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. 5131001/30003 Н.Е. Шевчук

М.А. Мальцев

Руководитель

ассистент ВШК ИКНК М.С. Иванов

Санкт-Петербург – 2025

Оглавление

[1 Цель 3](#_Toc198047752)

[2 Задачи 4](#_Toc198047753)

[2.1 Усовершенствование Telegram-бота 4](#_Toc198047754)

[2.2 Перенос на webhook 4](#_Toc198047755)

[2.3 Внедрение 5](#_Toc198047756)

[3 Ход работы 6](#_Toc198047757)

[3.1 Усовершенствование Telegram-бота 6](#_Toc198047758)

[3.1.1 Недостатки текущей реализации хранения пользовательских данных 6](#_Toc198047759)

[3.1.2 Усовершенствование метода хранения данных 8](#_Toc198047760)

[3.1.3 Добавление роли администратора 11](#_Toc198047761)

[3.2 Перенос на webhook 12](#_Toc198047762)

[3.2.1 Изучение функционала webhook из telegram API 12](#_Toc198047763)

[3.2.2 Сервис pythonanywhere и фреймворк flask 13](#_Toc198047764)

[3.2.3 Создание веб-приложения для обработки POST-запросов 14](#_Toc198047765)

[3.2.4 Регистрация созданного webhook в tg-боте 15](#_Toc198047766)

[3.2.5 Ознакомление с форматом полученного JSON сообщения. 16](#_Toc198047767)

[3.2.6 Реализация отправки ответных сообщений 17](#_Toc198047768)

[3.3 Внедрение 18](#_Toc198047769)

[4 Ответы на контрольные вопросы 19](#_Toc198047770)

[5 Вывод 23](#_Toc198047771)

[Список источников 24](#_Toc198047772)

[Приложение 1 25](#_Toc198047773)

[Приложение 2 34](#_Toc198047774)

[Приложение 3 36](#_Toc198047775)

# 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

## Усовершенствование Telegram-бота

### Недостатки текущей реализации хранения пользовательских данных

В текущей реализации кода Telegram-бота, выполненного в третьей лабораторной работе, пользовательские данный (а именно, ID пользователя и его хешированный пароль) хранятся в файле user.csv в формате CSV. Для безопасного хранения пароля было выбрано хеширование, так как оно, в отличие от шифрования, является односторонним процессом: после хеширования нет способа восстановить оригинальный пароль. Когда пользователь вводит пароль, система сравнивает хеш введённого пароля с хешем, хранящимся в базе данных. Если они совпадают, доступ предоставляется. В исходном коде пароли хешируются с использованием SHA-256. Доступ к файлу осуществляется через функции load\_users() и save\_user(). На листинге 1, с которым можно ознакомиться ниже, представлен фрагмент исходного кода, осуществляющий описанный действия.

Листинг 1 – Функции хеширования пароля, загрузки пользователей из файла и сохранения пользователя в файл в исходном коде.

|  |
| --- |
| *# Список авторизованных пользователей* signed\_in\_users = []  ...  *# Функция для хеширования пароля* def hash\_password(password):  return hashlib.sha256(password.encode()).hexdigest()  *# Функция для загрузки пользователей из файла* def load\_users():  users = {}  try:  with open('users.csv', mode='r') as file:  reader = csv.reader(file)  for row in reader:  if len(row) >= 2:  users[row[0]] = row[1]  except FileNotFoundError:  open('users.csv', mode='w').close()  return users   *# Функция для сохранения пользователя в файл* def save\_user(user\_id, hashed\_password):  with open('users.csv', mode='a', newline='') as file:  writer = csv.writer(file)  writer.writerow([user\_id, hashed\_password]) |

В текущей реализации можно выделить следующие недостатки:

1. Хранение данных в текстовом файле (CSV):

CSV-файл не защищён от несанкционированного доступа. Если злоумышленник получит доступ к серверу, он сможет прочитать или модифицировать файл. Несмотря на хэширование, злоумышленник может попытаться подобрать пароли методом перебора (особенно если пароли слабые).

1. Отсутствие проверки целостности файла

Нет механизма проверки, что файл users.csv не был изменён вручную или повреждён. Например, некорректные данные в файле могут привести к сбоям в работе бота.

1. Отсутствие ротации или резервного копирования данных

Если файл users.csv будет удалён или повреждён, данные пользователей будут потеряны.

1. Хранение паролей без соли

Пароли хэшируются SHA-256 без использования соли (случайной строки, добавляемой перед хэшированием). Это делает хэши уязвимыми к атакам с использованием радужных таблиц (специальный вариант таблиц поиска для обращения криптографических хеш-функций).

1. Отсутствие ограничения на длину и сложность пароля

Пользователь может задать слабый пароль (например, "123"), что упрощает его взлом (например, перебором).

### Усовершенствование метода хранения данных

Для устранения выявленных в предыдущем пункте недостатков было сделано следующее:

1. Переход на SQLite для хранения данных:

SQLite — лёгкая встроенная база данных, которая заменит CSV-файл.

Преимущества:

* Защита от некорректного формата данных.
* Возможность шифрования базы данных.

Реализация:

* Создать таблицу users для хранения ID, хэша пароля, соли и количества предсказаний.
* Создать таблицу admins для хранения ID администраторов.

На листинге 2 представлены модифицированные функции для инициализации базы данных, загрузки пользователей из базы данных и сохранения нового пользователя.

Листинг 2 - работа с SQLite базой данных

|  |
| --- |
| *# Инициализация базы данных* def init\_db():  conn = sqlite3.connect('users.db')  cursor = conn.cursor()  cursor.execute('''  CREATE TABLE IF NOT EXISTS users (  user\_id TEXT PRIMARY\_KEY,  hashed\_password TEXT NOT NULL,  salt TEXT NOT NULL,  prediction\_count INTEGER DEFAULT 0  )  ''')  cursor.execute('''  CREATE TABLE IF NOT EXISTS admins (  user\_id TEXT PRIMARY\_KEY  )  ''')  conn.commit() *#Cохраняет все изменения (создание таблиц) в файл users.db* conn.close()  …  *# Загрузка пользователей из базы данных* def load\_users():  conn = sqlite3.connect('users.db')  cursor = conn.cursor()  cursor.execute('SELECT user\_id, hashed\_password, salt, prediction\_count FROM users')  users = {row[0]: {'hash': row[1], 'salt': row[2], 'prediction\_count': row[3]} for row in cursor.fetchall()}  conn.close()  return users  *# Сохранение нового пользователя* def save\_user(user\_id, password):  if not is\_strong\_password(password):  return False, "Пароль должен быть минимум 8 символов, содержать буквы разного регистра, цифры и специальные символы."  hashed\_password, salt = hash\_password(password)  conn = sqlite3.connect('users.db')  cursor = conn.cursor()  try:  cursor.execute('INSERT INTO users (user\_id, hashed\_password, salt) VALUES (?, ?, ?)',  (user\_id, hashed\_password, salt))  conn.commit()  backup\_db() *# Создаём резервную копию* return True, "Регистрация успешна!"  except sqlite3.IntegrityError:  return False, "Пользователь уже зарегистрирован!"  finally:  conn.close() |

1. Добавление соли к паролям.

Для каждого пользователя будем генерировать случайную соль, которую будем хранить вместе с хешэм пароля в базе данных. Для генерации соли использовалась библиотека secrets.

На листинге 3 представлена обновленная функция хеширования пароля с солью.

Листинг 3 - Хэширование пароля

|  |
| --- |
| *# Функция для хэширования пароля с солью* def hash\_password(password, salt=None):  if salt is None:  salt = secrets.token\_hex(16)  salted\_password = password + salt  hashed = hashlib.sha256(salted\_password.encode()).hexdigest()  return hashed, salt |

1. Проверка сложности пароля:

При регистрации нового пользователя будем проверять пароль на минимальную длину (8 символов) и наличие букв, цифра и специальных символов.

На листинге 4 представлена функция для проверки пароля на сложность.

Листинг 4 - Проверка сложности пароля

|  |
| --- |
| *# Проверка сложности пароля* def is\_strong\_password(password):  if len(password) < 8:  return False  if not re.search(r'[A-Z]', password):  return False  if not re.search(r'[a-z]', password):  return False  if not re.search(r'\d', password):  return False  if not re.search(r'[!@#$%^&\*(),.?":{}|<>]', password):  return False  return True |

1. Резервное копирование

Периодически будем создавать резервные копии базы данных (копировать файл SQLite в отдельную директорию).

На листинге 5 представлена функция резервного копирования базы данных.

Листинг 5 - Резервное копирование базы данных

|  |
| --- |
| *# Функция для резервного копирования базы данных* def backup\_db():  try:  timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')  shutil.copy('users.db', f'backup/users\_{timestamp}.db')  except Exception as e:  print(f"Ошибка при создании резервной копии: {e}") |

### Добавление роли администратора

Пользователь становится администратором командой become\_admin, при вводе корректного секретного ключа. Список пользователей отправляется в виде текстового сообщения, где для каждого пользователя указаны user\_id и prediction\_id. Также добавлены следующие функции, реализующие функционал для администратора:

* is\_admin – проверка, является ли текущий пользователь администратором;
* has\_admins – проверка, есть ли уже зарегистрированные администраторы;
* add\_admin – добавление администратора;
* delete\_user – удаление пользователя (по его user\_id).

На рисунке 1 представлена регистрация первого администратора. На рисунке 2 представлен пример работы администратора.

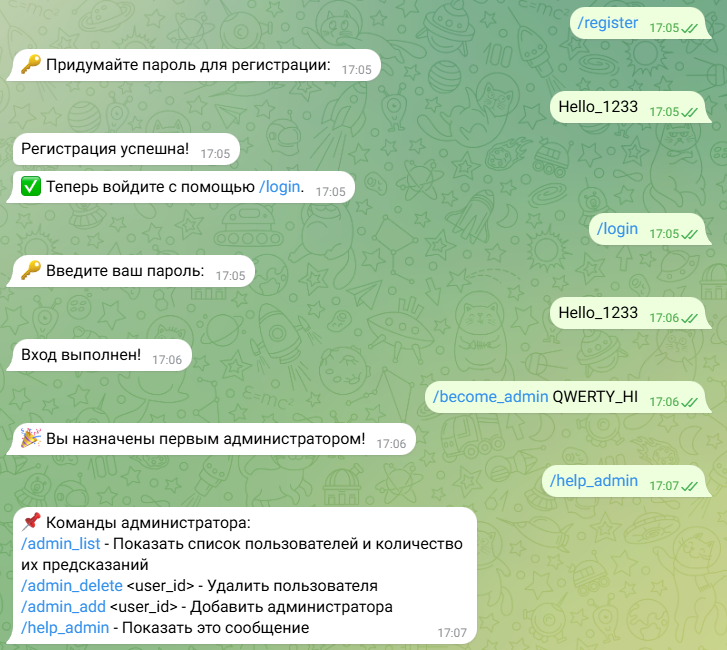


Рисунок 1 - Регистрация первого администратора

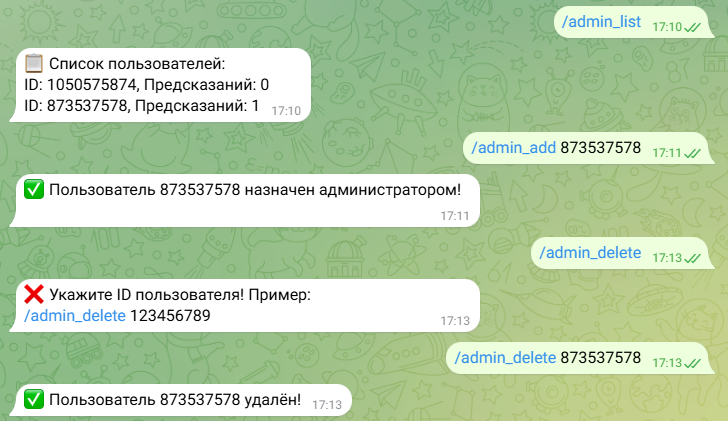


Рисунок 2 - Пример работы администратора

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.

### Создание веб-приложения для обработки POST-запросов

Была реализована простая программа, которая обрабатывает POST-запрос. На листинге 6 представлен пример кода обработчика POST-запросов.

Листинг 6 – Обработчик POST-запросов

|  |
| --- |
| from flask import Flask, request, jsonify  app = Flask(\_\_name\_\_)  # Ochoshaa страница  @app.route('/')  def home():  return "Webhook Server is Running"  # Webhook для POST-запросов  @app.route('/webhook/post', methods=['POST'])  def webhook\_post():  # Получаем данные из тела запроса  data = request.json  # Логируем полученные данные  print(f"Received POST webhook with data: {data}")  # Возвращаем ответ  return jsonify({  'status': 'success',  'method': 'POST',  'data\_received': data  }) |

### Регистрация созданного webhook в tg-боте

Для начала немного изменим код для обработки post-запросов. На листинге 7 представлен код изменённого обработчика post-запросов.

Листинг 7 - Изменённый обработчик post-запросов

|  |
| --- |
| # Webhook для POST-запросов из телеграмма  @app.route('/telegram\_webhook', methods=['POST'])  def telegram\_webhook():  update = request.get\_json()  # Обработка входящего сообщения  if 'message' in update:  chat\_id = update['message']['chat']['id']  text = update['message'].get('text', '')    # Простейший ответ  response = {  'method': 'sendMessage',  'chat\_id': chat\_id,  'text': f'Вы написали: {text}'  }  return jsonify(response)    return jsonify({'status': 'ok'}) |

Для регистрации вебхука можно использовать curl в bush командной строке или python-скрипт. На листинге 8 представлен пример регистрации вебхука в командной строке. На листинге 9 можно увидеть пример регистрации вебхука python-скриптом.

Листинг 8 - Регистрация вебхука в командной строке

|  |
| --- |
| curl “https://api.telegram.org/bot7842869515:AAEoqR7Dwe1dNh5UjSU6bDsRcO8S0ijmX5s/setWebhook?url=https://Shimu.pythonanywhere.com/telegram\_webhook” |

Листинг 9 – Регистрация вебхука с помощью python-скрипта

|  |
| --- |
| import requests  def set\_telegram\_webhook():  """Функция для регистрации вебхука"""  try:  response = requests.get(  f"https://api.telegram.org/bot{TOKEN}/setWebhook",  params={'url': WEBHOOK\_URL}  )  result = response.json()  print("Webhook setup result:", result)  return result.get('description', 'Webhook registered successfully')  except Exception as e:  return f"Error setting webhook: {str(e)}" |

В нашей реализации был использован python-скрипт. После выполнения данного участка кода telegram будет отправлять запросы только на указанный URL.

### Ознакомление с форматом полученного JSON сообщения.

POST-запрос в формате JSON — это HTTP-запрос, используемый для отправки структурированных данных на сервер. Он широко применяется в REST API, вебхуках и взаимодействии между клиентом и сервером. При отправлении сообщения боту, telegram отправил POST-запрос в формате JSON на указанный, при регистрации вебхука, URL. На листинге 10 представлен пример POST-запроса в формате JSON.

Листинг 10 – Пример POST-запроса в формате JSON

|  |
| --- |
| {'update\_id': 119617221,  'message': {  'message\_id': 791,  'from': {'id': 1347255109, 'is\_bot': False, 'first\_name': 'Масяо', 'username': 'ILShimu', 'language\_code': 'ru'},  'chat': {'id': 1347255109, 'first\_name': 'Масяо', 'username': 'ILShimu', 'type': 'private'},  'date': 1747075332,  'text': 'TRALALERA TRALALA'  }  } |

Здесь можно узнать имя пользователя, его ник, отправленный текст, дату отправки в формате UNIX TIMESTAMP, и другую разную информацию. Также можно заметить, что данный запрос от телеграмма пришел в виде json-словаря, что позволяет удобнее обрабатывать его.

### Реализация отправки ответных сообщений

Была реализована простейшая обработка и отправка ответного сообщения, с кодом которой можно ознакомиться на листинге 11.

Листинг 11 - Код обработки и отправки ответного сообщения

|  |
| --- |
| # Webhook для POST-запросов из телеграмма  @app.route('/telegram\_webhook', methods=['POST'])  def telegram\_webhook():  update = request.get\_json()  print("Received update:", update) # Логируем входящий запрос  # Обработка входящего сообщения  if 'message' in update:  chat\_id = update['message']['chat']['id']  text = update['message'].get('text', '')    # Простейший ответ  response = {  'method': 'sendMessage',  'chat\_id': chat\_id,  'text': f'Вы написали: {text}'  }  return jsonify(response)    return jsonify({'status': 'ok'}) |

## Внедрение

При выполнении переноса усовершенствованного Telegram-бота, с сохраненным функционалом из третьей лабораторной работы, на webhook’и произошел неприятный инцидент: при установке библиотек, необходимых для запуска и корректной работы web-приложения, в консоль вывелась ошибка о нехватке памяти на диске. Оказывается, бесплатная версия pythonanywhere имеет ограничение в 512 Мб дискового пространства. На рисунке 3 представлен консольный вывод ошибки о нехватке памяти при попытке установки необходимых python-библиотек.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 - Ошибка нехватки памяти

# 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.

# Список источников

1. <https://psono.com/ru/blog/evolution-of-password-hashing>
2. Дж. Уокер “Искусство хеширования”. <https://kurl.ru/KjOle>
3. <https://habr.com/ru/companies/otus/articles/786754/>
4. <https://habr.com/ru/articles/819971/>
5. <https://skillbox.ru/media/code/json-chto-eto-za-format-i-kak-s-nim-rabotat/>
6. <https://cloud.ru/blog/belye-i-serye-ip-dinamicheskie-i-staticheskie>

# Приложение 1

Листинг кода Telegram-бота:

import telebot  
import sqlite3  
import secrets *#Используется для создания соли для хэширования паролей*import re *#Используется для проверки сложности пароля*import os  
import shutil  
from datetime import datetime  
from telebot import types  
import numpy as np  
import tensorflow as tf  
from tensorflow.keras.preprocessing import image  
from PIL import Image  
import io  
import hashlib *#Используется для создания SHA-256 хэшей паролей  
  
# Инициализация бота*bot = telebot.TeleBot("7842869515:AAEoqR7Dwe1dNh5UjSU6bDsRcO8S0ijmX5s") *# Замените на свой токен  
  
# Список авторизованных пользователей*signed\_in\_users = []  
  
*# Загрузка модели*MODEL\_PATH = 'best\_human\_giraffe\_model.h5'  
try:  
 model = tf.keras.models.load\_model(MODEL\_PATH)  
 print("Модель успешно загружена")  
except Exception as e:  
 print(f"Ошибка загрузки модели: {e}")  
 model = None  
  
*# Инициализация базы данных*def init\_db():  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 cursor.execute('''  
 CREATE TABLE IF NOT EXISTS users (  
 user\_id TEXT PRIMARY\_KEY,  
 hashed\_password TEXT NOT NULL,  
 salt TEXT NOT NULL,  
 prediction\_count INTEGER DEFAULT 0  
 )  
 ''')  
 cursor.execute('''  
 CREATE TABLE IF NOT EXISTS admins (  
 user\_id TEXT PRIMARY\_KEY  
 )  
 ''')  
 conn.commit() *#Cохраняет все изменения (создание таблиц) в файл users.db* conn.close()  
  
*# Функция для резервного копирования базы данных*def backup\_db():  
 try:  
 os.makedirs('backup', exist\_ok=True)  
 timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')  
 shutil.copy('users.db', f'backup/users\_{timestamp}.db')  
 except Exception as e:  
 print(f"Ошибка при создании резервной копии: {e}")  
  
*# Функция для хэширования пароля с солью*def hash\_password(password, salt=None):  
 if salt is None:  
 salt = secrets.token\_hex(16)  
 salted\_password = password + salt  
 hashed = hashlib.sha256(salted\_password.encode()).hexdigest()  
 return hashed, salt  
  
*# Проверка сложности пароля*def is\_strong\_password(password):  
 if len(password) < 8:  
 return False  
 if not re.search(r'[A-Z]', password):  
 return False  
 if not re.search(r'[a-z]', password):  
 return False  
 if not re.search(r'\d', password):  
 return False  
 if not re.search(r'[\_!@#$%^&\*(),.?":{}|<>]', password):  
 return False  
 return True  
  
*# Загрузка пользователей из базы данных*def load\_users():  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 cursor.execute('SELECT user\_id, hashed\_password, salt, prediction\_count FROM users')  
 users = {row[0]: {'hash': row[1], 'salt': row[2], 'prediction\_count': row[3]} for row in cursor.fetchall()}  
 conn.close()  
 return users  
  
*# Сохранение нового пользователя*def save\_user(user\_id, password):  
 if not is\_strong\_password(password):  
 return False, "Пароль должен быть минимум 8 символов, содержать буквы разного регистра, цифры и специальные символы."  
 hashed\_password, salt = hash\_password(password)  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 try:  
 cursor.execute('INSERT INTO users (user\_id, hashed\_password, salt) VALUES (?, ?, ?)',  
 (user\_id, hashed\_password, salt))  
 conn.commit()  
 backup\_db() *# Создаём резервную копию* return True, "Регистрация успешна!"  
 except sqlite3.IntegrityError:  
 return False, "Пользователь уже зарегистрирован!"  
 finally:  
 conn.close()  
  
*# Проверка логина*def check\_login(user\_id, password):  
 users = load\_users()  
 if user\_id not in users:  
 return False, "Пользователь не зарегистрирован!"  
 hashed\_input, \_ = hash\_password(password, users[user\_id]['salt'])  
 if hashed\_input == users[user\_id]['hash']:  
 return True, "Вход выполнен!"  
 return False, "Неверный пароль!"  
  
*# Увеличение счётчика предсказаний*def increment\_prediction\_count(user\_id):  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 cursor.execute('UPDATE users SET prediction\_count = prediction\_count + 1 WHERE user\_id = ?', (user\_id,))  
 conn.commit()  
 conn.close()  
 backup\_db()  
  
*# Инициализация базы данных при старте*if os.path.exists('users.db'):  
 os.remove('users.db')  
if os.path.exists('backup'):  
 shutil.rmtree('backup')  
init\_db()  
  
*# Проверка, является ли пользователь администратором*def is\_admin(user\_id):  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 cursor.execute('SELECT user\_id FROM admins WHERE user\_id = ?', (user\_id,))  
 result = cursor.fetchone()  
 conn.close()  
 return result is not None  
  
*# Проверка, есть ли администраторы*def has\_admins():  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 cursor.execute('SELECT COUNT(\*) FROM admins')  
 count = cursor.fetchone()[0]  
 conn.close()  
 return count > 0  
  
*# Добавление администратора*def add\_admin(user\_id):  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 try:  
 cursor.execute('INSERT INTO admins (user\_id) VALUES (?)', (user\_id,))  
 conn.commit()  
 backup\_db()  
 except sqlite3.IntegrityError:  
 pass *# Пользователь уже администратор* finally:  
 conn.close()  
  
*# Удаление пользователя*def delete\_user(user\_id):  
 conn = sqlite3.connect('users.db')  
 cursor = conn.cursor()  
 cursor.execute('DELETE FROM users WHERE user\_id = ?', (user\_id,))  
 cursor.execute('DELETE FROM admins WHERE user\_id = ?', (user\_id,))  
 conn.commit()  
 conn.close()  
 backup\_db()  
  
*# Обработчик команды /start*@bot.message\_handler(commands=['start'])  
def send\_welcome(message):  
 bot.reply\_to(message, "Привет! Я бот для классификации изображений. Используй /help для списка команд.")  
  
*# Обработчик команды /help*@bot.message\_handler(commands=['help'])  
def show\_help(message):  
 help\_text = """  
📌 Доступные команды:  
/register - Зарегистрироваться (установить пароль)  
/login - Войти в систему (ввести пароль)  
/logout - Выйти из системы  
/predict - Классифицировать изображение (требуется авторизация)  
/become\_admin <code> - Стать первым администратором (требуется секретный код)  
/help\_admin - Показать команды администратора   
/help - Показать это сообщение  
 """  
 bot.send\_message(message.chat.id, help\_text)  
  
*# Обработчик команды /register*@bot.message\_handler(commands=['register'])  
def register\_user(message):  
 chat\_id = str(message.chat.id)  
 users = load\_users()  
  
 if chat\_id in users:  
 bot.send\_message(chat\_id, "❌ Вы уже зарегистрированы! Используйте /login.")  
 return  
  
 msg = bot.send\_message(chat\_id, "🔑 Придумайте пароль для регистрации:")  
 bot.register\_next\_step\_handler(msg, process\_password\_step)  
  
def process\_password\_step(message):  
 chat\_id = str(message.chat.id)  
 password = message.text.strip()  
 if not is\_strong\_password(password):  
 msg = bot.send\_message(chat\_id, "❌ Пароль должен быть минимум 8 символов, содержать буквы разного регистра, цифры и специальные символы. Попробуйте ещё раз:")  
 bot.register\_next\_step\_handler(msg, process\_password\_step)  
 return  
 success, msg = save\_user(chat\_id, password)  
 bot.send\_message(chat\_id, msg)  
 if success:  
 bot.send\_message(chat\_id, "✅ Теперь войдите с помощью /login.")  
  
*# Обработчик команды /login*@bot.message\_handler(commands=['login'])  
def login\_user(message):  
 chat\_id = str(message.chat.id)  
 users = load\_users()  
  
 if chat\_id not in users:  
 bot.send\_message(chat\_id, "❌ Вы не зарегистрированы! Используйте /register.")  
 return  
  
 if chat\_id in signed\_in\_users:  
 bot.send\_message(chat\_id, "❌ Вы уже вошли в систему! Используйте /logout для выхода.")  
 return  
  
 msg = bot.send\_message(chat\_id, "🔑 Введите ваш пароль:")  
 bot.register\_next\_step\_handler(msg, process\_login\_step)  
  
def process\_login\_step(message):  
 chat\_id = str(message.chat.id)  
 input\_password = message.text.strip()  
 success, msg = check\_login(chat\_id, input\_password)  
 bot.send\_message(chat\_id, msg)  
 if success:  
 signed\_in\_users.append(chat\_id)  
  
*# Обработчик команды /logout*@bot.message\_handler(commands=['logout'])  
def logout\_user(message):  
 chat\_id = str(message.chat.id)  
  
 if chat\_id in signed\_in\_users:  
 signed\_in\_users.remove(chat\_id)  
 bot.send\_message(chat\_id, "✅ Выход выполнен! Для использования /predict войдите снова /login.")  
 else:  
 bot.send\_message(chat\_id, "❌ Вы не вошли в систему!")  
  
*# Обработчик команды /predict (только для авторизованных)*@bot.message\_handler(commands=['predict'])  
def predict\_image(message):  
 chat\_id = str(message.chat.id)  
  
 if chat\_id not in signed\_in\_users:  
 bot.send\_message(chat\_id, "❌ Доступ запрещен! Зарегистрируйтесь /register и войдите /login.")  
 return  
  
 msg = bot.send\_message(chat\_id, "📷 Отправьте изображение для классификации:")  
 bot.register\_next\_step\_handler(msg, process\_image\_step)  
  
*# Обработчик команды /help\_admin*@bot.message\_handler(commands=['help\_admin'])  
def help\_admin(message):  
 chat\_id = str(message.chat.id)  
 if not is\_admin(chat\_id):  
 bot.send\_message(chat\_id, "❌ Вы не администратор!")  
 return  
 help\_text = """  
📌 Команды администратора:  
/admin\_list - Показать список пользователей и количество их предсказаний  
/admin\_delete <user\_id> - Удалить пользователя  
/admin\_add <user\_id> - Добавить администратора  
/help\_admin - Показать это сообщение  
 """  
 bot.send\_message(chat\_id, help\_text)  
  
*# Обработчик команды /become\_admin*@bot.message\_handler(commands=['become\_admin'])  
def become\_admin(message):  
 chat\_id = str(message.chat.id)  
 SECRET\_CODE = "QWERTY\_HI" *# Секретный код для назначения первого администратора* try:  
 code = message.text.split()[1]  
 except IndexError:  
 bot.send\_message(chat\_id, "❌ Укажите секретный код! Пример: /become\_admin SUPERSECRET123")  
 return  
 users = load\_users()  
 if chat\_id not in users:  
 bot.send\_message(chat\_id, "❌ Вы должны быть зарегистрированы! Используйте /register.")  
 return  
 if has\_admins():  
 bot.send\_message(chat\_id, "❌ Администратор уже назначен!")  
 return  
 if code != SECRET\_CODE:  
 bot.send\_message(chat\_id, "❌ Неверный секретный код!")  
 return  
 add\_admin(chat\_id)  
 bot.send\_message(chat\_id, "🎉 Вы назначены первым администратором!")  
  
*# Обработчик команды /admin\_list*@bot.message\_handler(commands=['admin\_list'])  
def admin\_list(message):  
 chat\_id = str(message.chat.id)  
 if not is\_admin(chat\_id):  
 bot.send\_message(chat\_id, "❌ Вы не администратор!")  
 return  
 users = load\_users()  
 if not users:  
 bot.send\_message(chat\_id, "📋 Список пользователей пуст.")  
 return  
 user\_list = "\n".join([f"ID: {uid}, Предсказаний: {data['prediction\_count']}" for uid, data in users.items()])  
 bot.send\_message(chat\_id, f"📋 Список пользователей:\n{user\_list}")  
  
*# Обработчик команды /admin\_delete*@bot.message\_handler(commands=['admin\_delete'])  
def admin\_delete(message):  
 chat\_id = str(message.chat.id)  
 if not is\_admin(chat\_id):  
 bot.send\_message(chat\_id, "❌ Вы не администратор!")  
 return  
 try:  
 user\_id = message.text.split()[1]  
 except IndexError:  
 bot.send\_message(chat\_id, "❌ Укажите ID пользователя! Пример: /admin\_delete 123456789")  
 return  
 users = load\_users()  
 if user\_id not in users:  
 bot.send\_message(chat\_id, "❌ Пользователь не найден!")  
 return  
 delete\_user(user\_id)  
 if user\_id in signed\_in\_users:  
 signed\_in\_users.remove(user\_id)  
 bot.send\_message(chat\_id, f"✅ Пользователь {user\_id} удалён!")  
  
*# Обработчик команды /admin\_add*@bot.message\_handler(commands=['admin\_add'])  
def admin\_add(message):  
 chat\_id = str(message.chat.id)  
 if not is\_admin(chat\_id):  
 bot.send\_message(chat\_id, "❌ Вы не администратор!")  
 return  
 try:  
 user\_id = message.text.split()[1]  
 except IndexError:  
 bot.send\_message(chat\_id, "❌ Укажите ID пользователя! Пример: /admin\_add 123456789")  
 return  
 users = load\_users()  
 if user\_id not in users:  
 bot.send\_message(chat\_id, "❌ Пользователь не найден!")  
 return  
 add\_admin(user\_id)  
 bot.send\_message(chat\_id, f"✅ Пользователь {user\_id} назначен администратором!")  
  
def process\_image\_step(message):  
 chat\_id = str(message.chat.id)  
  
 if not message.photo:  
 bot.send\_message(chat\_id, "❌ Это не изображение! Попробуйте снова /predict.")  
 return  
  
 if model is None:  
 bot.send\_message(chat\_id, "❌ Модель не загружена. Пожалуйста, свяжитесь с администратором.")  
 return  
  
 try:  
 *# Получаем файл изображения* file\_info = bot.get\_file(message.photo[-1].file\_id)  
 downloaded\_file = bot.download\_file(file\_info.file\_path)  
  
 *# Конвертируем изображение в нужный формат* img = Image.open(io.BytesIO(downloaded\_file))  
 img = img.resize((200, 200)) *# Изменяем размер до 200x200* x = image.img\_to\_array(img)  
 x = x / 255.0 *# Нормализация* x = np.expand\_dims(x, axis=0)  
  
 *# Предсказание* classes = model.predict(x, batch\_size=1)  
 probability = classes[0][0]  
 *# Определяем класс* if probability < 0.5:  
 result = "human"  
 prob\_text = f"🧍‍♂️🧍‍♀️ Вероятность человека: {(1 - probability):.4f}\n🦒 Вероятность жирафа: {probability:.4f}"  
 else:  
 result = "giraffe"  
 prob\_text = f"🧍‍♂️🧍‍♀️ Вероятность человека: {(1 - probability):.4f}\n🦒 Вероятность жирафа: {probability:.4f}"  
  
 bot.send\_message(chat\_id, f"Изображение классифицировано как {result}!\n{prob\_text}")  
 increment\_prediction\_count(chat\_id) *# Увеличиваем счётчик предсказаний* except Exception as e:  
 bot.send\_message(chat\_id, f"❌ Ошибка обработки изображения: {str(e)}. Попробуйте ещё раз.")  
  
*# Запуск бота*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Бот запущен...")  
 bot.infinity\_polling()

# Приложение 2

Листинг кода веб-приложения:

from flask import Flask, request, jsonify

import requests

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

# Конфигурационные данные

USERNAME = "Shimu"

TOKEN = "7842869515:AAEoqR7Dwe1dNh5UjSU6bDsRcO8S0ijmX5s"

WEBHOOK\_URL = f"https://{USERNAME}.pythonanywhere.com/telegram\_webhook"

def set\_telegram\_webhook():

"""Функция для регистрации вебхука"""

try:

response = requests.get(

f"https://api.telegram.org/bot{TOKEN}/setWebhook",

params={'url': WEBHOOK\_URL}

)

result = response.json()

print("Webhook setup result:", result)

return result.get('description', 'Webhook registered successfully')

except Exception as e:

return f"Error setting webhook: {str(e)}"

# Регистрируем вебхук при старте приложения

set\_telegram\_webhook()

# Основная страница

@app.route('/')

def home():

return "Webhook Server is Running"

# Webhook для POST-запросов из телеграмма

@app.route('/telegram\_webhook', methods=['POST'])

def telegram\_webhook():

update = request.get\_json()

print("Received update:", update) # Логируем входящий запрос

# Обработка входящего сообщения

if 'message' in update:

chat\_id = update['message']['chat']['id']

text = update['message'].get('text', '')

# Простейший ответ

response = {

'method': 'sendMessage',

'chat\_id': chat\_id,

'text': f'Вы написали: {text}'

}

return jsonify(response)

return jsonify({'status': 'ok'})

# Приложение 3

Листинг кода модифицированного бота внедренного на webhook’и

from flask import Flask, request, jsonify

import sqlite3

import secrets

import re

import os

import shutil

from datetime import datetime

import telebot

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing import image

from PIL import Image

import io

import hashlib

import requests

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

# Конфигурационные данные

USERNAME = "Shimu"

TOKEN = "7842869515:AAEoqR7Dwe1dNh5UjSU6bDsRcO8S0ijmX5s"

WEBHOOK\_URL = f"https://{USERNAME}.pythonanywhere.com/telegram\_webhook"

SECRET\_CODE = "QWERTY\_HI" # Секретный код для назначения первого администратора

# Инициализация бота

bot = telebot.TeleBot(TOKEN)

# Список авторизованных пользователей

signed\_in\_users = []

# Состояния пользователей для многошаговых команд

user\_states = {}

# Загрузка модели

MODEL\_PATH = 'best\_human\_giraffe\_model.h5'

try:

model = tf.keras.models.load\_model(MODEL\_PATH)

print("Модель успешно загружена")

except Exception as e:

print(f"Ошибка загрузки модели: {e}")

model = None

# Функции работы с базой данных

def init\_db():

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

user\_id TEXT PRIMARY KEY,

hashed\_password TEXT NOT NULL,

salt TEXT NOT NULL,

prediction\_count INTEGER DEFAULT 0

)

''')

cursor.execute('''

CREATE TABLE IF NOT EXISTS admins (

user\_id TEXT PRIMARY KEY

)

''')

conn.commit()

conn.close()

def backup\_db():

try:

os.makedirs('backup', exist\_ok=True)

timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

shutil.copy('users.db', f'backup/users\_{timestamp}.db')

except Exception as e:

print(f"Ошибка при создании резервной копии: {e}")

def hash\_password(password, salt=None):

if salt is None:

salt = secrets.token\_hex(16)

salted\_password = password + salt

hashed = hashlib.sha256(salted\_password.encode()).hexdigest()

return hashed, salt

def is\_strong\_password(password):

if len(password) < 8:

return False

if not re.search(r'[A-Z]', password):

return False

if not re.search(r'[a-z]', password):

return False

if not re.search(r'\d', password):

return False

if not re.search(r'[\_!@#$%^&\*(),.?":{}|<>]', password):

return False

return True

def load\_users():

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

cursor.execute('SELECT user\_id, hashed\_password, salt, prediction\_count FROM users')

users = {row[0]: {'hash': row[1], 'salt': row[2], 'prediction\_count': row[3]} for row in cursor.fetchall()}

conn.close()

return users

def save\_user(user\_id, password):

if not is\_strong\_password(password):

return False, "Пароль должен быть минимум 8 символов, содержать буквы разного регистра, цифры и специальные символы."

hashed\_password, salt = hash\_password(password)

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

try:

cursor.execute('INSERT INTO users (user\_id, hashed\_password, salt) VALUES (?, ?, ?)',

(user\_id, hashed\_password, salt))

conn.commit()

backup\_db()

return True, "Регистрация успешна!"

except sqlite3.IntegrityError:

return False, "Пользователь уже зарегистрирован!"

finally:

conn.close()

def check\_login(user\_id, password):

users = load\_users()

if user\_id not in users:

return False, "Пользователь не зарегистрирован!"

hashed\_input, \_ = hash\_password(password, users[user\_id]['salt'])

if hashed\_input == users[user\_id]['hash']:

return True, "Вход выполнен!"

return False, "Неверный пароль!"

def increment\_prediction\_count(user\_id):

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

cursor.execute('UPDATE users SET prediction\_count = prediction\_count + 1 WHERE user\_id = ?', (user\_id,))

conn.commit()

conn.close()

backup\_db()

def is\_admin(user\_id):

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

cursor.execute('SELECT user\_id FROM admins WHERE user\_id = ?', (user\_id,))

result = cursor.fetchone()

conn.close()

return result is not None

def has\_admins():

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

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

count = cursor.fetchone()[0]

conn.close()

return count > 0

def add\_admin(user\_id):

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

try:

cursor.execute('INSERT INTO admins (user\_id) VALUES (?)', (user\_id,))

conn.commit()

backup\_db()

except sqlite3.IntegrityError:

pass # Пользователь уже администратор

finally:

conn.close()

def delete\_user(user\_id):

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

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

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

conn.commit()

conn.close()

backup\_db()

# Инициализация базы данных при старте

if os.path.exists('users.db'):

os.remove('users.db')

if os.path.exists('backup'):

shutil.rmtree('backup')

init\_db()

# Webhook функции

def set\_telegram\_webhook():

"""Функция для регистрации вебхука"""

try:

response = requests.get(

f"https://api.telegram.org/bot{TOKEN}/setWebhook",

params={'url': WEBHOOK\_URL}

)

result = response.json()

print("Webhook setup result:", result)

return result.get('description', 'Webhook registered successfully')

except Exception as e:

return f"Error setting webhook: {str(e)}"

# Обработчики команд

def handle\_start(chat\_id):

bot.send\_message(chat\_id, "Привет! Я бот для классификации изображений. Используй /help для списка команд.")

def handle\_help(chat\_id):

help\_text = """

📌 Доступные команды:

/register - Зарегистрироваться (установить пароль)

/login - Войти в систему (ввести пароль)

/logout - Выйти из системы

/predict - Классифицировать изображение (требуется авторизация)

/become\_admin <code> - Стать первым администратором (требуется секретный код)

/help\_admin - Показать команды администратора

/help - Показать это сообщение

"""

bot.send\_message(chat\_id, help\_text)

def handle\_register(chat\_id):

users = load\_users()

if chat\_id in users:

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

return

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

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

def handle\_login(chat\_id):

users = load\_users()

if chat\_id not in users:

bot.send\_message(chat\_id, "❌ Вы не зарегистрированы! Используйте /register.")

return

if chat\_id in signed\_in\_users:

bot.send\_message(chat\_id, "❌ Вы уже вошли в систему! Используйте /logout для выхода.")

return

user\_states[chat\_id] = 'awaiting\_login\_password'

bot.send\_message(chat\_id, "🔑 Введите ваш пароль:")

def handle\_logout(chat\_id):

if chat\_id in signed\_in\_users:

signed\_in\_users.remove(chat\_id)

bot.send\_message(chat\_id, "✅ Выход выполнен! Для использования /predict войдите снова /login.")

else:

bot.send\_message(chat\_id, "❌ Вы не вошли в систему!")

def handle\_predict(chat\_id):

if chat\_id not in signed\_in\_users:

bot.send\_message(chat\_id, "❌ Доступ запрещен! Зарегистрируйтесь /register и войдите /login.")

return

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

bot.send\_message(chat\_id, "📷 Отправьте изображение для классификации:")

def handle\_help\_admin(chat\_id):

if not is\_admin(chat\_id):

bot.send\_message(chat\_id, "❌ Вы не администратор!")

return

help\_text = """

📌 Команды администратора:

/admin\_list - Показать список пользователей и количество их предсказаний

/admin\_delete <user\_id> - Удалить пользователя

/admin\_add <user\_id> - Добавить администратора

/help\_admin - Показать это сообщение

"""

bot.send\_message(chat\_id, help\_text)

def handle\_become\_admin(chat\_id, text):

try:

code = text.split()[1]

except IndexError:

bot.send\_message(chat\_id, "❌ Укажите секретный код! Пример: /become\_admin SUPERSECRET123")

return

users = load\_users()

if chat\_id not in users:

bot.send\_message(chat\_id, "❌ Вы должны быть зарегистрированы! Используйте /register.")

return

if has\_admins():

bot.send\_message(chat\_id, "❌ Администратор уже назначен!")

return

if code != SECRET\_CODE:

bot.send\_message(chat\_id, "❌ Неверный секретный код!")

return

add\_admin(chat\_id)

bot.send\_message(chat\_id, "🎉 Вы назначены первым администратором!")

def handle\_admin\_list(chat\_id):

if not is\_admin(chat\_id):

bot.send\_message(chat\_id, "❌ Вы не администратор!")

return

users = load\_users()

if not users:

bot.send\_message(chat\_id, "📋 Список пользователей пуст.")

return

user\_list = "\n".join([f"ID: {uid}, Предсказаний: {data['prediction\_count']}" for uid, data in users.items()])

bot.send\_message(chat\_id, f"📋 Список пользователей:\n{user\_list}")

def handle\_admin\_delete(chat\_id, text):

if not is\_admin(chat\_id):

bot.send\_message(chat\_id, "❌ Вы не администратор!")

return

try:

user\_id = text.split()[1]

except IndexError:

bot.send\_message(chat\_id, "❌ Укажите ID пользователя! Пример: /admin\_delete 123456789")

return

users = load\_users()

if user\_id not in users:

bot.send\_message(chat\_id, "❌ Пользователь не найден!")

return

delete\_user(user\_id)

if user\_id in signed\_in\_users:

signed\_in\_users.remove(user\_id)

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

def handle\_admin\_add(chat\_id, text):

if not is\_admin(chat\_id):

bot.send\_message(chat\_id, "❌ Вы не администратор!")

return

try:

user\_id = text.split()[1]

except IndexError:

bot.send\_message(chat\_id, "❌ Укажите ID пользователя! Пример: /admin\_add 123456789")

return

users = load\_users()

if user\_id not in users:

bot.send\_message(chat\_id, "❌ Пользователь не найден!")

return

add\_admin(user\_id)

bot.send\_message(chat\_id, f"✅ Пользователь {user\_id} назначен администратором!")

def process\_image(file\_id, chat\_id):

if model is None:

bot.send\_message(chat\_id, "❌ Модель не загружена. Пожалуйста, свяжитесь с администратором.")

return

try:

# Получаем файл изображения

file\_info = bot.get\_file(file\_id)

file\_url = f"https://api.telegram.org/file/bot{TOKEN}/{file\_info.file\_path}"

response = requests.get(file\_url)

img = Image.open(io.BytesIO(response.content))

# Конвертируем изображение в нужный формат

img = img.resize((200, 200))

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

x = x / 255.0 # Нормализация

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

# Предсказание

classes = model.predict(x, batch\_size=1)

probability = classes[0][0]

# Определяем класс

if probability < 0.5:

result = "human"

prob\_text = f"🧍‍♂️🧍‍♀️ Вероятность человека: {(1 - probability):.4f}\n🦒 Вероятность жирафа: {probability:.4f}"

else:

result = "giraffe"

prob\_text = f"🧍‍♂️🧍‍♀️ Вероятность человека: {(1 - probability):.4f}\n🦒 Вероятность жирафа: {probability:.4f}"

bot.send\_message(chat\_id, f"Изображение классифицировано как {result}!\n{prob\_text}")

increment\_prediction\_count(chat\_id)

except Exception as e:

bot.send\_message(chat\_id, f"❌ Ошибка обработки изображения: {str(e)}. Попробуйте ещё раз.")

# Основной обработчик вебхука

@app.route('/telegram\_webhook', methods=['POST'])

def telegram\_webhook():

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

update = request.get\_json()

handle\_update(update)

return jsonify({'status': 'ok'})

return jsonify({'status': 'error'}), 403

def handle\_update(update):

"""Обработка входящего обновления от Telegram"""

if 'message' not in update:

return

message = update['message']

chat\_id = str(message['chat']['id'])

text = message.get('text', '')

# Проверка состояний пользователя перед обработкой команд

if chat\_id in user\_states:

state = user\_states[chat\_id]

if state == 'awaiting\_password':

password = text.strip()

if not is\_strong\_password(password):

bot.send\_message(chat\_id, "❌ Пароль должен быть минимум 8 символов, содержать буквы разного регистра, цифры и специальные символы. Попробуйте ещё раз:")

return

success, msg = save\_user(chat\_id, password)

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

if success:

bot.send\_message(chat\_id, "✅ Теперь войдите с помощью /login.")

del user\_states[chat\_id]

return

elif state == 'awaiting\_login\_password':

input\_password = text.strip()

success, msg = check\_login(chat\_id, input\_password)

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

if success:

signed\_in\_users.append(chat\_id)

del user\_states[chat\_id]

return

# Обработка команд

if text.startswith('/'):

command = text.split()[0].lower()

if command == '/start':

handle\_start(chat\_id)

elif command == '/help':

handle\_help(chat\_id)

elif command == '/register':

handle\_register(chat\_id)

elif command == '/login':

handle\_login(chat\_id)

elif command == '/logout':

handle\_logout(chat\_id)

elif command == '/predict':

handle\_predict(chat\_id)

elif command == '/help\_admin':

handle\_help\_admin(chat\_id)

elif command == '/become\_admin':

handle\_become\_admin(chat\_id, text)

elif command == '/admin\_list':

handle\_admin\_list(chat\_id)

elif command == '/admin\_delete':

handle\_admin\_delete(chat\_id, text)

elif command == '/admin\_add':

handle\_admin\_add(chat\_id, text)

# Обработка изображений для команды /predict

elif 'photo' in message and chat\_id in signed\_in\_users and user\_states.get(chat\_id) == 'awaiting\_image':

file\_id = message['photo'][-1]['file\_id']

process\_image(file\_id, chat\_id)

del user\_states[chat\_id]

@app.route('/')

def home():

return "Webhook Server is Running"

# Регистрируем вебхук при старте приложения

set\_telegram\_webhook()