Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №6
«Разработка простого телеграм-бота на языке Python»

Выполнил:

Каженец Д.Н.

ИУ5-31Б

Подпись и дата:

Проверил:

Гапанюк Е.Ю.

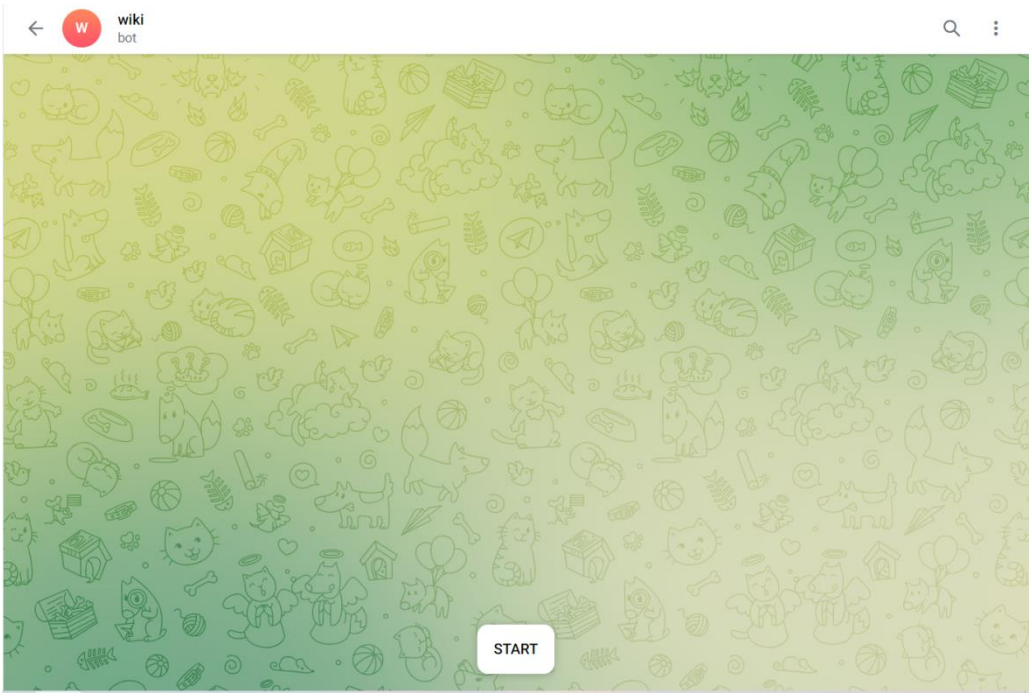
Подпись и дата:

В рамках лабораторной работы осуществлено программирование телеграм-бота на языке Python. Бот может выполнять различные команды, получаемые от пользователя. В табл. 1 представлены список команд и их описание.

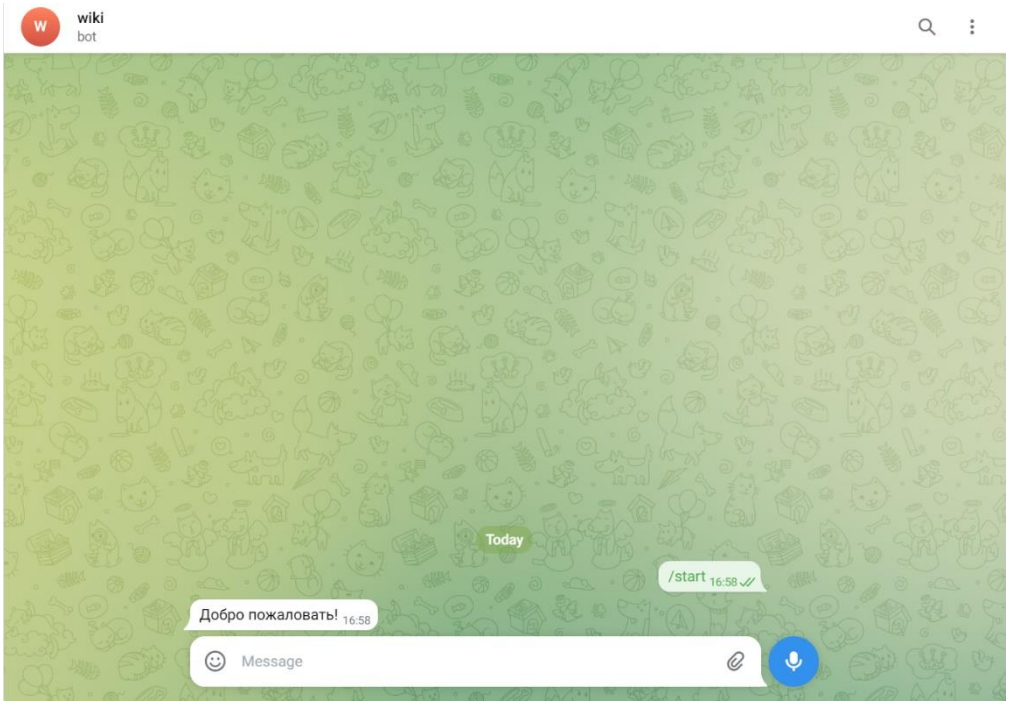
Таблица 1.

№ п.п.	Название команды	Описание работы команды
1.	/start	Запускает бота
2.	/help	Отправляет пользователю список всех команд
3.	/spent 2000	Делает запись о расходе на указанную сумм
4.	/earned 2000	Делает запись о доходе на указанную сумму
5.	/history	Отправляет пользователю историю операций за день
6.	/history day	Отправляет пользователю историю операций за день
7.	/history month	Отправляет пользователю историю операций за месяц
8.	/history year	Отправляет пользователю историю операций за год

Начало работы



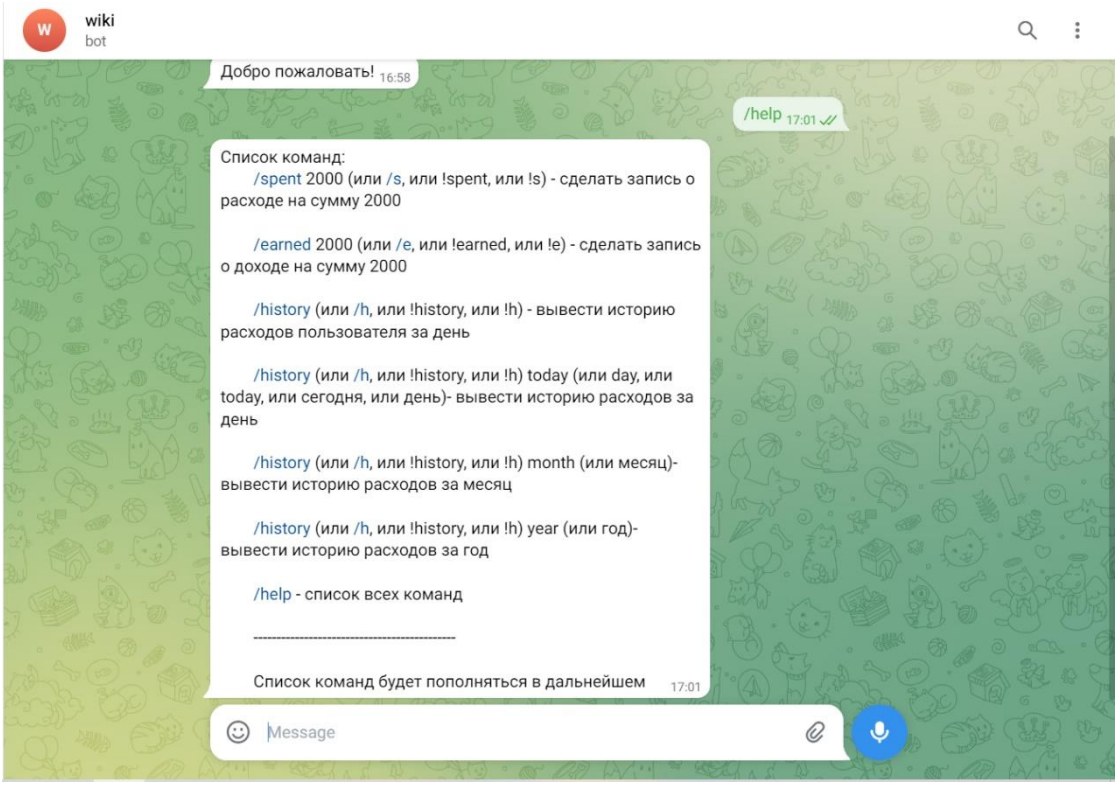
Для того чтобы запустить бота, необходимо нажать на кнопку «START».



После этого бот здоровается с пользователем и начинает работу

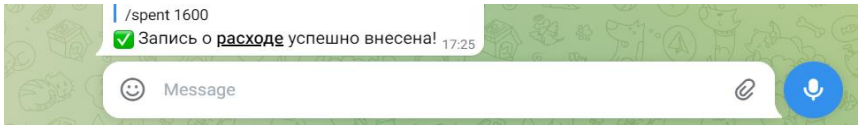
Команда «/help»

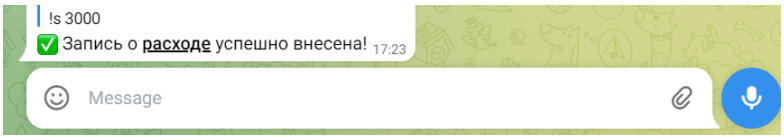
При вызове команды «/help» пользователю отправляется список всех доступных команд бота с их описанием.



Команда «/spent»

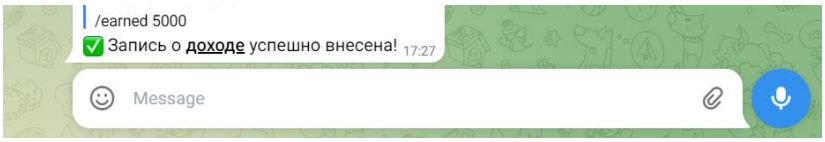
При вызове команды «/spent 2000» делается запись о расходе на сумму, указанную пользователем (также возможны варианты «/s», «!spent», «!s»)

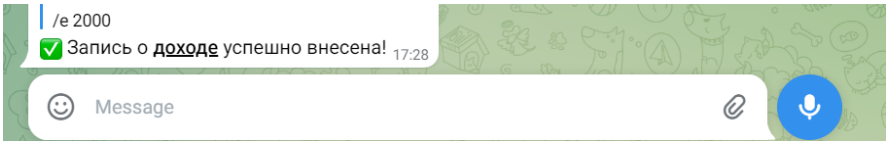




Команда «/earned»

При вызове команды «/earned 2000» делается запись о доходе на сумму, указанную пользователем (также возможны варианты «/e», или «!earned», или «!e»)





db.py

```
import sqlite3


class BotDB:


    def __init__(self, db_file):

        self.conn = sqlite3.connect(db_file)

        self.cursor = self.conn.cursor()


    def user_exists(self, user_id):

        result = self.cursor.execute("SELECT `id` FROM `users` WHERE `user_id` = ?",
(user_id,))

        return bool(len(result.fetchall()))


    def get_user_id(self, user_id):

        result = self.cursor.execute("SELECT `id` FROM `users` WHERE `user_id` = ?",
(user_id,))

        return result.fetchone()[0]


    def add_user(self, user_id):

        self.cursor.execute("INSERT INTO `users` (`user_id`) VALUES (?)", (user_id,))

        return self.conn.commit()


    def add_record(self, user_id, operation, value):
```

```
        self.cursor.execute("INSERT INTO `records` (`users_id`, `operation`, `value`) VALUES (?, ?, ?)",
            (self.get_user_id(user_id),
            operation == "+",
            value))
        return self.conn.commit()


    def get_records(self, user_id, within = "all"):


        if within == "day":
            result = self.cursor.execute("SELECT * FROM `records` WHERE `users_id` = ? AND `date` BETWEEN datetime('now', 'start of day') AND datetime('now', 'localtime') ORDER BY `date`",
                (self.get_user_id(user_id),))
        elif within == "week":
            result = self.cursor.execute("SELECT * FROM `records` WHERE `users_id` = ? AND `date` BETWEEN datetime('now', '-6 days') AND datetime('now', 'localtime') ORDER BY `date`",
                (self.get_user_id(user_id),))
        elif within == "month":
            result = self.cursor.execute("SELECT * FROM `records` WHERE `users_id` = ? AND `date` BETWEEN datetime('now', 'start of month') AND datetime('now', 'localtime') ORDER BY `date`",
                (self.get_user_id(user_id),))
        else:
            result = self.cursor.execute("SELECT * FROM `records` WHERE `users_id` = ? ORDER BY `date`",
                (self.get_user_id(user_id),))


        return result.fetchall()


    def close(self):
        self.connection.close()
```

dispatcher.py

```
import logging
from aiogram import Bot, Dispatcher
from filters import IsOwnerFilter, IsAdminFilter, MemberCanRestrictFilter
import config

# Configure logging
logging.basicConfig(level=logging.INFO)

# prerequisites
if not config.BOT_TOKEN:
    exit("No token provided")

# init
bot = Bot(token=config.BOT_TOKEN, parse_mode="HTML")
dp = Dispatcher(bot)

# activate filters
dp.filters_factory.bind(IsOwnerFilter)
dp.filters_factory.bind(IsAdminFilter)
dp.filters_factory.bind(MemberCanRestrictFilter)
```

filters.py

```
from aiogram import types
from aiogram.dispatcher.filters import BoundFilter
import config


class IsOwnerFilter(BoundFilter):
    key = "is_owner"

    def __init__(self, is_owner):
```

```
        self.is_owner = is_owner

    async def check(self, message: types.Message):
        return message.from_user.id == config.BOT_OWNER


class IsAdminFilter(BoundFilter):
    key = "is_admin"

    def __init__(self, is_admin: bool):
        self.is_admin = is_admin

    async def check(self, message: types.Message):
        member = await message.bot.get_chat_member(message.chat.id, message.from_user.id)
        return member.is_chat_admin() == self.is_admin


class MemberCanRestrictFilter(BoundFilter):
    key = 'member_can_restrict'

    def __init__(self, member_can_restrict: bool):
        self.member_can_restrict = member_can_restrict

    async def check(self, message: types.Message):
        member = await message.bot.get_chat_member(message.chat.id, message.from_user.id)
        return (member.is_chat_creator() or member.can_restrict_members) ==
self.member_can_restrict
```

personal_actions.py

```
from aiogram import types
from dispatcher import dp
import config
```

```
import re
from bot import BotDB

@dp.message_handler(commands = "start")
async def start(message: types.Message):
    if(not BotDB.user_exists(message.from_user.id)):
        BotDB.add_user(message.from_user.id)

    await message.bot.send_message(message.from_user.id, "Добро пожаловать!")



@dp.message_handler(commands = "help")
async def start(message: types.Message):
    if(not BotDB.user_exists(message.from_user.id)):
        BotDB.add_user(message.from_user.id)

    await message.bot.send_message(message.from_user.id, '''Список команд:
        /spent 2000 (или /s, или !spent, или !s) - сделать запись о расходе на сумму 2000

        /earned 2000 (или /e, или !earned, или !e) - сделать запись о доходе на сумму 2000

        /history (или /h, или !history, или !h) - вывести историю расходов пользователя за день

        /history (или /h, или !history, или !h) today (или day, или today, или сегодня, или день) - вывести историю расходов за день

        /history (или /h, или !history, или !h) month (или месяц) - вывести историю расходов за месяц

        /history (или /h, или !history, или !h) year (или год) - вывести историю расходов за год
```

```
        /help - список всех команд

        ------------------------------------------

        Список команд будет пополняться в дальнейшем
        ''')

@dp.message_handler(commands = ("spent", "earned", "s", "e"), commands_prefix = "/!")
async def start(message: types.Message):
    cmd_variants = (('/spent', '/s', '!spent', '!s'), ('/earned', '/e', '!earned', '!e'))
    operation = '-' if message.text.startswith(cmd_variants[0]) else '+'

    value = message.text
    for i in cmd_variants:
        for j in i:
            value = value.replace(j, '').strip()

    if(len(value)):
        x = re.findall(r"\d+(?:.\d+)?", value)
        if(len(x)):
            value = float(x[0].replace(',', '.'))

            BotDB.add_record(message.from_user.id, operation, value)

            if(operation == '-'):
                await message.reply("✅ Запись о <u><b>расходе</b></u> успешно внесена!")
            else:
                await message.reply("✅ Запись о <u><b>доходе</b></u> успешно внесена!")
        else:
            await message.reply("Не удалось определить сумму!")
    else:
        await message.reply("Не введена сумма!")
```

```
@dp.message_handler(commands = ("history", "h"), commands_prefix = "/!")
async def start(message: types.Message):
    cmd_variants = ('/history', '/h', '!history', '!h')
    within_als = {
        "day": ('today', 'day', 'сегодня', 'день'),
        "month": ('month', 'месяц'),
        "year": ('year', 'год'),
    }

    cmd = message.text
    for r in cmd_variants:
        cmd = cmd.replace(r, '').strip()

    within = 'day'
    if(len(cmd)):
        for k in within_als:
            for als in within_als[k]:
                if(als == cmd):
                    within = k

    records = BotDB.get_records(message.from_user.id, within)

    if(len(records)):
        answer = f"🕘 История операций за {within_als[within][-1]}\n\n"

        for r in records:
            answer += "<b>" + ("➖ Расход" if not r[2] else "➕ Доход") + "</b>"
            answer += f" - {r[3]}"
            answer += f" <i>({r[4]})</i>\n"

        await message.reply(answer)
```

```
else:
    await message.reply("Записей не обнаружено!")
```