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»

Выполнили

студенты гр. 5131001/30002 Н.К. Красин

А.Д. Турченко

Руководитель

ассистент ВШК ИКНК М.С. Иванов

Санкт-Петербург – 2025

Оглавление

[1 Цель 3](#_Toc199347472)

[2 Задачи 4](#_Toc199347473)

[2.1 Усовершенствование Telegram-бота 4](#_Toc199347474)

[2.2 Перенос на webhook 4](#_Toc199347475)

[2.3 Внедрение 5](#_Toc199347476)

[3 Ход работы 6](#_Toc199347477)

[3.1 Недостатки хранения пользовательских данных 6](#_Toc199347478)

[3.1.1 Усовершенствование метода хранения данных 7](#_Toc199347479)

[3.1.2 Выбранные способы реализации 8](#_Toc199347480)

[3.1.3 Добавление роли администратора 11](#_Toc199347481)

[3.2 Перенос на webhook 13](#_Toc199347482)

[3.2.1 Изучение функционала webhook из telegram API 13](#_Toc199347483)

[3.2.2 Сервис pythonanywhere и фреймворк flask 14](#_Toc199347484)

[4 Ответы на контрольные вопросы 18](#_Toc199347485)

[5 Вывод 22](#_Toc199347486)

[Приложение 1 23](#_Toc199347487)

[Приложение 2 35](#_Toc199347488)

# Цель

Получить навыки создания Telegram-ботов на языке программирования Python.

# Задачи

## Усовершенствование Telegram-бота

1. Выделить в текущей реализации Telegram-бота из л.р. 3 недостатки хранения пользовательских данных. Описать их в отчёте, описать к чему они могут привести, предложить способы противодействовать выделенным недостаткам.
2. Усовершенствовать метод хранения данных в соответствии с выделенными недостатками.
3. Добавить роль администратора в Telegram-бота. Администратор через Telegram-бота способен:

* просмотреть список пользователей и количество выполненных для него предсказаний;
* удалить пользователя;
* добавить нового администратора.

Способ добавления первого администратора, отображение пользователей и выполнение действий остаётся на усмотрение студента (например, первым администратором становиться первый написавший боту или тот, кто введёт сгенерированное значение, отображение просто сообщением и для выбора действий отсылать номер пользователя или сообщение с кнопками для выбора действия).

1. В отчете описать выбранные способы реализации, их преимущества и недостатки. Привести примеры работы, описать ключевые моменты реализации.

## Перенос на webhook

1. Изучить функционал webhook из Telegram API (<https://core.telegram.org/bots/api#setwebhook>, <https://core.telegram.org/bots/webhooks>). Кратко описать основные положения.
2. При помощи сервиса pythonanywhere.com (либо любого другого сервиса для хостинга веб-приложений) создать веб-приложение, которое будет содержать конечные точки, выступающие в роли webhook’ов. В качестве фреймворка для веб-приложения рекомендуется использовать Flask (<https://flask.palletsprojects.com/en/2.3.x/quickstart/>). Описать выбранный сервис, описать выбранный фреймворк.
3. Веб-приложение должно содержать обработчик POST-запросов. На данный обработчик в дальнейшем будут приходить запросы от Telegram API в формате JSON.
4. Зарегистрировать созданный webhook в Telegram-боте при помощи Telegram API (setWebhook). Описать процесс регистрации.
5. Отправить сообщение в чат бота, использующего webhook. Ознакомиться с форматом полученного веб-приложением сообщения.
6. Реализовать обработку получаемых сообщений и отправку ответных сообщений при помощи API sendMessage (<https://core.telegram.org/bots/api#sendmessage>). Описать формат получаемых сообщений и как происходит отправка ответных сообщений.

## Внедрение

1. Перенести усовершенствованного Telegram-бота (функционал из л.р. 3 должен сохраниться) на webhook’и. Описать основные моменты переноса.

# Ход работы

В предоставленном коде Telegram-бота для распознавания изображений (панд и людей) пользовательские данные хранятся в словаре users в оперативной памяти. Такой подход имеет несколько недостатков с точки зрения безопасности, масштабируемости и надежности. В данном отчёте выделены основные недостатки текущей реализации хранения данных, описаны потенциальные последствия этих недостатков и предложены способы их устранения.

## Недостатки хранения пользовательских данных

1. Хранение паролей в виде обычного текста

Описание: Пароли пользователей хранятся в словаре users в виде обычного текста (например, users[chat\_id] = {'password': password, 'logged\_in': False}). Это означает, что пароли не шифруются и не хешируются, что делает их уязвимыми для утечки.

Последствия:

* Утечка паролей при компрометации оперативной памяти сервера или дампа процесса.
* Возможность использования украденных паролей в других системах, если пользователи их повторно используют.
* Нарушение конфиденциальности, что может привести к потере доверия и юридическим последствиям.

Способы устранения:

* Использовать криптографически стойкое хеширование паролей (например, SHA-256 с солью или bcrypt).
* Применить библиотеку bcrypt для безопасного хранения паролей.

1. Хранение данных в оперативной памяти

Описание: Данные пользователей (пароли, статус входа) хранятся в словаре users в оперативной памяти, который сбрасывается при перезапуске бота.

Последствия:

* Потеря всех пользовательских данных при перезапуске бота.
* Ограниченная масштабируемость из-за роста потребления памяти.
* Отсутствие возможности резервного копирования или восстановления данных.

Способы устранения:

* Перейти на использование базы данных SQLite для постоянного хранения данных.
* Сохранять данные в JSON-файл при каждом изменении и загружать их при запуске.

1. Отсутствие ограничения на попытки входа

Описание: Нет ограничений на количество попыток ввода пароля при использовании команды /login, что делает систему уязвимой для атак brute-force.

Последствия:

* Риск компрометации учетных записей из-за перебора паролей.
* Возможная перегрузка бота при автоматизированных атаках.

Способы устранения:

* Ввести ограничение на количество попыток входа с временной блокировкой.
* Добавить CAPTCHA для предотвращения автоматизированных атак.
* Вести журнал попыток входа для анализа.

### Усовершенствование метода хранения данных

В обновлённой версии Telegram-бота для распознавания панд и людей реализованы улучшения в системе хранения пользовательских данных. Основные изменения включают использование базы данных SQLite, хеширование паролей с солью и ограничение попыток входа. В данном отчёте описаны выбранные способы реализации, их преимущества и недостатки, приведены примеры работы и ключевые моменты реализации.

### Выбранные способы реализации

1. Хранение пользовательских данных в SQLite

Описание реализации: Данные пользователей (chat\_id, пароль, статус входа, количество предсказаний) и администраторов хранятся в базе данных SQLite (users.db). Используются таблицы users и admins с SQL-запросами для управления данными. Пример программы представлен в листинге Листинг 1.

Листинг 1 - Хранение пользовательских данных в SQLite

|  |
| --- |
| conn = sqlite3.connect('users.db', check\_same\_thread=False)  cursor = conn.cursor()  cursor.execute('''CREATE TABLE IF NOT EXISTS users  (chat\_id INTEGER PRIMARY KEY,  password TEXT,  logged\_in BOOLEAN,  predictions\_count INTEGER DEFAULT 0)''')  cursor.execute('''CREATE TABLE IF NOT EXISTS admins  (chat\_id INTEGER PRIMARY KEY,  username TEXT)''')  conn.commit() |

Преимущества:

* Постоянное хранение данных на диске.
* Поддержка масштабируемости для умеренного числа пользователей.
* Простота интеграции без необходимости внешнего сервера.
* Гибкость при добавлении новых столбцов.

Недостатки:

* Ограниченная производительность при больших объёмах данных.
* Отсутствие транзакций в текущей реализации.
* Локальное хранение данных.

1. Хеширование паролей с использованием соли

Описание реализации: Пароли хешируются с помощью SHA-256 и случайной соли, генерируемой secrets.token\_hex. Хеш и соль сохраняются в формате salt:hashed. Пример программы представлен в листинге Листинг 2.

Листинг 2 - Хранение паролей с использованием соли

|  |
| --- |
| def hash\_password(password):  salt = secrets.token\_hex(8)  hashed = hashlib.sha256((password + salt).encode()).hexdigest()  return f"{salt}:{hashed}"  def check\_password(hashed\_password, user\_password):  salt, hashed = hashed\_password.split(':')  return hashed == hashlib.sha256((user\_password + salt).encode()).hexdigest() |

Преимущества:

* Защита от атак с использованием радужных таблиц.
* Простота реализации с использованием стандартных библиотек.
* Уникальность хешей благодаря соли.

Недостатки:

* SHA-256 менее устойчив к brute-force, чем bcrypt.
* Отсутствие адаптивной сложности вычислений.

1. Ограничение попыток входа

Описание реализации: После 3 неудачных попыток входа пользователь блокируется на 5 минут. Данные хранятся в словаре login\_attempts. Пример программы представлен в листинге Листинг 3.

Листинг 3 - Ограничение попыток входа

|  |
| --- |
| login\_attempts = {}  def login(message):  chat\_id = message.chat.id  if chat\_id in login\_attempts and login\_attempts[chat\_id]['attempts'] >= 3:  if datetime.now() < login\_attempts[chat\_id]['block\_time']:  remaining = (login\_attempts[chat\_id]['block\_time'] - datetime.now()).seconds // 60  bot.send\_message(chat\_id, f"🔒 Слишком много попыток. Подождите {remaining} минут.")  return  user\_states[chat\_id] = 'logging\_in'  bot.send\_message(chat\_id, "Введите пароль для входа:")  # В обработчике текстовых сообщений  if state == 'logging\_in':  cursor.execute("SELECT password FROM users WHERE chat\_id=?", (chat\_id,))  result = cursor.fetchone()  if result and check\_password(result[0], text):  cursor.execute("UPDATE users SET logged\_in=1 WHERE chat\_id=?", (chat\_id,))  conn.commit()  login\_attempts.pop(chat\_id, None)  bot.send\_message(chat\_id, "🔓 Вход выполнен успешно! Используйте /predict.")  else:  login\_attempts.setdefault(chat\_id, {'attempts': 0, 'block\_time': datetime.now()})  login\_attempts[chat\_id]['attempts'] += 1  if login\_attempts[chat\_id]['attempts'] >= 3:  login\_attempts[chat\_id]['block\_time'] = datetime.now() + timedelta(minutes=5)  remaining\_attempts = 3 - login\_attempts[chat\_id]['attempts']  bot.send\_message(chat\_id, f"❌ Неверный пароль. Осталось попыток: {remaining\_attempts}")  user\_states.pop(chat\_id, None) |

Преимущества:

* Защита от brute-force атак.
* Динамическая блокировка с таймером.
* Простота реализации.

Недостатки:

* Хранение данных в оперативной памяти.
* Отсутствие логирования попыток.
* Фиксированное время блокировки.

1. Ключевые моменты реализации
2. Инициализация SQLite: Использование check\_same\_thread=False для многопоточной среды и проверка структуры таблицы для обратной совместимости.
3. Хеширование паролей: Применение SHA-256 с солью для безопасности.
4. Ограничение попыток: Реализация счётчика и таймера блокировки в словаре login\_attempts.
5. Администрирование: Управление пользователями и администраторами через SQL-запросы.
6. Обработка ошибок: Проверка корректности ввода ID для предотвращения сбоев.

### Добавление роли администратора

В Telegram-боте реализован функционал управления администраторами через команду /add\_admin, а также функции is\_admin, has\_admins, add\_admin и delete\_user. Список пользователей отправляется с указанием chat\_id и predictions\_count. В отчёте описаны эти реализации, их преимущества и недостатки, примеры работы и ключевые моменты.

#### Реализация в коде

1. Команда/add\_admin:

* Доступна администраторам, запрашивает chat\_id нового администратора.
* Проверяет регистрацию пользователя и добавляет его в таблицу admins.

1. Список пользователей:

* Команда /list\_users возвращает текстовое сообщение с chat\_id, predictions\_count, статусом и ролью.

1. Функции администрирования:

* is\_admin(chat\_id): Проверяет наличие пользователя в таблице admins.
* has\_admins(): Проверяет наличие администраторов.
* add\_admin(chat\_id, username): Добавляет администратора в таблицу admins.
* delete\_user(user\_id): Удаляет пользователя из таблиц users и admins.

На Рисунке Рисунок 1 представлен запуск бота. На рисунке 2 представлен пример работы администратора.

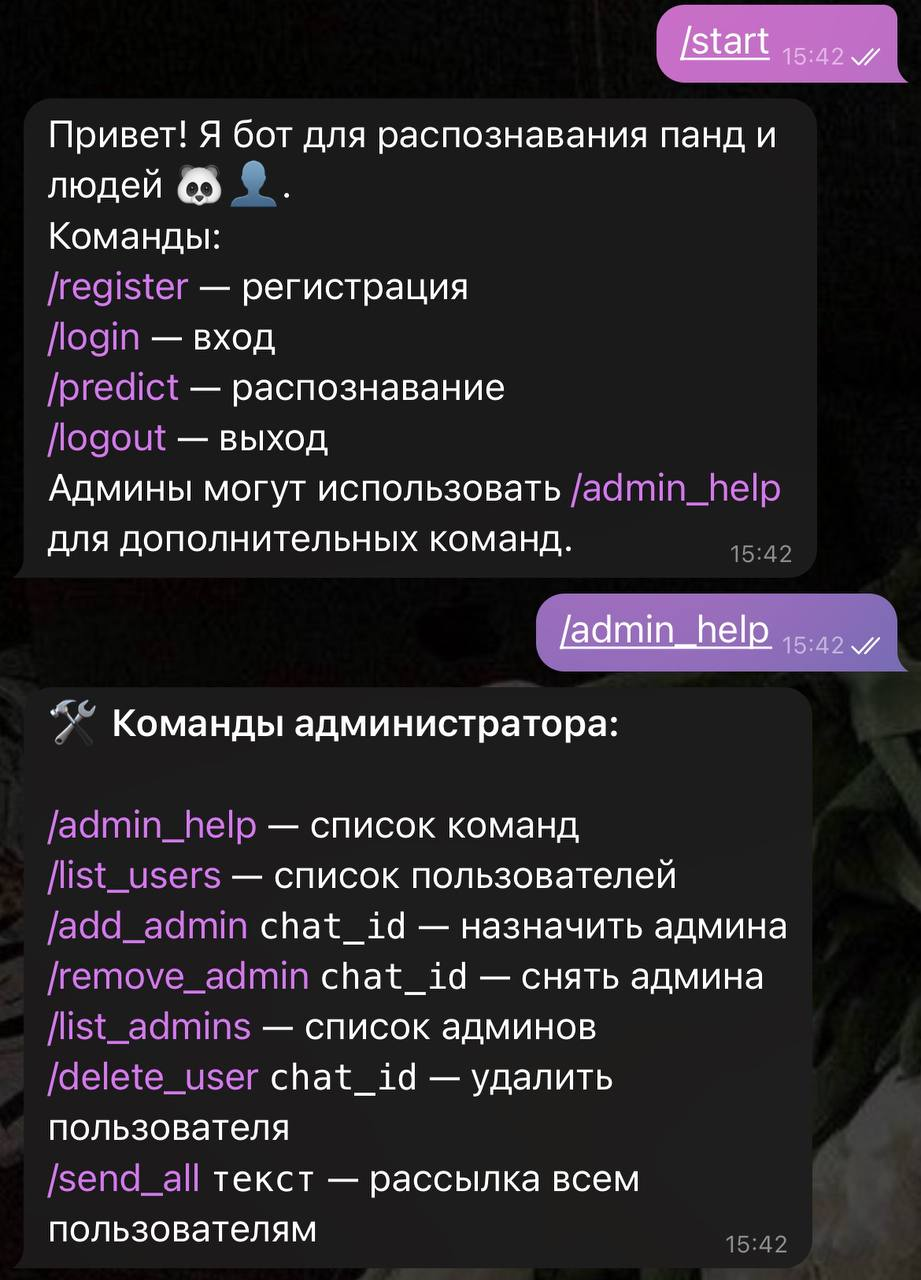


Рисунок 1 - Запуск бота

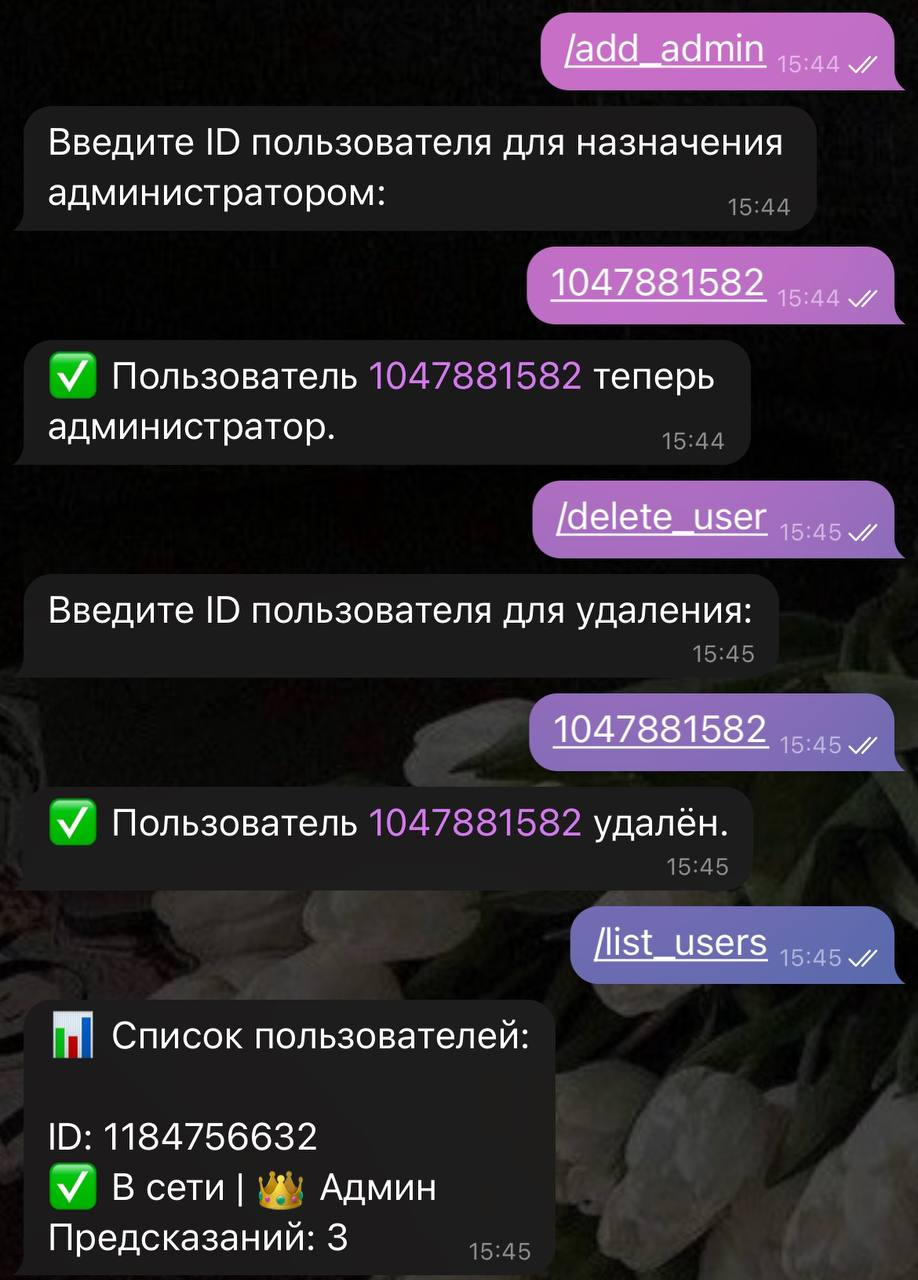


Рисунок 2 - Пример работы администратора

Полный модифицированный код Telegram-бота представлен в приложении 1.

## Перенос на webhook

### Изучение функционала webhook из telegram API

Webhook в Telegram API — это механизм, при котором сервер Telegram отправляет данные боту сразу, как только появляются новые обновления. Бот при этом не должен запрашивать данные — он ожидает «входящего» запроса.

Некоторые основные положения webhook:

* Мгновенная доставка. Обновления отправляются моментально, что улучшает отзывчивость бота.
* Экономия ресурсов. Так как бот получает обновления только в случае их появления, серверу бота не требуется поддерживать постоянные запросы к API Telegram.
* Асинхронность. Webhook подходит для приложений, работающих на асинхронных фреймворках, благодаря своей модели событийного взаимодействия.

Работа в несколько шагов:

* Регистрация вебхука. При запуске бота он должен зарегистрировать свой вебхук на сервере Telegram, передав ему URL, по которому будут поступать обновления.
* Обработка обновлений. Когда пользователь отправляет сообщение или команду боту, Telegram направляет запрос на зарегистрированный URL вебхука, содержащий все необходимые данные.
* Ответ на обновление. После обработки запроса бот может отправить ответ пользователю (например, сообщение с подтверждением выполнения команды), используя методы API Telegram.

Для передачи обновлений через webhook требуется **защищённое HTTPS-соединение**. Это необходимо для предотвращения перехвата данных между сервером Telegram и ботом.

### Сервис pythonanywhere и фреймворк flask

#### Описание сервиса pythonanywhere

PythonAnywhere — это онлайн-интегрированная среда разработки и служба веб-хостинга, основанная на языке программирования Python. Для работы нужны только доступ в интернет и браузер. Сервис подходит для работы на Mac, Windows и Linux. Основные особенности:

* Бесплатный тарифный план (с ограничениями).
* Предустановленные популярные Python-библиотеки.
* Простое развертывание веб-приложений.
* Встроенный терминал и редактор кода.
* Поддержка Flask, Django и других фреймворков.
* Возможность работы по расписанию (cron-задачи).
* Доступ к базе данных MySQL.

Идеально подходит для небольших проектов и образовательных целей.

#### Описание фреймворка Flask

Flask - это минималистичный веб-фреймворк на Python, предназначенный для создания веб-приложений и REST API. Он предоставляет простую и гибкую структуру, позволяющую быстро разработать как прототип, так и полнофункциональное веб-приложение. Ключевые характеристики:

* Минималистичный и гибкий.
* Простая настройка маршрутов (роутинга).
* Встроенный сервер для разработки.
* Поддержка шаблонов Jinja2.
* Легкая интеграция с расширениями.
* Подходит для создания RESTful API и вебхуков.

1. Настройка Flask и вебхука:

Описание: Бот использует Flask для создания веб-приложения, которое обрабатывает POST-запросы от Telegram API. Вебхук настроен на URL https://nikola.pythonanywhere.com/webhook. При запуске приложения существующий вебхук удаляется, и устанавливается новый с помощью bot.set\_webhook(url=WEBHOOK\_URL). Flask-сервер запускается для обработки входящих запросов. Пример программы представлен в листинге Листинг 4.

Листинг 4 - Настройка Flask и вебхука

|  |
| --- |
| from flask import Flask, request  TOKEN = 'YOUR\_NEW\_TOKEN'  WEBHOOK\_URL = 'https://nikola.pythonanywhere.com/webhook'  app = Flask(\_\_name\_\_)  bot = telebot.TeleBot(TOKEN)  @app.route('/webhook', methods=['POST'])  def webhook():  if request.headers.get('content-type') == 'application/json':  json\_string = request.get\_data().decode('utf-8')  update = telebot.types.Update.de\_json(json\_string)  bot.process\_new\_updates([update])  return 'OK', 200  return 'Invalid content type', 400  bot.delete\_webhook()  bot.set\_webhook(url=WEBHOOK\_URL)  if \_\_name\_\_ == '\_\_main\_\_':  app.run() |

**Обработчик POST-запросов:**

* Маршрут /webhook принимает только POST-запросы (methods=['POST']).
* Проверяется, что заголовок content-type равен application/json, что соответствует формату запросов от Telegram API.
* Тело запроса (JSON-данные) декодируется в строку (request.get\_data().decode('utf-8')).
* JSON преобразуется в объект Update с помощью telebot.types.Update.de\_json.
* Обновление передаётся в bot.process\_new\_updates, который вызывает соответствующие обработчики (например, для команд /start или /add\_admin).
* Возвращается ответ OK с кодом 200 для подтверждения успешной обработки. Если тип контента неверный, возвращается ошибка с кодом 400.

1. Интеграция с Telegram API:

* Механизм работы: После настройки вебхука через bot.set\_webhook, Telegram отправляет все обновления (сообщения, команды, фотографии) на указанный URL в формате JSON. Пример JSON-запроса от Telegram: Пример программы представлен в листинге Листинг 5.

Листинг 5 - Интеграция с Telegram API

|  |
| --- |
| {  "update\_id": 123456789,  "message": {  "message\_id": 100,  "chat": {  "id": 123456,  "type": "private"  },  "date": 1625068800,  "text": "/start"  }  } |

* Flask-сервер обрабатывает этот JSON, и библиотека pyTelegramBotAPI преобразует его в объект Update, который передаётся в обработчики команд или сообщений.

1. Адаптация для PythonAnywhere:

Описание: Код адаптирован для работы на PythonAnywhere, где файлы базы данных (users.db), весов модели (best\_weights.h5) и временных изображений хранятся в директории /home/user/www/. Пример программы представлен в листинге Листинг 6.

Листинг 6 - Адаптация для PythonAnywhere

|  |
| --- |
| conn = sqlite3.connect('/home/user/www/users.db', check\_same\_thread=False)  model.load\_weights('/home/user/www/best\_weights.h5')  img\_path = f"/home/user/www/temp\_{chat\_id}.jpg" |

Это обеспечивает совместимость с серверной средой, где пути к файлам отличаются от локальных.

# Ответы на контрольные вопросы

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

При реализации Telegram-бота через polling будут следующие преимущества:

* Не требует публичного сервера (можно запускать локально).
* Проще в настройке (не нужен HTTPS).
* Подходит для тестирования и разработки.

Также будут следующие недостатки:

* Задержки в получении сообщений (бот периодически опрашивает сервер).
* Высокая нагрузка на сервер при частых запросах.
* Ограниченная масштабируемость. Увеличение нагрузки на сервер с ростом числа пользователей.

При реализации Telegram-бота через webhook’и будут следующие преимущества:

* Почти мгновенная доставка сообщений (Telegram сам отправляет данные).
* Низкая нагрузка на сервер (сервер ждёт уведомлений).
* Эффективен для массового использования.
* Высокая масштабируемость. Эффективен при большом количестве пользователей.

Также будут следующие недостатки:

* Требуется публичный HTTPS-сервер (нельзя использовать локально без туннеля).
* Сложнее в настройке (нужен домен, SSL-сертификат).
* Зависимость от стабильности веб-сервера. [3].

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

Хэш-функции играют ключевую роль в безопасном хранении паролей. Вот зачем они нужны:

* Защита от утечек. Если злоумышленник получит доступ к базе данных с паролями в открытом виде, он сможет ими воспользоваться. Хэширование превращает пароли в нечитаемую строку, и даже при утечке восстановить исходный пароль сложно.
* Невозможность обратного преобразования**.**  Существует множество параметров, по которым определяется, считается ли хэш-функция хорошей. В первую очередь, она должна быстро вычисляться и давать минимальное количество коллизий. Примерами не очень хороших (содержащих большое количество коллизий) хэш-функций могут служить простейшие XOR-hash, rotating hash, Bernstein hash [2].
* Защита от rainbow-таблиц. Если хэшировать пароли без дополнительных мер, злоумышленники могут использовать заранее вычисленные таблицы хэшей (rainbow tables). Чтобы этого избежать, применяют "соль" (salt) — случайные данные, добавляемые к паролю перед хэшированием.
* Устойчивость к brute-force. Современные хэш-функции (например, bcrypt) намеренно медленные и ресурсоёмкие, что затрудняет подбор пароля перебором [1].
* Единообразие хранения. Даже если два пользователя имеют одинаковые пароли, их хэши (с разной "солью") будут разными, что повышает безопасность.

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

«Белый» IP-адрес (также называют публичным, внешним, глобальным) — это уникальный адрес в интернете, который позволяет устройствам напрямую взаимодействовать с другими серверами и пользователями в глобальной сети. Белые IP-адреса используются для:

* Размещение серверов. Если у вас есть веб-сайт, игровой сервер, VPN или почтовый сервис, нужен белый IP, чтобы к нему могли подключаться пользователи из интернета.
* Удалённый доступ к устройствам. Позволяет подключаться к камерам видеонаблюдения, промышленному оборудованию из любой точки мира.
* Игровые серверы и P2P. Некоторые онлайн-игры (например, Minecraft, Counter-Strike) требуют белого IP для создания собственного сервера.
* Обход NAT и ограничений. Без белого IP некоторые сервисы (например, VoIP, торренты) могут работать некорректно из-за преобразования адресов (NAT).
* Фиксированный адрес для бизнеса. Корпоративные сервисы (CRM, 1С) часто требуют статического белого IP для безопасного доступа. [6].

1. Для чего применяются базы данных?

Базы данных применяются для таких целей, как

* Хранение данных. Сохранение информации в структурированном виде (например, данные пользователей, товары в интернет-магазине, транзакции банка).
* Быстрый поиск и доступ. Позволяют мгновенно находить нужные данные (например, поиск книги в библиотеке или клиента в CRM).
* Обработка и анализ данных. Поддержка сложных запросов (аналитика продаж, отчёты, машинное обучение).
* Обеспечение целостности данных. Защита от дублирования, ошибок и потери информации (например, нельзя удалить клиента, у которого есть заказы).
* Многопользовательский доступ и безопасность. Разграничение прав (админ, модератор, пользователь) и защита от несанкционированного доступа.
* Резервное копирование и восстановление. Возможность восстановить данные после сбоя или атаки. [4].

1. Для чего применяется JSON формат файлов?

JSON формат файлов применяется в

* Передача данных между сервером и клиентом.
* Конфигурация приложений и программ.
* Хранение структурированных данных .
* Обмен данными между языками программирования, поскольку JSON поддерживается почти всеми языками программирования. [5].

# Вывод

В ходе лабораторной работы были получены навыки создания Telegram-ботов на языке программирования Python. Был модифицирован и улучшен механизм хранения пользовательских данных, созданный в третьей лабораторной работе. Был изучен функционал webhook из Telegram API и фреймворк Flask. Также получены и освоены, необходимые для создания web-приложений, содержащих обработчики POST-запросов, и необходимые для правильной регистрации созданного webhook’а в Telegram-боте, навыки. К сожалению, не получилось реализовать внедрение модифицированного бота на webhook’и из-за ограничений хостинг сервиса.

# Приложение 1

Листинг кода Telegram-бота:

import telebot

from telebot import types

import tensorflow as tf

import numpy as np

import os

import sqlite3

import hashlib

import secrets

from datetime import datetime, timedelta

from tensorflow.keras.preprocessing import image

# Конфигурация бота

TOKEN = '7593523748:AAGtmDsBVFQ1fVN2X\_yKHtJphDPHJRafefY'

bot = telebot.TeleBot(TOKEN)

# Настройка базы данных SQLite

conn = sqlite3.connect('users.db', *check\_same\_thread*=False)

cursor = conn.cursor()

# Создание таблицы пользователей, если она не существует

cursor.execute('''CREATE TABLE IF NOT EXISTS users

                 (chat\_id INTEGER PRIMARY KEY,

                  password TEXT,

                  logged\_in BOOLEAN,

                  predictions\_count INTEGER DEFAULT 0)''')

# Создание таблицы администраторов, если она не существует

cursor.execute('''CREATE TABLE IF NOT EXISTS admins

                 (chat\_id INTEGER PRIMARY KEY,

                  username TEXT)''')

conn.commit()

# Проверка наличия столбца predictions\_count в таблице пользователей

cursor.execute("PRAGMA table\_info(users)")

columns = [row[1] for row in cursor.fetchall()]

if "predictions\_count" not in columns:

    cursor.execute("ALTER TABLE users ADD COLUMN predictions\_count INTEGER DEFAULT 0")

    conn.commit()

# Управление состояниями пользователей

user\_states = {}

login\_attempts = {}

# Настройка модели TensorFlow

model = tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(32, (3, 3), *activation*='relu', *input\_shape*=(200, 200, 3)),

    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), *activation*='relu'),

    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, *activation*='relu'),

    tf.keras.layers.Dense(1, *activation*='sigmoid')

])

model.compile(*optimizer*='adam', *loss*='binary\_crossentropy', *metrics*=['accuracy'])

model.load\_weights('best\_weights.h5')

# Обработка паролей

*def* hash\_password(*password*):

    salt = secrets.token\_hex(8)

    hashed = hashlib.sha256((password + salt).encode()).hexdigest()

    return *f*"{salt}:{hashed}"

*def* check\_password(*hashed\_password*, *user\_password*):

    salt, hashed = hashed\_password.split(':')

    return hashed == hashlib.sha256((user\_password + salt).encode()).hexdigest()

# Проверка статуса администратора

*def* is\_admin(*chat\_id*):

    cursor.execute("SELECT \* FROM admins WHERE chat\_id=?", (chat\_id,))

    return cursor.fetchone() is not None

# Клавиатуры

*def* create\_main\_keyboard():

    markup = types.ReplyKeyboardMarkup(*resize\_keyboard*=True)

    markup.row(types.KeyboardButton('/register'), types.KeyboardButton('/login'))

    markup.row(types.KeyboardButton('/predict'), types.KeyboardButton('/logout'))

    markup.row(types.KeyboardButton('/user\_info'))

    return markup

*def* create\_admin\_keyboard():

    markup = types.ReplyKeyboardMarkup(*resize\_keyboard*=True)

    markup.row(types.KeyboardButton('/list\_users'), types.KeyboardButton('/delete\_user'))

    markup.row(types.KeyboardButton('/add\_admin'), types.KeyboardButton('/admin\_exit'))

    return markup

# Обработчики команд

@bot.message\_handler(*commands*=['start'])

*def* start(*message*):

    chat\_id = message.chat.id

    # Автоматическое назначение первого пользователя администратором, если администраторов нет

    cursor.execute("SELECT COUNT(\*) FROM admins")

    if cursor.fetchone()[0] == 0:

        cursor.execute("INSERT INTO admins VALUES (?, ?)",

                      (chat\_id, message.from\_user.username or *str*(chat\_id)))

        conn.commit()

        bot.send\_message(chat\_id, "👑 Вы стали первым администратором!")

    welcome\_message = (

        "Привет! Я бот для распознавания панд и людей 🐼👤.\n"

        "Команды:\n"

        "/register — регистрация\n"

        "/login — вход\n"

        "/predict — вывод приветствия\n"

        "/user\_info — информация о пользователе\n"

        "/logout — выход\n"

        "Админы могут использовать /admin\_help для дополнительных команд."

    )

    bot.send\_message(chat\_id, welcome\_message, *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*commands*=['admin\_help'])

*def* admin\_help(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "🚫 Доступ запрещён. Эта команда только для администраторов.")

        return

    admin\_commands = """

🛠 <b>Команды администратора:</b>

/admin\_help — список команд

/list\_users — список пользователей

/add\_admin <code>chat\_id</code> — назначить админа

/remove\_admin <code>chat\_id</code> — снять админа

/list\_admins — список админов

/delete\_user <code>chat\_id</code> — удалить пользователя

/send\_all <code>текст</code> — рассылка всем пользователям

"""

    bot.send\_message(chat\_id, admin\_commands, *parse\_mode*="HTML", *reply\_markup*=create\_admin\_keyboard())

@bot.message\_handler(*commands*=['admin'])

*def* admin\_panel(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    bot.send\_message(chat\_id, "Панель администратора:", *reply\_markup*=create\_admin\_keyboard())

@bot.message\_handler(*commands*=['admin\_exit'])

*def* admin\_exit(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    bot.send\_message(chat\_id, "Выход из панели администратора.", *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*commands*=['list\_users'])

*def* list\_users(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    cursor.execute('''SELECT u.chat\_id, u.logged\_in, u.predictions\_count,

                      a.chat\_id IS NOT NULL as is\_admin

                      FROM users u LEFT JOIN admins a ON u.chat\_id = a.chat\_id''')

    users = cursor.fetchall()

    if not users:

        bot.send\_message(chat\_id, "Нет зарегистрированных пользователей.")

        return

    response = "📊 Список пользователей:\n\n"

    for user in users:

        status = "✅ В сети" if user[1] else "❌ Не в сети"

        role = "👑 Админ" if user[3] else "👤 Пользователь"

        response += *f*"ID: {user[0]}\n{status} | {role}\nПредсказаний: {user[2]}\n\n"

    bot.send\_message(chat\_id, response)

@bot.message\_handler(*commands*=['delete\_user'])

*def* delete\_user(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_user\_id\_for\_deletion'

    bot.send\_message(chat\_id, "Введите ID пользователя для удаления:")

@bot.message\_handler(*commands*=['add\_admin'])

*def* add\_admin(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_admin\_id'

    bot.send\_message(chat\_id, "Введите ID пользователя для назначения администратором:")

@bot.message\_handler(*commands*=['remove\_admin'])

*def* remove\_admin(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_admin\_id\_for\_removal'

    bot.send\_message(chat\_id, "Введите ID администратора для снятия прав:")

@bot.message\_handler(*commands*=['list\_admins'])

*def* list\_admins(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    cursor.execute("SELECT chat\_id, username FROM admins")

    admins = cursor.fetchall()

    if not admins:

        bot.send\_message(chat\_id, "Нет администраторов.")

        return

    response = "👑 Список администраторов:\n\n"

    for admin in admins:

        response += *f*"ID: {admin[0]} | Имя: {admin[1] or 'Не указано'}\n"

    bot.send\_message(chat\_id, response)

@bot.message\_handler(*commands*=['send\_all'])

*def* send\_all(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_broadcast\_message'

    bot.send\_message(chat\_id, "Введите сообщение для рассылки всем пользователям:")

@bot.message\_handler(*commands*=['register'])

*def* register(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT \* FROM users WHERE chat\_id=?", (chat\_id,))

    if cursor.fetchone():

        bot.send\_message(chat\_id, "Вы уже зарегистрированы! Используйте /login.")

        return

    user\_states[chat\_id] = 'registering'

    bot.send\_message(chat\_id, "Введите пароль для регистрации:")

@bot.message\_handler(*commands*=['login'])

*def* login(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT \* FROM users WHERE chat\_id=?", (chat\_id,))

    if not cursor.fetchone():

        bot.send\_message(chat\_id, "Сначала зарегистрируйтесь через /register!")

        return

    if chat\_id in login\_attempts and login\_attempts[chat\_id]['attempts'] >= 3:

        if datetime.now() < login\_attempts[chat\_id]['block\_time']:

            remaining = (login\_attempts[chat\_id]['block\_time'] - datetime.now()).seconds // 60

            bot.send\_message(chat\_id, *f*"🔒 Слишком много попыток. Подождите {remaining} минут.")

            return

    user\_states[chat\_id] = 'logging\_in'

    bot.send\_message(chat\_id, "Введите пароль для входа:")

@bot.message\_handler(*commands*=['logout'])

*def* logout(*message*):

    chat\_id = message.chat.id

    cursor.execute("UPDATE users SET logged\_in=0 WHERE chat\_id=?", (chat\_id,))

    conn.commit()

    bot.send\_message(chat\_id, "Вы вышли из системы.", *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*commands*=['predict'])

*def* predict\_command(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT logged\_in FROM users WHERE chat\_id=?", (chat\_id,))

    result = cursor.fetchone()

    if not result:

        bot.send\_message(chat\_id, "Сначала зарегистрируйтесь через /register!")

        return

    if not result[0]:

        bot.send\_message(chat\_id, "Сначала выполните /login!")

        return

    # Увеличение счетчика предсказаний

    cursor.execute("UPDATE users SET predictions\_count = predictions\_count + 1 WHERE chat\_id=?", (chat\_id,))

    conn.commit()

    bot.send\_message(chat\_id, "привет")

@bot.message\_handler(*commands*=['user\_info'])

*def* user\_info(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT logged\_in, predictions\_count FROM users WHERE chat\_id=?", (chat\_id,))

    user = cursor.fetchone()

    if not user:

        bot.send\_message(chat\_id, "Сначала зарегистрируйтесь через /register!")

        return

    status = "✅ В сети" if user[0] else "❌ Не в сети"

    is\_admin\_status = "👑 Админ" if is\_admin(chat\_id) else "👤 Пользователь"

    response = (

*f*"📋 Информация о пользователе:\n\n"

*f*"ID: {chat\_id}\n"

*f*"Статус: {status}\n"

*f*"Роль: {is\_admin\_status}\n"

*f*"Предсказаний: {user[1]}"

    )

    bot.send\_message(chat\_id, response, *reply\_markup*=create\_main\_keyboard())

# Обработчик текстовых сообщений

@bot.message\_handler(*content\_types*=['text'])

*def* handle\_text(*message*):

    chat\_id = message.chat.id

    text = message.text.strip()

    state = user\_states.get(chat\_id)

    if state == 'registering':

        hashed\_password = hash\_password(text)

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

                       (chat\_id, hashed\_password, False, 0))

        conn.commit()

        user\_states.pop(chat\_id, None)

        bot.send\_message(chat\_id, "✅ Регистрация успешна! Используйте /login.")

    elif state == 'logging\_in':

        cursor.execute("SELECT password FROM users WHERE chat\_id=?", (chat\_id,))

        result = cursor.fetchone()

        if result and check\_password(result[0], text):

            cursor.execute("UPDATE users SET logged\_in=1 WHERE chat\_id=?", (chat\_id,))

            conn.commit()

            login\_attempts.pop(chat\_id, None)

            bot.send\_message(chat\_id, "🔓 Вход выполнен успешно! Используйте /predict.")

        else:

            login\_attempts.setdefault(chat\_id, {'attempts': 0, 'block\_time': datetime.now()})

            login\_attempts[chat\_id]['attempts'] += 1

            if login\_attempts[chat\_id]['attempts'] >= 3:

                login\_attempts[chat\_id]['block\_time'] = datetime.now() + timedelta(*minutes*=5)

            remaining\_attempts = 3 - login\_attempts[chat\_id]['attempts']

            bot.send\_message(chat\_id, *f*"❌ Неверный пароль. Осталось попыток: {remaining\_attempts}")

        user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_user\_id\_for\_deletion':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        try:

            user\_id = *int*(text)

            if user\_id == chat\_id:

                bot.send\_message(chat\_id, "❌ Нельзя удалить самого себя!")

            else:

                cursor.execute("DELETE FROM users WHERE chat\_id=?", (user\_id,))

                cursor.execute("DELETE FROM admins WHERE chat\_id=?", (user\_id,))

                conn.commit()

                bot.send\_message(chat\_id, *f*"✅ Пользователь {user\_id} удалён.")

        except *ValueError*:

            bot.send\_message(chat\_id, "❌ Неверный формат ID. Введите число.")

        finally:

            user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_admin\_id':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        try:

            new\_admin\_id = *int*(text)

            cursor.execute("SELECT \* FROM users WHERE chat\_id=?", (new\_admin\_id,))

            if not cursor.fetchone():

                bot.send\_message(chat\_id, "❌ Пользователь не найден. Он должен зарегистрироваться.")

            else:

                cursor.execute("INSERT OR IGNORE INTO admins VALUES (?, ?)",

                               (new\_admin\_id, *f*"admin\_{new\_admin\_id}"))

                conn.commit()

                bot.send\_message(chat\_id, *f*"✅ Пользователь {new\_admin\_id} теперь администратор.")

                bot.send\_message(new\_admin\_id, "👑 Вас назначили администратором!")

        except *ValueError*:

            bot.send\_message(chat\_id, "❌ Неверный формат ID. Введите число.")

        finally:

            user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_admin\_id\_for\_removal':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        try:

            admin\_id = *int*(text)

            if admin\_id == chat\_id:

                bot.send\_message(chat\_id, "❌ Нельзя снять права с самого себя!")

            else:

                cursor.execute("DELETE FROM admins WHERE chat\_id=?", (admin\_id,))

                conn.commit()

                bot.send\_message(chat\_id, *f*"✅ Пользователь {admin\_id} больше не администратор.")

                bot.send\_message(admin\_id, "❌ Ваши права администратора были сняты.")

        except *ValueError*:

            bot.send\_message(chat\_id, "❌ Неверный формат ID. Введите число.")

        finally:

            user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_broadcast\_message':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        cursor.execute("SELECT chat\_id FROM users")

        users = cursor.fetchall()

        for user in users:

            try:

                bot.send\_message(user[0], *f*"📢 Сообщение от админа: {text}")

            except:

                continue

        bot.send\_message(chat\_id, "✅ Сообщение отправлено всем пользователям.")

        user\_states.pop(chat\_id, None)

# Обработчик фотографий (сохранено для возможного использования в будущем)

@bot.message\_handler(*content\_types*=['photo'])

*def* handle\_photo(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT logged\_in FROM users WHERE chat\_id=?", (chat\_id,))

    result = cursor.fetchone()

    if not result or not result[0]:

        bot.send\_message(chat\_id, "Сначала выполните вход через /login.")

        return

    file\_info = bot.get\_file(message.photo[-1].file\_id)

    downloaded\_file = bot.download\_file(file\_info.file\_path)

    img\_path = *f*"temp\_{chat\_id}.jpg"

    with **open**(img\_path, 'wb') as f:

        f.write(downloaded\_file)

    try:

        prediction, confidence = predict\_image(img\_path)

        cursor.execute("UPDATE users SET predictions\_count = predictions\_count + 1 WHERE chat\_id=?", (chat\_id,))

        conn.commit()

        bot.send\_message(chat\_id, *f*"🔍 На изображении: {prediction} (вероятность: {confidence*:.2f*}%)")

    except *Exception* as e:

        bot.send\_message(chat\_id, *f*"❌ Ошибка при распознавании: {*str*(e)}")

    finally:

        if os.path.exists(img\_path):

            os.remove(img\_path)

# Функция предсказания (сохранено для возможного использования в будущем)

*def* predict\_image(*img\_path*):

    img = image.load\_img(img\_path, *target\_size*=(200, 200))

    x = image.img\_to\_array(img)

    x = np.expand\_dims(x, *axis*=0) / 255.0

    prediction = model.predict(x)

    return ("панда 🐼", (1 - prediction[0][0]) \* 100) if prediction[0] < 0.5 else ("человек 👤", prediction[0][0] \* 100)

# Запуск бота

bot.polling(*none\_stop*=True)

# Приложение 2

Листинг кода модифицированного бота внедренного на webhook’и

import telebot

from telebot import types

from flask import Flask, request

import tensorflow as tf

import numpy as np

import os

import sqlite3

import hashlib

import secrets

from datetime import datetime, timedelta

from tensorflow.keras.preprocessing import image

TOKEN = 'YOUR\_NEW\_TOKEN'

WEBHOOK\_URL = 'https://nikola.pythonanywhere.com/webhook'

app = Flask(\_\_name\_\_)

bot = telebot.TeleBot(TOKEN)

conn = sqlite3.connect('/home/nikola/mysite/users.db', *check\_same\_thread*=False)

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS users

                 (chat\_id INTEGER PRIMARY KEY,

                  password TEXT,

                  logged\_in BOOLEAN,

                  predictions\_count INTEGER DEFAULT 0)''')

cursor.execute('''CREATE TABLE IF NOT EXISTS admins

                 (chat\_id INTEGER PRIMARY KEY,

                  username TEXT)''')

conn.commit()

cursor.execute("PRAGMA table\_info(users)")

columns = [row[1] for row in cursor.fetchall()]

if "predictions\_count" not in columns:

    cursor.execute("ALTER TABLE users ADD COLUMN predictions\_count INTEGER DEFAULT 0")

    conn.commit()

user\_states = {}

login\_attempts = {}

model = tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(32, (3, 3), *activation*='relu', *input\_shape*=(200, 200, 3)),

    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), *activation*='relu'),

    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, *activation*='relu'),

    tf.keras.layers.Dense(1, *activation*='sigmoid')

])

model.compile(*optimizer*='adam', *loss*='binary\_crossentropy', *metrics*=['accuracy'])

model.load\_weights('/home/nikola/mysite/best\_weights.h5')  # Adjust path for PythonAnywhere

*def* hash\_password(*password*):

    salt = secrets.token\_hex(8)

    hashed = hashlib.sha256((password + salt).encode()).hexdigest()

    return *f*"{salt}:{hashed}"

*def* check\_password(*hashed\_password*, *user\_password*):

    salt, hashed = hashed\_password.split(':')

    return hashed == hashlib.sha256((user\_password + salt).encode()).hexdigest()

*def* is\_admin(*chat\_id*):

    cursor.execute("SELECT \* FROM admins WHERE chat\_id=?", (chat\_id,))

    return cursor.fetchone() is not None

*def* create\_main\_keyboard():

    markup = types.ReplyKeyboardMarkup(*resize\_keyboard*=True)

    markup.row(types.KeyboardButton('/register'), types.KeyboardButton('/login'))

    markup.row(types.KeyboardButton('/predict'), types.KeyboardButton('/logout'))

    markup.row(types.KeyboardButton('/user\_info'))

    return markup

*def* create\_admin\_keyboard():

    markup = types.ReplyKeyboardMarkup(*resize\_keyboard*=True)

    markup.row(types.KeyboardButton('/list\_users'), types.KeyboardButton('/delete\_user'))

    markup.row(types.KeyboardButton('/add\_admin'), types.KeyboardButton('/admin\_exit'))

    return markup

@bot.message\_handler(*commands*=['start'])

*def* start(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT COUNT(\*) FROM admins")

    if cursor.fetchone()[0] == 0:

        cursor.execute("INSERT INTO admins VALUES (?, ?)",

                      (chat\_id, message.from\_user.username or *str*(chat\_id)))

        conn.commit()

        bot.send\_message(chat\_id, "👑 Вы стали первым администратором!")

    welcome\_message = (

        "Привет! Я бот для распознавания панд и людей 🐼👤.\n"

        "Команды:\n"

        "/register — регистрация\n"

        "/login — вход\n"

        "/predict — вывод приветствия\n"

        "/user\_info — информация о пользователе\n"

        "/logout — выход\n"

        "Админы могут использовать /admin\_help для дополнительных команд."

    )

    bot.send\_message(chat\_id, welcome\_message, *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*commands*=['admin\_help'])

*def* admin\_help(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "🚫 Доступ запрещён. Эта команда только для администраторов.")

        return

    admin\_commands = """

🛠 <b>Команды администратора:</b>

/admin\_help — список команд

/list\_users — список пользователей

/add\_admin <code>chat\_id</code> — назначить админа

/remove\_admin <code>chat\_id</code> — снять админа

/list\_admins — список админов

/delete\_user <code>chat\_id</code> — удалить пользователя

/send\_all <code>текст</code> — рассылка всем пользователям

"""

    bot.send\_message(chat\_id, admin\_commands, *parse\_mode*="HTML", *reply\_markup*=create\_admin\_keyboard())

@bot.message\_handler(*commands*=['admin'])

*def* admin\_panel(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    bot.send\_message(chat\_id, "Панель администратора:", *reply\_markup*=create\_admin\_keyboard())

@bot.message\_handler(*commands*=['admin\_exit'])

*def* admin\_exit(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    bot.send\_message(chat\_id, "Выход из панели администратора.", *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*commands*=['list\_users'])

*def* list\_users(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    cursor.execute('''SELECT u.chat\_id, u.logged\_in, u.predictions\_count,

                      a.chat\_id IS NOT NULL as is\_admin

                      FROM users u LEFT JOIN admins a ON u.chat\_id = a.chat\_id''')

    users = cursor.fetchall()

    if not users:

        bot.send\_message(chat\_id, "Нет зарегистрированных пользователей.")

        return

    response = "📊 Список пользователей:\n\n"

    for user in users:

        status = "✅ В сети" if user[1] else "❌ Не в сети"

        role = "👑 Админ" if user[3] else "👤 Пользователь"

        response += *f*"ID: {user[0]}\n{status} | {role}\nПредсказаний: {user[2]}\n\n"

    bot.send\_message(chat\_id, response)

@bot.message\_handler(*commands*=['delete\_user'])

*def* delete\_user(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_user\_id\_for\_deletion'

    bot.send\_message(chat\_id, "Введите ID пользователя для удаления:")

@bot.message\_handler(*commands*=['add\_admin'])

*def* add\_admin(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_admin\_id'

    bot.send\_message(chat\_id, "Введите ID пользователя для назначения администратором:")

@bot.message\_handler(*commands*=['remove\_admin'])

*def* remove\_admin(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_admin\_id\_for\_removal'

    bot.send\_message(chat\_id, "Введите ID администратора для снятия прав:")

@bot.message\_handler(*commands*=['list\_admins'])

*def* list\_admins(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    cursor.execute("SELECT chat\_id, username FROM admins")

    admins = cursor.fetchall()

    if not admins:

        bot.send\_message(chat\_id, "Нет администраторов.")

        return

    response = "👑 Список администраторов:\n\n"

    for admin in admins:

        response += *f*"ID: {admin[0]} | Имя: {admin[1] or 'Не указано'}\n"

    bot.send\_message(chat\_id, response)

@bot.message\_handler(*commands*=['send\_all'])

*def* send\_all(*message*):

    chat\_id = message.chat.id

    if not is\_admin(chat\_id):

        bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

        return

    user\_states[chat\_id] = 'awaiting\_broadcast\_message'

    bot.send\_message(chat\_id, "Введите сообщение для рассылки всем пользователям:")

@bot.message\_handler(*commands*=['register'])

*def* register(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT \* FROM users WHERE chat\_id=?", (chat\_id,))

    if cursor.fetchone():

        bot.send\_message(chat\_id, "Вы уже зарегистрированы! Используйте /login.")

        return

    user\_states[chat\_id] = 'registering'

    bot.send\_message(chat\_id, "Введите пароль для регистрации:")

@bot.message\_handler(*commands*=['login'])

*def* login(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT \* FROM users WHERE chat\_id=?", (chat\_id,))

    if not cursor.fetchone():

        bot.send\_message(chat\_id, "Сначала зарегистрируйтесь через /register!")

        return

    if chat\_id in login\_attempts and login\_attempts[chat\_id]['attempts'] >= 3:

        if datetime.now() < login\_attempts[chat\_id]['block\_time']:

            remaining = (login\_attempts[chat\_id]['block\_time'] - datetime.now()).seconds // 60

            bot.send\_message(chat\_id, *f*"🔒 Слишком много попыток. Подождите {remaining} минут.")

            return

    user\_states[chat\_id] = 'logging\_in'

    bot.send\_message(chat\_id, "Введите пароль для входа:")

@bot.message\_handler(*commands*=['logout'])

*def* logout(*message*):

    chat\_id = message.chat.id

    cursor.execute("UPDATE users SET logged\_in=0 WHERE chat\_id=?", (chat\_id,))

    conn.commit()

    bot.send\_message(chat\_id, "Вы вышли из системы.", *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*commands*=['predict'])

*def* predict\_command(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT logged\_in FROM users WHERE chat\_id=?", (chat\_id,))

    result = cursor.fetchone()

    if not result:

        bot.send\_message(chat\_id, "Сначала зарегистрируйтесь через /register!")

        return

    if not result[0]:

        bot.send\_message(chat\_id, "Сначала выполните /login!")

        return

    cursor.execute("UPDATE users SET predictions\_count = predictions\_count + 1 WHERE chat\_id=?", (chat\_id,))

    conn.commit()

    bot.send\_message(chat\_id, "привет")

@bot.message\_handler(*commands*=['user\_info'])

*def* user\_info(*message*):

    chat\_id = message.chat.id

    cursor.execute("SELECT logged\_in, predictions\_count FROM users WHERE chat\_id=?", (chat\_id,))

    user = cursor.fetchone()

    if not user:

        bot.send\_message(chat\_id, "Сначала зарегистрируйтесь через /register!")

        return

    status = "✅ В сети" if user[0] else "❌ Не в сети"

    is\_admin\_status = "👑 Админ" if is\_admin(chat\_id) else "👤 Пользователь"

    response = (

*f*"📋 Информация о пользователе:\n\n"

*f*"ID: {chat\_id}\n"

*f*"Статус: {status}\n"

*f*"Роль: {is\_admin\_status}\n"

*f*"Предсказаний: {user[1]}"

    )

    bot.send\_message(chat\_id, response, *reply\_markup*=create\_main\_keyboard())

@bot.message\_handler(*content\_types*=['text'])

*def* handle\_text(*message*):

    chat\_id = message.chat.id

    text = message.text.strip()

    state = user\_states.get(chat\_id)

    if state == 'registering':

        hashed\_password = hash\_password(text)

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

                       (chat\_id, hashed\_password, False, 0))

        conn.commit()

        user\_states.pop(chat\_id, None)

        bot.send\_message(chat\_id, "✅ Регистрация успешна! Используйте /login.")

    elif state == 'logging\_in':

        cursor.execute("SELECT password FROM users WHERE chat\_id=?", (chat\_id,))

        result = cursor.fetchone()

        if result and check\_password(result[0], text):

            cursor.execute("UPDATE users SET logged\_in=1 WHERE chat\_id=?", (chat\_id,))

            conn.commit()

            login\_attempts.pop(chat\_id, None)

            bot.send\_message(chat\_id, "🔓 Вход выполнен успешно! Используйте /predict.")

        else:

            login\_attempts.setdefault(chat\_id, {'attempts': 0, 'block\_time': datetime.now()})

            login\_attempts[chat\_id]['attempts'] += 1

            if login\_attempts[chat\_id]['attempts'] >= 3:  # Fixed typo

                login\_attempts[chat\_id]['block\_time'] = datetime.now() + timedelta(*minutes*=5)

            remaining\_attempts = 3 - login\_attempts[chat\_id]['attempts']

            bot.send\_message(chat\_id, *f*"❌ Неверный пароль. Осталось попыток: {remaining\_attempts}")

        user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_user\_id\_for\_deletion':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        try:

            user\_id = *int*(text)

            if user\_id == chat\_id:

                bot.send\_message(chat\_id, "❌ Нельзя удалить самого себя!")

            else:

                cursor.execute("DELETE FROM users WHERE chat\_id=?", (user\_id,))

                cursor.execute("DELETE FROM admins WHERE chat\_id=?", (user\_id(user\_id,))

                conn.commit()

                bot.send\_message(chat\_id, *f*"✅ Пользователь {user\_id} удалён.")

        except *ValueError*:

            bot.send\_message(chat\_id, "❌ Неверный формат ID. Введите число.")

        finally:

            user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_admin\_id':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        try:

*new\_admin\_id* = *int*(text)

            cursor.execute("SELECT \* FROM users WHERE chat\_id=?", (new\_admin\_id,))

            if not cursor.fetchone():

                bot.send\_message(chat\_id, "❌ Пользователь не найден. Он должен зарегистрироваться.")

            else:

                cursor.execute("INSERT OR IGNORE INTO admins VALUES (?, ?)",

                               (new\_admin\_id, *f*"admin\_{new\_admin\_id}"))

                conn.commit()

                bot.send\_message(chat\_id, *f*"✅ Пользователь {new\_admin\_id} теперь администратор.")

                bot.send\_message(new\_admin\_id, "👑 Вас назначили администратором!")

        except *ValueError*:

            bot.send\_message(chat\_id, "❌ Неверный формат ID. Введите число.")

        finally:

            user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_admin\_id\_for\_removal':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        try:

*admin\_id* = *int*(text)

            if admin\_id == chat\_id:

                bot.send\_message(chat\_id, "❌ Нельзя снять права с самого себя!")

            else:

                cursor.execute("DELETE FROM admins WHERE chat\_id=?", (admin\_id,))

                conn.commit()

                bot.send\_message(chat\_id, *f*"✅ Пользователь {admin\_id} больше не администратор.")

                bot.send\_message(admin\_id, "❌ Ваши права администратора были сняты.")

        except *ValueError*:

            bot.send\_message(chat\_id, "❌ Неверный формат ID. Введите число.")

        finally:

            user\_states.pop(chat\_id, None)

    elif state == 'awaiting\_broadcast\_message':

        if not is\_admin(chat\_id):

            bot.send\_message(chat\_id, "❌ У вас нет прав администратора!")

            user\_states.pop(chat\_id, None)

            return

        cursor.execute("SELECT chat\_id FROM users")

*users* = cursor.fetchall()

        for user in users:

            try:

                bot.send\_message(user[0], *f*"📢 Сообщение от админа: {text}")

            except:

                continue

        bot.send\_message(chat\_id, "✅ Сообщение отправлено всем пользователям.")

        user\_states.pop(chat\_id, None)

@bot.message\_handler(*content\_types*=['photo'])

def handle\_photo(message):

*chat\_id* = message.chat.id

    cursor.execute("SELECT logged\_in FROM users WHERE chat\_id=?", (chat\_id,))

*result* = cursor.fetchone()

    if not result or not result[0]:

        bot.send\_message(chat\_id, "Сначала выполните вход через /login.")

        return

*file\_info* = bot.get\_file(message.photo[-1].file\_id)

*downloaded\_file* = bot.download\_file(file\_info.file\_path)

*img\_path* = *f*"/home/nikola/mysite/temp\_{chat\_id}.jpg"  # Adjust path for PythonAnywhere

    with **open**(img\_path, 'wb') as f:

        f.write(downloaded\_file)

    try:

        prediction, *confidence* = predict\_image(img\_path)

        cursor.execute("UPDATE users SET predictions\_count = predictions\_count + 1 WHERE chat\_id=?", (chat\_id,))

        conn.commit()

        bot.send\_message(chat\_id, *f*"🔍 На изображении: {prediction} (вероятность: {confidence*:.2f*}%)")

    except *Exception* as e:

        bot.send\_message(chat\_id, *f*"❌ Ошибка при распознавании: {*str*(e)}")

    finally:

        if os.path.exists(img\_path):

            os.remove(img\_path)

def predict\_image(img\_path):

*img* = image.load\_img(img\_path, *target\_size*=(200, 200))

*x* = image.img\_to\_array(img)

*x* = np.expand\_dims(x, *axis*=0) / 255.0

*prediction* = model.predict(x)

    return ("панда 🐼", (1 - prediction[0][0]) \* 100) if prediction[0] < 0.5 else ("человек 👤", prediction[0][0] \* 100)

@app.route('/webhook', *methods*=['POST'])

def webhook():

    if request.headers.get('content-type') == 'application/json':

*json\_string* = request.get\_data().decode('utf-8')

*update* = telebot.types.Update.de\_json(json\_string)

        bot.process\_new\_updates([update])

        return 'OK', 200

    return 'Invalid content type', 400

bot.delete\_webhook()

bot.set\_webhook(*url*=WEBHOOK\_URL)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run()