Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Использование Python-библиотек для работы с ИИ**

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 Реализация классификатора. 4](#__RefHeading___3)

[2.2 Реализация Telegram-бота. 6](#__RefHeading___4)

[Команда /help 6](#__RefHeading___Toc2345_2481771433)

[Команда /register 7](#__RefHeading___Toc2347_2481771433)

[Команда /login и /logout 9](#__RefHeading___Toc2349_2481771433)

[2.3 Объединение Telegram-бота и классификатора. 12](#__RefHeading___5)

[2.4 Ответы на контрольные вопросы. 14](#__RefHeading___Toc2365_2481771433)

[3 ВЫВОД 16](#__RefHeading___7)

# **ЦЕЛЬ РАБОТЫ**

Цель работы – получение навыков работы с методами ИИ для решения задачи классификации данных с использованием языка программирования Python.

# **ХОД РАБОТЫ**

Лабораторная работа была выполнена с использованием Python версии **3.12.9**. Для создания Telegram-бота был использован **aiogram3,** для обучения модели **tensorflow.** В качестве **БД** для Telegram-бота использовалась **SQLite,** вместе с **SQLAlchemy**. Виртуальное окружение было создано с помощью **venv.** Был получен вариант 16 – Медведь.

## **Реализация классификатора.**

Так как для обучения модели, было необходимо собрать как можно больше материала для обучения, были созданы 2 датасета: с людьми и с медведями.

В каждом датасете для обучения использовалось примерно по **600** картинок (Рисунок 1). Также для каждого обучающего датасета, был собран валидационный датасет. В каждом из валидационных датасетов используется примерно по **100** картинок. Таким образом, соотношение размеров валидационного и тренировочного датасетов: 1/6**.**

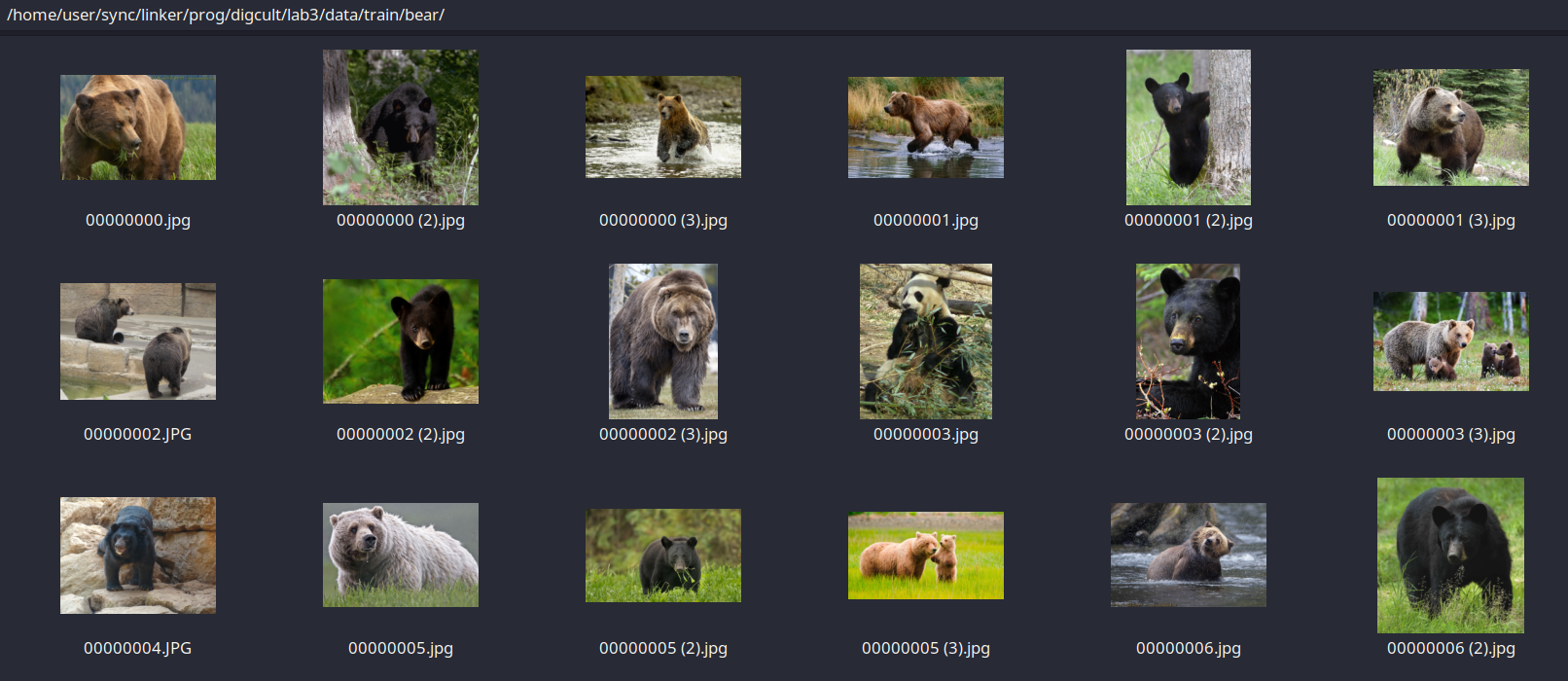


Рисунок 1 – собранный, тренировочный датасет с медведями

Классификатор был реализован на основе данного в методических рекомендациях скрипта с обучением модели. Были внесены некоторые модификации, а именно: восстановление лучших весов, из эпохи с минимальным значением параметря потерь.

Таким образом, по окончании обучения, модель возвращается к той эпохе, в которой параметр потерь принял наименьшее значение. Это реализовано с помощью переданного объекта **callback**, который представляет собой объект **EarlyStopping.** (Листинг 1)

Листинг 1 – реализация возвращения к лучшим весам

|  |
| --- |
| callback = EarlyStopping(  monitor="loss",  mode="min",  restore\_best\_weights=True,  patience=4,  start\_from\_epoch=10,  verbose=1  )  . . .  history = model.fit(  train\_generator,  steps\_per\_epoch=10,  epochs=15,  verbose=1,  validation\_data = validation\_generator,  validation\_steps=10,  callbacks=[callback]  ) |

Также, для классификации отдельно взятой картинки была написана функция, принимающая на вход путь до файла и объект модели, и, возвращающая класс объекта, представленного на картинке (Листинг 2).

Листинг 2 – реализация классификации отдельно взятой картинки

|  |
| --- |
| def recognize\_picture(filename: str, model: Any):  img = image.load\_img(filename, target\_size=(200, 200))  x = image.img\_to\_array(img)  plt.imshow(x / 255.)  x = np.expand\_dims(x, axis=0)  images = np.vstack([x])  classes = model.predict(images, batch\_size=10)  if classes[0] < 0.5:  return (0, classes[0][0]) # bear  else:  return (1, classes[0][0]) # human |

## **Реализация Telegram-бота.**

Telegram-бот был реализован с помощью библиотеки **aiogram3.** Бот поддерживает такие команды как: **/help, /register, /login, /logout, /predict.**

Система диалогов с ботом реализована с помощью технологии **FSM** (Машина Конечных Состояний), встроенной в **aiogram3**. Так, на каждом из этапов, каждому пользователю присваивается некоторое состояние, все из которых описаны в отдельном файле **states.py:** (Листинг 3)

Листинг 3 – реализация состояний для FSM

|  |
| --- |
| 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() |

Далее, в процессе диалога, отдельные хендлеры настроены на сообщения, которые отправлены пользователем в определённом состоянии. И это состояние изменяется по завершении какого-либо действия, или при отмене действия

Давайте рассмотрим, как бот реагирует на команды, описанные выше.

### Команда /help

Эта команда была реализована для того, чтобы пользователь мог ознакомиться со всем функционалом бота. При отправки команды **/help** боту, приходит соответствующий ответ: (Рисунок 2)

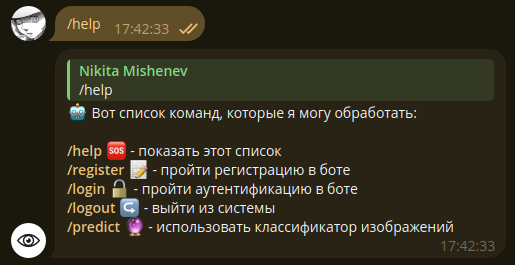


Рисунок 2 – ответ на команду **/help**

Хэндлер, который отвечает за обработку этой команды выглядит следующим образом (Листинг 4):

Листинг 4 – реализация хэндлера для команды **/help**

|  |
| --- |
| @router.message(Command("start", "help"))  async def cmd\_help(message: types.Message):  await message.reply(REPLIES["help"]) |

### Команда /register

Эта команда была реализована для того, чтобы пользователь мог зарегистрироваться в системе, для дальнейшего использования команды с классификатором. Если пользователь не зарегистрирован в системе, то у него не получится войти в нее, и как следствие, не получится воспользоваться классификатором.

При отправке команды **/register** боту, если пользователь уже зарегистрирован в системе, приходит сответствующий ответ: (Рисунок 3)

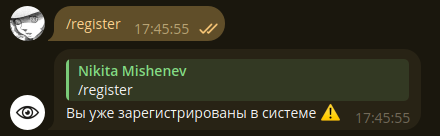


Рисунок 3 – ответ на команду **/register**, если пользователь зарегистрирован

Если же пользователь еще не зарегистрирован, запускается процесс регистрации пользователя: (Рисунок 4)

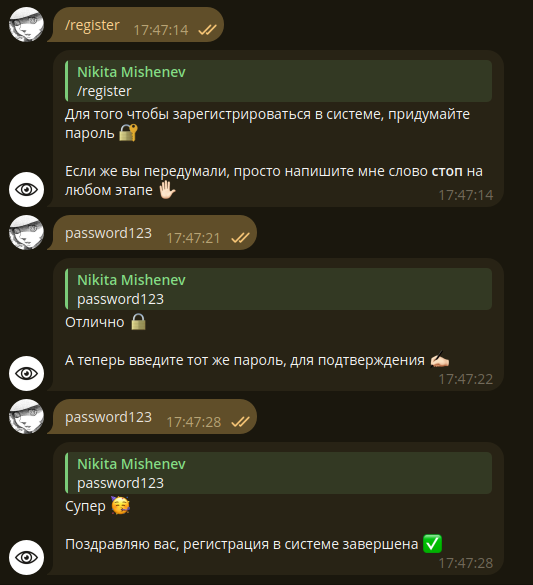


Рисунок 4 – процесс регистрации пользователя

Хэндлер, отвечающий за регистрацию пользователя выглядит следующим образом: (Листинг 5)

Листинг 5 – реализация хэндлера регистрации пользователя.

|  |
| --- |
| @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"]) |

Пароль пользователя не сохраняется в базу данных и никак не запоминается системой. Вместо этого, используется его **хэш,** как при сравнении паролей на стадии регистрации, так и при сравнении паролей на стадии входа в систему.

### Команда /login и /logout

Эта команда была реализована для входа в систему, для проведения процессов аутентификации и авторизации.

При отправке боту команды **/login**, если пользователь уже вошел в систему, он получит следующее сообщение: (Рисунок 5)

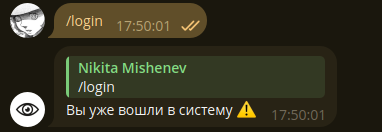


Рисунок 5 – ответ вошедшему пользователю на команду **/login**

Если же пользователь даже не зарегистрировался, то он получит следующее сообщение: (Рисунок 6)

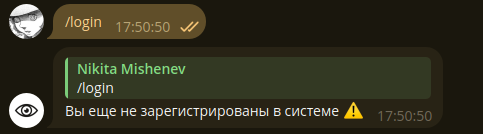


Рисунок 6 – ответ незарегистрированному пользователю на команду **/login**

В случае когда пользователь зарегистрировался, но не вошел в систему, запускается процесс авторизации: (Рисунок 7)

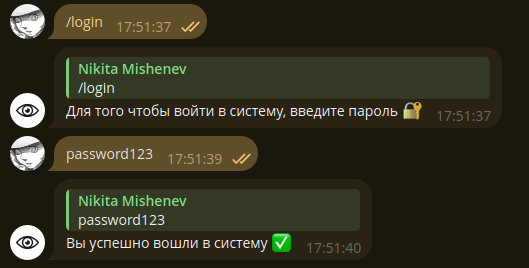


Рисунок 7 – процесс авторизации

Хэндлер, отслеживающий работу с этой командой, реализован следующим образом: (Листинг 6)

Листинг 6 – реализация хэндлера для команды **/login**

|  |
| --- |
| @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"]) |

При использовании команды **/logout,** пользователь выходит из системы, и не может больше пользоваться её возможностями, пока снова не пройдет процесс авторизации. Ответ авторизованному пользователю на эту команду, выглядит следующим образом: (Рисунок 8)

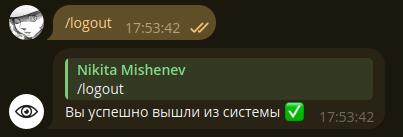


Рисунок 8 – ответ авторизованному пользователю на команду **/logout**

Хэндлер, для обработки этой команды, выглядит следующим образом: (Листинг 7)

Листинг 7 – реализация хэндлера для команды **/logout**

|  |
| --- |
| @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"]) |

Последняя доступная в боте команда, отвечающая за вызов классификатора, рассмотрена в следующем разделе.

## **Объединение Telegram-бота и классификатора.**

Объединение бота и классификатора проиходит при выполнении команды /predict. Если пользователь зарегистрирован и авторизован, то бот предлагает ему отправить картинку (Рисунок 9), иначе, говорит о невозможности продолжении диалога: (Рисунок 10)

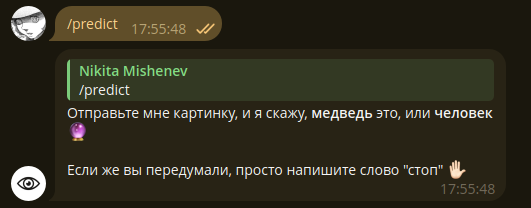


Рисунок 9 – ответ бота пользователю, авторизованному в системе

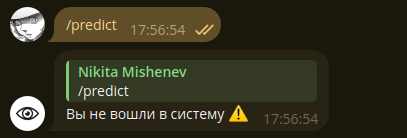


Рисунок 10 – ответ бота пользователю, не авторизованному в системе

После этого, в хендлере для этой команды, происходит сохранение фотографии от пользователя и ее дальнейшая классификация: (Листинг 8)

Листинг 8 – реализация соединения классификатора и бота

|  |
| --- |
| @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, \_ = recognize\_picture(filename, model)  match answer:  case 0:  await message.reply(REPLIES["is\_bear"])  case 1:  await message.reply(REPLIES["is\_human"])  case \_:  await message.reply(REPLIES["error"])  await message.reply(REPLIES["predict\_prompt"]) |

После проведения классификации, бот отправляет результат в ответном сообщении пользователю (Рисунок 11)

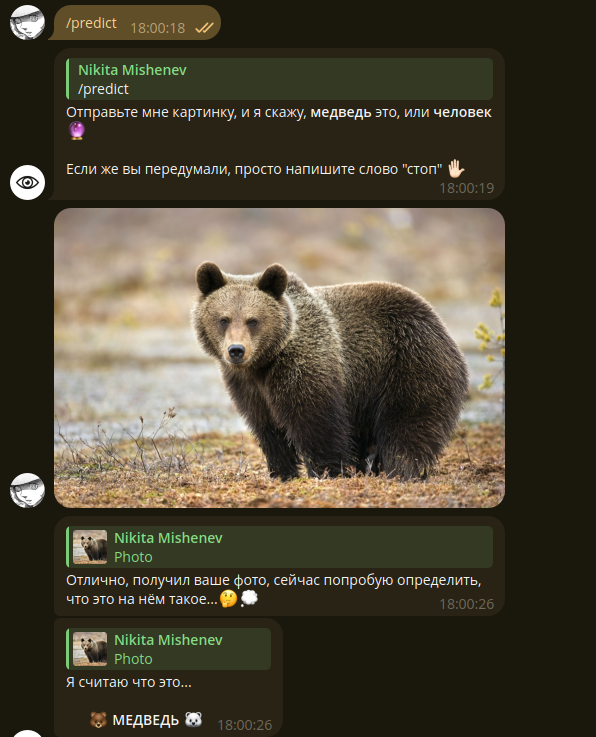


Рисунок 11 – ответ, после проведения классификации изображения

## Ответы на контрольные вопросы.

1. *Какие классы задач могут быть решены с помощью методов искусственного интеллекта?*

В наше время, подавляющее большинство классов задач могут быть решены с использованием ИИ. Например, классификация, предсказание и распознавание. В последнее время набирает популярность генерация различных типов медиа, таких как фотографии и видеоролики. Также существует возможность генерировать код для самый различных областей, путём составления соответствующего запроса для ИИ.

1. *Чем отличается обучающий набор данных от тестового?*

Обучающий набор данных, используется для изначального построения модели, настройки ее основных параметров. Тестовый же набор данных, используется для непредвзятой оценки окончательной модели, настроенной с помощью обучающего набора данных.

1. *Что такое признак в контексте методов искусственного интеллекта? Что такое метка в контексте методов искусственного интеллекта? В чем их разница?*

Признак в [машинном обучении](https://ru.wikipedia.org/wiki/Машинное_обучение) – индивидуальное измеримое свойство или характеристика наблюдаемого явления. Метка в машинном обучении – это описательный элемент, сообщающий модели, чем является отдельный элемент исследуемых данных, чтобы она могла учиться, основываясь на чем-то.

Таким образом, признак, просто описывает какую-то характеристику исследуемой единицы данных, а метка однозначно связывает единицу данных с конечным результатом.

1. *Чем методы глубокого обучения отличаются от других методов искусственного интеллекта?*

Методы глубокого обучения отличаются от других своей сложностью. Обычно, глубокое обучение используют для решения объемных/сложных задач, которые требуют гораздо больше ресурсов. Поэтому, при использовании глубокого обучения, чаще всего, алгоритмы более сложны, данных для обучения гораздо больше, а также сами эти данные гораздо сложнее перевести, например, в числовой вид.

1. *Из чего состоит слой в нейронной сети? Какие слои бывают? Что такое нейрон?*

Слой в нейронной сети состоит из некоторого количества нейронов. Слои бывают: входные, скрытые, выходные. Входной слой, получает информацию и передат ее в скрытый слой. В скрытом слое производятся основные вычисления нейронной сети. Далее, информация передаётся в выходной слой, количество нейронов в котором, соответствует количеству классов в задаче классификации. Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше.

1. *Что такое аутентификация?*

В процессе аутентификации, устанавливается личность пользователя. Даётся ответ на вопрос, а правда ли этот пользователь тот, за кого себя выдаёт (например, процесс проверки пароля от учётной записи).

1. *Что такое авторизация?*

В процессе авторизации, определяется, какими правами обладает аутентифицированный пользователь, что он может делать (например, к какой группе относится пользователь: администраторы, или посетители)

1. *Чем аутентификация отличается от авторизации?*

Этап аутентификации предшествует этапу авторизации. Именно на этапе аутентификации, пользователя могут не пустить в систему, если он не будет знать, например, секретное слово или пароль.

1. *Для чего нужен токен Telegram-бота?*

Токен telegram-бота, необходим для того, чтобы созданный бот, мог обращаться к серверам Telegram и получать информацию о произошедших событиях в чатах, к которым он имеет доступ. Такой токен можно получить у специального бота **@BotFather**, который предоставляет полную настройку каждого бота.

# **ВЫВОД**

В ходе работы были получены навыки работы с методами ИИ для решения задачи классификации данных с использованием языка программирования **Python**.

Были составлены выборки изображений для двух классов существ. Также был написан скрипт-классификатор, который определяет принадлежность полученной фотографии к одному из двух классов. Классификатор реализован на языке **Python** с помощью возможностей библиотеки **tensorflow.**

В ходе работы был написан Telegram-бот с использованием **aiogram3.** В нём реализованы функции, который осуществляют доступ к классификатору, а также реализована система пользователей, которая подразумевает регистрацию и ввод пароля, перед использованием возможностей бота.

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

**ПРИЛОЖЕНИЕ 1. ПРОГРАММНЫЙ КОД TELEGRAM-БОТА**

**app.py:**

|  |
| --- |
| import asyncio  from handlers import default  from loader import bot, dp  async def main():  dp.include\_router(default.router)  await dp.start\_polling(bot)  if \_\_name\_\_ == "\_\_main\_\_":  print("Bot has been launched...")  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 database.dbworker import create\_db\_engine  from scripts.predictor import tensorflow\_init  model = tensorflow\_init()  load\_dotenv(".env")  bot = Bot(  token=environ.get("TOKEN"),  default=DefaultBotProperties(  parse\_mode=ParseMode.HTML  )  )  engine = create\_db\_engine()  dp = Dispatcher(storage=MemoryStorage()) |

**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  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, \_ = recognize\_picture(filename, model)  match answer:  case 0:  await message.reply(REPLIES["is\_bear"])  case 1:  await message.reply(REPLIES["is\_human"])  case \_:  await message.reply(REPLIES["error"])  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) |

**dbworker.py:**

|  |
| --- |
| from typing import Optional  from sqlalchemy import create\_engine  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]  )  session.add(user)  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 😔",  "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 🐻‍❄️</b>",  "is\_human": "Я считаю что это...\n\n\t\t\t\t\t\t<b>🙋🏻‍♂️ ЧЕЛОВЕК 🙋🏻‍♀️</b>",  "error": "Ой, совсем не понимаю что это такое получилось...🫤"  } |

**ПРИЛОЖЕНИЕ 2. ПРОГРАММНЫЙ КОД КЛАССИФИКАТОРА**

**predictor.py:**

|  |
| --- |
| from typing import Any  import matplotlib.pyplot as plt  import numpy as np  import tensorflow as tf  from tensorflow.keras.callbacks import EarlyStopping # type: ignore  from tensorflow.keras.preprocessing import image # type: ignore  from tensorflow.keras.preprocessing.image import ImageDataGenerator # type: ignore  def tensorflow\_init() -> Any:  train\_datagen = ImageDataGenerator(rescale=1/255)  validation\_datagen = ImageDataGenerator(rescale=1/255)  train\_generator = train\_datagen.flow\_from\_directory(  './data/train/',  classes = ['bear', 'human'],  target\_size=(200, 200),  batch\_size=7,  class\_mode='binary'  )  validation\_generator = validation\_datagen.flow\_from\_directory(  './data/valid/',  classes = ['bear', 'human'],  target\_size=(200, 200),  batch\_size=20,  class\_mode='binary',  shuffle=False  )  callback = EarlyStopping(  monitor="loss",  mode="min",  restore\_best\_weights=True,  patience=4,  start\_from\_epoch=10,  verbose=1  )  model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input\_shape = (200,200,3)),  tf.keras.layers.Dense(128, activation=tf.nn.relu),  tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)])  model.summary()  model.compile(optimizer = tf.keras.optimizers.Adam(),  loss = 'binary\_crossentropy',  metrics=['accuracy'])  history = model.fit(  train\_generator,  steps\_per\_epoch=10,  epochs=15,  verbose=1,  validation\_data = validation\_generator,  validation\_steps=10,  callbacks=[callback]  )  return model  def recognize\_picture(filename: str, model: Any):  img = image.load\_img(filename, target\_size=(200, 200))  x = image.img\_to\_array(img)  plt.imshow(x / 255.)  x = np.expand\_dims(x, axis=0)  images = np.vstack([x])  classes = model.predict(images, batch\_size=10)  if classes[0] < 0.5:  return (0, classes[0][0]) # bear  else:  return (1, classes[0][0]) # human |