Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №5
«Разработка простого бота для Telegram с использованием языка Python»

Выполнил:
студент группы ИУ5-32Б
Мажитов В.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Описание задания

Разработайте простого бота для Telegram. Бот должен использовать функциональность создания кнопок.

Текст программы

Файл bot.py

```python
import asyncio
import logging

from aiogram import Bot, Dispatcher
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.contrib.fsm_storage.redis import RedisStorage2

from tgbot.config import load_config
from tgbot.handlers.dart import register_dart
from tgbot.handlers.dice import register_dice
from tgbot.handlers.bowl import register_soccer
from tgbot.handlers.user import register_user

logger = logging.getLogger(__name__)


def register_all_handlers(dp):
    register_user(dp)

    register_dice(dp)
    register_dart(dp)
    register_soccer(dp)


async def main():
    config = load_config()

    storage = MemoryStorage()
    bot = Bot(token=config.telegram.token, parse_mode='HTML')
    dp = Dispatcher(bot, storage=storage)

    register_all_handlers(dp)

    try:
        await dp.start_polling()
    finally:
        await dp.storage.close()
        await dp.storage.wait_closed()
        await bot.session.close()


if __name__ == '__main__':
    try:
        asyncio.run(main())
```

```
    except (KeyboardInterrupt, SystemExit):
        logger.error("Bot stopped!")
```

Файл Dockerfile

```
FROM python:3.9-buster

WORKDIR /usr/src/app/tg_bot

COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .

CMD [ "python", "bot.py" ]
```

Файл deployments/Chart.yaml

```
apiVersion: v2
name: tgbot-service
description: A Helm chart for Kubernetes

# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into
versioned archives
# to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer.
They're included as
# a dependency of application charts to inject those utilities and functions into
the rendering
# pipeline. Library charts do not define any templates and therefore cannot be
deployed.
type: application

# This is the chart version. This version number should be incremented each time
you make changes
# to the chart and its templates, including the app version.
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.1.0

# This is the version number of the application being deployed. This version number
should be
# incremented each time you make changes to the application. Versions are not
expected to
# follow Semantic Versioning. They should reflect the version the application is
using.
# It is recommended to use it with quotes.
appVersion: "1.0.0"
```

Файл deployments/values.yaml

```yaml
replicaCount: 1

image:
  repository: registry.gitlab.com/trackerbot/pcpl_lab5
  pullPolicy: IfNotPresent
  pullSecretName: trackerbot-cred

  tag: "latest"

service:
  secret_name: tgbot-secret
```

Файл deployments/template/deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "tgbot-service.fullname" . }}
  namespace: {{ .Release.Namespace }}
  labels:
    {{- include "tgbot-service.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app.kubernetes.io/name: {{ include "tgbot-service.name" . }}
      app.kubernetes.io/instance: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app.kubernetes.io/name: {{ include "tgbot-service.name" . }}
        app.kubernetes.io/instance: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default
.Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          env:
            - name: BOT_TOKEN
              valueFrom:
                secretKeyRef:
                  name: {{ .Values.service.secret_name }}
                  key: bot_token

      imagePullSecrets:
        - name: {{ .Values.image.pullSecretName }}
```

Файл tgbot/handlers/bowl.py

```
from aiogram import types, Dispatcher
from aiogram.dispatcher.filters import Text

from tgbot.keyboards.reply import bowl_button_text
from tgbot.utils.process_result import process_result


async def bowl(message: types.Message):
    result = await message.answer_dice('🎳')
    await process_result(message, result.dice.value)


def register_soccer(dp: Dispatcher):
    dp.register_message_handler(bowl, Text(equals=bowl_button_text))
```

Файл tgbot/handlers/dart.py

```
from aiogram import types, Dispatcher
from aiogram.dispatcher.filters import Text

from tgbot.keyboards.reply import dart_button_text
from tgbot.utils.process_result import process_result


async def dart(message: types.Message):
    result = await message.answer_dice('🎯')
    await process_result(message, result.dice.value)


def register_dart(dp: Dispatcher):
    dp.register_message_handler(dart, Text(equals=dart_button_text))
```

Файл tgbot/handlers/dice.py

```
from aiogram import types, Dispatcher
from aiogram.dispatcher.filters import Text

from tgbot.keyboards.reply import dice_button_text
from tgbot.utils.process_result import process_result


async def dice(message: types.Message):
    result = await message.answer_dice('🎲')
    await process_result(message, result.dice.value)


def register_dice(dp: Dispatcher):
    dp.register_message_handler(dice, Text(equals=dice_button_text))
```

Файл tgbot/handlers/user.py

```python
from aiogram import Dispatcher
from aiogram.types import Message

from tgbot.keyboards.reply import menu_keyboard


async def user_start(message: Message):
    await message.reply(
        "Привет! Выбери что хочешь сделать с помощью меню",
        reply_markup=menu_keyboard(),
    )


def register_user(dp: Dispatcher):
    dp.register_message_handler(user_start, commands=["start"], state="*")
```

Файл tgbot/keyboards/reply.py

```python
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

dice_button_text = "Бросить кубик"
bowl_button_text = "Сыграть в боулинг"
dart_button_text = "Сыграть в дартс"


def menu_keyboard() -> ReplyKeyboardMarkup:
    return (
        ReplyKeyboardMarkup(resize_keyboard=True)
        .row(
            KeyboardButton(text=dice_button_text),
        )
        .row(
            KeyboardButton(text=dart_button_text),
        )
        .row(
            KeyboardButton(text=bowl_button_text),
        )
    )
```

Файл tgbot/utils/process_result.py

```python
import asyncio

from aiogram.types import Message


async def process_result(message: Message, value: int):
    if value == 6:
        await asyncio.sleep(2)
        await message.answer("Ты выиграл!")
```

Файл tgbot/config.py

```python
from dataclasses import dataclass

from environs import Env


@dataclass
class Telegram:
    token: str


@dataclass
class Config:
    telegram: Telegram


def load_config():
    env = Env()

    return Config(
        telegram=Telegram(
            token=env.str("BOT_TOKEN"),
        ),
    )
```

Пример выполнения программы