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](#__RefHeading___1)

[2 ХОД РАБОТЫ 4](#__RefHeading___2)

[2.1 Усовершенствование Telegram-бота. 4](#__RefHeading___3)

[2.2 Перенос бота на вебхук. 7](#__RefHeading___5)

[2.3 Ответы на контрольные вопросы. 9](#__RefHeading___Toc2365_2481771433)

[3 ВЫВОД 11](#__RefHeading___7)

# **ЦЕЛЬ РАБОТЫ**

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

# **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.**

## **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 выполненых предсказаний. **(Рисунок 1)**

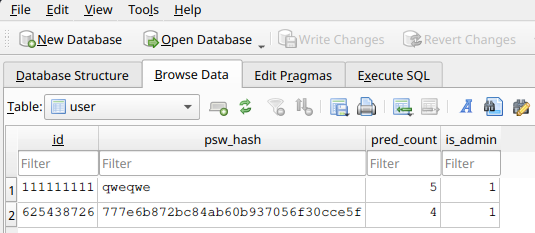
****

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 ситстеме с помощью команды **/stats:** (Рисунок 2)

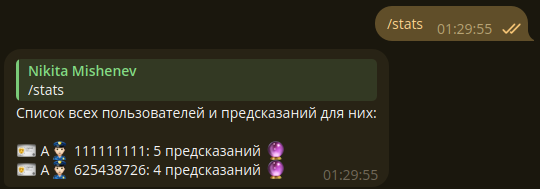


Рисунок 2 – ответ бота на запрос статистики с помощью команды /stats

Данная команда просто собирает всех пользователей и форматирует вывод для администратора в удобночитаемом 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 релизован 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 |

## **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)

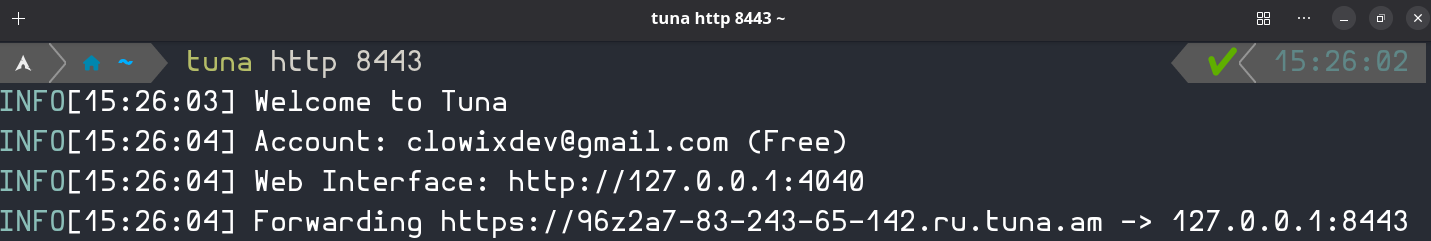


Рисунок 3 – запущенный сервер на порте 8443

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.

## Ответы на контрольные вопросы.

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

Боты, реализованные с использованием polling’а, переодически 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 случится, и дополнительной настройки SSL-сертификата.

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

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

1. *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.

1. *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.

1. *Для чего применяется JSON формат файлов?*

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

# **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() == "стоп"):  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() == "стоп"):  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() == "стоп"):  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() == "стоп"):  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": "Введённые 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<b>🐻 МЕДВЕДЬ ({probability}%) 🐻‍❄️</b>",  "is\_human": "Я считаю что это...\n\n\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"]) |