URL shortener

Hi,

I am sricharan.I am a 3rd year undergraduate student at JNTUH college of Engineering Jagtial.I am a coding enthusiast and have good skills in python.I am an intermediate in data structures and algorithms.You can look at my resume [here](https://drive.google.com/file/d/1wCjTP4aabBEe_9400LHAYEZvtcE7Xixz/view?usp=sharing).

This is a little intro about me,let’s dive into the problem.

**Basically we will will be provided with a long URL , we need to shorten the URL so that whenever a user access this short URL it should redirect to the long URL**

The statement may be seen as simple but there are many complex issues in it.

The basic idea is we will be as follows,

We will store the long URL along with a randomly generated short string (we will call it slug for now) or algorithmically encoded string in a db instance table.So that when a short URL is hit we can return it’s respective long URL.

**urlshort**(db Table)

|  |  |
| --- | --- |
| **url(character Field)** | **slug(character Field)** |
| https://vodnalasricharan.github.io/ | vhwhdnvgrl |

The factors you need to consider:

1)short URL length

2)RDBMS or NoSQL

3)Encoding Algorithm

Let us discuss one by one:

1)short URL length

The length of the short URL really matters if you have only one server and you have a large amount of users.

Consider average long URL size is 2KB

And the short URL size is 15 bytes for 15 characters.

Suppose we have 30M active users per month then we need 30000000\*2.017=6,05,10,000 ~= 60.5GB per month

Like this very huge amount of data will be collected over the years.

So I restricted the length of short URL to **maximum 15 characters** and encoded only upto **10 characters.**

This will save a lot of space in our db.

2)RDBMS vs NoSQL

In case of heavy traffic NoSQL will perform better than SQL databases.Although we have ACID properties in SQL ,there will be scalability issues when users get increases.

But due to deployment constraints , I am going to use POSTGRESQL in my project.

3)Encoding algorithm

Any algorithm will be fine unless it is easily crackable.In this project I am not going to use any of such complex algorithms. I will use a simple python random function to choose the character.

But it is good to use the MD5 algorithm or any such one.

This will reduce the data redundancy in the database.If the amount of users are high then there is high probability that our short URL gets repeated so we need to check if that short URL is repeated then we need to generate it again until the short URL is unique.

**IMPLEMENTATION**

I have chosen to implement it using the *django rest framework* for implementing the backend.I have designed an API for performing this task.

The basic database table is as follows

class urlshort(models.Model):

url = models.CharField(max\_length=200)

slug = models.CharField(max\_length=15)

def \_\_str\_\_(self):

return self.slug

---->First task is to encode the given long URL into Short URL and insert it into database.

The below code performs that task.

@api\_view(['POST'])

def createurl(request):

if request.method =='POST':

data= JSONParser().parse(request)

try:

shorturl=urlshort.objects.get(url=data['url'])

serializer=urlshorserializer(shorturl)

return Response(serializer.data)

except urlshort.DoesNotExist:

while True:

slug = ''.join(random.choice('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890')

for x in range(10))

try:

ret = urlshort.objects.get(slug=slug)

continue

except urlshort.DoesNotExist:

break

data2=dict()

data2['url']=data['url']

data2['slug']=slug

serializer=urlshorserializer(data=data2)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

*NOTE: We generate the slug until the generated slug is not found in the database*

--->Second task is to return the long URL when short URL is provided

@api\_view(['GET'])

def urlRedirect(request,slugs):

try:

ret = urlshort.objects.get(slug=slugs)

except urlshort.DoesNotExist:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

serializer=urlshorserializer(ret)

return Response(serializer.data['url'])

That’s it our basic building block of our api is finished.This is not easy as discussed above.I just gave a gist of it.

**API GUIDE**

CREATE A SHORT URL

To create a short URL,we need to make a **POST** request at: <https://squat-url.herokuapp.com/api/>

import requests

import json

URL='https://squat-url.herokuapp.com/api/'

def post():

data={

'url':'https://url\_to\_be\_encoded.com'

}

json\_data=json.dumps(data)

r=requests.post(url=URL,data=json\_data)

print(r.json())

post()

Output :

{'url': 'https://url\_to\_be\_encoded.com', 'slug': 'ed7tHbMZfk'}

GET LONG URL

import requests

import json

URL='https://squat-url.herokuapp.com/api/'

def get(slug):

url\_to=URL+str(slug)

r=requests.get(url=url\_to)

print(r.json())

#get('pass-slug-value')

get('ed7tHbMZfk')

Output:

https://url\_to\_be\_encoded.com

**LINKS**

You can check out code at : <https://drive.google.com/drive/folders/1uORbNyKR4FaC8XaLnJuY3lyu9cJRkMS_?usp=sharing>

The API is LIVE at :

<https://squat-url.herokuapp.com/api/>

PS: Access API through python code only,the browsable API interface is not working properly.

THANK YOU

[Vodnala Sricharan](https://www.linkedin.com/in/sricharanvodnala/)