Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет
Петра Великого» (ФГАОУ ВО СПбПУ)
Институт компьютерных наук и кибербезопасности
Высшая школа искусственного интеллекта

# КУРСОВОЙ ПРОЕКТ

По дисциплине «Разработка приложений на Python»

Тема: «Система распознавания лиц по фотографиям с телеграма»
(семестр V)

Выполнили студентки
группы 5130203/20101 Т. Г. Гончарова
А.Д. Михеева
Ф. И. Карталиева

Оценка выполненной студентом работы:

Преподаватель Н. А. Вениаминов

Санкт-Петербург – 2024

## Содержание

## Введение

Курсовой проект направлен на отработку теоретических знаний курса по разработке приложений на языке Python.

Цель проекта: создание интегрированной системы, способной не только распознавать лица на фотографиях, но и отслеживать количество появлений каждой личности, сохраняя эти данные в базе данных.

Для достижения поставленной цели проект предусматривает решение следующих задач:

1. Разработка бота для Telegram, который будет принимать фотографии от пользователей и передавать их для дальнейшей обработки.
2. Реализация системы распознавания лиц.
3. Создание базы данных для хранения информации о распознанных лицах, включая количество их появлений.
4. Интеграция всех компонентов системы для обеспечения её эффективного функционирования.
5. Тестирование.

Таким образом, в рамках данного курсового проекта будет создана полноценная система, объединяющая в себе три ключевых компонента: телеграмм-бот, модуль распознавания лиц и базу данных, обеспечивающую хранение и управление полученными данными.

# 1. Разработка бота для Telegram

## 1.1. Выбор технологий для создания телеграмм бота

Для реализации Telegram-бота в курсовой работе была выбрана библиотека python-telegram-bot. Эта библиотека предоставляет удобный интерфейс для взаимодействия с API Telegram, который позволяет разработать функционального бота с минимальными усилиями. Она поддерживает следующие ключевые функции:

- Позволяет легко создавать обработчики для различных команд и текстовых сообщений от пользователей.
- Поддерживает обработку текстовых сообщений, документов, фотографий и других типов контента.
- Позволяет обрабатывать несколько запросов одновременно, что улучшает производительность бота.
- Позволяет отправлять текстовые сообщения, изображения, документы и другие типы медиа.

Причины выбора python-telegram-bot:

- Библиотека имеет интуитивно понятный интерфейс, позволяющий быстро начать разработку бота без необходимости глубокого изучения API Telegram.
- Поддерживает множество функций, включая inline-режим, кнопки, клавиатуры и другие интерактивные элементы, что позволяет создавать более сложные и привлекательные интерфейсы для пользователей.
- Библиотека имеет обширную документацию и активное сообщество (легко найти решение возникшей проблемы)
- Легкая интеграция с другими библиотеками.

## 1.2. Описание файла app.py

1) Описание класса TelegramBot

Класс TelegramBot отвечает за создание и управление функциональностью бота. Рассмотрим основные методы класса и их назначения.

```
class TelegramBot:
    def __init__(self, token: str):
        self.token = token
        self.application = ApplicationBuilder().token(self.token).build()
```

Конструктор класса, инициализирующий экземпляр бота с заданным токеном API. Токен используется для аутентификации при взаимодействии с API Telegram.

Обработчики команд:

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):

await update.message.reply_text("Привет! Отправь мне изображение для распознавания лиц.")

2) start(update: Update, context: ContextTypes.DEFAULT_TYPE):

Этот метод обрабатывает команду /start. Запускается при активации бота и отвечает за отправку пользователю меню с выбором действий. Она создает две кнопки: одну для очистки данных и другую для распознавания человека. Кнопки оформлены с помощью InlineKeyboardButton и помещены в разметку InlineKeyboardMarkup. После создания кнопок, функция отправляет сообщение с просьбой выбрать действие, прикрепляя к нему разметку с кнопками.

3) Функция handle_menu_choice

Функция обрабатывает выбор пользователя из меню, созданного функцией start. Она получает информацию о нажатой кнопке через query.callback_query и отвечает на запрос. Если пользователь выбрал опцию очистки базы данных, функция вызывает метод clear_database из объекта database и сообщает пользователю о результате операции. В случае успеха отправляется сообщение о том, что база данных очищена, а в случае ошибки - сообщение с описанием проблемы. Если пользователь выбрал опцию распознавания человека, ему предлагается отправить фотографию для дальнейшей обработки. Если выбор не соответствует ни одной из предусмотренных опций, отправляется сообщение об ошибке.

4) Класс Handle_document(handle_photo):

Функция handle_document предназначена для обработки документов, отправленных пользователями в Telegram. Она выполняет проверку формата файла, загружает его, обрабатывает и возвращает результат пользователю.

Параметры

- update: объект, содержащий информацию о полученном обновлении (например, сообщение от пользователя).
- context: объект, содержащий контекст выполнения, который может включать дополнительные данные о состоянии бота и пользователе.
- Проверка формата файла:
    - Сначала функция проверяет, имеет ли файл расширение .jpg. Если нет, отправляется сообщение с просьбой отправить корректный файл.
- Скачивание файла:

    - Если файл имеет правильный формат, он скачивается с использованием метода get_file и сохраняется в указанной папке.
- Проверка формата изображения:
    - После загрузки файла вызывается функция is_jpg_file для проверки, является ли файл корректным изображением в формате JPG. Если файл не подходит, он удаляется, и пользователю отправляется соответствующее сообщение.
- Обработка изображения:
    - Если файл корректный, происходит его обработка с использованием метода recognize_and_update класса FaceRecognizer, который выполняет распознавание лиц.
- Отправка результата:
    - Результат обработки анализируется. Если это строка, она разбивается на ключи и значения для дальнейшей интерпретации статуса. В зависимости от статуса (ошибка, неоднозначно, успешно) пользователю отправляется соответствующее сообщение.
- Обработка исключений:
    - Если возникает ошибка на любом этапе, она логируется, и пользователю отправляется сообщение об ошибке.

# 2. Реализация системы распознавания лиц

## 2.1. Выбор библиотеки

Для реализации системы распознавания лиц в курсовой работе была выбрана библиотека face_recognition. Эта библиотека построена на основе мощной библиотеки машинного обучения dlib и предоставляет простой интерфейс для выполнения сложных задач распознавания лиц. Она поддерживает следующие ключевые функции:

- Обнаружение лиц на изображениях.
- Определение ключевых точек лица (например, глаз, носа, рта).
- Извлечение эмбеддингов лиц для их дальнейшего сравнения.
- Сравнение лиц и определение степени их сходства.

Причины выбора face_recognition:

- Библиотека предлагает интуитивно понятные функции и методы, которые позволяют быстро интегрировать распознавание лиц в проект без необходимости глубокого погружения в сложные алгоритмы машинного обучения.
- Обеспечивает высокую точность распознавания лиц.
- Предоставляет готовые к использованию функции для извлечения эмбеддингов, которые затем могут быть использованы для сравнения лиц, что идеально подходит для целей проекта.
- Библиотека легко интегрируется с другими инструментами и библиотеками Python.

## 2.2. Описание класса FaceRecognizer

Класс FaceRecognizer предназначен для распознавания лиц и обновления информации о них в базе данных. Кратко рассмотрим значения методов в классе:

1) __init__(self, database: IDatabase, threshold=0.6)

Конструктор класса, инициализирующий экземпляр с заданной базой данных и порогом для определения сходства лиц. Порог определяет максимальное допустимое евклидово расстояние между эмбеддингами двух лиц для их идентификации как одного человека.

2) recognize_and_update(self, image_path)

Основной метод класса, который принимает путь к изображению, извлекает эмбеддинг лица с помощью метода _extract_embedding_from_image, и на основе эмбеддинга определяет, принадлежит ли лицо известному человеку из базы данных. Если лицо не распознано, добавляется новая запись. Если лицо совпадает с существующей записью, база данных обновляется новым эмбеддингом.

3) _extract_embedding_from_image(self, image_path)

Вспомогательный метод, который загружает изображение, извлекает из него эмбеддинг лица с использованием библиотеки face_recognition. Возвращает эмбеддинг, если на изображении найдено одно лицо, иначе возвращает None.

4) _recognize_and_update_from_embedding(self, embedding)

Метод, который сравнивает полученный эмбеддинг с эмбеддингами, хранящимися в базе данных. Если эмбеддинг совпадает с одним из существующих, база данных обновляется. В противном случае добавляется новая запись с новым идентификатором.

5) _calculate_distance(self, embedding1, embedding2)

Метод для вычисления евклидова расстояния между двумя эмбеддингами. Используется для определения схожести лиц.

6) _format_result(self, result)

Форматирует результат работы метода recognize_and_update в удобочитаемую строку для вывода.

Соответствие принципам SOLID:

1) S (Single Responsibility Principle)

Каждый метод класса FaceRecognizer выполняет одну конкретную задачу. Например, метод _extract_embedding_from_image отвечает только за извлечение эмбеддинга, а _calculate_distance - за вычисление расстояния между эмбеддингами.

2) O (Open/Closed Principle)

Класс открыт для расширения, но закрыт для модификации. Например, можно добавить новые методы для обработки эмбеддингов или изменения логики сравнения, не изменяя существующий код.

3) L (Liskov Substitution Principle)

Класс FaceRecognizer использует интерфейс IDatabase для работы с базой данных, что позволяет заменять реализацию базы данных на другую, не изменяя логику работы класса.

4) I (Interface Segregation Principle)

Интерфейс IDatabase обеспечивает только необходимые методы для работы с базой данных, избегая предоставления избыточных функций.

5) D (Dependency Inversion Principle)

Класс FaceRecognizer зависит от абстракции IDatabase, а не от конкретной реализации базы данных. Это делает код легко адаптируемым к изменениям в реализации базы данных.

# 3. Создание базы данных

## 3.1. Выбор технологий для создания базы данных

В качестве СУБД был выбрал PostgreSQL. PostgreSQL обеспечивает стабильную и надежную работу с данными, что критично для реализации системы распознавания лиц.

В нашем приложении мы использовали систему Railway, которая является облачной платформой для развертывания и управления приложениями. Она предоставляет удобный интерфейс для работы с базами данных, упрощая процесс настройки.

Преимущества данной системы:

1. Простота использования: Railway имеет интуитивно понятный интерфейс, что облегчает настройку баз данных, включая PostgreSQL, с минимальными усилиями.
2. Бесплатный тариф: Для начальных этапов разработки Railway предоставляет бесплатный тариф с достаточным лимитом ресурсов.
3. Удобный доступ: Доступ к базе данных возможен из любого места через предоставленные URL и учетные данные.

## 3.2. Проектирование базы данных

Для нашей задачи база данных должна была иметь возможность хранения следующих данных:

1. Для каждого человека:
    - Уникальный идентификатор;
    - Количество появлений на фото.
2. Для каждого эмбеддинга:
    - Уникальный идентификатор;
    - Связь с человеком (чтобы определить, к какому человеку относится эмбеддинг);

- Сам эмбеддинг в виде массива чисел (многомерный вектор);
- Уникальный хэш для идентификации эмбеддинга.

Определение сущностей и их атрибутов:

1. Person (Человек):
   - id (Integer): уникальный идентификатор (первичный ключ);
   - appearance_count (Integer): число появлений человека.

   Связь с сущностью Embedding (один ко многим).

2. Embedding (Эмбеддинг):
   - id (Integer): уникальный идентификатор (первичный ключ).
   - person_id (Integer): внешний ключ, связывающий эмбеддинг с человеком.
   - embedding (ARRAY(Float)): массив чисел (многомерный вектор).
   - embedding_hash (BIGINT): уникальный хэш для быстрого сравнения эмбеддингов.

   Связь между Person и Embedding:

   - Отношение "один ко многим" (один человек может иметь много эмбеддингов).
   - Связь устанавливается через внешний ключ person_id в таблице embedding.
   - Каскадное удаление должно гарантировать, что при удалении записи о человеке все связанные эмбеддинги будут автоматически удалены.

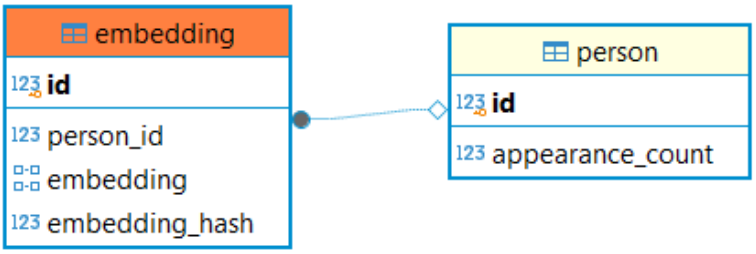



Рисунок 3.1. Схема БД.

### 3.2. Создание базы данных

Для реализации таблиц используется SQLAlchemy как ORM, что позволяет работать с базой данных на уровне объектов.

ORM (Object-Relational Mapping) — это технология, которая позволяет взаимодействовать с базой данных, используя объектно-ориентированный подход вместо прямого написания SQL-запросов. ORM автоматически преобразует объекты в строки базы данных (и наоборот), упрощая работу с данными.

```python
class Person(Base):
    __tablename__ = 'person'
    id = Column(Integer, primary_key=True)
    appearance_count = Column(Integer, default=0)
    embeddings = relationship(
        "Embedding",
        back_populates="person",
        cascade="all, delete-orphan",
        lazy="dynamic",
    )

class Embedding(Base):
    __tablename__ = 'embedding'
    id = Column(Integer, primary_key=True)
    person_id = Column(Integer, ForeignKey('person.id', ondelete="CASCADE"))
    embedding = Column(ARRAY(Float), nullable=False)
    embedding_hash = Column(BIGINT, unique=True, nullable=False)
    person = relationship("Person", back_populates="embeddings")
```

Рисунок. 3.2. Классы для создания таблиц.

```python
class FaceDatabase(IDatabase):
    def __init__(self, database_url, cache_manager):
        """
        Инициализация базы данных и системы кэширования.
        """
        try:
            self.engine = create_engine(database_url)
            self.Session = sessionmaker(bind=self.engine)
            self._initialize_database()
        except Exception as e:
            raise ConnectionError(f"Не удалось подключиться к базе данных: {str(e)}")

        self.cache_manager = cache_manager

    def _initialize_database(self):
        Base.metadata.create_all(self.engine)
```

Рисунок 3.3. Инициализация БД.

## 3.3. Работы с базой данных

В файле db.py определены следующие элементы:

1. Класс IDatabase(ABC)- Интерфейс для работы с базой данных.
2. Класс CacheManager - Класс для управления кэшированием.
3. Классы Person, Embedding.
4. Класс FaceDatabase(IDatabase) - Класс для работы с нашей БД.

### *3.3.1. Интерфейс IDatabase*

Интерфейс IDatabase определяет методы для взаимодействия с базой данных, обеспечивая абстракцию над реализацией.

Методы интерфейса:

1. add_person_with_embedding(embedding) – Добавляет нового человека вместе с эмбеддингом в базу данных.
2. add_embedding(person_id, embedding) – Добавляет эмбеддинг к существующему человеку.
3. get_all_embeddings() - Возвращает все эмбеддинги из базы данных.

4. get_embeddings(person_id) – Возвращает список эмбеддингов для конкретного человека.
5. increment_appearance(person_id) – Увеличивает счётчик появления для конкретного человека.
6. get_appearance_count(person_id) – Возвращает количество появлений для конкретного человека.
7. clear_database() – Очищает базу данных, удаляя все записи.

Интерфейс обеспечивает единообразие при взаимодействии с различными реализациями базы данных и позволяет легко изменять реализацию при необходимости.

### *3.3.2. Класс CacheManager*

CacheManager — это класс для управления кэшированием данных, чтобы минимизировать обращения к базе данных и повысить производительность.

Атрибуты класса:

1. cache – хранит кэшированные данные.
2. cache_timestamp – сохраняет время последнего обновления кэша и используется для проверки актуальности данных.
3. cache_lifetime – указывает, как долго кэш считается актуальным (в секундах).

Методы класса:

1. __init__(cache_lifetime=60) – Инициализирует объект кэша с заданным временем жизни.
2. is_cache_valid() – Проверяет, является ли текущий кэш актуальным.
3. get_cache(force_refresh=False) – Кэшированные данные или None, если кэш неактуален или отсутствует.
4. refresh_cache(data) – Обновляет кэш с новыми данными.

### *3.3.2. Класс FaceDatabase*

FaceDatabase — это реализация интерфейса IDatabase, предназначенная для работы с базой данных, содержащей информацию о людях и их эмбеддингах. Класс использует SQLAlchemy для взаимодействия с базой данных и интеграции с системой кэширования через CacheManager.

Атрибуты класса:

1. engine – объект SQLAlchemy Engine для подключения к базе данных.
2. Session – сессия SQLAlchemy для выполнения операций с базой данных.
3. cache_manager – объект класса CacheManager, который управляет кэшированием данных.

Методы класса:

1. __init__(database_url, cache_manager) – Инициализирует подключение к базе данных и объект кэширования.
2. _initialize_database() – Создаёт таблицы в базе данных на основе моделей SQLAlchemy.
3. calculate_embedding_hash(embedding)
    - Вычисляет хэш для эмбеддинга, чтобы проверить его уникальность.
    - Возвращает хэш эмбеддинга.
4. validate_embedding(embedding)
    - Проверяет корректность эмбеддинга.
    - Генерирует ValueError при некорректных данных.
5. add_person_with_embedding(embedding)
    - Добавляет нового человека с эмбеддингом. Возвращает ID созданного человека.
    - Возвращает ID созданного человека.

6. add_embedding(person_id, embedding)
    - Добавляет эмбеддинг для существующего человека.
7. get_all_embeddings_from_db()
    - Получает все эмбеддинги из базы данных без использования кэша.
    - Возвращает словарь {person_id: [embeddings]}
8. get_all_embeddings()
    -
9. get_embeddings(person_id)
    - Возвращает все эмбеддинги для заданного человека.
10. increment_appearance(person_id)
    - Увеличивает счётчик появления для человека.
11. get_appearance_count(person_id)
    - Возвращает количество появлений человека.
    - Количество появлений (int).
12. clear_database()
    - Удаляет все данные из базы данных.
    - Сбрасывает кэш.

### *3.3.4. Соответствие принципам SOLID*

Программа хорошо соответствует принципам SOLID:

- Она разделяет ответственность между классами, не нарушая принцип единой ответственности.
- Программа открыта для расширения, но закрыта для изменения.
- Используются абстракции, обеспечивающие подстановку классов и инверсию зависимостей, что также соответствует принципам подстановки Лисков и инверсии зависимостей.
- Интерфейс достаточно специфичен и не перегружен лишними методами, что соответствует принципу разделения интерфейса.

Таким образом, архитектура программы спроектирована в соответствии с SOLID, что способствует ее гибкости, расширяемости и поддерживаемости.

## 4. Интеграция всех компонентов системы

В предыдущих главах было описана разработка отдельных компонентов системы. Проект состоит из нескольких взаимосвязанных частей: модуля с распознаванием лиц на фото, базы данных для хранения информации о лицах и эмбеддингах, системы кэширования, а также Telegram-бота для взаимодействия с пользователями. Интеграция этих компонентов обеспечила слаженную работу всей системы.

Класс FaceRecognizer отвечает за обработку изображений, извлечение эмбеддингов лиц и их сравнение с существующими записями в базе данных. Он взаимодействует с базой данных через функции find_user_by_embedding и add_user для поиска и добавления записей. Обменивается данными с кэшем (CacheManager), чтобы оптимизировать производительность.

Для хранения информации о пользователях и их эмбеддингах используется PostgreSQL. Взаимодействие с базой данных реализовано через ORM SQLAlchemy. FaceRecognizer обращается к базе данных через ORM.

Система кэширования реализована с целью уменьшения количества запросов к базе данных и ускорения работы системы. CacheManager управляет кэшированием эмбеддингов и проверяет их актуальность в течение заданного времени. Если кэш устарел, данные обновляются из базы данных. FaceRecognizer проверяет кэш перед обработкой изображения и сохраняет в него результаты для будущего использования.

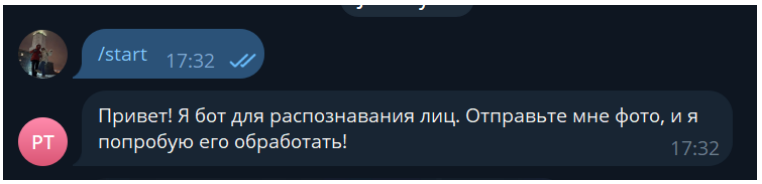
Для взаимодействия с пользователями был разработан Telegram-бот, который принимает изображения, обрабатывает их через модуль FaceRecognizer и возвращает пользователю результат распознавания. Бот Использует библиотеку python-telegram-bot для работы с Telegram API.

Интеграция началась с разработки и тестирования каждого компонента отдельно. После успешного тестирования части были объединены. Telegram-бот взаимодействует с модулем FaceRecognizer, который использует базу данных для хранения и получения эмбеддингов. Система кэширования также интегрирована в работу с базой данных для повышения производительности. Итоговая система была протестирована на разных изображениях для проверки корректности работы всех компонентов.

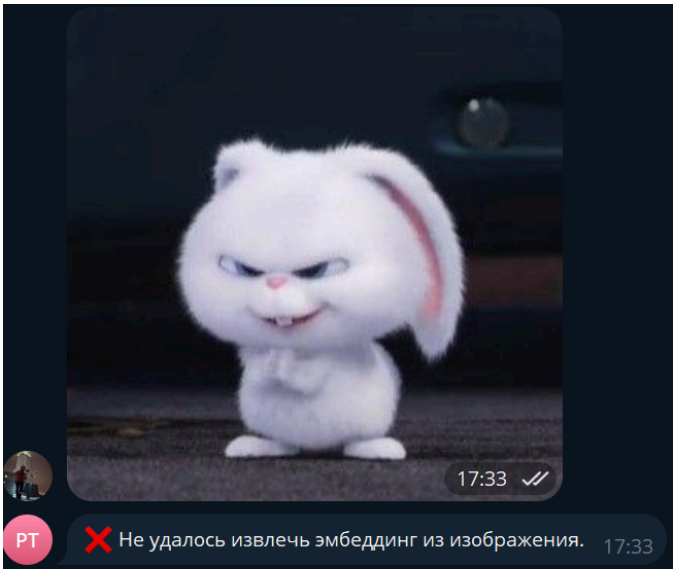
Взаимодействие компонентов в итоге выглядит так:

1) Telegram-бот получает изображение от пользователя.
2) Изображение передается в FaceRecognizer, где оно обрабатывается, и извлекается эмбеддинг.
3) FaceRecognizer проверяет кэш через CacheManager. Если эмбеддинг уже есть в кэше, он используется сразу.
4) Если эмбеддинг новый, FaceRecognizer обращается к базе данных для поиска соответствующего пользователя.
5) Если пользователь найден, бот отправляет сообщение, о том, что лицо успешно распознано, выводит id личности с фотографии и количество сканирования этой личности. Если нет, бот добавляет нового человека в базу данных и уведомляет об этом пользователя.

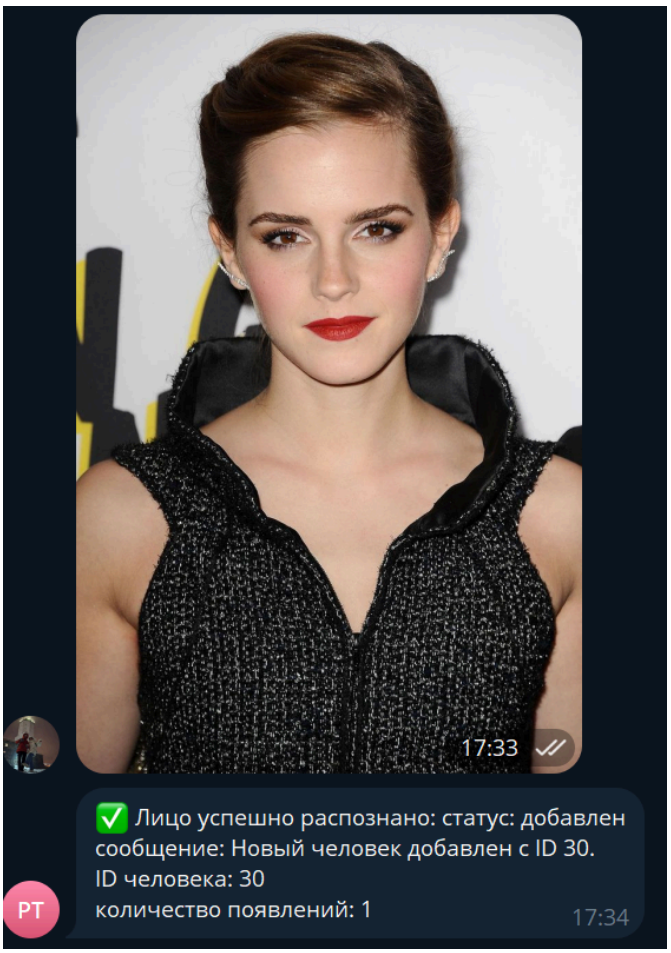
## 5. Тестирование

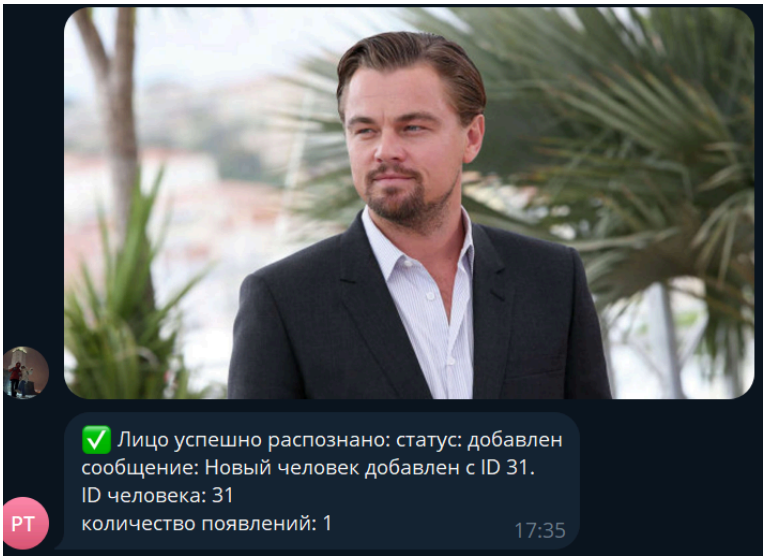



После запуска команды /start бот приветствует пользователя сообщением, в котором сообщает, что ему нужно сделать.

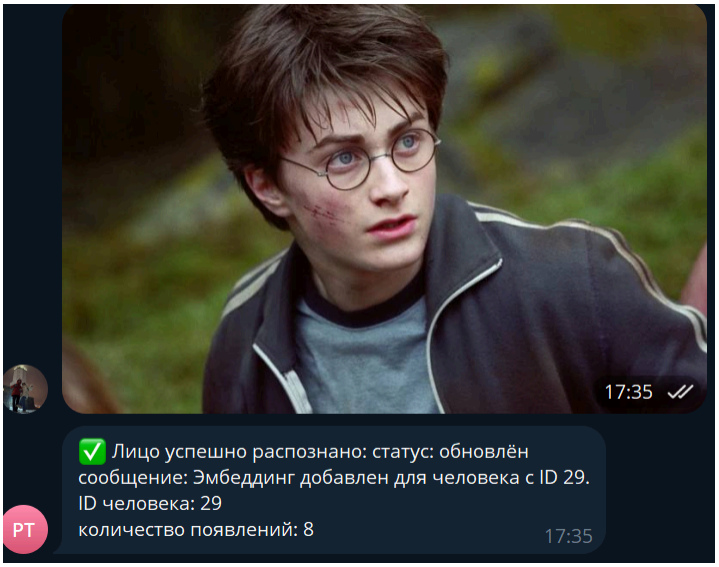



Этот тест показывает, что бот корректно работает с картинками, где нет человеческих лиц.




Еще примеры по распознаванию лиц.


Эти тесты демонстрируют процесс распознавания лица, которое уже было добавлено в систему. Бот успешно находит совпадение и возвращает информацию о пользователе (id, количество появлений).

## Заключение

В рамках данного курсового проекта была разработана и успешно протестирована система распознавания лиц, интегрирующая Telegram-бота, модуль распознавания лиц и базу данных. Все части системы работают слаженно, обеспечивая корректную обработку изображений.

Проект позволил применить на практике изученные в рамках курса концепции и технологии по разработке приложений на языке Python. Это поспособствовало углублению понимания теоретических основ и приобретению практических навыков, необходимых для будущей профессиональной деятельности.

Проект может быть расширен и улучшен в будущем для обработки больших объемов данных и повышения точности распознавания.