import importlib

import time

import re

from sys import argv

from typing import Optional

from ERNEST import (

ALLOW\_EXCL,

CERT\_PATH,

DONATION\_LINK,

LOGGER,

OWNER\_ID,

PORT,

SUPPORT\_CHAT,

TOKEN,

URL,

WEBHOOK,

SUPPORT\_CHAT,

dispatcher,

StartTime,

telethn,

pbot,

updater,

)

# needed to dynamically load modules

# NOTE: Module order is not guaranteed, specify that in the config file!

from ERNEST.modules import ALL\_MODULES

from ERNEST.modules.helper\_funcs.chat\_status import is\_user\_admin

from ERNEST.modules.helper\_funcs.misc import paginate\_modules

from telegram import InlineKeyboardButton, InlineKeyboardMarkup, ParseMode, Update

from telegram.error import (

BadRequest,

ChatMigrated,

NetworkError,

TelegramError,

TimedOut,

Unauthorized,

)

from telegram.ext import (

CallbackContext,

CallbackQueryHandler,

CommandHandler,

Filters,

MessageHandler,

)

from telegram.ext.dispatcher import DispatcherHandlerStop, run\_async

from telegram.utils.helpers import escape\_markdown

def get\_readable\_time(seconds: int) -> str:

count = 0

ping\_time = ""

time\_list = []

time\_suffix\_list = ["s", "m", "h", "days"]

while count < 4:

count += 1

remainder, result = divmod(seconds, 60) if count < 3 else divmod(seconds, 24)

if seconds == 0 and remainder == 0:

break

time\_list.append(int(result))

seconds = int(remainder)

for x in range(len(time\_list)):

time\_list[x] = str(time\_list[x]) + time\_suffix\_list[x]

if len(time\_list) == 4:

ping\_time += time\_list.pop() + ", "

time\_list.reverse()

ping\_time += ":".join(time\_list)

return ping\_time

PM\_START\_TEXT = """

Hi✨

Itz ERNEST Here...

I am a Pro Group Manager😎Which helps to manage big groups easily with my features 😌

•know More about me click on \*\*About Me✨\*\*

• Get Help by Clicking on \*\*HELP\*\* or Use `/help`

"""

buttons = [

[

InlineKeyboardButton(

text="➕️ ADD ME TO YOUR GROUP ➕️", url="t.me/?startgroup=true"),

],

[

InlineKeyboardButton(text="ABOUT Me✨", callback\_data="ernest\_"),

InlineKeyboardButton(text="Help👷", callback\_data="help\_back"),

],

[

InlineKeyboardButton(

text="Dev👨‍💻", callback\_data="source\_"),

InlineKeyboardButton(text="My Owner😎", url="https://t.me/Black\_lord\_on\_fire"),

],

[

InlineKeyboardButton(

text="👥 SUPPORT", url="https://t.me/leavingtgsupport"

),

],

]

HELP\_STRINGS = """

\*Hi Welcome ✨\n\nBelow is my features:\*"""

DONATE\_STRING = """will update shortly ….. please wait"""

IMPORTED = {}

MIGRATEABLE = []

HELPABLE = {}

STATS = []

USER\_INFO = []

DATA\_IMPORT = []

DATA\_EXPORT = []

CHAT\_SETTINGS = {}

USER\_SETTINGS = {}

for module\_name in ALL\_MODULES:

imported\_module = importlib.import\_module("ERNEST.modules." + module\_name)

if not hasattr(imported\_module, "\_\_mod\_name\_\_"):

imported\_module.\_\_mod\_name\_\_ = imported\_module.\_\_name\_\_

if imported\_module.\_\_mod\_name\_\_.lower() not in IMPORTED:

IMPORTED[imported\_module.\_\_mod\_name\_\_.lower()] = imported\_module

else:

raise Exception("Can't have two modules with the same name! Please change one")

if hasattr(imported\_module, "\_\_help\_\_") and imported\_module.\_\_help\_\_:

HELPABLE[imported\_module.\_\_mod\_name\_\_.lower()] = imported\_module

# Chats to migrate on chat\_migrated events

if hasattr(imported\_module, "\_\_migrate\_\_"):

MIGRATEABLE.append(imported\_module)

if hasattr(imported\_module, "\_\_stats\_\_"):

STATS.append(imported\_module)

if hasattr(imported\_module, "\_\_user\_info\_\_"):

USER\_INFO.append(imported\_module)

if hasattr(imported\_module, "\_\_import\_data\_\_"):

DATA\_IMPORT.append(imported\_module)

if hasattr(imported\_module, "\_\_export\_data\_\_"):

DATA\_EXPORT.append(imported\_module)

if hasattr(imported\_module, "\_\_chat\_settings\_\_"):

CHAT\_SETTINGS[imported\_module.\_\_mod\_name\_\_.lower()] = imported\_module

if hasattr(imported\_module, "\_\_user\_settings\_\_"):

USER\_SETTINGS[imported\_module.\_\_mod\_name\_\_.lower()] = imported\_module

# do not async

def send\_help(chat\_id, text, keyboard=None):

if not keyboard:

keyboard = InlineKeyboardMarkup(paginate\_modules(0, HELPABLE, "help"))

dispatcher.bot.send\_message(

chat\_id=chat\_id,

text=text,

parse\_mode=ParseMode.MARKDOWN,

disable\_web\_page\_preview=True,

reply\_markup=keyboard,

)

@run\_async

def test(update: Update, context: CallbackContext):

# pprint(eval(str(update)))

# update.effective\_message.reply\_text("Hola tester! \_I\_ \*have\* `markdown`", parse\_mode=ParseMode.MARKDOWN)

update.effective\_message.reply\_text("This person edited a message")

print(update.effective\_message)

@run\_async

def start(update: Update, context: CallbackContext):

args = context.args

uptime = get\_readable\_time((time.time() - StartTime))

if update.effective\_chat.type == "private":

if len(args) >= 1:

if args[0].lower() == "help":

send\_help(update.effective\_chat.id, HELP\_STRINGS)

elif args[0].lower().startswith("ghelp\_"):

mod = args[0].lower().split("\_", 1)[1]

if not HELPABLE.get(mod, False):

return

send\_help(

update.effective\_chat.id,

HELPABLE[mod].\_\_help\_\_,

InlineKeyboardMarkup(

[[InlineKeyboardButton(text="⬅️ BACK", callback\_data="help\_back")]]

),

)

elif args[0].lower().startswith("stngs\_"):

match = re.match("stngs\_(.\*)", args[0].lower())

chat = dispatcher.bot.getChat(match.group(1))

if is\_user\_admin(chat, update.effective\_user.id):

send\_settings(match.group(1), update.effective\_user.id, False)

else:

send\_settings(match.group(1), update.effective\_user.id, True)

elif args[0][1:].isdigit() and "rules" in IMPORTED:

IMPORTED["rules"].send\_rules(update, args[0], from\_pm=True)

else:

update.effective\_message.reply\_text(

PM\_START\_TEXT,

reply\_markup=InlineKeyboardMarkup(buttons),

parse\_mode=ParseMode.MARKDOWN,

timeout=60,

)

else:

update.effective\_message.reply\_text(

"I'm awake already!\n<b>Haven't slept since:</b> <code>{}</code>".format(

uptime

),

parse\_mode=ParseMode.HTML,

)

def error\_handler(update, context):

"""Log the error and send a telegram message to notify the developer."""

# Log the error before we do anything else, so we can see it even if something breaks.

LOGGER.error(msg="Exception while handling an update:", exc\_info=context.error)

# traceback.format\_exception returns the usual python message about an exception, but as a

# list of strings rather than a single string, so we have to join them together.

tb\_list = traceback.format\_exception(

None, context.error, context.error.\_\_traceback\_\_

)

tb = "".join(tb\_list)

# Build the message with some markup and additional information about what happened.

message = (

"An exception was raised while handling an update\n"

"<pre>update = {}</pre>\n\n"

"<pre>{}</pre>"

).format(

html.escape(json.dumps(update.to\_dict(), indent=2, ensure\_ascii=False)),

html.escape(tb),

)

if len(message) >= 4096:

message = message[:4096]

# Finally, send the message

context.bot.send\_message(chat\_id=OWNER\_ID, text=message, parse\_mode=ParseMode.HTML)

# for test purposes

def error\_callback(update: Update, context: CallbackContext):

error = context.error

try:

raise error

except Unauthorized:

print("no nono1")

print(error)

# remove update.message.chat\_id from conversation list

except BadRequest:

print("no nono2")

print("BadRequest caught")

print(error)

# handle malformed requests - read more below!

except TimedOut:

print("no nono3")

# handle slow connection problems

except NetworkError:

print("no nono4")

# handle other connection problems

except ChatMigrated as err:

print("no nono5")

print(err)

# the chat\_id of a group has changed, use e.new\_chat\_id instead

except TelegramError:

print(error)

# handle all other telegram related errors

@run\_async

def help\_button(update, context):

query = update.callback\_query

mod\_match = re.match(r"help\_module\((.+?)\)", query.data)

prev\_match = re.match(r"help\_prev\((.+?)\)", query.data)

next\_match = re.match(r"help\_next\((.+?)\)", query.data)

back\_match = re.match(r"help\_back", query.data)

print(query.message.chat.id)

try:

if mod\_match:

module = mod\_match.group(1)

text = (

"「 \*HELP FOR\* \*{}\* 」\n".format(

HELPABLE[module].\_\_mod\_name\_\_

)

+ HELPABLE[module].\_\_help\_\_

)

query.message.edit\_text(

text=text,

parse\_mode=ParseMode.MARKDOWN,

disable\_web\_page\_preview=True,

reply\_markup=InlineKeyboardMarkup(

[[InlineKeyboardButton(text="Main Menu", callback\_data="help\_back")]]

),

)

elif prev\_match:

curr\_page = int(prev\_match.group(1))

query.message.edit\_text(

text=HELP\_STRINGS,

parse\_mode=ParseMode.MARKDOWN,

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(curr\_page - 1, HELPABLE, "help")

),

)

elif next\_match:

next\_page = int(next\_match.group(1))

query.message.edit\_text(

text=HELP\_STRINGS,

parse\_mode=ParseMode.MARKDOWN,

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(next\_page + 1, HELPABLE, "help")

),

)

elif back\_match:

query.message.edit\_text(

text=HELP\_STRINGS,

parse\_mode=ParseMode.MARKDOWN,

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(0, HELPABLE, "help")

),

)

# ensure no spinny white circle

context.bot.answer\_callback\_query(query.id)

# query.message.delete()

except BadRequest:

pass

@run\_async

def Ernest\_about\_callback(update: Update, context: CallbackContext):

query = update.callback\_query

if query.data == "ernest\_":

query.message.edit\_text(

text=""" Hi✨ I am ENNEST 😊A Powerfull group manager\n\n•I can maintain your group very well\n•I can protect your group from Scammers\n •You can download songs Via YouTube\n\n||•||↓↓My ideas↓↓||•||\nMY OWNER:- [『мR』╚»𝐁𝐋𝐀𝐂𝐊★𝐋𝐎𝐑𝐃«╝](https://t.me/Black\_lord\_on\_fire)\n [ℂ𝕃𝔸ℕ]( https://t.me/leavingtgsupport)\nThe [update Channel](NOT READY RIGHT NOW)\n My dev is [Alone\_loverboy](https://t.me/Alone\_loverboy)""",

parse\_mode=ParseMode.MARKDOWN,

disable\_web\_page\_preview=false,

reply\_markup=InlineKeyboardMarkup(

[

[

InlineKeyboardButton(text="Back🧑‍🦽", callback\_data="ernest\_back")

]

]

),

)

elif query.data == "ernest\_back":

query.message.edit\_text(

PM\_START\_TEXT,

reply\_markup=InlineKeyboardMarkup(buttons),

parse\_mode=ParseMode.MARKDOWN,

timeout=60,

disable\_web\_page\_preview=False,

)

@run\_async

def Source\_about\_callback(update: Update, context: CallbackContext):

query = update.callback\_query

if query.data == "source\_":

query.message.edit\_text(

text=""" Hi..🤗 Itz ERNEST Here✨

\nHere is the [My Dev](t.me/Alone\_loverboy)\nAnd My Pro Owner:- [『мR』╚»𝐁𝐋𝐀𝐂𝐊★𝐋𝐎𝐑𝐃«╝](https://t.me/Black\_lord\_on\_fire)""",

parse\_mode=ParseMode.MARKDOWN,

disable\_web\_page\_preview=True,

reply\_markup=InlineKeyboardMarkup(

[

[

InlineKeyboardButton(text="Back🧑‍🦽", callback\_data="source\_back")

]

]

),

)

elif query.data == "source\_back":

query.message.edit\_text(

PM\_START\_TEXT,

reply\_markup=InlineKeyboardMarkup(buttons),

parse\_mode=ParseMode.MARKDOWN,

timeout=60,

disable\_web\_page\_preview=False,

)

@run\_async

def get\_help(update: Update, context: CallbackContext):

chat = update.effective\_chat # type: Optional[Chat]

args = update.effective\_message.text.split(None, 1)

# ONLY send help in PM

if chat.type != chat.PRIVATE:

if len(args) >= 2 and any(args[1].lower() == x for x in HELPABLE):

module = args[1].lower()

update.effective\_message.reply\_text(

f"Contact me in PM to get help of {module.capitalize()}",

reply\_markup=InlineKeyboardMarkup(

[

[

InlineKeyboardButton(

text="Help",

url="t.me/{}?start=ghelp\_{}".format(

context.bot.username, module

),

)

]

]

),

)

return

update.effective\_message.reply\_text(

"Contact me in PM to get the list of possible commands.",

reply\_markup=InlineKeyboardMarkup(

[

[

InlineKeyboardButton(

text="Help",

url="t.me/{}?start=help".format(context.bot.username),

)

]

]

),

)

return

elif len(args) >= 2 and any(args[1].lower() == x for x in HELPABLE):

module = args[1].lower()

text = (

"Here is the available help for the \*{}\* module:\n".format(

HELPABLE[module].\_\_mod\_name\_\_

)

+ HELPABLE[module].\_\_help\_\_

)

send\_help(

chat.id,

text,

InlineKeyboardMarkup(

[[InlineKeyboardButton(text="Back", callback\_data="help\_back")]]

),

)

else:

send\_help(chat.id, HELP\_STRINGS)

def send\_settings(chat\_id, user\_id, user=False):

if user:

if USER\_SETTINGS:

settings = "\n\n".join(

"\*{}\*:\n{}".format(mod.\_\_mod\_name\_\_, mod.\_\_user\_settings\_\_(user\_id))

for mod in USER\_SETTINGS.values()

)

dispatcher.bot.send\_message(

user\_id,

"These are your current settings:" + "\n\n" + settings,

parse\_mode=ParseMode.MARKDOWN,

)

else:

dispatcher.bot.send\_message(

user\_id,

"Seems like there aren't any user specific settings available :'(",

parse\_mode=ParseMode.MARKDOWN,

)

else:

if CHAT\_SETTINGS:

chat\_name = dispatcher.bot.getChat(chat\_id).title

dispatcher.bot.send\_message(

user\_id,

text="Which module would you like to check {}'s settings for?".format(

chat\_name

),

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(0, CHAT\_SETTINGS, "stngs", chat=chat\_id)

),

)

else:

dispatcher.bot.send\_message(

user\_id,

"Seems like there aren't any chat settings available :'(\nSend this "

"in a group chat you're admin in to find its current settings!",

parse\_mode=ParseMode.MARKDOWN,

)

@run\_async

def settings\_button(update: Update, context: CallbackContext):

query = update.callback\_query

user = update.effective\_user

bot = context.bot

mod\_match = re.match(r"stngs\_module\((.+?),(.+?)\)", query.data)

prev\_match = re.match(r"stngs\_prev\((.+?),(.+?)\)", query.data)

next\_match = re.match(r"stngs\_next\((.+?),(.+?)\)", query.data)

back\_match = re.match(r"stngs\_back\((.+?)\)", query.data)

try:

if mod\_match:

chat\_id = mod\_match.group(1)

module = mod\_match.group(2)

chat = bot.get\_chat(chat\_id)

text = "\*{}\* has the following settings for the \*{}\* module:\n\n".format(

escape\_markdown(chat.title), CHAT\_SETTINGS[module].\_\_mod\_name\_\_

) + CHAT\_SETTINGS[module].\_\_chat\_settings\_\_(chat\_id, user.id)

query.message.reply\_text(

text=text,

parse\_mode=ParseMode.MARKDOWN,

reply\_markup=InlineKeyboardMarkup(

[

[

InlineKeyboardButton(

text="Back",

callback\_data="stngs\_back({})".format(chat\_id),

)

]

]

),

)

elif prev\_match:

chat\_id = prev\_match.group(1)

curr\_page = int(prev\_match.group(2))

chat = bot.get\_chat(chat\_id)

query.message.reply\_text(

"Hi there! There are quite a few settings for {} - go ahead and pick what "

"you're interested in.".format(chat.title),

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(

curr\_page - 1, CHAT\_SETTINGS, "stngs", chat=chat\_id

)

),

)

elif next\_match:

chat\_id = next\_match.group(1)

next\_page = int(next\_match.group(2))

chat = bot.get\_chat(chat\_id)

query.message.reply\_text(

"Hi there! There are quite a few settings for {} - go ahead and pick what "

"you're interested in.".format(chat.title),

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(

next\_page + 1, CHAT\_SETTINGS, "stngs", chat=chat\_id

)

),

)

elif back\_match:

chat\_id = back\_match.group(1)

chat = bot.get\_chat(chat\_id)

query.message.reply\_text(

text="Hi there! There are quite a few settings for {} - go ahead and pick what "

"you're interested in.".format(escape\_markdown(chat.title)),

parse\_mode=ParseMode.MARKDOWN,

reply\_markup=InlineKeyboardMarkup(

paginate\_modules(0, CHAT\_SETTINGS, "stngs", chat=chat\_id)

),

)

# ensure no spinny white circle

bot.answer\_callback\_query(query.id)

query.message.delete()

except BadRequest as excp:

if excp.message not in [

"Message is not modified",

"Query\_id\_invalid",

"Message can't be deleted",

]:

LOGGER.exception("Exception in settings buttons. %s", str(query.data))

@run\_async

def get\_settings(update: Update, context: CallbackContext):

chat = update.effective\_chat # type: Optional[Chat]

user = update.effective\_user # type: Optional[User]

msg = update.effective\_message # type: Optional[Message]

# ONLY send settings in PM

if chat.type != chat.PRIVATE:

if is\_user\_admin(chat, user.id):

text = "Click here to get this chat's settings, as well as yours."

msg.reply\_text(

text,

reply\_markup=InlineKeyboardMarkup(

[

[

InlineKeyboardButton(

text="Settings",

url="t.me/{}?start=stngs\_{}".format(

context.bot.username, chat.id

),

)

]

]

),

)

else:

text = "Click here to check your settings."

else:

send\_settings(chat.id, user.id, True)

@run\_async

def donate(update: Update, context: CallbackContext):

user = update.effective\_message.from\_user

chat = update.effective\_chat # type: Optional[Chat]

bot = context.bot

if chat.type == "private":

update.effective\_message.reply\_text(

DONATE\_STRING, parse\_mode=ParseMode.MARKDOWN, disable\_web\_page\_preview=True

)

if OWNER\_ID != 254318997 and DONATION\_LINK:

update.effective\_message.reply\_text(

"You can also donate to the person currently running me "

"[here]({})".format(DONATION\_LINK),

parse\_mode=ParseMode.MARKDOWN,

)

else:

try:

bot.send\_message(

user.id,

DONATE\_STRING,

parse\_mode=ParseMode.MARKDOWN,

disable\_web\_page\_preview=True,

)

update.effective\_message.reply\_text(

"I've PM'ed you about donating to my creator!"

)

except Unauthorized:

update.effective\_message.reply\_text(

"Contact me in PM first to get donation information."

)

def migrate\_chats(update: Update, context: CallbackContext):

msg = update.effective\_message # type: Optional[Message]

if msg.migrate\_to\_chat\_id:

old\_chat = update.effective\_chat.id

new\_chat = msg.migrate\_to\_chat\_id

elif msg.migrate\_from\_chat\_id:

old\_chat = msg.migrate\_from\_chat\_id

new\_chat = update.effective\_chat.id

else:

return

LOGGER.info("Migrating from %s, to %s", str(old\_chat), str(new\_chat))

for mod in MIGRATEABLE:

mod.\_\_migrate\_\_(old\_chat, new\_chat)

LOGGER.info("Successfully migrated!")

raise DispatcherHandlerStop

def main():

if SUPPORT\_CHAT is not None and isinstance(SUPPORT\_CHAT, str):

try:

dispatcher.bot.sendMessage(f"@{SUPPORT\_CHAT}", " I AM BACK😈😈☠☠")

except Unauthorized:

LOGGER.warning(

"Bot isnt able to send message to support\_chat, go and check!"

)

except BadRequest as e:

LOGGER.warning(e.message)

test\_handler = CommandHandler("test", test)

start\_handler = CommandHandler("start", start)

help\_handler = CommandHandler("help", get\_help)

help\_callback\_handler = CallbackQueryHandler(help\_button, pattern=r"help\_.\*")

settings\_handler = CommandHandler("settings", get\_settings)

settings\_callback\_handler = CallbackQueryHandler(settings\_button, pattern=r"stngs\_")

about\_callback\_handler = CallbackQueryHandler(Ernest\_about\_callback, pattern=r"ernest\_")

source\_callback\_handler = CallbackQueryHandler(Source\_about\_callback, pattern=r"source\_")

donate\_handler = CommandHandler("donate", donate)

migrate\_handler = MessageHandler(Filters.status\_update.migrate, migrate\_chats)

# dispatcher.add\_handler(test\_handler)

dispatcher.add\_handler(start\_handler)

dispatcher.add\_handler(help\_handler)

dispatcher.add\_handler(about\_callback\_handler)

dispatcher.add\_handler(source\_callback\_handler)

dispatcher.add\_handler(settings\_handler)

dispatcher.add\_handler(help\_callback\_handler)

dispatcher.add\_handler(settings\_callback\_handler)

dispatcher.add\_handler(migrate\_handler)

dispatcher.add\_handler(donate\_handler)

dispatcher.add\_error\_handler(error\_callback)

if WEBHOOK:

LOGGER.info("Using webhooks.")

updater.start\_webhook(listen="0.0.0.0", port=PORT, url\_path=TOKEN)

if CERT\_PATH:

updater.bot.set\_webhook(url=URL + TOKEN, certificate=open(CERT\_PATH, "rb"))

else:

updater.bot.set\_webhook(url=URL + TOKEN)

else:

LOGGER.info("Using long polling.")

updater.start\_polling(timeout=15, read\_latency=4, clean=True)

if len(argv) not in (1, 3, 4):

telethn.disconnect()

else:

telethn.run\_until\_disconnected()

updater.idle()

if \_\_name\_\_ == "\_\_main\_\_":

LOGGER.info("Successfully loaded modules: " + str(ALL\_MODULES))

telethn.start(bot\_token=TOKEN)

pbot.start()

main()