import re

import motor.motor\_asyncio # pylint: disable=import-error

from bot import DB\_URI # pylint: disable=import-error

class Singleton(type):

\_\_instances\_\_ = {}

def \_\_call\_\_(cls, \*args, \*\*kwargs):

if cls not in cls.\_\_instances\_\_:

cls.\_\_instances\_\_[cls] = super(Singleton, cls).\_\_call\_\_(\*args, \*\*kwargs)

return cls.\_\_instances\_\_[cls]

class Database(metaclass=Singleton):

def \_\_init\_\_(self):

self.\_client = motor.motor\_asyncio.AsyncIOMotorClient(DB\_URI)

self.db = self.\_client["Adv\_Auto\_Filter"]

self.col = self.db["Main"]

self.acol = self.db["Active\_Chats"]

self.fcol = self.db["Filter\_Collection"]

self.cache = {}

self.acache = {}

async def create\_index(self):

"""

Create text index if not in db

"""

await self.fcol.create\_index([("file\_name", "text")])

def new\_chat(self, group\_id, channel\_id, channel\_name):

"""

Create a document in db if the chat is new

"""

try:

group\_id, channel\_id = int(group\_id), int(channel\_id)

except:

pass

return dict(

\_id = group\_id,

chat\_ids = [{

"chat\_id": channel\_id,

"chat\_name": channel\_name

}],

types = dict(

audio=False,

document=True,

video=True

),

configs = dict(

accuracy=0.80,

max\_pages=5,

max\_results=50,

max\_per\_page=10,

pm\_fchat=True,

show\_invite\_link=True

) )

async def status(self, group\_id: int):

"""

Get the total filters, total connected

chats and total active chats of a chat

"""

group\_id = int(group\_id)

total\_filter = await self.tf\_count(group\_id)

chats = await self.find\_chat(group\_id)

chats = chats.get("chat\_ids")

total\_chats = len(chats) if chats is not None else 0

achats = await self.find\_active(group\_id)

if achats not in (None, False):

achats = achats.get("chats")

if achats == None:

achats = []

else:

achats = []

total\_achats = len(achats)

return total\_filter, total\_chats, total\_achats

async def find\_group\_id(self, channel\_id: int):

"""

Find all group id which is connected to a channel

for add a new files to db

"""

data = self.col.find({})

group\_list = []

for group\_id in await data.to\_list(length=50): # No Need Of Even 50

for y in group\_id["chat\_ids"]:

if int(y["chat\_id"]) == int(channel\_id):

group\_list.append(group\_id["\_id"])

else:

continue

return group\_list

# Related TO Finding Channel(s)

async def find\_chat(self, group\_id: int):

"""

A funtion to fetch a group's settings

"""

connections = self.cache.get(str(group\_id))

if connections is not None:

return connections

connections = await self.col.find\_one({'\_id': group\_id})

if connections:

self.cache[str(group\_id)] = connections

return connections

else:

return self.new\_chat(None, None, None)

async def add\_chat(self, group\_id: int, channel\_id: int, channel\_name):

"""

A funtion to add/update a chat document when a new chat is connected

"""

new = self.new\_chat(group\_id, channel\_id, channel\_name)

update\_d = {"$push" : {"chat\_ids" : {"chat\_id": channel\_id, "chat\_name" :

channel\_name}}}

prev = await self.col.find\_one({'\_id':group\_id})

if prev:

await self.col.update\_one({'\_id':group\_id}, update\_d)

await self.update\_active(group\_id, channel\_id, channel\_name)

await self.refresh\_cache(group\_id)

return True

self.cache[str(group\_id)] = new

await self.col.insert\_one(new)

await self.add\_active(group\_id, channel\_id, channel\_name)

await self.refresh\_cache(group\_id)

return True

async def del\_chat(self, group\_id: int, channel\_id: int):

"""

A Funtion to delete a channel and its files from db of a chat connection

"""

group\_id, channel\_id = int(group\_id), int(channel\_id) # group\_id and channel\_id Didnt

type casted to int for some reason

prev = self.col.find\_one({"\_id": group\_id})

if prev:

await self.col.update\_one(

{"\_id": group\_id},

{"$pull" :

{"chat\_ids" :

{"chat\_id":

} },

False,

True )

channel\_id }

await self.del\_active(group\_id, channel\_id)

await self.refresh\_cache(group\_id)

return True

return False

async def in\_db(self, group\_id: int, channel\_id: int):

"""

Check whether if the given channel id is in db or not...

"""

connections = self.cache.get(group\_id)

if connections is None:

connections = await self.col.find\_one({'\_id': group\_id})

check\_list = []

if connections:

for x in connections["chat\_ids"]:

check\_list.append(int(x.get("chat\_id")))

if int(channel\_id) in check\_list:

return True

return False

async def update\_settings(self, group\_id: int, settings):

"""

A Funtion to update a chat's filter types in db

"""

group\_id = int(group\_id)

prev = await self.col.find\_one({"\_id": group\_id})

if prev: try:

except Exception as e:

print (e)

return False

print("You Should First Connect To A Chat To Use This Funtion..... 'databse.py/#201' ")

return False

async def update\_configs(self, group\_id: int, configs):

"""

A Funtion to update a chat's configs in db

"""

prev = await self.col.find\_one({"\_id": group\_id})

if prev: try:

except Exception as e:

print (e)

return False

print("You Should First Connect To A Chat To Use This")

return False

async def delete\_all(self, group\_id: int):

"""

A Funtion to delete all documents related to a

chat from db

"""

prev = await self.col.find\_one({"\_id": group\_id})

if prev:

await self.delall\_active(group\_id)

await self.delall\_filters(group\_id)

await self.del\_main(group\_id)

await self.refresh\_cache(group\_id)

return

async def del\_main(self, group\_id: int):

"""

A Funtion To Delete the chat's main db document

"""

await self.col.delete\_one({"\_id": group\_id})

await self.refresh\_cache(group\_id)

return True

async def refresh\_cache(self, group\_id: int):

"""

A Funtion to refresh a chat's chase data

in case of update in db

"""

if self.cache.get(str(group\_id)):

self.cache.pop(str(group\_id))

prev = await self.col.find\_one({"\_id": group\_id})

if prev:

self.cache[str(group\_id)] = prev

return True

# Related To Finding Active Channel(s)

async def add\_active(self, group\_id: int, channel\_id: int, channel\_name):

"""

A Funtion to add a channel as an active chat the a connected group

(This Funtion will be used only if its the first time)

"""

templ = {"\_id": group\_id, "chats":[{"chat\_id": channel\_id, "chat\_name": channel\_name}]}

try:

await self.acol.insert\_one(templ)

await self.refresh\_acache(group\_id)

except Exception as e:

print(e)

return False

return True

async def del\_active(self, group\_id: int, channel\_id: int):

"""

A funtion to delete a channel from active chat colletion in db

"""

templ = {"$pull": {"chats": dict(chat\_id = channel\_id)}}

try:

await self.acol.update\_one({"\_id": group\_id}, templ, False, True)

except Exception as e:

print(e)

pass

await self.refresh\_acache(group\_id)

return True

async def update\_active(self, group\_id: int, channel\_id: int, channel\_name):

"""

A Funtion to add a new active chat to the connected group

"""

group\_id, channel\_id = int(group\_id), int(channel\_id)

prev = await self.acol.find\_one({"\_id": group\_id})

templ = {"$push" : {"chats" : dict(chat\_id = channel\_id, chat\_name = channel\_name)}}

in\_c = await self.in\_active(group\_id, channel\_id)

if prev:

if not in\_c:

await self.acol.update\_one({"\_id": group\_id}, templ)

else:

return False

else:

await self.add\_active(group\_id, channel\_id, channel\_name)

return True

async def find\_active(self, group\_id: int):

"""

A Funtion to find all active chats of

a group from db

"""

if self.acache.get(str(group\_id)):

self.acache.get(str(group\_id))

connection = await self.acol.find\_one({"\_id": group\_id})

if connection:

self.acache[str(group\_id)] = connection

return connection

return False

async def in\_active(self, group\_id: int, channel\_id: int):

"""

A Funtion to check if a chat id is in the active

chat id list in db

"""

prev = await self.acol.find\_one({"\_id": group\_id})

if prev:

for x in prev["chats"]:

if x["chat\_id"] == channel\_id:

return True

return False

return False

async def delall\_active(self, group\_id: int):

"""

A Funtion to Delete all active chats of

a group from db

"""

await self.acol.delete\_one({"\_id":int(group\_id)})

await self.refresh\_acache(group\_id)

return

async def refresh\_acache(self, group\_id: int):

"""

A Funtion to refresh a active chat's chase data

in case of update in db

"""

if self.acache.get(str(group\_id)):

self.acache.pop(str(group\_id))

prev = await self.acol.find\_one({"\_id": group\_id})

if prev:

self.acache[str(group\_id)] = prev

return True

# Related To Finding Filter(s)

async def add\_filters(self, data):

"""

A Funtion to add document as

a bulk to db

"""

try:

await self.fcol.insert\_many(data)

except Exception as e:

print(e)

return True

async def del\_filters(self, group\_id: int, channel\_id: int):

"""

A Funtion to delete all filters of a specific

chat and group from db

"""

group\_id, channel\_id = int(group\_id), int(channel\_id)

try:

await self.fcol.delete\_many({"chat\_id": channel\_id, "group\_id": group\_id})

print(await self.cf\_count(group\_id, channel\_id))

return True

except Exception as e:

print(e)

return False

async def delall\_filters(self, group\_id: int):

"""

A Funtion To delete all filters of a group

"""

await self.fcol.delete\_many({"group\_id": int(group\_id)})

return True

async def get\_filters(self, group\_id: int, keyword: str):

"""

A Funtion to fetch all similar results for a keyowrd

from using text index

"""

achats = await self.find\_active(group\_id)

achat\_ids=[]

if not achats:

return False

for chats in achats["chats"]:

achat\_ids.append(chats.get("chat\_id"))

filters = []

pattern = keyword.lower().strip().replace(' ','.\*')

raw\_pattern = r"\b{}\b".format(pattern)

regex = re.compile(raw\_pattern, flags=re.IGNORECASE)

db\_list = self.fcol.find({"group\_id": group\_id,"file\_name": regex})

for document in await db\_list.to\_list(length=600):

if document["chat\_id"] in achat\_ids:

filters.append(document)

else:

continue

return filters

async def get\_file(self, unique\_id: str):

"""

A Funtion to get a specific files using its

unique id

"""

file = await self.fcol.find\_one({"unique\_id": unique\_id})

file\_id = None

file\_type = None

file\_name = None

file\_caption = None

if file:

file\_id = file.get("file\_id")

file\_name = file.get("file\_name")

file\_type = file.get("file\_type")

file\_caption = file.get("caption")

return file\_id, file\_name, file\_caption, file\_type

async def cf\_count(self, group\_id: int, channel\_id: int):

"""

A Funtion To count number of filter in channel

w.r.t the connect group

"""

return await self.fcol.count\_documents({"chat\_id": channel\_id, "group\_id": group\_id})

async def tf\_count(self, group\_id: int):

"""

A Funtion to count total filters of a group

"""

return await self.fcol.count\_documents({"group\_id": group\_id})

await self.col.update\_one({"\_id": group\_id}, {"$set": {"types": settings}})

await self.refresh\_cache(group\_id)

return True

await self.col.update\_one(prev, {"$set":{"configs": configs}})

await self.refresh\_cache(group\_id)

return True