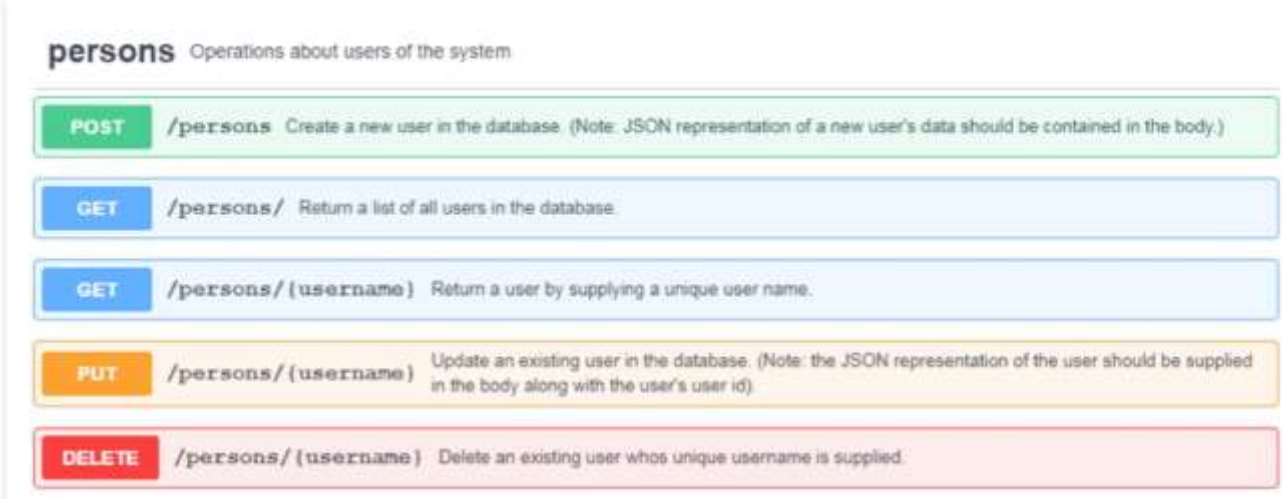
MESSAGE`esa bu response xabari.("Not Found" , "Bad Request")`



Agar e'tibor bergan bo'lsangiz , **iT** sohasida bo'sh ish o'rinlarini taklif qiluvchi vakansiyalarda *"restful API bilan ishlash ko'nikmasi"* ko'plab vakansiyalarda takrorlangan***** Bu degani siz xoh frontend , android , ios va hk. dasturchi bo'ling **API** bilan ishlashni bilishingiz muhim.

*****Опыт проектирования и поддержки клиент-серверного **API**, Понимание **RESTful** API , Опыт работы с протоколами **REST**/JSON **API**, **RESTful API** и HTTP/S va shunga o'xshash talablar.

API endpoint hamda route haqida



- **Route** (yoki marshrut) - bu **API**ning qandaydur qismi nima ish bajarishiga ishora qilib turuvchi nom. Boshqacha qilib aytganda sizning so'rovingiz yuborilayotgan manzil marshrut deb ataladi. Unga HTTP metodlari bo'yicha so'rov yuboriladi. Bir route bir nechta **endpointl**arga ega bo'lishi mumkin.
- **Endpoin**t (yakuniy nuqta **- конечная точка**) - bu marshrutga (route)ga alohida HTTP metodlari yordamida murojaat qilishdir. Har bir endpoint aniq bir masalani hal etadi va ular har bir klientdan (klient haqida o'tgan safargi maqolamda aytib o'tgan edim) parametr(sozlamalar)ni oladi so'ngra unga shu bo'yicha ma'lumotlarni yetkazadi.

Yuqoridagilar haqida tasavvurga ega bo'lish uchun quyidagi misolni ko'rib chiqamiz:

http://example.com/api/v1/users/1

`api/v1/users/1`- Bu marshrut`(route) , api/`esa rest API ga olib boruvchi standart yo'l.

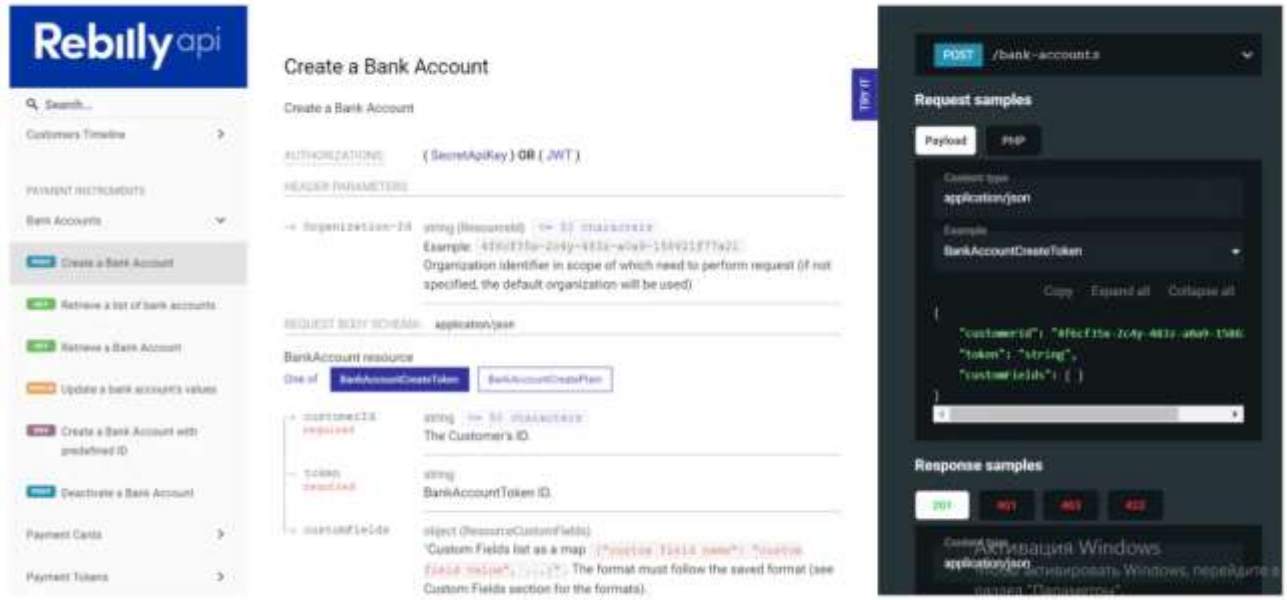
Yuqoridagi marshrut 4 ta endpointga ega deb ayta olamiz:

- **GET** - bu metod bilan murojaat qilinganda **id**si 1ga teng foydalanuvchi haqidagi ma'lumot server tomonidan bizga yuboriladi.
- **PUT / PATCH** - bu metod orqali biz so'rov yuboramiz. Ya'ni **id**si 1ga teng foydalanuvchi haqidagi ma'lumotni yangilash uchun. So'rov `json` formatida yuborilishi mumkin.
- **POST** - bu metod orqali serverga yangi resurs qo'shiladi .`/api/v1/users` ga bu metod bilan murojaat qilinganda yangi foydalanuvchi qo'shish mumkin.
- **DELETE** - bu metod vazifasi nomidan ma'lum. Foydalanuvchi haqidagi ma'lumotni serverdan o'chiradi.

	Metod	URL	Vazifasi
1.	**GET**	`api/v1/users/`	Barcha foydalanuvchilar kolleksiyasi.
2.	**GET**	`api/v1/users/1`	Alohida resursga ega bo'lish uchun (**id**si 1ga teng user).
3.	**POST**	`api/v1/users/`	Yangi foydalanuvchi qo'shish.
4.	**DELETE**	`api/v1/users/1`	**id**si 1ga teng userni o'chirish.
5.	**PUT/PATCH**	`api/v1/users/1`	**id**si 1ga teng user ma'lumotlarini yangilash.

Boshqa dasturlash tillariga nisbatan **python**da ko'proq kod yozaman. Balkim shuning uchundur u menga ko'proq yoqadi hamda qulay. Maqolam davomida esa **python** dasturlash tilida **API** yaratish imkonini beruvchi freymvorklarni keltirib o'tmoqchiman.

API qo'llanmasi (API documentation)



Har bir **API** : qo'llanmaga (boshqacha qilib aytganda documentationga) ega bo'lishi shart. **Buning nima ahamiyati bor?** - qo'llanmada sizning **API**ingiz qanday va nima vaziyatda qo'llanilishi , barcha **endpoinlar** hamda **route**larga yuboriladigan so'rovlar va ularning vazifasi to'liq yoritilgan bo'ladi(ko'p hollarda so'rovlar oldindan tayyorlab qo'yilgan bo'ladi). Bu esa siz o'z **API**ingizni`public API`ko'rinishida namoyish etishingizda katta rol o'ynaydi. Hozirda ko'plab freymvorklar xususan: **Django-rest-framework** hamda **FastAPI** sizning **API**ingiz uchun qo'llanmani o'zi avtomatik tarzda generatisya qilib beradi.

Tavsiya etilgan: Barchamizga tanish **Click UZ**ning **API** qo'llanmasi: https://docs.click.uz/click-api-request/ barchasi to'liq yoritilgan.

Pythonda API yaratish imkoniyatini beruvchi freymvorklar

FastAPI

Bu freymvork oxirgi vaqtlarda o'zimga ham juda ma'qul keldi. Chunki unda API yaratish boshqalariga nisbatan tezroq (nomi bilan **FAST !!!**). Bu freymvorkni yuqori tezlikga ega **HTTP API** serverlarni yaratishda qo'llashyapdi. Bu freymvork **Starlette** asosida qurilgan , validatsiyaga esa **Pydantic** javob beradi. Umuman olganda bu freymvorkni har bir python dasturchi ishlatib ko'rishi zarur. **FastAPI** Python dasturlash tilidagi eng tez backend freymvorklar qatoriga kiradi.(`Django`dan tez)

*P.S **FastAPI** mikrofreymvorklar sirasiga kiradi.*

Men oxirgi kunlarda FastAPI ishlatib kelmoqdaman. Ayni damda jamoamiz bilan birgalikda **Microsoftdagi ML servislarimizda** FastAPIdan foydalanishni reja qilib kelmoqdamiz. **Windows** hamda **Ofis** mahsulotlarimizni ham u bilan integratsiya qilish rejamizda bor.

Kabir Khan - **Microsoft**
Foydali havola: https://fastapi.tiangolo.com/

Flask

Mikrofreymvork

Flask - Python dasturchilar orasida juda ham ommabop bo'lgan freymvork. Githubda 50000dan ortiq **"star"**lari mavjud. Bu fremvorkdan python dasturlash tilida veb ilovalar qurishda keng qo'llaniladi.

So'ngi yillarda **Flask** ko'p yangilandi va juda qulay o'zgartirishlar kiritildi. Bu o'zgarishlar sabab biz uni full stack freymvork deb atashimiz ham mumkin. Uning minimalistikligi ya'ni veb ilovalar qurilishi soddaligi dasturchilar uchun aynan muddao. Maqolam davomida **Python** dasturlash tilidagi **API**ni aynan Flaskda yaratamiz!

Foydali havola: https://flask.palletsprojects.com/en/1.1.x/

Django

Full stack freymvork.

Django - Python dasturlash tilida yaratilgan Full Stack veb freymvork. Imkoniyatli juda ham ko'p , hamda boshqa freymvorklarga nisbatan qulayroq. Ushbu freymvorkni maqolamga bog'laydigan bo'lsak , bu borada Django ancha yetakchi! Django bazasida ishlab chiqilgan **Django REST** freymvorki mavjud. Unda asosan **RESTful API**lar quriladi va bu borada ancha loyihalar amalga oshgan.

Bu borada qiziqishingiz yuqori bo'lsa izohlarda qoldiring , albatta inobatga olaman.

Foydali havolalar:

`Django Rest Framework` - https://github.com/encode/django-rest-framework

`Django` - https://github.com/django/django

Maqolamning amaliy qismiga o'tish vaqti keldi **:)**

API yaratamiz !

Muhitni sozlaymiz.

Buning uchun sizga quyidagilar zarur bo'ladi:

- Kod muharrir (men **visual studio code**ni ishlataman)
- **Postman** - **API** testing uchun ya'ni so'rovlar yuborish uchun. **CURL**ga nisbatan ancha qulay.
- Kompyuteringizga **Python** interpretatori o'rnatilgan bo'lishi lozim.

Yuqoridagilarda sizda kamchilik mavjud bo'lsa quyidagi `community.uzbekcoders.uz`da chop etilgan maqolani o'qing:

Python dasturlarni ishga tushirish.

Kompyuteringizda bo'sh papka hosil qiling va istalgan nom bilan python fayli yarating(`app.py`).

```
$ mkdir project
$ touch app.py
$ code .
```

Endi esa kerakli kutubxonani o'rnatishimiz lozim:

```
$ pip install Flask
```

Muhit sozlandi, endi esa kod yozamiz.

Foydali havola: https://pypi.org/project/Flask/

Boshlang'ich kodlarni kiritamiz

Kutubxonadan zarur bo'lgan class, funksiya va hk.larni import qilamiz:

```
from flask import Flask,request,jsonify
```

- **`Flask`** - bu haqida bilib oldingiz.
- **`Request`**- so'rovlar bilan ishlash uchun o'ta muhim.
- **`Jsonify`** - *.json formatdagi ma'lumotlar bilan ishlash uchun.

Endi esa **Flask** freymvorkidagi birinchi ilovangiz kodini kiritamiz:

```
from flask import Flask,request,jsonify
```

```
app = Flask(__name__)

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

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

`app = Flask(__name__)` **Flask**dan ekzemplyar olgan holda , yangi flask ilova yaratadi hamda uni `app` deb nomlaydi.

`@app.route` –bu esa dekorator. Python dasturlash tilida dekorator tushunchasi mavjud.

Route haqida yuqorida aytib o'tganman , `'/'` bo'yicha o'tganimizda **`api_route()`** funksiyasi ishga tushadi. Ya'ni **Hello World** yozuvi chop etiladi.So'ngi qatordagi kod ilova ishga tushganda qanday funksiya bajarilishini ham aniqlab beradi. Bizda esa python fayl tushganda ilova **debug rejim**da ishga tushishi lozim.

```
$ python app.py
```

Endi esa `http://127.0.0.1:5000/` bo'yicha o'tib natijada guvoh bo'ling.

Biz yaratmoqchi bo'lgan API nima vazifani bajaradi? Bizning API asosan dasturlash tillariga oid ma'lumotlar bilan bog'liq. Har bir dasturlash tilida o'zi **id** siga ega. API yordamida ular haqida ma'lumotga ega bo'lish , yangisini qo'shish ,mavjud bo'lganini yangilash va o'chirish mumkin. Qisqa qilib aytganda sodda **restful api**. Maqolamda **API** yaratish davomida ma'lumotlar omboridan foydalanmay turamiz. Sababi , ma'lumot omborini modellashtirish ham alohida katta mavzu. Ma'lumotlarni shunchaki kodning o'zida saqlab ketamiz:

```
languages = [...]
```

Kodning asosiy qismlarini yozamiz

Dasturlash tillariga oid ma'lumotlarni saqlaymiz:

```
languages = [
    {
        "id":0,
        "author": "Bjarne Stroustrup",
        "language": "C++"
    },
    {
        "id":1,
        "author": "Rasmus Lerdorf",
        "language": "PHP"
    },
    {
        "id":2,
        "author": "Dennis Ritchie",
        "language": "C"
    }
]
```

Endi esa yangi **route** yaratamiz. U orqali barcha dasturlash tillari(resurs) haqidagi ma'lumotni olish hamda serverga biror bir resurs joylash mumkin:

```
@app.route('/languages', methods = ['GET','POST'])
```

```
def get_or_post():
    if request.method == 'GET':
        if len(languages) > 0:
            return jsonify(languages)
        else:
            return jsonify({"Resurslar mavjud emas": True})
```

`/languages` ga faqatgina **GET** hamda **POST** metodi bilan murojaat qilish mumkin. Agar request metodi **GET**ga teng bo'lsa quyidagi funksiya bajariladi:

- `languages` ya'ni ma'lumotlar soni tekshiriladi , agar ma'lumotlar soni 0dan katta bo'lsa ularning barchasi json formatda chop etiladi (`return jsonify(languages)`)
- `languages` da ma'lumot mavjud bo'lmasa ,shunchaki json formatda sodda response qaytaramiz. `Resurslar mavjud emas = Rost` qiymatini oladi.

GET metodi bilan bog'liq vaziyatni hal etdik. Endi esa foydalanuvchi biror bir resursni joylay olishi kerak:

```
    if request.method == 'POST':
        iD = languages[-1]['id']+1
        new_obj = {
            'id': iD,
            'author': request.json['author'],
            'language': request.json['language'],
        }

        languages.append(new_obj)

        return jsonify(languages)
```

Agar request metodi **POST**ga teng bo'lsa quyidagi funksiya bajariladi:

- Yangi obyekt **id**si - `languages`dagi so'ngi obyektning **id**sidan 1 taga ortiq bo'ladi (`iD = languages[-1]['id']+1`)

Yangi obyekt yaratamiz va u quyidagi qiymatlarga ega:

- Yangi obyekt **id**si yuqoridagi **iD** ning qiymatiga teng.
- `'author'` ya'ni dasturlash tili muallifi esa biz json formatda yuborgan so'rovdagi `author`ning qiymatiga teng. `request.json['author']`- bu esa aynan muddao (so'rovdan kelgan **author**ning qiymatini saralab oladi).
- `'language'`da ham yuqoridagi vaziyat.

So'ngra biz yaratgan obyektni `languages`ga qo'shib qo'yamiz(`languages.append(new_obj)`) va **API** foydalanuvchisiga json formatda barcha ma'lumotlarni qaytaramiz.

Python dasturlash tilida yuqorida darajalik bo'lganligi uchun inson tiliga biroz yaqin hamda uni o'qib berish juda oson.

Bir **route** uchun yozilgan umumiy kodimiz(**GET** bilan **POST**ni o'z ichiga olgan):

```
@app.route('/languages', methods = ['GET','POST'])
def get_or_post():
    if request.method == 'GET':
        if len(languages) > 0:
            return jsonify(languages)
        else:
            return jsonify({"Resurslar mavjud emas": True})

    if request.method == 'POST':
        iD = languages[-1]['id']+1
```

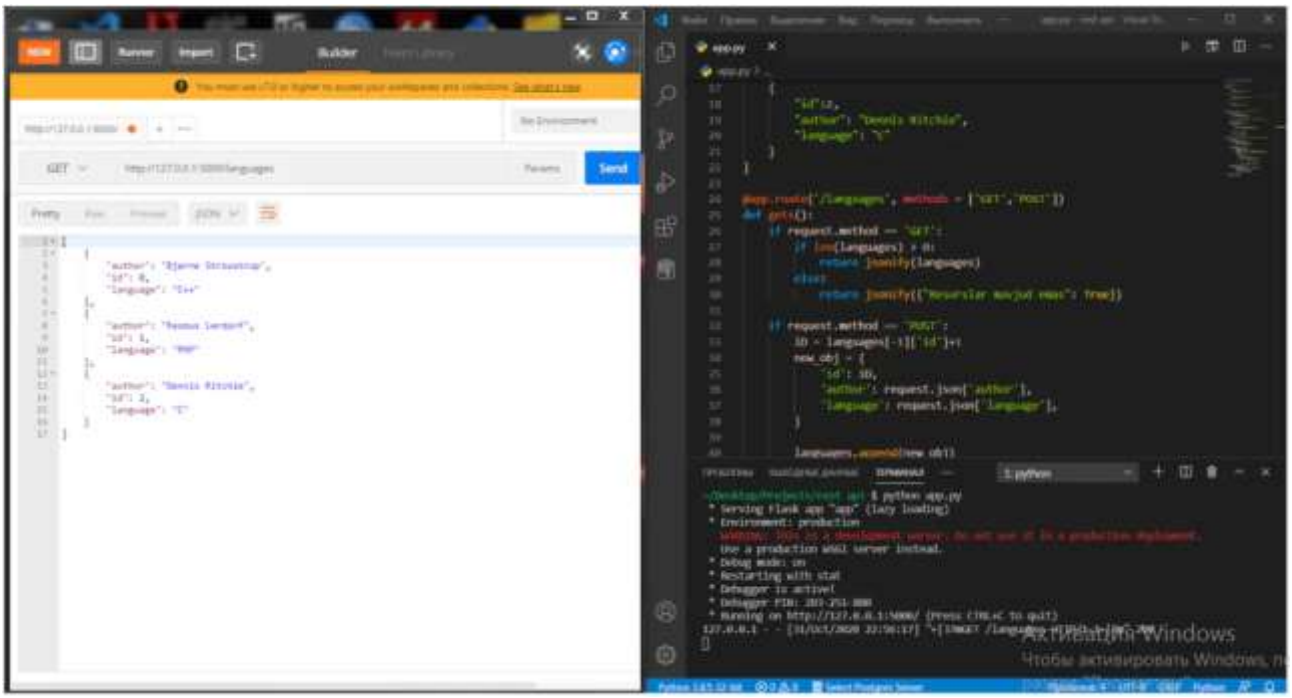
```
        new_obj = {
            'id': iD,
            'author': request.json['author'],
            'language': request.json['language'],
        }

        languages.append(new_obj)

        return jsonify(languages)
```

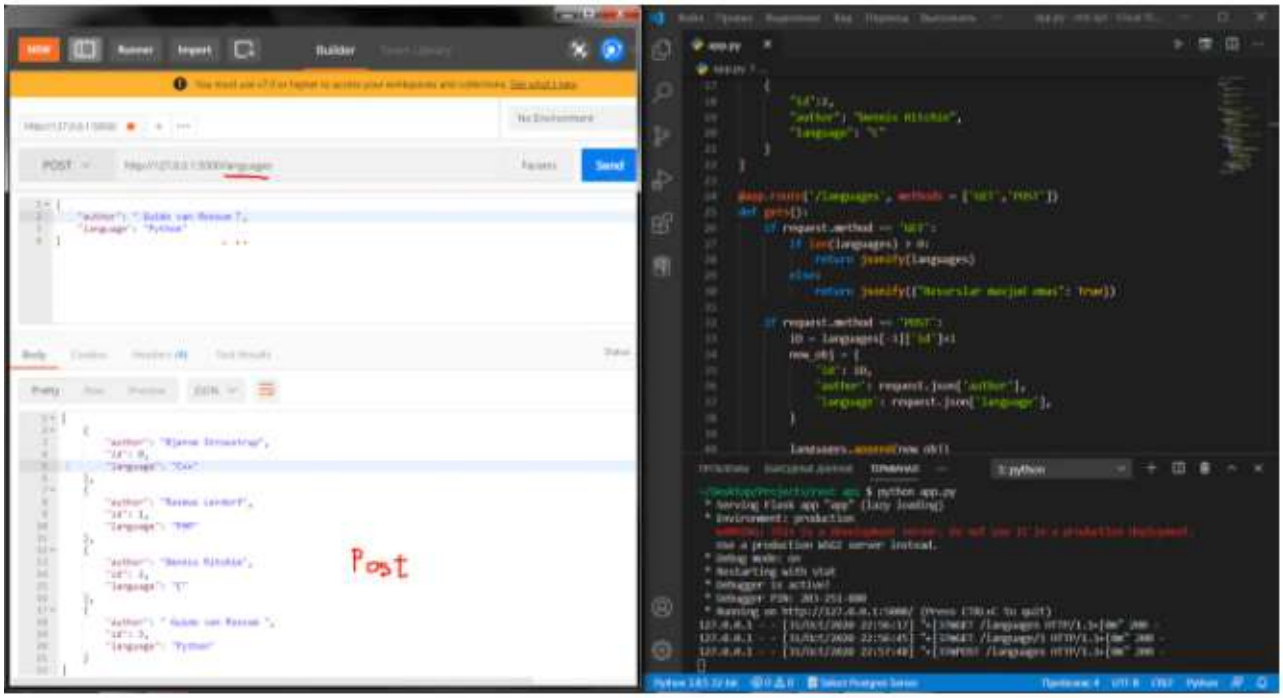
Backend dasturlashda **API testing** tushunchasi bor. Bunda siz yozgan **API** maxsus instrumentlar orqali sinovdan o'tkaziladi. Men bu jarayonda **Postman**dan foydalanaman.

Keling yuqoridagi kodni qanday natija berishini sinab ko'rsak:



GET ALL

http://127.0.0.1:5000/languagesga **Get** metodi orqali so'rov yubordik hamda rasmdagidek ma'lumotlarga ega bo'ldik.



POST

http://127.0.0.1:5000/languagesga **Post** metodi(hamda json formatdagi ma'lumot. Python dasturlash tili haqidagi ma'lumot) orqali murojaat qildik hamda rasmdagidek o'zimiz yo'llagan ma'lumot bilan barcha ma'lumotlarga ega bo'ldik(kodimizda aynan shu narsalar yozilgan edi).

Xo'sh , yuqoridagi yozganlarimiz amalga oshdi. Endi esa ma'lumotni yangilash hamda o'chirish funksiyalarini yozish qoldi xolos.

Eslatib o'taman, bu veb ilovada funksiya nomi unikal bo'lishi lozim.

Yangi route yaratamiz hamda unga murojaat qilish mumkin bo'lgan metodlarni keltiramiz:

```
@app.route('/language/<int:id>', methods=['GET','PUT','DELETE'])
def put_or_delete(id):
    if request.method == 'GET':
        for language in languages:
            if language['id'] == id:
                return jsonify(language)

    if request.method == 'PUT':
        for language in languages:
            if language['id'] == id:
                language['author'] = request.json['author']
                language['language'] = request.json['language']

                updated_language = {
                    'id': id,
                    'author': language['author'],
                    'language': language['language'],
                }

                return jsonify(updated_language)

    if request.method == 'DELETE':
        for index, language in enumerate(languages):
            if language['id'] == id:
```

```
                languages.pop(index)
                return jsonify(languages)
```

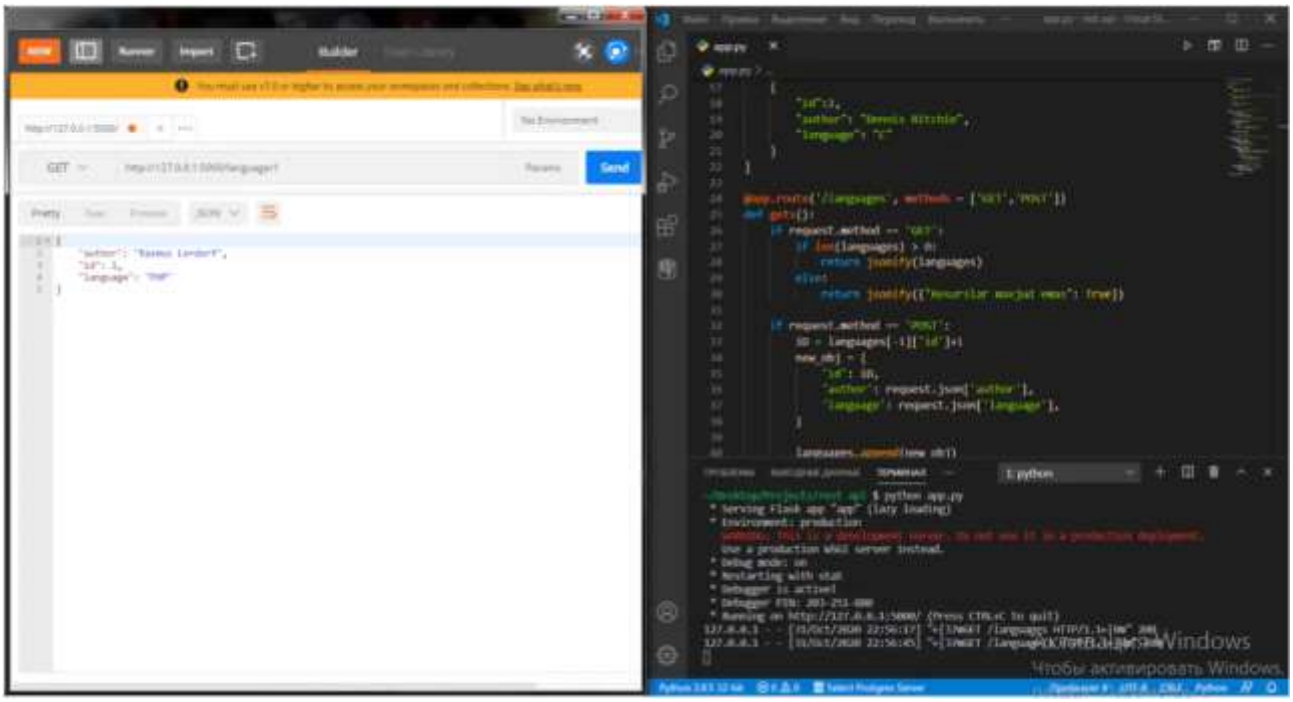
Bu routega esa alohida **id** orqali murojaat qilinadi. Shuning uchun route: `/language/<int:id>`kabi berilgan. `<int:id>` ya'ni **id** har doim son bo'lishi lozim. Alohida route bilan berilishining sababi: biz bu yerda bajarayotgan funksiyalarimiz alohida bir obyekt uchundir ya'ni u haqida alohida ma'lumotga ega bo'lishimiz , uni yangilashimiz va o'chirishimiz kabi funksiyalar. Har bir funksiyani tahlil qilib chiqsak:

- `GET`- alohida obyekt haqidagi ma'lumotni qaytaradi. Funksiya juda oddiy , **id** hamda **GET** metodi orqali murojaat qilinganda `languages` ya'ni ma'lumotlar ichidan mos keluvchi **id**ni izlaydi. Har bir elementni `language`deb belgilab oladi. Agarda `languages`dan olingan `language`ning **id**si murojaat qilingan **id**ga teng bo'lsa: u haqida ma'lumotni **json** formatda qaytariladi.

- `PUT` - alohida obyektni yangilashimiz zarur. Bunda **PUT** metodi hamda **json** formatdagi ma'lumot bilan so'rov yo'llaniladi (ma'lum bir **id**ga). Funksiya yuqoridagidek **id**lar mosligini tekshiradi , agar moslik mavjud bo'lsa: request(so'rov) orqali kelgan ma'lumotlarni avvalgi ma'lumotlarga tenglashtirish orqali ma'lumotni yangilaymiz.`language['author']` bu bizda avval mavjud bo'lgan ma'lumot biz esa uni `request.json['author']`ya'ni **request**dan kelgan ma'lumot bilan tenglashtiryapmiz. Barcha tenglashtirishlardan so'ng `updated_language` nomli obyekt yaratib uni **json** formatida chop etamiz.

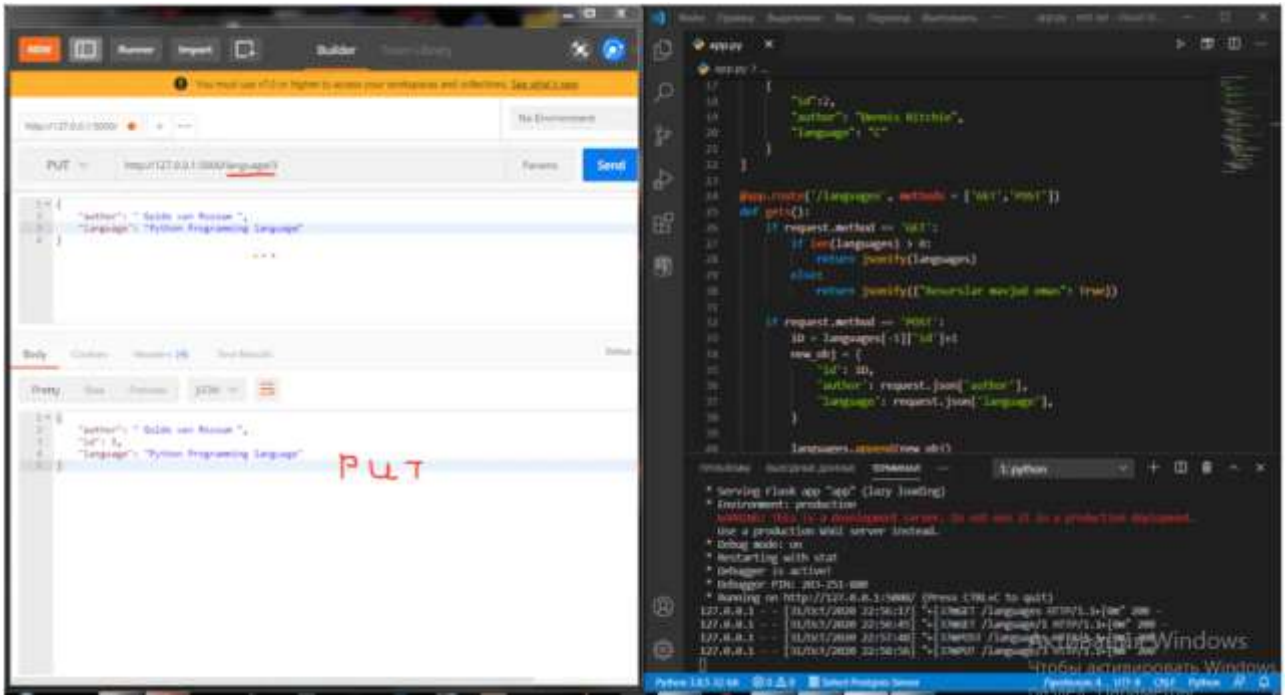
- **DELETE** - alohida obyektni o'chiramiz. Agar **DELETE** metodi bilan ma'lum bir obyektga qarata murojaat qilinsa quyidagi funksiya bajariladi: sodda qilib aytganda`enumerate`orqali languagesdagi ma'lumotlarni raqamlab chiqamiz (**Foydali havola:** https://www.geeksforgeeks.org/enumerate-in-python/). `For` tsikli orqali `language` hamda `index`o'zgaruvchilarining qiymatlariga ega bo'lamiz. Yana `language`ning **id**si yuborilgan so'rovdagi **id** bilan mosligini tekshiramiz. So'ngra `pop()` metodi **index** nomlik argument oladi hamda listdagi ma'lum elementni o'chiradi(https://www.w3schools.com/python/ref_list_pop.asp). Oxirida esa funksiyamiz barcha mavjud ma'lumotlarni **json** formatda qaytaradi.

Endi esa barcha yozganlarimizni birma-bir tekshirib chiqsak:



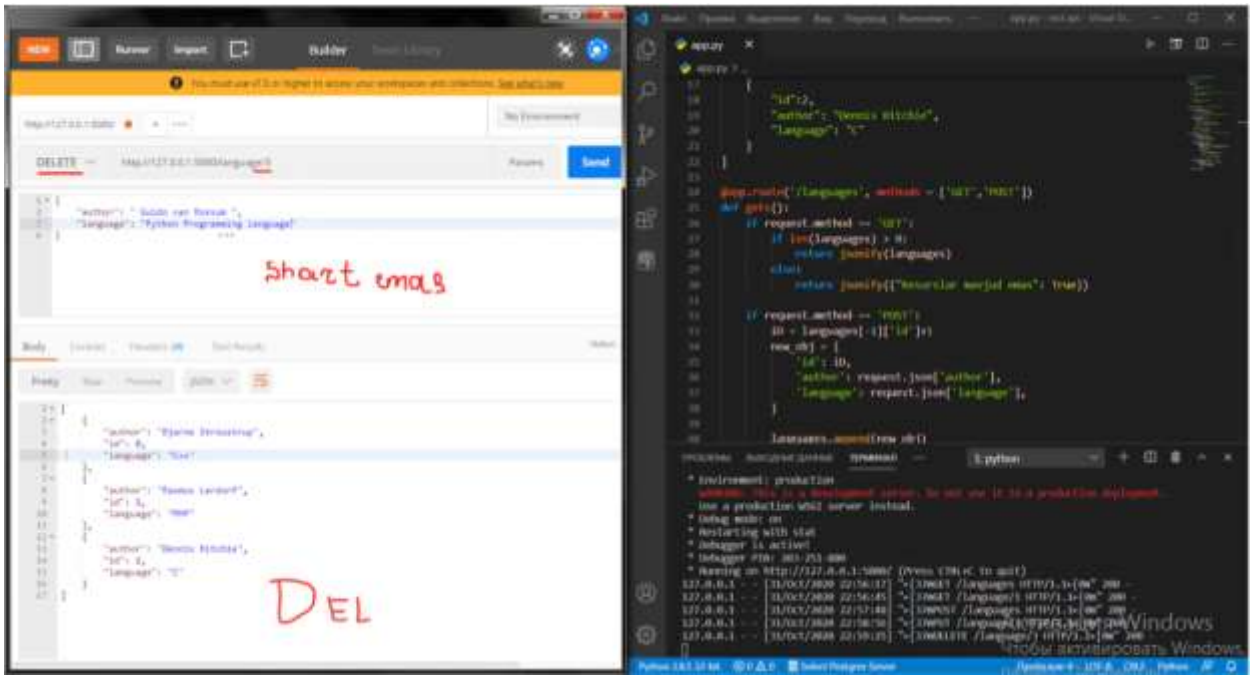
GET smth (single)

idsi 1ga teng obyekt uchun so'rov yubordik hamda **id**si 1ga teng bo'lgan **PHP** obyektiga ega bo'ldik.



PUT smth

idsi 3ga teng bo'lgan Python nomli obyektni yangiladik. Avvalgi ma'lumot "Python" edi. Endi esa "Python Programming Language" ga o'zgardi xolos. **PUT** metodi orqali`languages/3` ga murojaat yo'lladik , response sifatida yangilangan ma'lumotni oldik.



DELETE smth

Biror bir ma'lumotni o'chirishda `request body` talab etilmaydi (ya'ni json formatdagi biror bir ma'lumot). Shunchaki `languages/3` bo'yicha **DELETE** metodi orqali murojaat qilinadi va **id**si 3ga teng bo'lgan element barcha elementlar qatoridan o'chib ketadi.
*Bu amaliy mashg'ulot **API** qanday ishlashini tushunishingiz uchun edi albatta. Kamchiliklari shundaki bu **API** hozircha xatoliklarni boshqara olmaydi , ma'lumotlar omboriga bog'lanmagan(bular o'zi alohida mavzu). Umuman olganda boshida bu kabilarga bosh qotirmasdan sodda tushunchalardan boshlagan ma'qulroq , shundagina asta-sekin samaradorlik ortadi.*

"Pythonni HTMLga qanday ulash mumkin?"

To'g'risi bu kichik mavzuni maqolamga qo'shish fikri Full Stack guruhida shu mavzu bilan bog'liq ko'plab savollar orqali paydo bo'ldi. Savol biroz boshqacharoq tuyulsa ham ancha o'rinli. Shuning uchun bu savolga sodda javobni keltirmoqchiman:

Flask veb freymvorkidan `render_template`ni import qilib olishimiz mumkin. Python dasturlash tilida backend yozishi davomida **HTML** sahifalar bilan ishlaganda shablonizatorlarning roli katta. Bunday shablonizatorlardan biri - **`Jinja2 (Flask bilan birgalikda avtomatik tarzda o'rnatiladi)`**. U bizga **HTML** sahifalar (**templates** deb yuritiladi) bilan ishlashda ancha qo'l keladi. **Jinja template engine** deb ham yuritiladi.

Loyiha papkasida `templates` nomlik yana bir papka hosil qilamiz va uning ichiga `index.html`faylini yaratamiz(Umuman olganda veb sahifa fayllari shu yerda saqlanadi):

```
$ mkdir templates
$ cd templates
$ touch index.html
$ code .
```

`index.html`:

```
<!DOCTYPE html>
<html>
<head>
  <title>Flask</title>
</head>
<body>
   <h1> HTML <-> PYTHON </h1>
</body>
</html>
```

`app.py`:

```
from flask import Flask ,render_template


app = Flask(__name__)

@app.route('/')
def hello():
        """
        do something
        """
    return render_template('index.html')

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

Python faylni shunchaki ishga tushurasiz va **Python & HTML** bir biriga bog'langanligiga guvoh bo'lasiz :**)**

```
$ python app.py
```

Foydali havola: https://flask.palletsprojects.com/en/1.1.x/api/#flask.render_template

Xulosa

Maqolaning oxirigacha o'qigan holda **"API haqida tushunchaga ega bo'ldim"** deb ayta olgan bo'lsangiz , biz ham "maqsadimga erishdim deb ayta olaman". Maqolada maksimal darajadagi sodda misollar hamda terminlardan foydalandik va **backend** dasturlashga oid boshlang'ich tushunchalarni keltirib o'tdik. *O'rganish har doim qiyin bo'lgan , qiyinchilikga duch keldingizmi? - bu degani boshlagan ishingizni tashlab ketish degani emas*. O'rganishdan va izlanishdan charchamang , salomat bo'ling.

Maqola muallifi Abduaziz Ziyodov**.**

E'tiboringiz uchun tashakkur.