

```

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": channel_id}}}, False, True)
            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:
                    await self.col.update_one({'_id': group_id}, {"$set": {"types": settings}})
                except Exception as e:
                    print(e)
                    return False
            print("You Should First Connect To A Chat To Use This Funtion..... 'database.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:
                    await self.col.update_one(prev, {"$set": {"configs": configs}})
                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 colleition 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})

```

