Пояснительная записка

1. Описание

Uchitelskiy\_bot – это Telegram-бот, который работает наподобие VK, нужен для общения пользователей

2. Функционал

Всего используется 5 функций:

1. Вход/выход/регистрация

Следующие функции нужны для регистрации, входа и выхода соответственно

async def start(update, context):  
 if not (body):  
 await update.message.reply\_text(  
 "Здравствуйте. Для начала войдите в свой аккаунт или создайте новый!\n"  
 "Хотите изучить интерфейс? нажмите на кнопку help или вбейте /help!",  
 reply\_markup=markup)  
 else:  
 await update.message.reply\_text(  
 "Вы уже вошли в свой аккаунт")  
  
  
async def signupper(update,  
 context): # 1 шаг: Начало созания аккаунта, введение имени пользователя(Вход после регистрации выполнятся не будет)  
 global signin  
 signin = []  
 if len(cur.execute(  
 f'SELECT \* FROM accounts WHERE id = {update.message.from\_user.id}').fetchall()) > 0: # Если id уже есть в БД, регистрация невозможна  
 await update.message.reply\_text("Извините, но вы уже зарегистрированы")  
 return ConversationHandler.END  
 await update.message.reply\_text("Введите свое имя:")  
 return 1 # переход на шаг sig\_1  
  
  
async def sig\_1(update, context): # 2 шаг: введение фамилии пользователя  
 global signin  
 signin.append(update.message.text.capitalize()) # Добавляем введенное имя  
 await update.message.reply\_text("Введите свою фамилию:")  
 return 2 # переход на шаг sig\_2  
  
  
async def sig\_2(update, context): # шаг 3: вводим пароль  
 global signin  
 signin.append(update.message.text.capitalize()) # Добавляем введенную фамилию  
 await update.message.reply\_text("Придумайте пароль (длина более 5 символов, должны быть цифры и"  
 " буквы разных регистров, иных символов быть не должно, "  
 "после введения ваше сообщение с паролем удалится, поэтому запомните его сейчас):")  
 return 3 # переход на шаг sig\_3  
  
  
async def sig\_3(update, context):  
 global signin  
 if password\_level(update.message.text) == 'Надежный пароль': # Если пароль прошел проверку  
 await update.message.delete()  
 signin.insert(0, update.message.text) # Добавляем пароль  
 cur.execute(f'CREATE TABLE IF NOT EXISTS accounts '  
 f'(id INTEGER PRIMARY KEY UNIQUE, '  
 f'password TEXT, '  
 f'first\_name TEXT,'  
 f'last\_name TEXT,'  
 f'chat\_id INTEGER);') # Создаем таблицу accounts, если она не существует  
 cur.execute(f"INSERT INTO accounts(id,password,first\_name,last\_name,chat\_id) VALUES(?,?,?,?,?)",  
 (update.message.from\_user.id,  
 \*signin, update.message.chat\_id)) # Создаем в таблице accounts запись об пользователе  
 cur.execute(f'CREATE TABLE IF NOT EXISTS {signin[1].capitalize()}\_{signin[2].capitalize()} '  
 f'(id INTEGER UNIQUE, first\_name TEXT, last\_name TEXT);') # Таблица для контактов - друзей пользователя  
 con.commit()  
 await update.message.reply\_text("Поздравляем с регистрацией!")  
 return ConversationHandler.END  
 else:  
 await update.message.reply\_text(f'{password\_level(update.message.text)} Введите новый пароль')  
 return 3  
  
  
async def loginner(update, context): # Вход в аккаунт, стартовая точка  
 global body  
 if not (body):  
 res = cur.execute(f'SELECT \* FROM accounts WHERE id = {update.message.from\_user.id}').fetchall()  
 if len(res) == 0:  
 await update.message.reply\_text("Вы еще не зарегистрированы! Пожалуйста, зарегистрируйтесь!")  
 return -1  
 await update.message.reply\_text("Пожалуйста, введите ваш пароль:",  
 reply\_markup=ReplyKeyboardRemove()) # Просьба ввести пароль,  
 # удаляем клавиатуру  
 return 1 # переход на шаг login  
 else:  
 await update.message.reply\_text('Вход уже выполнен!')  
 return ConversationHandler.END  
  
  
async def login(update, context): # Сам процесс входа  
 global body  
 res = cur.execute(f"SELECT first\_name, last\_name FROM accounts "  
 f"WHERE password='{update.message.text}'").fetchall() # Проверяем введенный пароль  
 if len(res) == 0:  
 await update.message.reply\_text(f'Неверный пароль! Поменяйте его или введите заново')  
 return 1  
 else:  
 await update.message.reply\_text(f'С возвращением, {res[0][0]} {res[0][1]}', reply\_markup=mrkup)  
 await update.message.delete() # Удаляем сообщение с правильным паролем  
 body = (res[0][0], res[0][1]) # Вход выполнен  
 return ConversationHandler.END

async def exitor(update, context): # Выполнить выход  
 global body  
 body = 0 # Выход выполнен  
 await update.message.reply\_text('Выход выполнен. До свидания!',  
 reply\_markup=ReplyKeyboardMarkup([['/start'], ['/help']], one\_time\_keyboard=False))  
 return ConversationHandler.END # Если был активен диалог, он завершается

start – начинаем диалог. Предлагает войти или зарегистрироваться

signupper, sig\_1, sig\_2, sig\_3 – регистрация(signupper – точка начала регистрации; sig\_1, sig\_2, sig\_3 – этапы регистрации)

loginner, login – вход в с систему(loginner – точка начала входа, login – обработка пароля)

exitor – выход из системы

2. Запрос на дружбу

Следующие функции нужны для отправки запроса дружбы(данные заносятся в базу данных в таблицу пользователя-отправителя и пользователя-получателя)

async def friender(update, context): # Отправим контакту запрос на дружбу  
 global body  
 if not (body):  
 await update.message.reply\_text("Для начала войдите в аккаунт!")  
 return ConversationHandler.END  
 await update.message.reply\_text('Отправьте имя и фамилию пользователя, с кем хотите дружить (через пробел)',  
 reply\_markup=ReplyKeyboardMarkup([['/cancel'], ['/exit']], one\_time\_keyboard=False))  
 return 1  
  
  
async def friend\_1(update, context):  
 global body  
 text = list(update.message.text.split())  
 res = cur.execute('SELECT chat\_id FROM accounts WHERE first\_name=? and last\_name=?',  
 (text[0], text[1])).fetchall() # Проверим существование аккаунта  
 if len(res) == 0:  
 await update.message.reply\_text('Такого пользователя нет! Попробуйте еще раз!')  
 return 1  
 else:  
 otp = cur.execute(f'SELECT id, first\_name, last\_name FROM accounts '  
 f'WHERE chat\_id = {res[0][0]}').fetchall()  
 params = {  
 'chat\_id': res[0][0],  
 'text': f"Вам отправлен запрос на дружбу от {body[0]} {body[1]}"  
 }  
 cur.execute(f'INSERT INTO {body[0]}\_{body[1]}(id, first\_name, last\_name) VALUES(?,?,?);',  
 \*otp) # В таблицу отправителя добавляется получатель  
 cur.execute(f'INSERT INTO {otp[0][1]}\_{otp[0][2]}(id, first\_name, last\_name) VALUES(?,?,?);',  
 (update.message.from\_user.id, \*body)) # В таблицу получателя добавляется отправитель  
 con.commit()  
 response = requests.get(f'https://api.telegram.org/bot{TOKEN}/sendMessage',  
 params=params) # Отправим сообщение получателю  
 await update.message.reply\_text('Запрос отправлен!')  
 return ConversationHandler.END

Здесь friender – точка начала процесса запроса на дружбу, friend\_1 – обработка запроса

3. Отправить сообщение

Функции берут данные из базы данных(таблица с аккаунтами) и с помощью этих данных и модуля requests отправляют сообщение получателю

sync def send(update, context): # Отправим сообщение своему контакту  
 global body  
 if not (not (body)):  
 res = cur.execute(  
 f'SELECT first\_name, last\_name FROM {body[0]}\_{body[1]}').fetchall() # Получаем список контактов  
 if len(res) > 0: # Есть ли вообще контакты  
 buttons = [[' '.join(x)] for x in res]  
 buttons.append(['/cancel'])  
 buttons.append(['/exit'])  
 contacts = ReplyKeyboardMarkup(buttons, one\_time\_keyboard=True)  
 await update.message.reply\_text(  
 "Выберите пользователя, которому хотите отправить сообщение:", reply\_markup=contacts)  
 return 1  
 else: # Если нет, завершаем диалог  
 await update.message.reply\_text('У вас нет друзей!')  
 return ConversationHandler.END  
 else:  
 await update.message.reply\_text(  
 "Для начала войдите в аккаунт!")  
 return ConversationHandler.END  
  
  
async def idir(update, context): # Проверка корректности аккаунта и ввод сообщения  
 global idk # Сюда запишем id чата бота и получателя  
 first\_name, last\_name = update.message.text.split()  
 res = cur.execute(f'SELECT chat\_id FROM accounts '  
 f'WHERE first\_name=? AND last\_name=?',  
 (first\_name, last\_name)).fetchall() # Нужно будет для сообщения  
 if len(res) == 0: # Нет такого пользователя  
 await update.message.reply\_text('Такого пользователя нет! Попробуйте еще раз')  
 return 1  
 else:  
 idk = res[0][0]  
 await update.message.reply\_text('Введите ваше сообщение:',  
 reply\_markup=ReplyKeyboardMarkup([['/cancel'], ['/exit']],  
 one\_time\_keyboard=False))  
 return 2  
  
  
async def is\_sent(update, context): # Отправляем сообщение  
 global idk  
 text = f'Вам сообщение от {body[0]} {body[1]}:\n{update.message.text}'  
 params = {  
 'chat\_id': idk,  
 'text': text  
 }  
 response = requests.get(f'https://api.telegram.org/bot{TOKEN}/sendMessage', params=params)  
 await update.message.reply\_text("Все отправлено!", reply\_markup=mrkup)

Здесь send – точка начала процесса отправки сообщения, idir – вбивается сообщение, is\_sent – процедура отправки сообщения

4. Отправить анонимное сообщение

Функции берут данные из базы данных(таблица с аккаунтами) и с помощью этих данных и модуля requests отправляют анонимное сообщение получателю

# Следующие 3 команды работают аналогично 3 предыдущим, но для анонимного чата  
async def anonym\_send(update, context): # Отправим сообщение своему контакту  
 global body  
 if not (not (body)):  
 res = cur.execute(f'SELECT first\_name, last\_name FROM {body[0]}\_{body[1]}').fetchall()  
 if len(res) > 0:  
 buttons = [[' '.join(x)] for x in res]  
 buttons.append(['/cancel'])  
 buttons.append(['/exit'])  
 contacts = ReplyKeyboardMarkup(buttons, one\_time\_keyboard=True)  
 await update.message.reply\_text(  
 "Выберите пользователя, которому хотите отправить сообщение:", reply\_markup=contacts)  
 return 1  
 else:  
 await update.message.reply\_text('У вас нет друзей!')  
 return ConversationHandler.END  
 else:  
 await update.message.reply\_text(  
 "Для начала войдите в аккаунт!")  
 return ConversationHandler.END  
  
  
async def anonym\_idir(update, context):  
 global idk  
 first\_name, last\_name = update.message.text.split()  
 res = cur.execute(f'SELECT chat\_id FROM accounts '  
 f'WHERE first\_name=? AND last\_name=?', (first\_name, last\_name)).fetchall()  
 if len(res) == 0:  
 await update.message.reply\_text('Такого пользователя нет! Попробуйте еще раз')  
 return 1  
 else:  
 idk = res[0][0]  
 await update.message.reply\_text('Введите ваше сообщение:',  
 reply\_markup=ReplyKeyboardMarkup([['/cancel'], ['/exit']],  
 one\_time\_keyboard=False))  
 return 2  
  
  
async def anonym\_is\_sent(update, context):  
 global idk  
 text = f'Вам сообщение от анонимуса:\n"{update.message.text}"'  
 params = {  
 'chat\_id': idk,  
 'text': text  
 }  
 response = requests.get(f'https://api.telegram.org/bot{TOKEN}/sendMessage', params=params)  
 await update.message.reply\_text("Все отправлено!", reply\_markup=mrkup)

5. Показать друзей

Показывается список друзей пользователя (единственная функция show\_friends)

async def show\_friends(update, context): # Показать список друзей  
 global body  
 if not (body): # Если вход не выполнен  
 await update.message.reply\_text('Для начала войдите в аккаунт!')  
 return 0  
 res = cur.execute(f"SELECT first\_name, last\_name FROM {body[0]}\_{body[1]}").fetchall()  
 if len(res) > 0:  
 text = '\n'.join(list(' '.join(res)))  
 await update.message.reply\_text(f'Ваш список друзей:\n{text}')  
 else:  
 await update.message.reply\_text('У вас нет друзей!')

6. Основное тело(функция main)

main создает каркас приложения

def main():  
 application = Application.builder().token(TOKEN).build()  
 # Строим тг-бота  
 conv\_handler = ConversationHandler(  
 entry\_points=[CommandHandler('start', start)],  
 states={  
 1: [CommandHandler('signup', signupper)],  
 2: [CommandHandler('login', loginner)]  
 },  
 fallbacks=[CommandHandler('cancel', cancel)]  
 )  
 # Начальный диалог для регистрации/входа в аккаунт  
 log = ConversationHandler(  
 entry\_points=[CommandHandler('login', loginner)],  
 states={  
 1: [MessageHandler(filters.TEXT & ~filters.COMMAND, login)]  
 },  
 fallbacks=[CommandHandler('cancel', cancel)]  
 )  
 # Диалог для входа в аккаунт  
 sig = ConversationHandler(entry\_points=[CommandHandler('signup', signupper)],  
 states={  
 1: [MessageHandler(filters.TEXT & ~filters.COMMAND, sig\_1)],  
 2: [MessageHandler(filters.TEXT & ~filters.COMMAND, sig\_2)],  
 3: [MessageHandler(filters.TEXT & ~filters.COMMAND, sig\_3)]  
 },  
 fallbacks=[CommandHandler('cancel', cancel)])  
 # Диалог для регистрации аккаунта  
 friends = ConversationHandler(  
 entry\_points=[CommandHandler('send\_friend', friender)],  
 states={  
 1: [MessageHandler(filters.TEXT & ~filters.COMMAND, friend\_1)]  
 },  
 fallbacks=[CommandHandler('cancel', cancel)]  
 )  
 # Диалог для отправки запроса в друзья  
 sender = ConversationHandler(  
 entry\_points=[CommandHandler('send', send)],  
 states={  
 1: [MessageHandler(filters.TEXT & ~filters.COMMAND, idir)],  
 2: [MessageHandler(filters.TEXT & ~filters.COMMAND, is\_sent)]  
 },  
 fallbacks=[CommandHandler('cancel', cancel)]  
 )  
 # Диалог для отправления сообщения контакту  
 anonym\_sender = ConversationHandler(  
 entry\_points=[CommandHandler('anonym\_send', anonym\_send)],  
 states={  
 1: [MessageHandler(filters.TEXT & ~filters.COMMAND, anonym\_idir)],  
 2: [MessageHandler(filters.TEXT & ~filters.COMMAND, anonym\_is\_sent)]  
 },  
 fallbacks=[CommandHandler('cancel', cancel)]  
 )  
 # Анонимный диалог  
 # Добавляем диалоги  
 application.add\_handler(conv\_handler)  
 application.add\_handler(log)  
 application.add\_handler(sig)  
 application.add\_handler(sender)  
 application.add\_handler(friends)  
 application.add\_handler(anonym\_sender)  
 # Добавляем стартовые точки/точки стопа  
 application.add\_handler(CommandHandler('login', loginner))  
 application.add\_handler(CommandHandler('signup', signupper))  
 application.add\_handler(CommandHandler('exit', exitor))  
 application.add\_handler(CommandHandler('friends', show\_friends))  
 application.add\_handler(CommandHandler('help', helper))  
 application.run\_polling()

3. Используемые модули и классы

import sqlite3  
from telegram.ext import Application, MessageHandler, filters  
from telegram.ext import CommandHandler, ConversationHandler  
from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove  
import requests

sqlite3 – для работы с базой данных

telegram – для создания интерфеса телеграм-бота

requests – для отправки сообщения пользователю

4. Системные требования

|  |  |
| --- | --- |
| 64-битная операционная система Microsoft Windows | * Windows 10 |
| Компьютер и процессор | * 1,6 ГигаГерц (ГГц) или быстрее * 2 или больше ядер |
| Память | * 6 Мбайт оперативной памяти |
| Дисплей | * Разрешение 1280 x 720 или выше |

5. Список используемой литературы

* <https://tlgrm.ru/docs/bots>
* [https://python-course.readthedocs.io](https://python-course.readthedocs.io/)