|  |
| --- |
| """ |
|  | telegrambot.py |
|  | Starter code to create a telegram bot using the python-telegram-bot library. |
|  | Includes call to Google Maps API to extract map coordinates from location. |
|  |  |
|  | Author: liuhh02 https://medium.com/@liuhh02 |
|  | """ |
|  |  |
|  | import logging |
|  | import telegram |
|  | from telegram import (ReplyKeyboardMarkup, ReplyKeyboardRemove) |
|  | from telegram.ext import (Updater, CommandHandler, MessageHandler, Filters, |
|  | ConversationHandler) |
|  | from googlemaps import Client as GoogleMaps |
|  | import os |
|  |  |
|  | # Enable logging |
|  | logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', |
|  | level=logging.INFO) |
|  |  |
|  | logger = logging.getLogger(\_\_name\_\_) |
|  |  |
|  | LOCATION, PHOTO, DIET, SERVINGS, TIME, CONFIRMATION = range(6) |
|  |  |
|  | reply\_keyboard = [['Confirm', 'Restart']] |
|  | markup = ReplyKeyboardMarkup(reply\_keyboard, resize\_keyboard=True, one\_time\_keyboard=True) |
|  | TOKEN = 'YOURTELEGRAMBOTTOKEN' |
|  | bot = telegram.Bot(token=TOKEN) |
|  | chat\_id = 'YOURTELEGRAMCHANNEL' |
|  | GMAPSAPI = 'YOURGOOGLEMAPSAPITOKEN' |
|  | gmaps = GoogleMaps(GMAPSAPI) |
|  |  |
|  | PORT = int(os.environ.get('PORT', 5000)) |
|  |  |
|  | def facts\_to\_str(user\_data): |
|  | facts = list() |
|  |  |
|  | for key, value in user\_data.items(): |
|  | facts.append('{} - {}'.format(key, value)) |
|  |  |
|  | return "\n".join(facts).join(['\n', '\n']) |
|  |  |
|  |  |
|  | def start(update, context): |
|  | update.message.reply\_text( |
|  | "Hi! I am your posting assistant to help you advertise your leftover food to reduce food waste. " |
|  | "To start, please type the location of the leftover food.") |
|  | return LOCATION |
|  |  |
|  |  |
|  | def location(update, context): |
|  | user = update.message.from\_user |
|  | user\_data = context.user\_data |
|  | category = 'Location' |
|  | text = update.message.text |
|  | user\_data[category] = text |
|  | logger.info("Location of %s: %s", user.first\_name, update.message.text) |
|  |  |
|  | update.message.reply\_text('I see! Please send a photo of the leftovers, ' |
|  | 'so users will know how the food looks like, or send /skip if you don\'t want to.') |
|  | return PHOTO |
|  |  |
|  |  |
|  | def photo(update, context): |
|  | user = update.message.from\_user |
|  | user\_data = context.user\_data |
|  | photo\_file = update.message.photo[-1].get\_file() |
|  | photo\_file.download('user\_photo.jpg') |
|  | category = 'Photo Provided' |
|  | user\_data[category] = 'Yes' |
|  | logger.info("Photo of %s: %s", user.first\_name, 'user\_photo.jpg') |
|  | update.message.reply\_text('Great! Is the food halal? Vegetarian? Please type in the dietary specifications of the food.') |
|  |  |
|  | return DIET |
|  |  |
|  |  |
|  | def skip\_photo(update, context): |
|  | user = update.message.from\_user |
|  | user\_data = context.user\_data |
|  | category = 'Photo Provided' |
|  | user\_data[category] = 'No' |
|  | logger.info("User %s did not send a photo.", user.first\_name) |
|  | update.message.reply\_text('Is the food halal? Vegetarian? Please type in the dietary specifications of the food.') |
|  |  |
|  | return DIET |
|  |  |
|  |  |
|  | def diet(update, context): |
|  | user = update.message.from\_user |
|  | user\_data = context.user\_data |
|  | category = 'Dietary Specifications' |
|  | text = update.message.text |
|  | user\_data[category] = text |
|  | logger.info("Dietary Specification of food: %s", update.message.text) |
|  | update.message.reply\_text('How many servings are there?') |
|  |  |
|  | return SERVINGS |
|  |  |
|  | def servings(update, context): |
|  | user = update.message.from\_user |
|  | user\_data = context.user\_data |
|  | category = 'Number of Servings' |
|  | text = update.message.text |
|  | user\_data[category] = text |
|  | logger.info("Number of servings: %s", update.message.text) |
|  | update.message.reply\_text('What time will the food be available until?') |
|  |  |
|  | return TIME |
|  |  |
|  | def time(update, context): |
|  | user = update.message.from\_user |
|  | user\_data = context.user\_data |
|  | category = 'Time to Take Food By' |
|  | text = update.message.text |
|  | user\_data[category] = text |
|  | logger.info("Time to Take Food By: %s", update.message.text) |
|  | update.message.reply\_text("Thank you for providing the information! Please check the information is correct:" |
|  | "{}".format(facts\_to\_str(user\_data)), reply\_markup=markup) |
|  |  |
|  | return CONFIRMATION |
|  |  |
|  | def confirmation(update, context): |
|  | user\_data = context.user\_data |
|  | user = update.message.from\_user |
|  | update.message.reply\_text("Thank you! I will post the information on the channel @" + chat\_id + " now.", reply\_markup=ReplyKeyboardRemove()) |
|  | if (user\_data['Photo Provided'] == 'Yes'): |
|  | del user\_data['Photo Provided'] |
|  | bot.send\_photo(chat\_id=chat\_id, photo=open('user\_photo.jpg', 'rb'), |
|  | caption="<b>Food is Available!</b> Check the details below: \n {}".format(facts\_to\_str(user\_data)) + |
|  | "\n For more information, message the poster {}".format(user.name), parse\_mode=telegram.ParseMode.HTML) |
|  | else: |
|  | del user\_data['Photo Provided'] |
|  | bot.sendMessage(chat\_id=chat\_id, |
|  | text="<b>Food is Available!</b> Check the details below: \n {}".format(facts\_to\_str(user\_data)) + |
|  | "\n For more information, message the poster {}".format(user.name), parse\_mode=telegram.ParseMode.HTML) |
|  | geocode\_result = gmaps.geocode(user\_data['Location']) |
|  | lat = geocode\_result[0]['geometry']['location'] ['lat'] |
|  | lng = geocode\_result[0]['geometry']['location']['lng'] |
|  | bot.send\_location(chat\_id=chat\_id, latitude=lat, longitude=lng) |
|  |  |
|  | return ConversationHandler.END |
|  |  |
|  | def cancel(update, context): |
|  | user = update.message.from\_user |
|  | logger.info("User %s canceled the conversation.", user.first\_name) |
|  | update.message.reply\_text('Bye! Hope to see you again next time.', |
|  | reply\_markup=ReplyKeyboardRemove()) |
|  |  |
|  | return ConversationHandler.END |
|  |  |
|  |  |
|  | def error(update, context): |
|  | """Log Errors caused by Updates.""" |
|  | logger.warning('Update "%s" caused error "%s"', update, context.error) |
|  |  |
|  |  |
|  | def main(): |
|  | # Create the Updater and pass it your bot's token. |
|  | # Make sure to set use\_context=True to use the new context based callbacks |
|  | # Post version 12 this will no longer be necessary |
|  | updater = Updater(TOKEN, use\_context=True) |
|  |  |
|  | # Get the dispatcher to register handlers |
|  | dp = updater.dispatcher |
|  |  |
|  | # Add conversation handler with the states GENDER, PHOTO, LOCATION and BIO |
|  | conv\_handler = ConversationHandler( |
|  | entry\_points=[CommandHandler('start', start)], |
|  |  |
|  | states={ |
|  |  |
|  | LOCATION: [CommandHandler('start', start), MessageHandler(Filters.text, location)], |
|  |  |
|  | PHOTO: [CommandHandler('start', start), MessageHandler(Filters.photo, photo), |
|  | CommandHandler('skip', skip\_photo)], |
|  |  |
|  | DIET: [CommandHandler('start', start), MessageHandler(Filters.text, diet)], |
|  |  |
|  | SERVINGS: [CommandHandler('start', start), MessageHandler(Filters.text, servings)], |
|  |  |
|  | TIME: [CommandHandler('start', start), MessageHandler(Filters.text, time)], |
|  |  |
|  | CONFIRMATION: [MessageHandler(Filters.regex('^Confirm$'), |
|  | confirmation), |
|  | MessageHandler(Filters.regex('^Restart$'), |
|  | start) |
|  | ] |
|  |  |
|  | }, |
|  |  |
|  | fallbacks=[CommandHandler('cancel', cancel)] |
|  | ) |
|  |  |
|  | dp.add\_handler(conv\_handler) |
|  |  |
|  | # log all errors |
|  | dp.add\_error\_handler(error) |
|  |  |
|  | updater.start\_webhook(listen="0.0.0.0", port=int(PORT), url\_path=TOKEN) |
|  | updater.bot.setWebhook('https://YOURHEROKUAPPNAME.herokuapp.com/' + TOKEN) |
|  |  |
|  | # Run the bot until you press Ctrl-C or the process receives SIGINT, |
|  | # SIGTERM or SIGABRT. This should be used most of the time, since |
|  | # start\_polling() is non-blocking and will stop the bot gracefully. |
|  | updater.idle() |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |