**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «ПиКЯП (Парадигмы и конструкции языков программирования)»

Отчет по лабораторной работе №5-6

«**Разработка простого бота для Telegram с использованием языка Python/ Разработка бота на основе конечного автомата для Telegram с использованием языка Python**»

*Засчитывается за 2 выполненных лабораторных работы*

***Выполнение данной работы согласовано с преподавателем!***

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| Студент группы ИУ5-31Б |  | Преподаватель каф. ИУ5 |
| Нагдасёв Д.М. |  | Гапанюк Ю.Е. |
|  |  |  |

Москва, 2024 г

**Цель лабораторной работы:** изучение разработки ботов в Telegram, изучение разработки ботов в Telegram.

**Краткое описание программы:** бот-Википедия в Telegram, который обращается к оригинальному сайту Википедии и возвращается первый абзац информации по заданному пользователю слову.

**Код программы:**

import wikipediaapi

from telegram import Update, ReplyKeyboardMarkup, KeyboardButton, ReplyKeyboardRemove, Bot

from telegram.ext import ApplicationBuilder, CommandHandler, CallbackContext, MessageHandler, filters, ConversationHandler

import threading

import asyncio

from collections import defaultdict

from datetime import datetime, timedelta

user\_languages = {}

index\_lang = 0

connected\_users = {}

active\_users = {}

counter = 1

counter\_active = 1

current\_id = 0

flag = -1

flag\_active = -1

can\_search = False

user\_last\_message\_time = defaultdict(lambda: datetime.now())

tasks = {}

forbidden\_symbols = "!@\"\'#№$~`:;[]{}/\\|?.,<>()\*&^%-\_=+"

allowed\_characters = [ ["абвгдеёжзийклмнопрстуфхцчшщъыьэюя", "", "0123456789"], # 0 - Русский

                           ["abcdefghijklmnopqrstuvwxyz", "", "0123456789"] ]   # 1 - English

allowed\_characters[0][1] = allowed\_characters[0][0].upper()

allowed\_characters[1][1] = allowed\_characters[1][0].upper()

bot\_token = "BOT\_TOKEN" # ВСТАВИТЬ ТОКЕН БОТА

async def send\_delayed\_message(chat\_id: int):

    global counter\_active

    await asyncio.sleep(16)

    if datetime.now() - user\_last\_message\_time[chat\_id] >= timedelta(seconds=15):

        for i in range(len(active\_users)): # active\_users.pop(active\_users.index(chat\_id))

            if active\_users[i] == chat\_id: # counter\_active -= 1

                active\_users.pop(i)

                counter\_active -= 1

async def start(update: Update, context: CallbackContext) -> None:

    global counter

    global counter\_active

    global flag

    global flag\_active

    global can\_search

    can\_search = False

    chat\_id = update.message.chat\_id

    user\_last\_message\_time[chat\_id] = datetime.now()

    tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

    if chat\_id in tasks:

        tasks[chat\_id].cancel()

    tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

    user = update.message.from\_user

    user\_info = (

        f"User ID: {user.id}\n"

        f"Username: {user.username}\n"

        f"Full Name: {user.full\_name}\n"

        f"Language Code: {user.language\_code}"

    )

    for i in range(len(connected\_users)):

        if connected\_users[i] == user.id:

            flag = i

            break

    for i in range(len(active\_users)):

        if active\_users[i] == user.id:

            flag\_active = i

            break

    if flag\_active == -1:

        active\_users[counter\_active - 1] = user.id

        counter\_active += 1

    flag\_active = -1

    if flag == -1:

        print(counter, ". Successful connection", sep = "")

        connected\_users[counter - 1] = user.id

        counter += 1

    else:

        print("\nUser with ID:", connected\_users[flag], "successfully reconnected")

        flag = -1

    print(user\_info)

    await update.message.reply\_text('Привет! Я бот, который поможет найти значение любого слова в Википедии! Только сначала выбери язык при помощи команды /language\n\nHello! I\'m a bot that will help you find the meaning of any word on Wikipedia! Just first select the language using the /language command')

async def language(update: Update, context: CallbackContext) -> None:

    global can\_search

    global counter

    global counter\_active

    global flag

    global flag\_active

    can\_search = False

    chat\_id = update.message.chat\_id

    user\_last\_message\_time[chat\_id] = datetime.now()

    if chat\_id in tasks:

        tasks[chat\_id].cancel()

    tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

    user = update.message.from\_user

    for i in range(len(active\_users)):

        if active\_users[i] == user.id:

            flag\_active = i

            break

    if flag\_active == -1:

        active\_users[counter\_active - 1] = user.id

        counter\_active += 1

    flag\_active = -1

    for i in range(len(connected\_users)):

        if connected\_users[i] == user.id:

            flag = i

            break

    if flag == -1:

        connected\_users[counter - 1] = user.id

        counter += 1

    else:

        flag = -1

    print("\nUser with ID", user.id, "choosing language")

    #print(user\_info)

    keyboard = [

        [KeyboardButton("Русский"), KeyboardButton("English")]

    ]

    reply\_markup = ReplyKeyboardMarkup(keyboard, one\_time\_keyboard=True, resize\_keyboard=True)

    await update.message.reply\_text('Выберите язык Википедии / Choose a language Wiki:', reply\_markup=reply\_markup)

async def language\_choice(update: Update, context: CallbackContext) -> None:

    global index\_lang

    global counter\_active

    global flag\_active

    chat\_id = update.message.chat\_id

    text = update.message.text

    user = update.message.from\_user

    user\_last\_message\_time[chat\_id] = datetime.now()

    for i in range(len(active\_users)):

        if active\_users[i] == user.id:

            flag\_active = i

            break

    if flag\_active == -1:

        active\_users[counter\_active - 1] = user.id

        counter\_active += 1

    flag\_active = -1

    if text == "Русский":

        index\_lang = 0

        user\_languages[chat\_id] = 'ru'

        if chat\_id in tasks:

            tasks[chat\_id].cancel()

        tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

        await update.message.reply\_text('Вы выбрали русский язык. Напишите /search <ваше слово> для поиска (или /search для ввода слова для поиска следующим сообщением).', reply\_markup=ReplyKeyboardRemove())

    elif text == "English":

        index\_lang = 1

        user\_languages[chat\_id] = 'en'

        if chat\_id in tasks:

            tasks[chat\_id].cancel()

        tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

        await update.message.reply\_text('You have selected English. Type /search <your word> to search (or /search to input word for search in next message).', reply\_markup=ReplyKeyboardRemove())

def get\_first\_paragraph(summary: str) -> str:

    return summary.split('\n')[0]

async def search(update: Update, context: CallbackContext) -> None:

    global current\_id

    global can\_search

    global counter\_active

    global flag\_active

    chat\_id = update.message.chat\_id

    user = update.message.from\_user

    user\_last\_message\_time[chat\_id] = datetime.now()

    for i in range(len(active\_users)):

        if active\_users[i] == user.id:

            flag\_active = i

            break

    if flag\_active == -1:

        active\_users[counter\_active - 1] = user.id

        counter\_active += 1

    flag\_active = -1

    if chat\_id in tasks:

        tasks[chat\_id].cancel()

    tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

    query = ' '.join(context.args)

    if chat\_id not in user\_languages:

        await update.message.reply\_text('Пожалуйста, сначала выберите язык с помощью /language \n\n Please select a language first using /language.')

        return ConversationHandler.END

    can\_search = True

    await choose\_word(update, context, query)

async def choose\_word(update: Update, context: CallbackContext, query: str) -> None:

    global current\_id

    global counter\_active

    global flag\_active

    chat\_id = update.message.chat\_id

    user = update.message.from\_user

    user\_last\_message\_time[chat\_id] = datetime.now()

    for i in range(len(active\_users)):

        if active\_users[i] == user.id:

            flag\_active = i

            break

    if flag\_active == -1:

        active\_users[counter\_active - 1] = user.id

        counter\_active += 1

    flag\_active = -1

    if chat\_id in tasks:

        tasks[chat\_id].cancel()

    tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

    if chat\_id not in user\_languages:

        await update.message.reply\_text('Пожалуйста, сначала выберите язык с помощью /language \n\n Please select a language first using /language.')

        return

    wiki = wikipediaapi.Wikipedia(language=user\_languages[chat\_id], user\_agent='YourAppName/1.0 (yourname@example.com)')

    if not query:

        if index\_lang == 0:

            await update.message.reply\_text('Введите слово для поиска :)')

        elif index\_lang == 1:

            await update.message.reply\_text('Input word to search :D')

        return

    # for index in query:

    #     for i in range(len(forbidden\_symbols)):

    #         if index == forbidden\_symbols[i]:

    #             if index\_lang == 0:

    #                 st = 'Слово содержит недопустимый символ: ' + index

    #                 await update.message.reply\_text(st)

    #             elif index\_lang == 1:

    #                 st = 'The word contains an invalid character: ' + index

    #                 await update.message.reply\_text(st)

    #             return

    all\_characters\_right = True #dont work!!!!!!!!!!!!

    for index in query:

        character\_right = False

        for syms in allowed\_characters[index\_lang]:

            for sym in syms:

                if index == sym:

                    character\_right = True

        if character\_right == False:

            all\_characters\_right = False

            break

    if all\_characters\_right == False:

        if index\_lang == 0:

            st = 'Слово содержит недопустимый символ: ' + index

            await update.message.reply\_text(st)

        elif index\_lang == 1:

            st = 'The word contains an invalid character: ' + index

            await update.message.reply\_text(st)

        return

    all\_pos\_var\_of\_word = [query, query.lower(), query.upper(), query[0].upper() + query[1:].lower()]

    find = False

    page = wiki.page(query)

    user = update.message.from\_user

    user\_info = (f"User ID: {user.id}\n")

    if can\_search == True:

        for word in all\_pos\_var\_of\_word:

            if find == True:

                break

            page = wiki.page(word)

            if page.exists():

                find = True

                if current\_id != user.id:

                    current\_id = user.id

                    print("\n", user\_info, end = "")

                print("Searching '", query, "' (modify ", word, ") : Successful\n", sep = "")

                first\_paragraph = get\_first\_paragraph(page.summary)

                await update.message.reply\_text(first\_paragraph)

        if find == False:

            if current\_id != user.id:

               current\_id = user.id

               print("\n", user\_info, end = "")

            print("Searching '", query, "': Failed\nIn all Variations: ", all\_pos\_var\_of\_word, "\n", sep = "")

            await update.message.reply\_text('Page not found.')

    else:

        if index\_lang == 0:

            await update.message.reply\_text('Надо использовать команду /search для поиска.')

        elif index\_lang == 1:

            await update.message.reply\_text('You need to use command /search at first.')

async def qwerty(update: Update, context: CallbackContext) -> None:

    chat\_id = update.message.chat\_id

    user\_last\_message\_time[chat\_id] = datetime.now()

    if chat\_id in tasks:

        tasks[chat\_id].cancel()

    tasks[chat\_id] = asyncio.create\_task(send\_delayed\_message(chat\_id))

    query = update.message.text

    await choose\_word(update, context, query)

def terminal\_input\_listener(loop):

    while True:

        user\_input = input()

        if user\_input.strip().lower() == "exit":

            print("Exiting...")

            for task in tasks.values():

                task.cancel()

            for i in range(len(active\_users)):

                loop.call\_soon\_threadsafe(loop.create\_task, notify\_all\_users(i))

            loop.call\_soon\_threadsafe(loop.stop)

            break

async def notify\_all\_users(index) -> int:

    bot = Bot(token=bot\_token)

    print(f"Message sent to user with ID", active\_users[index])

    try:

        if index\_lang == 0:

            await bot.send\_message(chat\_id=active\_users[index], text="Бот остановлен администратором. Повторите попытку позже :(")

        elif index\_lang == 1:

            await bot.send\_message(chat\_id=active\_users[index], text="Bot stopped by administrator. Try again later D:")

    except Exception as e:

        print(f"Failed to send message to {active\_users[index]}: {e}")

def main():

    loop = asyncio.new\_event\_loop()

    asyncio.set\_event\_loop(loop)

    application = ApplicationBuilder().token(bot\_token).build()

    application.add\_handler(CommandHandler("start", start))

    application.add\_handler(CommandHandler("language", language))

    application.add\_handler(CommandHandler("search", search))

    application.add\_handler(MessageHandler(filters.Regex('^(Русский|English)$'), language\_choice))

    application.add\_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, qwerty))

    input\_thread = threading.Thread(target=terminal\_input\_listener, args=(loop,), daemon=True)

    input\_thread.start()

    print("System started. Ready for connections.")

    application.run\_polling()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Результаты выполнения программы**



**Вывод в консоль**

