

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

ОТЧЁТ ПО ТЕХНОЛОГИЧЕСКОЙ (ПРОЕКТНО-ТЕХНОЛОГИЧЕСКОЙ)

ПРАКТИКЕ НА ТЕМУ:

Телеграм бот на Python 3

Выполнил студент 5.205-2 гр.

_____ Р.М. Плахотнюк

Научный руководитель:

Проверил: Ст. преп. каф. ВТ и Э

_____ И.А. Шмаков

Курсовая работа защищена:

«____» _____ 2024 г.

Оценка _____

Барнаул 2024

РЕФЕРАТ

Полный объём работы составляет 29 страниц, включая 3 рисунка и 0 таблиц, 14 использованных источников.

ТЕЛЕГРАМ БОТ, ПОГОДА, ВЛАЖНОСТЬ, ПОХОЖИЕ ИЗОБРАЖЕНИЯ, PYTELEGRAMBOTAPI, OPENWEATHER API, GOOGLE API, GOOGLE CX, TRANSFORMERS, HUGGING FACE, PYTHON 3.

В работе показана разработка телеграм-бота на Python 3, который предоставляет информацию о погоде в различных городах. Бот выводит следующие данные: Состояние погоды, температуру, максимальную и минимальную температуры, влажность, давление и скорость ветра. Также бот умеет сравнивать влажность в нескольких городах и выводить город с наименьшей влажностью. Кроме того, бот может искать похожие изображения по отправленной картинке. Также был проведен успешный тест работоспособности бота.

Отчет оформлен с помощью текстового редактора Microsoft Word.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	5
1.1. Основные параметры погоды.....	5
1.2. Основные возможности телеграм-ботов	6
1.3. API и модели для разработки телеграм-бота.....	6
1.4. Работа с ответами от API в формате JSON.....	8
1.5. Обоснование выбора языка программирования.....	9
1.6. Библиотека PIL (Python Imaging Library)	10
1.7. Модели BlipProcessor и BlipForConditionalGeneration	11
1.8. Обоснование выбора среды разработки программного приложения .	11
2. РАЗРАБОТКА ПРОГРАММЫ.....	13
2.1. Требования к приложению.....	13
2.2. Блок-схема программы	14
2.3. Описание процесса разработки.....	15
2.4. Проверка работоспособности (тестирование) программы	19
ЗАКЛЮЧЕНИЕ	21
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	22
ПРИЛОЖЕНИЕ.....	24

ВВЕДЕНИЕ

В современном мире скорее всего у каждого человека в кармане есть смартфон, на котором в качестве мессенджера установлен Telegram. Телеграм-боты становятся все более популярными и востребованными, помогая пользователям получать необходимую информацию быстро и удобно. Одной из таких полезных функций является предоставление данных о погоде в различных городах.

Цель работы: Создание телеграм-бота на Python 3, который выводит информацию о погоде, сравнивает влажность в нескольких городах и ищет похожие изображения по отправленной картинке.

Задачи, для достижения цели работы, были поставлены следующие:

- Изучить существующие телеграм-боты с аналогичным функционалом;
- Изучить основные приемы работы с API для получения данных о погоде и поиска изображений;
- Сформулировать перечень требований к боту;
- Разработать телеграм-бота в соответствии с требованиями;
- Выполнить проверку работоспособности бота.

Практическая значимость:

- Развитие навыков: Работа над проектом позволит развить навыки программирования, тестирования и отладки.
- Создание готового продукта: Результатом работы станет функциональный телеграм-бот, который можно использовать для получения информации о погоде, сравнения влажности в различных городах и поиска похожих изображений по отправленной картинке.
- Возможность применения: Полученные навыки могут быть применены для разработки других телеграм-ботов или программных решений, связанных с обработкой данных о погоде и изображениях.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. Основные параметры погоды

Погода [1] — это состояние атмосферы в определенный момент времени и в определенном месте. Она включает в себя различные параметры, такие как температура, влажность, давление, скорость ветра и состояние неба.

Самые важные параметры погоды, которые будет предоставлять телеграм-бот:

- **Состояние погоды** - описание текущего состояния неба, которое может включать такие понятия, как ясно, пасмурно, дождь, снег и другие. Этот параметр помогает пользователям понять, какая погода ожидается в ближайшее время.
- **Температура** - измеряется в градусах Цельсия (°C) и включает в себя текущую температуру, а также максимальную и минимальную температуру за день. Этот параметр важен для планирования дневной активности и подбора одежды.
- **Влажность** — это количество водяного пара в воздухе, измеряемое в процентах (%). Влажность влияет на ощущение температуры и комфорт. Например, высокая влажность может делать жаркую погоду еще более некомфортной.
- **Давление** — это сила, с которой воздух давит на поверхность, измеряемая в миллиметрах ртутного столба (мм рт. ст.). Атмосферное давление может влиять на погодные условия и самочувствие человека.
- **Скорость ветра** - измеряется в метрах в секунду (м/с) и показывает, насколько сильный ветер ожидается в данном месте. Скорость ветра важна для различных видов деятельности, таких как парусный спорт, авиация и строительство.

Эти параметры погоды являются основными и наиболее часто используемыми для описания текущих погодных условий.

1.2. Основные возможности телеграм-ботов

Телеграм-боты[2] — это программные агенты, которые взаимодействуют с пользователями через мессенджер Telegram, предоставляя им различные функции и услуги. Они могут выполнять широкий спектр задач, от простых запросов информации до сложных автоматизированных процессов.

- **Интерактивное взаимодействие** - боты могут отправлять и получать сообщения, обрабатывать команды и запросы пользователей, а также взаимодействовать с внешними API для получения и обработки данных.
- **Автоматизация задач** - боты могут автоматизировать рутинные задачи, такие как напоминания, отправка уведомлений, сбор и анализ данных.
- **Интеграция с внешними сервисами** - боты могут интегрироваться с различными внешними сервисами и API для получения данных о погоде, новостях, курсах валют и многого другого.
- **Обработка мультимедийного контента** - боты могут обрабатывать и анализировать изображения, видео и аудиофайлы, предоставляя пользователям полезную информацию и рекомендации.

1.3. API и модели для разработки телеграм-бота

API (Application Programming Interface) — это набор инструментов и протоколов, которые позволяют разработчикам создавать программное обеспечение, взаимодействующее с другими сервисами и системами. В контексте разработки телеграм-бота для получения информации о погоде, сравнения влажности и поиска похожих изображений, используются следующие API и модели:

- **PyTelegramBotApi[3]** - библиотека для создания и управления телеграм-ботами на Python. Она позволяет отправлять и получать сообщения, а

также обрабатывать команды и запросы пользователей. Основные параметры pytelegrambotapi:

- **Токен бота** - уникальный идентификатор, который используется для аутентификации бота в Telegram.
- **Обработчики команд** - функции, которые выполняются при получении определенных команд от пользователей.
- **Обработчики сообщений** - функции, которые выполняются при получении текстовых сообщений от пользователей.
- **Openweathermap api[4]** - сервис для получения данных о погоде. Он предоставляет информацию о текущей погоде, прогнозах и исторических данных. Основные параметры openweathermap api:
 - **API ключ** - уникальный идентификатор, который используется для доступа к данным о погоде.
 - **Запросы к API** - параметры, которые определяют, какие данные о погоде будут получены (например, текущая погода, прогноз на несколько дней).
 - **Формат ответа** - формат данных, в котором будет предоставлена информация о погоде (например, JSON).
- **Google api[5] и google cx[6]** - сервисы для поиска изображений. Они позволяют искать похожие изображения по заданным критериям. Основные параметры google api и google cx:
 - **API ключ** - уникальный идентификатор, который используется для доступа к поисковым функциям.
 - **Запросы к API** - параметры, которые определяют, какие изображения будут найдены (например, ключевые слова, фильтры).
 - **Формат ответа** - формат данных, в котором будут предоставлены результаты поиска (например, JSON).

- **Transformers от Hugging Face[7]** - библиотека предобученных моделей и процессоров для обработки естественного языка и изображений. Основные параметры transformers:
 - **Модели** - предобученные модели, которые используются для анализа текста и изображений.
 - **Процессоры** - инструменты для предварительной обработки данных перед их анализом.
 - **Функции анализа** - функции, которые выполняют анализ данных и возвращают результаты (например, описание объектов на изображении).

1.4. Работа с ответами от API в формате JSON

Ответы от API обычно предоставляются в формате JSON[8] (JavaScript Object Notation), который является текстовым форматом обмена данными, легко читаемым как для человека, так и для машины. JSON использует структуру "ключ-значение" для хранения данных, что делает его удобным для парсинга и извлечения информации.

Работа с JSON-ответами включает несколько ключевых шагов:

1. **Отправка запроса к API:** Для получения данных необходимо отправить запрос к API с использованием соответствующих параметров, таких как API ключ и запрашиваемые данные.
2. **Получение ответа:** После отправки запроса API возвращает ответ в формате JSON. Этот ответ содержит все необходимые данные, запрашиваемые пользователем.
3. **Парсинг JSON-ответа:** Полученный JSON-ответ необходимо распарсить, чтобы извлечь нужные данные. В Python для этого часто используется библиотека json. Парсинг включает преобразование JSON-строки в объект Python[9], такой как словарь или список.
4. **Извлечение данных:** После парсинга JSON-ответа можно извлечь необходимые данные, такие как состояние погоды, температура,

влажность, давление и скорость ветра. Это делается путем обращения к соответствующим ключам в JSON-объекте.

5. **Обработка данных:** Извлеченные данные могут быть обработаны и преобразованы в удобный для пользователя формат. Например, температура может быть преобразована из Кельвинов в Цельсии, а давление - из гектопаскалей в миллиметры ртутного столба.
6. **Отправка данных пользователю:** После обработки данные отправляются пользователю в удобном для него формате, например, в виде текстового сообщения в Telegram.

1.5. Обоснование выбора языка программирования

Python 3[10] — это современный высокоуровневый язык программирования, который широко используется для разработки различных типов приложений, включая веб-приложения, научные вычисления, анализ данных и искусственный интеллект. Вот несколько ключевых причин, почему Python 3 был выбран для разработки телеграм-бота:

- Простота и читаемость кода: Python известен своей простотой и читаемостью кода, что делает его идеальным для быстрого прототипирования и разработки. Это позволяет разработчикам сосредоточиться на решении задач, а не на синтаксисе языка.
- Богатая стандартная библиотека: Python 3 имеет обширную стандартную библиотеку, которая включает модули для работы с сетью, файловой системой, веб-сервисами и многим другим. Это упрощает разработку и уменьшает необходимость в сторонних библиотеках.
- Поддержка библиотек и фреймворков: Python 3 поддерживает множество библиотек и фреймворков, таких как `pytelegrambotapi`, `requests`, `transformers` и другие, которые значительно упрощают работу с API и обработку данных.
- Кроссплатформенность: Python 3[11] является кроссплатформенным языком, что позволяет запускать код на различных операционных

системах, таких как Windows, macOS и Linux, без необходимости внесения значительных изменений.

- **Сообщество и поддержка:** Python имеет большое и активное сообщество разработчиков, что обеспечивает наличие множества ресурсов, документации и примеров кода. Это упрощает процесс обучения и решения проблем.
- **Управление памятью:** Python 3[12] автоматически управляет памятью, что освобождает разработчика от необходимости вручную управлять выделением и освобождением памяти. Это снижает вероятность ошибок, связанных с управлением памятью.
- **Интеграция с другими технологиями:** Python 3 легко интегрируется с другими технологиями и сервисами, такими как базы данных, веб-сервисы и облачные платформы, что делает его универсальным инструментом для разработки комплексных приложений.

1.6. Библиотека PIL (Python Imaging Library)

PIL (Python Imaging Library)[13] — это библиотека для работы с изображениями в Python. Она предоставляет широкий набор функций для открытия, манипуляции и сохранения различных форматов изображений. PIL поддерживает множество форматов файлов, включая JPEG, PNG, BMP, GIF и другие.

Основные возможности PIL включают:

- **Открытие и сохранение изображений:** PIL позволяет открывать изображения из файлов и сохранять их в различных форматах.
- **Манипуляция изображениями:** Библиотека предоставляет функции для изменения размера изображений, обрезки, поворота, зеркального отображения и других преобразований.
- **Рисование на изображениях:** PIL позволяет рисовать линии, прямоугольники, эллипсы и другие фигуры на изображениях, а также добавлять текст.

- **Фильтры и эффекты:** Библиотека включает различные фильтры и эффекты, такие как размытие, резкость, контраст и другие.
- **Анализ изображений:** PIL предоставляет инструменты для анализа изображений, такие как получение информации о пикселях, гистограммы и другие статистические данные.

1.7. Модели BlipProcessor и BlipForConditionalGeneration

Библиотека transformers от Hugging Face предоставляет модели и процессоры для обработки естественного языка и изображений. Две важные модели, используемые в боте, это BlipProcessor и BlipForConditionalGeneration от Salesforce.

- **BlipProcessor:** Этот процессор используется для предварительной обработки изображений перед их анализом моделью. Он преобразует изображения в формат, который может быть использован моделью для генерации описаний.
- **BlipForConditionalGeneration:** Эта модель используется для генерации текстовых описаний изображений. Она принимает на вход предварительно обработанные изображения и генерирует текстовые описания, которые могут быть использованы для различных задач, таких как создание подписей к изображениям или анализ содержания изображений.

1.8. Обоснование выбора среды разработки программного приложения

Google Colab[14] — это бесплатная облачная среда для выполнения кода, которая предоставляет доступ к графическим процессорам (GPU) и другим ресурсам для выполнения сложных вычислений и машинного обучения. Google Colab позволяет разработчикам и исследователям создавать, тестировать и развертывать модели машинного обучения без необходимости установки сложного программного обеспечения на локальные машины.

Основные особенности Google Colab:

- **Интерфейс:** Google Colab предоставляет современный интерфейс для работы с кодом, который включает множество полезных функций, таких как быстрый запуск, управление файлами, интеграция с Google Drive и др. Интерфейс может быть настроен, включая изменение цветовой схемы, расположения окон, сниппетов и др.
- **Редактор кода:** Редактор кода Google Colab содержит функции автодополнения, предложения кода, рефакторинг, отображение ошибок и др. Редактор кода также может использоваться для отладки, создания точек останова и проверки переменных.
- **Управление проектами:** Google Colab позволяет создавать и управлять проектами, включая Jupyter notebook, скрипты Python и другие типы файлов. Дополнительные инструменты, такие как интеграция с Google Drive и Git, облегчают поддержку проекта и управление зависимостями.
- **Debugging:** Google Colab имеет широкий набор инструментов для отладки приложений, включая поддержку всех типов точек восстановления, интеграцию с инструментами отладки и возможность удаленной отладки.
- **Развертывание:** Google Colab позволяет легко развернуть приложение, предоставляя стандартные возможности установки и обновления, а также интеграцию с различными облачными платформами, такими как Google Cloud Platform (GCP) и другие.
- **Различные инструменты разработки:** Google Colab предоставляет различные инструменты и средства разработки, такие как редакторы для сложных типов файлов, инструменты для тестирования и отладки, редакторы кода для JavaScript и HTML, интеграция с Google Cloud Platform и т. д.

2. РАЗРАБОТКА ПРОГРАММЫ

2.1. Требования к приложению

Необходимо определить функционал и ключевые особенности разрабатываемого телеграм-бота. Результатом работы бота является предоставление пользователям информации о погоде, сравнение влажности в различных городах и поиск похожих изображений по отправленной картинке. К разрабатываемому боту предъявляются следующие функциональные требования:

- **Получение и вывод информации о погоде:** Бот должен уметь получать и выводить информацию о текущей погоде в различных городах, включая состояние погоды, температуру, максимальную и минимальную температуру, влажность, давление и скорость ветра.
- **Сравнение влажности:** Бот должен уметь сравнивать влажность в нескольких городах и выводить город с наименьшей влажностью.
- **Поиск похожих изображений:** Бот должен уметь искать похожие изображения по отправленной пользователем картинке, используя предобученные модели и процессоры из библиотеки transformers от Hugging Face.
- **Интеграция с API:** Бот должен интегрироваться с различными API для получения данных о погоде (openweathermap api) и поиска изображений (google api и google cx).
- **Обработка команд и сообщений:** Бот должен уметь обрабатывать команды и сообщения от пользователей, предоставляя соответствующие ответы и информацию.
- **Интеграция с Google Colab:** Бот должен поддерживать возможность запуска и тестирования в облачной среде Google Colab, что позволяет разработчикам и исследователям использовать мощные

вычислительные ресурсы без необходимости установки сложного программного обеспечения на локальные машины.

Приложение должно соответствовать следующим техническим требованиям:

- Приложение консольного типа: Бот должен быть консольным приложением, которое может запускаться и управляться через командную строку или терминал.
- Портативность: Бот должен быть портативным, что позволяет запускать его на любом компьютере без необходимости установки дополнительного программного обеспечения.
- Возможность запуска на любом устройстве с Telegram: Бот должен поддерживать работу на любом устройстве, где установлен Telegram, обеспечивая совместимость и удобство использования.

2.2. Блок-схема программы

Блок-схема работы программы с функциями представлена ниже.

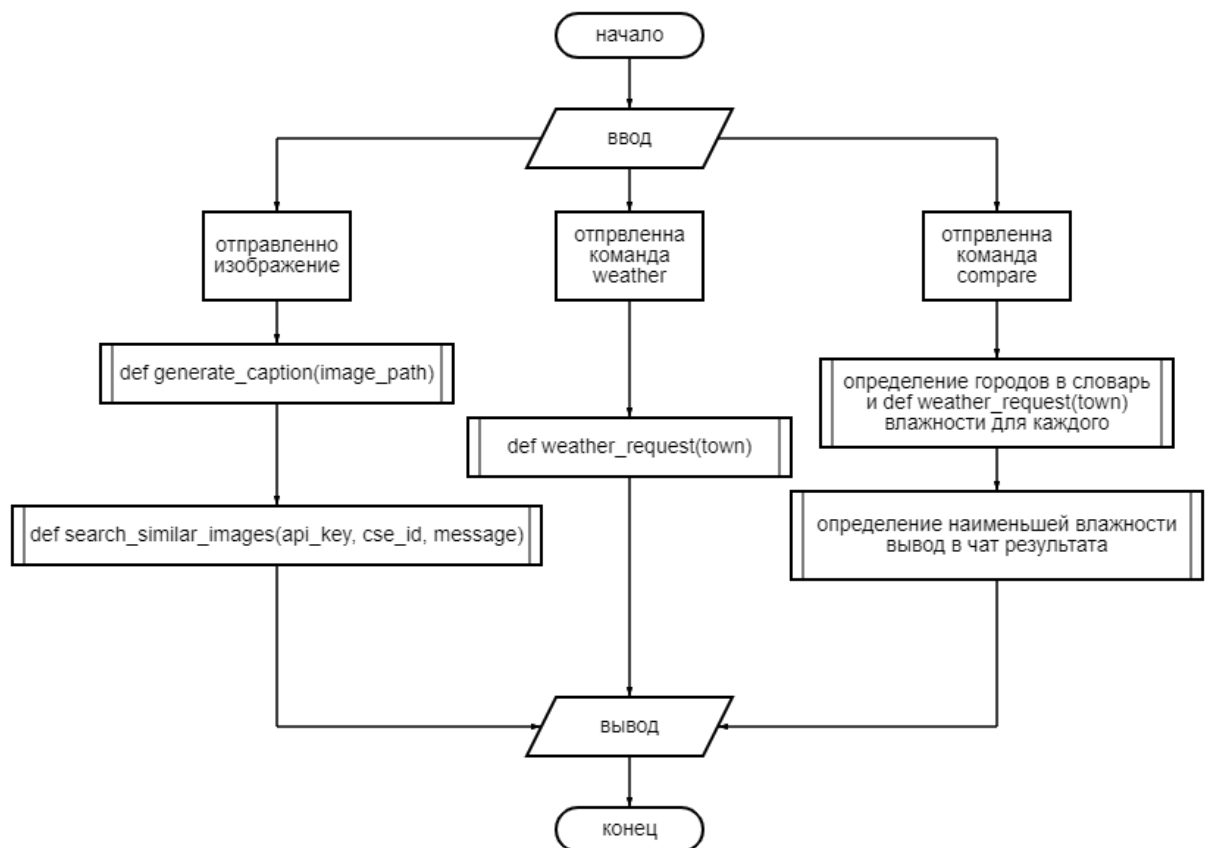


Рис. 2.1 Блок-схема логики телеграм бота.

2.3. Описание процесса разработки

Выбор стоял между языками C++ и Python. С текущими задачами был выбран Python за счет его простоты и читаемости кода, а также наличия богатой стандартной библиотеки, которая включает модули для работы с сетью, файловой системой, веб-сервисами и многим другим. Это упрощает разработку и уменьшает необходимость в сторонних библиотеках.

В Python также имеется широкая поддержка библиотек и фреймворков, таких как `pytelegrambotapi`, `requests`, `transformers` и другие, которые значительно упрощают работу с API и обработку данных. Кроме того, Python является кроссплатформенным языком, что позволяет запускать код на различных операционных системах, таких как Windows, macOS и Linux, без необходимости внесения значительных изменений.

В C++ за счет отсутствия аналогичных библиотек и фреймворков необходимо было бы прописывать многие функции вручную, что усложнило бы процесс разработки и увеличило бы время на написание и отладку кода. Это и повлияло на выбор в пользу языка Python.

Сначала устанавливаются пакеты `pytelegrambotapi`, `transformers` и `torch`. Далее в проект подключаются библиотеки `requests`, `json`, `random`, `telebot`, `pillow` и `transformers`, также добавляем `build` для создания клиента для взаимодействия с конкретным Google API.

Листинг 2.1

Установка пакетов и подключение библиотек

```
!pip3 install pytelegrambotapi
!pip3 install transformers
!pip3 install torch
!pip3 install google-api-python-client
!pip3 install pillow
import requests
import json
```

```

import random
import telebot
from PIL import Image
from transformers import BlipProcessor, BlipForConditionalGeneration
from googleapiclient.discovery import build

```

Затем прописываются токены телеграм бота, google api и google cx.

Далее указывается местоположение временного фото и открытие списка с поддерживаемыми городами. Ниже указываем какие процессор и модель используем и передаем токен боту.

Листинг 2.2

Первоначальные настройки

```

TOKEN = "yourTelegramBotToken"
GOOGLE_API_KEY = yourGoogleApi'
GOOGLE_CX = yourGoogleCx
image_path = "/content/photo.jpg"
with open('city.list.json', 'r') as file:
    cityLists = json.load(file)
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base")
bot = telebot.TeleBot(TOKEN)

```

После этого идут функции выполняющие основные работы бота.

Две функции `weather_request` и `weather_request_id`, выполняющие обращение к API и передающие полученную информацию в обработку.

Листинг 2.3

Функции `weather_request` и `weather_request_id`

```

def weather_request(town):

```



```

    return 'https://api.openweathermap.org/data/2.5/weather?q=' + town +
    '&appid=owTokenHere&units=metric&lang=Ru'
def weather_request_id(cityId):
    return 'https://api.openweathermap.org/data/2.5/weather?id=' + str(cityId)
    + '&appid=owTokenHere&units=metric&lang=Ru'

```

Далее идет функция определения объектов на фото и создания текстового описания используется библиотеку PIL для открытия изображения, processor преобразует изображение в формат, который может быть использован моделью, generate принимает преобразованные входные данные и генерирует последовательность токенов (идентификаторов), представляющих текст, decode для преобразования сгенерированных идентификаторов токенов (generated_ids) в человекочитаемый текст.

Листинг 2.4

Функция generate_caption

```

def generate_caption(image_path):
    image = Image.open(image_path)
    inputs = processor(images=image, return_tensors="pt")
    generated_ids = model.generate(**inputs)
    caption = processor.decode(generated_ids[0], skip_special_tokens=True)
    return caption

```

Следующей идет функция поиска похожих изображений по текстовому описанию, создается сервис для взаимодействия с Google Custom Search JSON API, задаются параметры searchType указывает, что мы ищем изображения, num указывает количество результатов, которые мы хотим получить (в данном случае 10), res.get('items', []): Этот метод извлекает список результатов из ответа. Если результатов нет, возвращается пустой список, [item['link']] for item in res.get('items', []): Этот генератор списка извлекает URL изображений из

каждого элемента в списке результатов. В конце возвращается список URL изображений из функции.

Листинг 2.5

Функция search_similar_images

```
def search_similar_images(api_key, cse_id, message):
    service = build("customsearch", "v1", developerKey=api_key)
    try:
        res = service.cse().list(
            q=generate_caption(image_path),
            cx=cse_id,
            searchType='image',
            num=10
        ).execute()
        image_urls = [item['link'] for item in res.get('items', [])]
        return image_urls
    except Exception as e:
        print(f"Произошла ошибка: {e}")
        return []
```

Последняя функция send_images, позволяет сгруппировать все изображения для отправки одним сообщением, создается список media_group, в который с помощью цикла записываются URL на полученные изображения.

Листинг 2.6

Функция send_images

```
def send_images(chat_id, image_urls):
    media_group = []
    for url in image_urls:
        media_group.append(telebot.types.InputMediaPhoto(url))
    bot.send_media_group(chat_id, media_group)
```

2.4. Проверка работоспособности (тестирование) программы

Методика тестирования:

Для тестирования были выбраны 2 устройства с разными комплектующими и операционными системами:

- Ноутбук: Операционная система – Windows 10 x64 22H2
- Смартфон: Операционная система – Android 11 RKQ1.200826.002

Результаты тестирования:

В ходе тестирования было выявлено, что программа корректно выводит информацию о погоде, сравнивает влажность в различных городах и находит похожие изображения по отправленной картинке на устройствах под управлением разных операционных систем. Результат показан на Рис. 2.2 и Рис. 2.3.

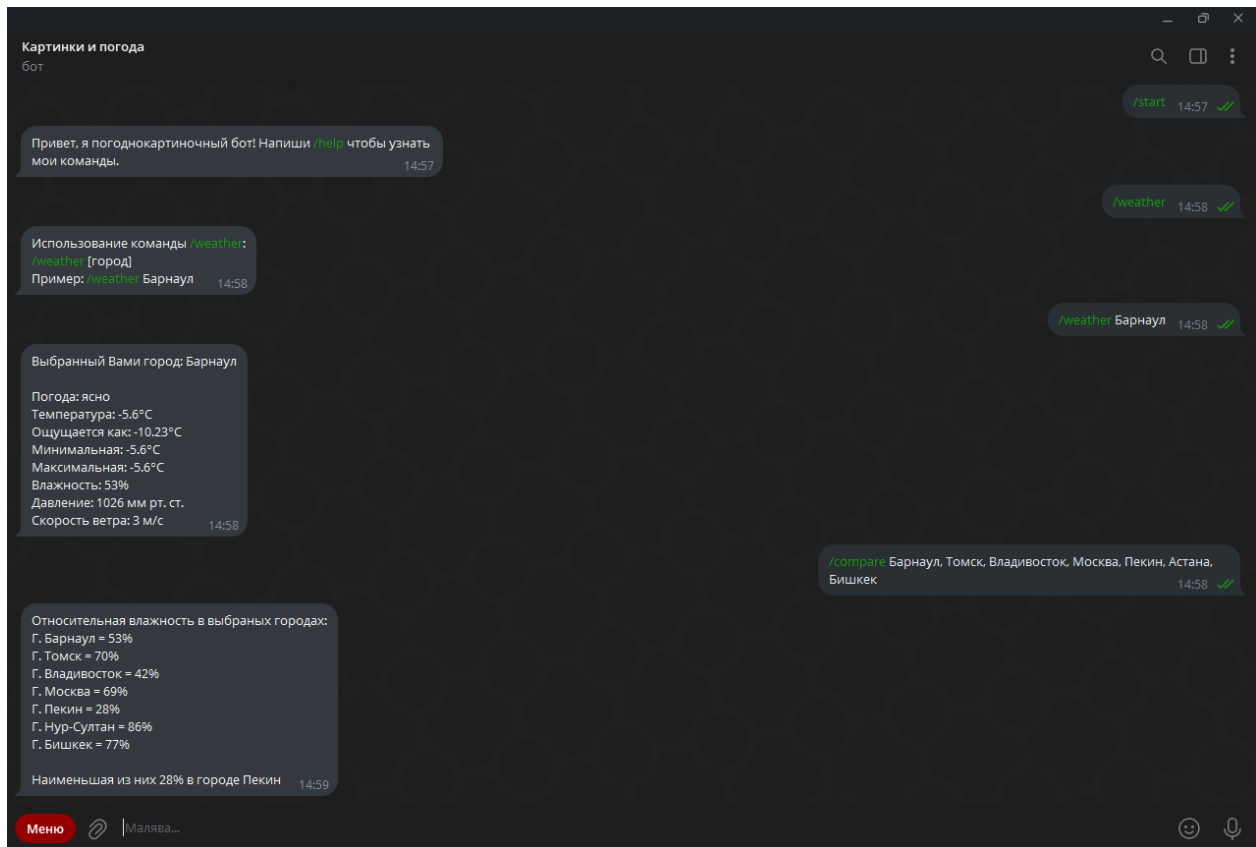


Рис. 2.2 Демонстрация погодного модуля на ноутбуке.



Рис. 2.3 Демонстрация поиска похожих фотографий на смартфоне.

3. ЗАКЛЮЧЕНИЕ

В ходе проделанной работы был изучен процесс разработки телеграм ботов для вывода информации о погоде, сравнения влажности в различных городах и поиска похожих изображений по отправленной картинке. Были изучены основные виды API и их интерфейсы. Также рассмотрены особенности работы с библиотеками `pytelegrambotapi`, `requests`, `transformers` и другими на языке Python.

Во второй части работы показана разработка телеграм-бота для вывода информации о погоде, сравнения влажности и поиска похожих изображений. Также был разобран листинг программного кода, построена блок-схема работы программного кода. Затем проведено тестирование, которое показало работоспособность бота на разных устройствах с разными операционными системами.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Alan Watts. Weather Handbook. Справочник по погоде. — London.: Adlard Coles Nautical, 2017. — 126 с.
2. Telegram Bot API [Электронный ресурс] Официальный сайт Telegram — Режим доступа: <https://core.telegram.org/api> (дата обращения 20.11.2024).
3. PyTelegramBotAPI Documentation [Электронный ресурс] Официальный сайт pyTelegramBotAPI — Режим доступа: <https://pytba.readthedocs.io/en/latest/> (дата обращения 20.11.2024).
4. OpenWeatherMap API [Электронный ресурс] Официальный сайт Open Weather — Режим доступа: <https://openweathermap.org/api> (дата обращения 20.11.2024).
5. Google API Client Library for Python Docs [Электронный ресурс] Официальный репозиторий Google API Client на GitHub — Режим доступа: <https://github.com/googleapis/google-api-python-client/blob/main/docs/README.md> (дата обращения 20.11.2024).
6. Custom Search JSON API [Электронный ресурс] Официальный сайт Google — Режим доступа: <https://developers.google.com/custom-search/v1/overview> (дата обращения 20.11.2024).
7. Transformers by Hugging Face [Электронный ресурс] Официальный сайт Hugging Face — Режим доступа: <https://huggingface.co/docs/transformers/index> (дата обращения 20.11.2024).
8. Requests: HTTP for Humans [Электронный ресурс] Официальный сайт Python requests — Режим доступа: <https://docs.python-requests.org/en/latest/> (дата обращения 20.11.2024).
9. Python 3.13.1 documentation [Электронный ресурс] Официальный сайт Python — Режим доступа: <https://docs.python.org/3/> (дата обращения 20.11.2024).

10. Введение в Python [Электронный ресурс] Официальный сайт Metanit — Режим доступа: <https://metanit.com/python/tutorial/1.1.php> (дата обращения 20.11.2024).
11. Любанович Билл. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.
12. Доусон М. Програмируем на Python. — СПб.: Питер, 2014. — 416 с.
13. Pillow (PIL Fork) Documentation [Электронный ресурс] Официальный сайт pillow — Режим доступа: <https://pillow.readthedocs.io/en/stable/> (дата обращения 20.11.2024).
14. Overview of Colaboratory Features [Электронный ресурс] Официальный сайт Google colab — Режим доступа: https://colab.research.google.com/notebooks/basic_features_overview.ipynb (дата обращения 20.11.2024).

ПРИЛОЖЕНИЕ

Текст программы

```
import requests
import json
import random
import telebot

from PIL import Image
from transformers import BlipProcessor, BlipForConditionalGeneration
from googleapiclient.discovery import build

TOKEN = "yourTelegramBotToken"
GOOGLE_API_KEY = yourGoogleApi'
GOOGLE_CX = yourGoogleCx

image_path = "photo.jpg"
with open('city.list.json', 'r') as file:
    cityLists = json.load(file)

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
captioning-base")

bot = telebot.TeleBot(TOKEN)
```



```

def weather_request(town):
    return 'https://api.openweathermap.org/data/2.5/weather?q=' + town +
    '&appid=owTokenHere&units=metric&lang=Ru'

def weather_request_id(cityId):
    return 'https://api.openweathermap.org/data/2.5/weather?id=' + str(cityId)
    + '&appid=owTokenHere&units=metric&lang=Ru'

def generate_caption(image_path):
    image = Image.open(image_path)
    inputs = processor(images=image, return_tensors="pt")
    generated_ids = model.generate(**inputs)
    caption = processor.decode(generated_ids[0], skip_special_tokens=True)
    return caption

def search_similar_images(api_key, cse_id, message):
    service = build("customsearch", "v1", developerKey=api_key)
    try:
        res = service.cse().list(
            q=generate_caption(image_path),
            cx=cse_id,
            searchType='image',
            num=10
        ).execute()
        image_urls = [item['link'] for item in res.get('items', [])]
        return image_urls
    except Exception as e:
        print(f"Произошла ошибка: {e}")
        return []

```

```

def send_images(chat_id, image_urls):
    media_group = []
    for url in image_urls:
        media_group.append(telebot.types.InputMediaPhoto(url))

    bot.send_media_group(chat_id, media_group)

@bot.message_handler(commands=['start'])
def send_start(message):
    bot.send_message(message.chat.id, "Привет, я погоднокартиночный бот!
    Напиши /help чтобы узнать мои команды.")

@bot.message_handler(commands=['help'])
def send_help(message):
    bot.send_message(message.chat.id, "Справка:\n /barnaul - показывает
    текущую погоду в Барнауле.\n /weather <Название города> - показывает
    погоду в выбранном городе\n /compare <Названия городов> - сравнивает
    влажность в разных городах\n /random - показывает погоду в случайном
    городе\n\nА если отправить мне картинку я постараюсь скинуть тебе 10
    похожих.")

@bot.message_handler(commands=['barnaul'])
def send_weather_Barnaul(message):
    response = requests.get(weather_request('Barnaul'))
    if response.status_code == 200:
        data = response.json()

        bot.send_message(message.chat.id, f"Выбранный Вами город:
        {data.get('name')}\n\nПогода: {data.get('weather',
        [{ }])[0].get('description')}\nТемпература: {data.get('main',

```

```

    {}).get('temp'))°C\nОщущается          как:          {data.get('main',
    {}).get('feels_like'))°C\nМинимальная:          {data.get('main',
    {}).get('temp_min'))°C\nМаксимальная:          {data.get('main',
    {}).get('temp_max'))°C\nВлажность:          {data.get('main',
    {}).get('humidity'))%\nДавление: {data.get('main', {}).get('pressure')} мм рт.
ст.\nСкорость ветра: {data.get('wind', {}).get('speed')} м/с\n")

```

```

else:

```

```

    bot.send_message(message.chat.id, f"Ошибка при выполнении запроса:
    {response.status_code}")

```

```

@bot.message_handler(commands=['random'])

```

```

def send_weather_random(message):

```

```

    CityIds = [item['id'] for item in cityLists]

```

```

    response = requests.get(weather_request_id(random.choice(CityIds)))

```

```

    if response.status_code == 200:

```

```

        data = response.json()

```

```

        bot.send_message(message.chat.id, f"Выбранный Вами город:
        {data.get('name')}\n\nПогода:          {data.get('weather',
        [{}])[0].get('description')}\nТемпература:          {data.get('main',
        {}).get('temp'))°C\nОщущается          как:          {data.get('main',
        {}).get('feels_like'))°C\nМинимальная:          {data.get('main',
        {}).get('temp_min'))°C\nМаксимальная:          {data.get('main',
        {}).get('temp_max'))°C\nВлажность:          {data.get('main',
        {}).get('humidity'))%\nДавление: {data.get('main', {}).get('pressure')} мм рт.
ст.\nСкорость ветра: {data.get('wind', {}).get('speed')} м/с\n")

```

```

    else:

```

```

        bot.send_message(message.chat.id, f"Ошибка при выполнении запроса:
        {response.status_code}")

```

```

@bot.message_handler(commands=['weather'])
def send_weather_Town(message):
    args = message.text.split()[1:]
    if not args:
        help_text = (
            "Использование команды /weather:\n/weather [город]\nПример: /weather\nБарнаул")
        bot.send_message(message.chat.id, help_text)
    else:
        city = ' '.join(args)
        response = requests.get(weather_request(city))
        if response.status_code == 200:
            data = response.json()

            bot.send_message(message.chat.id, f"Выбранный Вами город:
{data.get('name')}\n\nПогода: {data.get('weather',
[{}])[0].get('description')}\nТемпература: {data.get('main',
{ }).get('temp')}°C\nОщущается как: {data.get('main',
{ }).get('feels_like')}°C\nМинимальная: {data.get('main',
{ }).get('temp_min')}°C\nМаксимальная: {data.get('main',
{ }).get('temp_max')}°C\nВлажность: {data.get('main',
{ }).get('humidity')}%\nДавление: {data.get('main', { }).get('pressure')} мм рт.
ст.\nСкорость ветра: {data.get('wind', { }).get('speed')} м/с\n")
        else:
            bot.send_message(message.chat.id, f"Ошибка при выполнении запроса,
возможно было введено название с ошибкой или сразу несколько городов.
Статус: {response.status_code}")

@bot.message_handler(commands=['about'])
def handle_about(message):

```

```
bot.send_photo(message.chat.id, "https://sun9-80.userapi.com/impf/c851136/v851136988/1061aa/boTLclsDmHI.jpg?size=2560x1707&quality=96&sign=256c2921bbd5bc9641d1deb3caccb71e&type=album",
"Бота создал Плахотнюк Роман Максимович студент второго курса АГУ, из группы 5.205.2\nОбратная связь через вконтакте vk.com/id453485745")
```

```
@bot.message_handler(commands=['compare'])
def send_weather_compare(message):
    text = message.text.replace('/compare', '').strip()
    if not text:
        bot.send_message(message.chat.id, "Введите города через запятую после команды /compare. Например: /compare Барнаул, Томская область, Новосибирск")
    else:
        cities = [city.strip() for city in text.split(',')]
        if len(cities) == 1:
            bot.send_message(message.chat.id, "Введите еще один город через запятую. Например: /compare Барнаул, Томская область")
        else:
            humidity_dict = {}
            humidityList = "Относительная влажность в выбранных городах:\n"
            for town in cities:
                response = requests.get(weather_request(town))
                if response.status_code == 200:
                    data = response.json()
                    humidity_dict[data.get('name')] = data.get('main', {}).get('humidity')
                    humidityList += f"Г. {data.get('name')} = {data.get('main', {}).get('humidity')}%\n"
            else:
```

```
bot.send_message(message.chat.id, f"Название города: {town} -  
содержит ошибку, оно было автоматически исключено из сравнения")
```

```
bot.send_message(message.chat.id, f"{humidityList}\nНаименьшая из них  
{humidity_dict[min(humidity_dict, key=humidity_dict.get)]}% в городе  
{min(humidity_dict, key=humidity_dict.get)}")
```

```
@bot.message_handler(content_types=['photo'])
```

```
def photo_search(message):
```

```
    photo = message.photo[-1]
```

```
    file_info = bot.get_file(photo.file_id)
```

```
    downloaded_file = bot.download_file(file_info.file_path)
```

```
    with open('photo.jpg', 'wb') as new_file:
```

```
        new_file.write(downloaded_file)
```

```
    waiter = bot.reply_to(message, "Ищу похожие картинки, минуточку")
```

```
    try:
```

```
        send_images(message.chat.id, search_similar_images(GOOGLE_API_KEY,  
GOOGLE_CX, message))
```

```
        bot.delete_message(chat_id=message.chat.id, message_id=waiter.message_id)
```

```
    except:
```

```
        bot.delete_message(chat_id=message.chat.id, message_id=waiter.message_id)
```

```
        bot.send_message(message.chat.id, "Ничего не найдено")
```

```
bot.infinity_polling()
```