

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА №4
Реализация Telegram-бота на языке программирования Python

по дисциплине
«Цифровая культура»

Выполнил
студент гр. 5131001/30002

Н. С. Мишенёв

Руководитель
ассистент ВШК ИКНК

М. С. Иванов

Санкт-Петербург – 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ.....	3
2 ХОД РАБОТЫ.....	4
2.1 Усовершенствование Telegram-бота.....	4
2.2 Перенос бота на вебхук.....	7
2.3 Ответы на контрольные вопросы.....	9
3 ВЫВОД.....	11

1 ЦЕЛЬ РАБОТЫ

Цель работы – получение навыков создания Telegram-ботов на языке программирования Python.

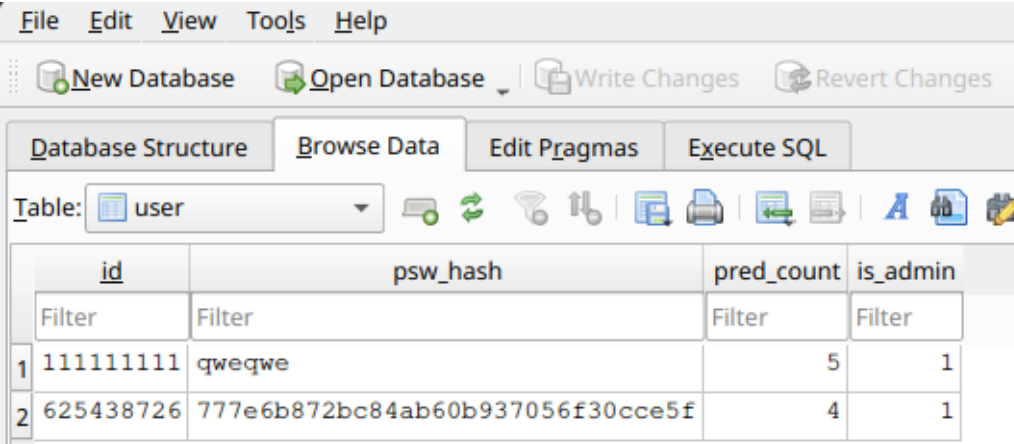
2 ХОД РАБОТЫ

Лабораторная работа была выполнена с использованием Python версии **3.10.17**. Для создания Telegram-бота был использован **aiogram3**. В качестве БД для Telegram-бота использовалась **SQLite**, вместе с **SQLAlchemy**. В качестве внутреннего сервера для создания вебхука использовалась **Tuna**. Для написания небольшого обработчика POST-запросов, использовался **Flask**. Виртуальное окружение было создано с помощью **venv**.

2.1 Усовершенствование Telegram-бота.

В бота было добавлено несколько функций, позволяющих добавлять и удалять администраторов и пользователей, а также функция, которая собирала статистику о сделанных предсказаниях для каждого пользователя.

Для того чтобы добавить роль администратора, была модифицирована таблица базы данных. Было добавлено поле **<is_admin>**, которое показывает принадлежность пользователя к классу администраторов. Также было добавлено поле для подсчёта выполненных предсказаний. (Рисунок 1)



	id	psw_hash	pred_count	is_admin
Filter	Filter	Filter	Filter	Filter
1	11111111	qweqwe	5	1
2	625438726	777e6b872bc84ab60b937056f30cce5f	4	1

Рисунок 1 – изменённая структура таблицы БД.

Самым первым администратором в системе становится пользователь, ID которого указан в файле. После этого, он может добавлять и удалять пользователей с помощью соответствующих команд с использованием telegram-ID целевого пользователя:

- **/addadm <id>**

Данная команда изменяет значение поля роли администратора, если передан корректный ID пользователя, иначе ничего не происходит и пользователь получает сообщение об ошибке. Данная команда реализована следующим образом (Листинг 1)

Листинг 1 – реализация хендлера для команды /addadm

```
@router.message(Command("addadm"))
@admin_required
async def adm_cmd_addadm(message: types.Message, command: CommandObject):
    if command.args is None:
        await message.answer(REPLIES["no_args_add"])
        return

    user = get_user(command.args, engine)
    if (user):
        add_adm_user(user.id, engine)
        await message.answer(REPLIES["add_user"].format(id=command.args))
    else:
        await message.answer(REPLIES["incorrect_id"])
```

- **/rmuser <id>**

Данная команда удаляет пользователя из базы данных, если передан корректный ID пользователя, иначе ничего не происходит и пользователь получает сообщение об ошибке. Данная команда реализована следующим образом (Листинг 2)

Листинг 2 – реализация хендлера для команды /rmuser

```
@router.message(Command("rmuser"))
@admin_required
async def adm_cmd_rmuser(message: types.Message, command: CommandObject):
    if command.args is None:
        await message.answer(REPLIES["no_args_rm"])
        return

    user = get_user(command.args, engine)
    if (user):
        delete_user(user.id, engine)
        await message.answer(REPLIES["delete_user"].format(id=command.args))
    else:
        await message.answer(REPLIES["incorrect_id"])
```

Также администратор может получать статистику обо всех пользователях в системе с помощью команды **/stats:** (Рисунок 2)

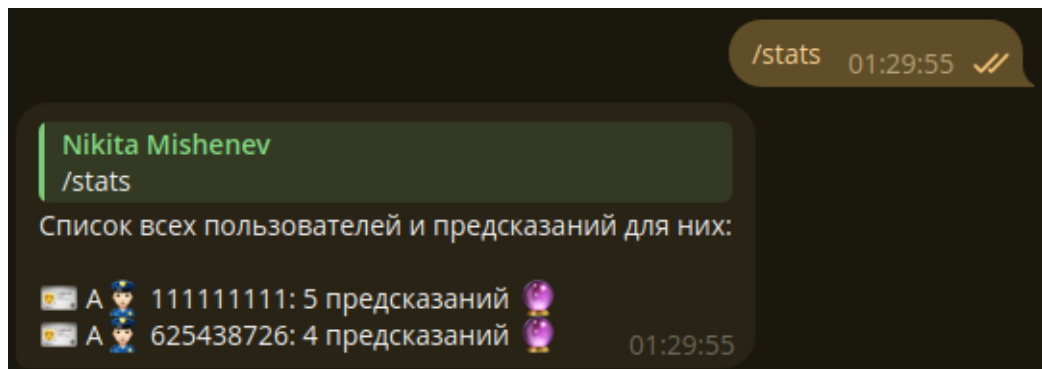


Рисунок 2 – ответ бота на запрос статистики с помощью команды /stats

Данная команда просто собирает всех пользователей и форматирует вывод для администратора в удобочитаемом виде. Данная команда реализована следующим образом: (Листинг 3)

Листинг 3 – реализация хендлера для команды /stats

```
@router.message(Command("stats"))
@admin_required
async def adm_cmd_stats(message: types.Message):
    users = get_all_users(engine)
    users_msg = ""
    for user in users:
        users_msg += REPLIES["user_stat"].format(
            role="A👮" if user.is_admin else "U👤",
            id=user.id,
            predictions=user.pred_count
        )

    await message.reply(REPLIES["all_users"].format(users=users_msg))
```

Каждый из вышеприведённых хендлеров обернут в декоратор **@admin_required**. Данный декоратор защищает данные команды от использования пользователями, у которых нет прав администратора. Он реализован следующим образом: (Листинг 4)

Листинг 4 – реализация декоратора @admin_required

```
@def admin_required(func: Callable) -> Any:
    @wraps(func)
    async def wrapper(*args, **kwargs):
        user = get_user(args[0].from_user.id, engine)
        if (not user or not user.is_admin):
            await args[0].reply(REPLIES["insufficient_rights"])
            return
        return await func(*args, **kwargs)
    return wrapper
```

2.2 Перенос бота на вебхук.

Для переноса бота на webhook, был написан простейший POST-обработчик на Flask (Листинг 5), который должен был быть использован на сервере для перенаправления всех апдейтов с сообщениями через вебхук.

Листинг 5 – простое Flask-приложение с обработчиком POST запросов

```
from flask import (Flask,
                   render_template,
                   request)
from os import environ
import requests

app = Flask(__name__)
app.secret_key = environ.get("FKEY")

@app.route('/')
def view_form():
    return render_template('index.html')

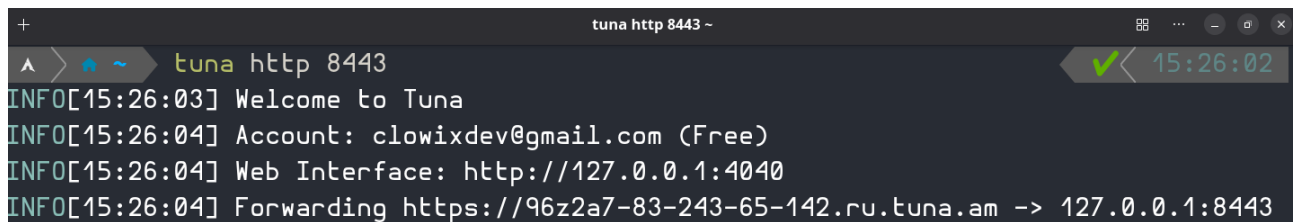
@app.route('/post', methods=['GET', 'POST'])
def handle_post():
    if request.method == 'POST':
        print(request.data)
        requests.post("http://127.0.0.1:8443", json=request.data)

    return "sent an answer"

if __name__ == '__main__':
    app.run(debug=False,
            port=8443,
            ssl_context="adhoc")
```

Однако, в силу ограниченности рабочего пространства и процессорного времени на бесплатном сервере, в качестве вебхука выступал сервер, поднятый локально с помощью утилиты **Tuna**.

После установки, настройки и запуска сервера, утилита показывает нам адрес созданной страницы, которую необходимо будет зарегистрировать как вебхук. (Рисунок 3)



```
+ tuna http 8443 ~
INFO[15:26:03] Welcome to Tuna
INFO[15:26:04] Account: clowixdev@gmail.com (Free)
INFO[15:26:04] Web Interface: http://127.0.0.1:4040
INFO[15:26:04] Forwarding https://96z2a7-83-243-65-142.ru.tuna.am -> 127.0.0.1:8443
```

Рисунок 3 – запущенный сервер на порте 8443

Далее, при запуске бота необходимо зарегистрировать полученный вебхук, а также запустить сервер на этом порте, для того чтобы бот получал передаваемые апдейты (Листинг 6)

Листинг 6 – регистрация вебхука и запуск сервера

```
async def main():
    dp.include_router(admin.router)
    dp.include_router(default.router)

    # dp.startup.register(on_startup)
    await bot.set_webhook(f"{TUNA}{WEBHOOK_PATH}")
    await bot.send_message(chat_id=ADMIN_ID, text="started!")

    # dp.shutdown.register(on_shutdown)
    await bot.send_message(chat_id=ADMIN_ID, text="shuted down...")
    await bot.delete_webhook(drop_pending_updates=True)
    await bot.session.close()

    app = web.Application()

    webhook_requests_handler = SimpleRequestHandler(
        dispatcher=dp,
        bot=bot
    )

    webhook_requests_handler.register(app, path=WEBHOOK_PATH)
    setup_application(app, dp, bot=bot)
    await web._run_app(app, host=HOST, port=PORT)
```

При запуске бота происходит регистрация вебхука, для того чтобы telegram-сервер перенаправлял все апдейты на наш вебхук, а при выключении бота происходит удаление вебхука и завершение сессии прослушивания.

Также, устанавливается **SimpleRequestHandler(...)**, который и будет форматировать и перенаправлять все приходящие от сервера запросы на уже разработанные обработчики сообщений.

2.3 Ответы на контрольные вопросы.

1. Какие преимущества и недостатки реализаций Telegram-ботов с использованием polling'a и webhook'ов.

Боты, реализованные с использованием polling'a, периодически отправляют запросы на Telegram-сервер, чтобы узнать, не произошло ли в их чатах что-нибудь. Если есть какие-то изменения, то они узнают об этом только тогда, когда отправят запрос. Из этого вытекает самый большой недостаток таких ботов – частые запросы на сервер и перегрузка железа лишней работой. Также, при очень большом количестве апдейтов, производительность такого бота может очень сильно снизиться. Однако такие боты очень просты и неприхотливы в настройке и использовании, например для маленьких проектов или локального тестирования.

Боты, реализованные с использованием webhook'ов не обращаются на сервер, а сервер обращается к ним, с информацией о полученном апдейте. Такие же боты очень удобны, так как не тратят ресурсы хоста на постоянные запросы к серверу. Также, при большой нагрузке, такие боты лучше справляются с наплывом апдейтов. Однако такие боты требуют отдельного сервера для прослушивания апдейтов, с которым тоже может что-то случиться, и дополнительной настройки SSL-сертификата.

2. Для чего используются хэш-функции при работе с паролями?

Хэш функции используются для безопасного хранения паролей пользователей, так как пароли можно преобразовать в хэши, а хэши в пароли преобразовать нельзя. Таким образом, взаимодействуя с хэшем, риск утечки пароля в сетькратно уменьшается.

3. Что такое «белый» IP-адрес? Зачем применяется?

Белые IP-адреса это публичные IP-адреса, к которым можно обращаться в сети интернет с других устройств. Так как только белые IP-адреса могут существовать в интернете, то, все сервисы которыми мы пользуемся имеют белые IP-адреса, а иногда и несколько, на случай отказа одного из них.

4. Для чего применяются базы данных?

Базы данных используются для долгосрочного хранения информации, упрощения манипуляций с данными, а также для быстрого доступа и фильтрации этих данных.

5. Для чего применяется JSON формат файлов?

JSON формат файлов применяется для обмена данными между серверами и приложениями. Также JSON формат применяется при взаимодействии с какими-либо API. JSON формат предоставляет удобную структуру хранения каких либо данных, которая представлена вложенными словарями.

3 ВЫВОД

В ходе работы были получены навыки создания **Telegram**-ботов с использованием языка программирования **Python**.

В ходе работы был написан Telegram-бот с использованием **aiogram3**. В нём реализованы функции, который осуществляют доступ к классификатору изображений, а также реализована система пользователей и администраторов, которая подразумевает регистрацию и ввод пароля, перед использованием возможностей бота. Также класс администратора имеет возможности добавлять других администраторов, удалять пользователей и просматривать статистику пользователей.

Хранение хешей всех паролей, хешированных по алгоритму **md5**, осуществляется в базе данных **SQLite**, взаимодействие с которой производится с помощью **SQLAlchemy**.

Было изучено понятие **webhook**'а, а также бот был перенесён на **webhook**'и, с использованием сервера **Tuna**. Также был реализован простейший **POST**-обработчик с помощью фреймворка **Flask**.

ПРИЛОЖЕНИЕ 1. ПРОГРАММНЫЙ КОД TELEGRAM-БОТА

app.py:

```
import asyncio
from aiohttp import web
from aiogram.webhook.aiohttp_server import SimpleRequestHandler, setup_application

from handlers import default, admin
from loader import bot, dp, TUNA, WEBHOOK_PATH, ADMIN_ID, HOST, PORT

from sys import argv

async def on_startup() -> None:
    await bot.set_webhook(f'{TUNA}{WEBHOOK_PATH}')
    await bot.send_message(chat_id=ADMIN_ID, text="started!")

async def on_shutdown() -> None:
    await bot.send_message(chat_id=ADMIN_ID, text="shuted down...")
    await bot.delete_webhook(drop_pending_updates=True)
    await bot.session.close()

async def main():
    dp.include_router(admin.router)
    dp.include_router(default.router)

    dp.startup.register(on_startup)
    dp.shutdown.register(on_shutdown)
    app = web.Application()

    webhook_requests_handler = SimpleRequestHandler(
        dispatcher=dp,
        bot=bot
    )

    webhook_requests_handler.register(app, path=WEBHOOK_PATH)
    setup_application(app, dp, bot=bot)
    await web._run_app(app, host=HOST, port=PORT)

if __name__ == "__main__":
    if (argv.__len__() > 1 and argv[1] != None):
        TUNA = argv[1]
    print(f"Launching bot on {TUNA}...")
    asyncio.run(main())
```

loader.py:

```
from os import environ

from aiogram import Bot, Dispatcher
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from aiogram.fsm.storage.memory import MemoryStorage
from dotenv import load_dotenv
from keras import models

from database.dbworker import create_db_engine
# from scripts.predictor import tensorflow_init

# model = tensorflow_init()
# model.save("./scripts/model.keras")
model = models.load_model("./scripts/model.keras")

load_dotenv(".env")

bot = Bot(
    token=environ.get("TOKEN"),
    default=DefaultBotProperties()
```

```

        parse_mode=ParseMode.HTML
    )
)

engine = create_db_engine()
dp = Dispatcher(storage=MemoryStorage())

ADMIN_ID = 625438726
BOT_TOKEN = environ.get("TOKEN")
HOST = environ.get("HOST")
PORT = int(environ.get("PORT"))
WEBHOOK_PATH = f"/{BOT_TOKEN}"
TUNA = environ.get("TUNA")

```

states.py:

```

from aiogram.fsm.state import StatesGroup, State

class Registration(StatesGroup):
    entering_psw = State()
    confirming_psw = State()

class Login(StatesGroup):
    entering_psw = State()

class Predict(StatesGroup):
    waiting_pic = State()

class Session(StatesGroup):
    logged_in = State()

```

default.py:

```

from hashlib import md5 as hash

from aiogram import F, Router, types, Bot
from aiogram.filters import Command
from aiogram.fsm.context import FSMContext

from database.dbworker import get_user, add_user, add_user_pred
from database.msg import REPLIES
from loader import engine, model
from scripts.predictor import recognize_picture
from states.states import Registration, Session, Login, Predict

router = Router()

@router.message(Command("start", "help"))
async def cmd_help(message: types.Message):
    await message.reply(REPLIES["help"])

@router.message(Command("register"))
async def cmd_register(message: types.Message, state: FSMContext):
    if (get_user(message.from_user.id, engine)):
        await message.reply(REPLIES["registration_already"])
        return
    else:
        await message.reply(REPLIES["register_start"])
        await state.set_state(Registration.entering_psw)

@router.message(Registration.entering_psw)
async def reg_enter_psw(message: types.Message, state: FSMContext):
    if (message.text.lower() == "cton"):
        await message.reply(REPLIES["stop"])
        await state.set_state(Session.logged_in)

    return

```

```

        await state.update_data(psw_hash=hash(message.text.encode()).hexdigest())
        await message.reply(REPLIES["register_confirm"])
        await state.set_state(Registration.confirming_psw)

@router.message(Registration.confirming_psw)
async def reg_confirm_psw(message: types.Message, state: FSMContext):
    if (message.text.lower() == "cton"):
        await message.reply(REPLIES["stop"])
        await state.set_state(Session.logged_in)

    return

    user_data = await state.get_data()
    if (user_data["psw_hash"] == hash(message.text.encode()).hexdigest()):
        add_user([message.from_user.id, user_data["psw_hash"]], engine)
        await message.reply(REPLIES["registration_completed"])
        await state.set_state(None)
    else:
        await message.reply(REPLIES["passwords_are_not_same"])
        await message.reply(REPLIES["register_confirm"])

@router.message(Command("login"))
async def cmd_login(message: types.Message, state: FSMContext):
    if (await state.get_state() == Session.logged_in):
        await message.reply(REPLIES["already_logged"])

    return

    if (get_user(message.from_user.id, engine) is None):
        await message.reply(REPLIES["not_registered"])
        return
    else:
        await message.reply(REPLIES["login_password"])
        await state.set_state(Login.entering_psw)

@router.message(Login.entering_psw)
async def log_enter_psw(message: types.Message, state: FSMContext):
    if (message.text.lower() == "cton"):
        await message.reply(REPLIES["stop"])
        await state.set_state(None)

    return

    psw_hash = hash(message.text.encode()).hexdigest()
    cur_user_hash = get_user(message.from_user.id, engine).psw_hash
    if (psw_hash == cur_user_hash):
        await message.reply(REPLIES["logged_in"])
        await state.set_state(Session.logged_in)
    else:
        await message.reply(REPLIES["incorrect_psw"])

@router.message(Command("predict"))
async def cmd_predict(message: types.Message, state: FSMContext):
    if (await state.get_state() == Session.logged_in):
        await message.reply(REPLIES["predict_prompt"])
        await state.set_state(Predict.waiting_pic)
    else:
        await message.reply(REPLIES["not_logged"])

@router.message(Predict.waiting_pic)
async def predict_waiting_pic(message: types.Message, state: FSMContext, bot: Bot):
    if (message.text and message.text.lower() == "cton"):
        await message.reply(REPLIES["stop"])
        await state.set_state(Session.logged_in)
        return
    elif (message.text):
        await message.reply(REPLIES["not_a_photo"])
        return
    if (message.photo):
        filename = f"./data/content/{message.from_user.id}.jpg"

```

```

        await bot.download(message.photo[-1], filename)
        await message.reply(REPLIES["is_predicting"])

    answer, probability = recognize_picture(filename, model)
    match answer:
        case 0:
            await message.reply(REPLIES["is_bear"].format(
                probability=(1 - int(probability)) * 100
            ))
        case 1:
            await message.reply(REPLIES["is_human"].format(
                probability=probability * 100
            ))
        case _:
            await message.reply(REPLIES["error"])

    user = get_user(message.from_user.id, engine)
    add_user_pred(user.id, engine)

    await message.reply(REPLIES["predict_prompt"])

@router.message(Command("logout"))
async def cmd_logout(message: types.Message, state: FSMContext):
    if (await state.get_state() != Session.logged_in):
        await message.reply(REPLIES["not_logged"])
    else:
        await state.set_state(None)
        await message.reply(REPLIES["logout"])

@router.message(F)
async def no_cmd(message: types.Message):
    await message.reply(REPLIES["miss"])

```

models.py:

```

from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = "user"

    id = Column(Integer, primary_key=True)
    psw_hash = Column(String)
    pred_count = Column(Integer)
    is_admin = Column(Integer)

```

dbworker.py:

```

from typing import Optional

from sqlalchemy import create_engine, select
from sqlalchemy.engine import Engine
from sqlalchemy.orm import Session, sessionmaker

from .models import Base, User

def create_db_engine() -> Engine:
    engine = create_engine('sqlite+pysqlite:///database/database.db')
    Base.metadata.create_all(engine)

    return engine

def create_session(engine: Engine) -> Session:
    Session = sessionmaker(bind=engine)
    return Session()

```

```

def get_user(user_id: Optional[int], engine: Engine) -> User:
    session = create_session(engine)
    user = []
    try:
        if user_id != None:
            user = session.query(User).filter_by(id=user_id).first()
            if not user:
                return None

            session.expunge(user)
    except BaseException as e:
        print(e)
        session.rollback()
    finally:
        session.close()

    return user

def add_user(userdata: list, engine: Engine) -> User:
    session = create_session(engine)
    try:
        user = User(
            id=userdata[0],
            psw_hash=userdata[1],
            pred_count=0,
            is_admin=0 if userdata[0] != 625438726 else 1
        )

        session.add(user)
        session.commit()
    except BaseException as e:
        print(e)
        session.rollback()
    finally:
        session.close()

def get_all_users(engine: Engine) -> list[User]:
    session = create_session(engine)
    users = []
    try:
        all_users = session.execute(select(User).order_by(User.id)).all()
        for user in all_users:
            users += user
    except Exception as e:
        print(e)
        session.rollback()
    finally:
        session.close()

    return users

def delete_user(user_id: int, engine: Engine) -> None:
    session = create_session(engine)
    try:
        if user_id != None:
            user = session.query(User).filter_by(id=user_id).first()
            if not user:
                return None

            session.delete(user)
            session.commit()
    except BaseException as e:
        print(e)
        session.rollback()
    finally:
        session.close()

def add_adm_user(user_id: int, engine: Engine) -> None:
    session = create_session(engine)
    try:

```



```

        if user_id != None:
            user = session.query(User).filter_by(id=user_id).first()
            if not user:
                return None

            user.is_admin = 1
            session.commit()
        except BaseException as e:
            print(e)
            session.rollback()
        finally:
            session.close()

def add_user_pred(user_id: int, engine: Engine) -> None:
    session = create_session(engine)
    try:
        if user_id != None:
            user = session.query(User).filter_by(id=user_id).first()
            if not user:
                return None

            user.pred_count += 1
            session.commit()
        except BaseException as e:
            print(e)
            session.rollback()
        finally:
            session.close()

```

msg.py:

```

REPLIES = {
    "help": "🤖 Вот список команд, которые я могу обработать:\n\n<b>/help 🆘</b> -  
показать этот список\n<b>/register 📝</b> - пройти регистрацию в боте\n<b>/login 🔒</b> -  
пройти аутентификацию в боте\n<b>/logout 🚪</b> - выйти из системы\n<b>/predict 🧠</b> -  
использовать классификатор изображений",
    "stop": "Действие отменено 🛑",
    "miss": "То что вы мне отправили, к сожалению, не является командой, которую я  
понимаю 😞",

    "all_users": "Список всех пользователей и предсказаний для них:\n\n{users}",
    "user_stat": "📊 {role} {id}: {predictions} предсказаний 🧠\n",
    "no_args_rm": "Не передан ID пользователя для удаления ⚠️",
    "no_args_add": "Не передан ID пользователя для добавления ⚠️",
    "incorrect_id": "Некорректный ID пользователя 🛑",
    "delete_user": "Пользователь {id} успешно удалён ✅",
    "add_user": "Пользователь {id} успешно назначен администратором ✅",
    "insufficient_rights": "У вас недостаточно прав 🛑",

    "logout": "Вы успешно вышли из системы ✅",
    "not_logged": "Вы не вошли в систему ⚠️",
    "logged_in": "Вы успешно вошли в систему ✅",
    "already_logged": "Вы уже вошли в систему ⚠️",
    "login_password": "Для того чтобы войти в систему, введите пароль 🔒",
    "incorrect_psw": "Вы ввели неверный пароль 🛑\n\nПопробуйте еще раз, или отправьте  
мне слово \"стоп\", чтобы прекратить попытку входа 🙅",

    "register_start": "Для того чтобы зарегистрироваться в системе, придумайте пароль 🔒  
\n\nЕсли же вы передумали, просто напишите мне слово <b>стоп</b> на любом этапе 🙅",
    "registration_completed": "Супер 🥳\n\nПоздравляю вас, регистрация в системе  
завершена ✅",
    "register_confirm": "Отлично 🔒\n\nА теперь введите тот же пароль, для подтверждения  
👉",
    "registration_already": "Вы уже зарегистрированы в системе ⚠️",
    "passwords_are_not_same": "Введенные пароли не совпадают ⚠️\n\nПопробуйте еще раз  
👉",
    "not_registered": "Вы еще не зарегистрированы в системе ⚠️",

    "predict_prompt": "Отправьте мне картинку, и я скажу, <b>медведь</b> это, или  
<b>человек</b> 🧠\n\nЕсли же вы передумали, просто напишите слово \"стоп\" 🙅",
    "not_a_photo": "Это не фотография, и не команда \"стоп\" ⚠️\n\nПопробуйте еще раз  
👉",

```

```
{  
    "is_predicting": "Отлично, получил ваше фото, сейчас попробую определить, что это на  
нём такое... 🤖💭",  
    "is_bear": "Я считаю что это...\n\n\t\t\t\t\t\t\t\t<b>🐻 МЕДВЕДЬ ({probability}%  
🐾</b>",  
    "is_human": "Я считаю что это...\n\n\t\t\t\t\t\t\t\t<b>👤 ЧЕЛОВЕК ({probability}%  
🧑</b>",  
    "error": "Ой, совсем не понимаю что это такое получилось... 😞"  
}
```

admin.py:

```
from aiogram import Router, types
from aiogram.filters import Command, CommandObject

from database.dbworker import get_all_users, delete_user, get_user, add_adm_user
from database.msg import REPLIES
from loader import engine
from scripts.decorators import admin_required

router = Router()

@router.message(Command("stats"))
@admin_required
async def adm_cmd_stats(message: types.Message):
    users = get_all_users(engine)
    users_msg = ""
    for user in users:
        users_msg += REPLIES["user_stat"].format(
            role="A👤" if user.is_admin else "U👤",
            id=user.id,
            predictions=user.pred_count
        )

    await message.reply(REPLIES["all_users"].format(users=users_msg))

@router.message(Command("rmuser"))
@admin_required
async def adm_cmd_rmuser(message: types.Message, command: CommandObject):
    if command.args is None:
        await message.answer(REPLIES["no_args_rm"])
        return

    user = get_user(command.args, engine)
    if (user):
        delete_user(user.id, engine)
        await message.answer(REPLIES["delete_user"].format(id=command.args))
    else:
        await message.answer(REPLIES["incorrect_id"])

@router.message(Command("addadm"))
@admin_required
async def adm_cmd_addadm(message: types.Message, command: CommandObject):
    if command.args is None:
        await message.answer(REPLIES["no_args_add"])
        return

    user = get_user(command.args, engine)
    if (user):
        add_adm_user(user.id, engine)
        await message.answer(REPLIES["add_user"].format(id=command.args))
    else:
        await message.answer(REPLIES["incorrect_id"])
```