МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

ОТЧЕТ

по учебной практике

за период с 10 марта по 22 марта 2025 г.

Авторы отчета

\_\_\_\_\_\_\_\_\_\_\_ Закусилов Денис Сергеевич ПрИ-201

подпись инициалы, фамилия

\_\_\_\_\_\_\_\_\_\_\_ Мясников Юрий Алексеевич ПИ-201

подпись инициалы, фамилия

\_\_\_\_\_\_\_\_\_\_\_ Луценко Дмитрий Андреевич ПрИ-202

подпись инициалы, фамилия

\_\_\_\_\_\_\_\_\_\_\_ Александр Максимович Смирнов ПрИ-202

подпись инициалы, фамилия

Зав. кафедрой ИТиЭИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  / С.А. Скрипов /

подпись инициалы, фамилия

Отчет защищен \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата оценка

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись                                                   инициалы, фамилия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись                                                   инициалы, фамилия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись                                                   инициалы, фамилия

Челябинск 2025г.

Содержание

[Содержание 2](#_Toc194779527)

[Введение 3](#_Toc194779528)

[1. Анализ предметной области 4](#_Toc194779529)

[1.1 Бизнес-проблема 4](#_Toc194779530)

[1.2 Бизнес-требования 6](#_Toc194779531)

[2. Концепция решения 8](#_Toc194779532)

[2.1 Описание решения 7](#_Toc194779533)

[2.1 Словарь предметной области 9](#_Toc194779534)

[2.2 Модель прецендентов 11](#_Toc194779535)

[3. Разработка 13](#_Toc194779536)

[3.1 Стек технологий 13](#_Toc194779537)

[3.2 Архитектура проекта 14](#_Toc194779538)

[3.4 Основные модули 15](#_Toc194779540)

[3.5 Фрагменты кода 16](#_Toc194779541)

[4. Тестирование 21](#_Toc194779536)

[5. Заключение 25](#_Toc194779536)

Введение

Важной тенденцией последних лет стала автоматизация рутинных операций, позволяющая значительно повысить эффективность работы и улучшить пользовательский опыт. Особенно актуальны эти задачи в контексте информационной перегруженности и необходимости оперативного выполнения множества разнообразных операций.

В связи с этим актуальность проекта «Telegram-бот «Jarvis» обусловлена необходимостью автоматизировать выполнение повседневных задач пользователей посредством голосовых и текстовых команд, обеспечивая простой и интуитивно понятный интерфейс взаимодействия с различными цифровыми сервисами. Важным преимуществом разрабатываемого бота является возможность отправки команд в свободной форме, без необходимости придерживаться строгой структурности запроса. Данное решение призвано снизить затраты времени на рутинные процессы, повысить удобство и качество взаимодействия пользователя с сервисами Google (Gmail, Docs, Calendar, Drive, Contacts).

Целью проекта является создание многофункционального персонального помощника, работающего через платформу Telegram, способного оперативно и точно выполнять разнообразные пользовательские команды с использованием современных технологий, в том числе локальной нейросети и внешних API-сервисов.

1. Анализ предметной области

1.1 Бизнес-проблема

В современных условиях информационной перегруженности и динамичного темпа работы многие пользователи сталкиваются с рутинными задачами, выполнение которых требует значительных временных и когнитивных затрат.

|  |  |
| --- | --- |
| **Проблема** | Пользователи ежедневно сталкиваются с большим количеством рутинных операций и действий при работе с цифровыми сервисами, особенно когда не могут использовать традиционные способы ввода данных (например, за рулём или в спешке), что приводит к потере времени и снижению производительности. |
| Воздействует на | Пользователей цифровых сервисов и информационных систем, особенно тех, кто часто перемещается и испытывает недостаток времени. |
| Результатом чего является | Снижение эффективности работы, высокая вероятность ошибок из-за неудобства и ограниченности традиционных методов ввода информации. |
| Выигрыш от | Предлагаемый Telegram-бот автоматизирует рутинные задачи с помощью голосовых и неформальных текстовых команд, позволяя выполнять задачи оперативно даже при отсутствии возможности ввода текста вручную. |
| Успешное решение должно | Обеспечить удобный и быстрый способ выполнения задач, минимизировать необходимость ручного ввода и взаимодействия с интерфейсами, а также повысить удобство и эффективность повседневной деятельности пользователей |

1.2 Бизнес-требования

Бизнес-требования к проекту можно представить следующим образом:

* Повышение производительности и эффективности работы пользователей путем автоматизации рутинных операций. Голосовое управление с минимальной необходимостью взаимодействия с экраном (важно). Возможность использования неструктурированных команд (важно)
* Пользовательский интерфейс должен быть максимально простым, понятным и доступным для широкого круга пользователей
* Интеграция с цифровыми сервисами. Бот должен безупречно взаимодействовать с ключевыми сервисами Google (Gmail, Docs, Calendar, Drive, Contacts), обеспечивая выполнение задач напрямую из мессенджера

2. Концепция решения

2.1 Описание решения

Проектная команда выбрала подход, ориентированный на удовлетворение повседневных потребностей пользователей, испытывающих нехватку времени и неудобства при использовании традиционных методов взаимодействия с цифровыми сервисами.

Функциональность решения включает:

* Telegram-бот, который принимает голосовые и текстовые команды в свободной форме, используя Whisper для распознавания речи.
* Для распознавания команд, вводимых пользователем в свободной форме, используется локально развернутая нейросеть, которая посредством заранее составленных примеров (few-shot prompting) преобразует запросы в строго структурированный JSON. Это позволяет снизить зависимость от жестких шаблонов и обеспечить гибкость взаимодействия
* Проект реализован в виде набора специализированных модулей (агентов), каждый из которых отвечает за работу с отдельными сервисами (Gmail, Calendar и др.). Центральным элементом является модуль-оркестратор, который принимает входящие команды и маршрутизирует их к соответствующим агентам

Анализ рисков проекта:

* Несмотря на использование локальной нейросети для обработки естественного языка, возможны случаи неверной интерпретации команд, что может привести к ошибкам при выполнении задач.
* Интеграция с сервисами Google и сторонними источниками влечет за собой риск сбоев или изменений в API, что может повлиять на работоспособность отдельных функций. Необходимо регулярно обновлять модули и следить за изменениями в документации внешних сервисов

Удобство использования готового продукта:

* Интеграция с Telegram обеспечивает привычную и доступную платформу для большинства пользователей. Возможность ввода команд в свободной форме снижает необходимость обучения и повышает удовлетворенность сервисом.

Перспективы развития:

* Модульная архитектура позволяет без труда внедрять новые функции и интегрировать дополнительные сервисы. В будущем возможно расширение ассортимента поддерживаемых API и реализация новых сценариев автоматизации.

2.2 Словарь предметной области

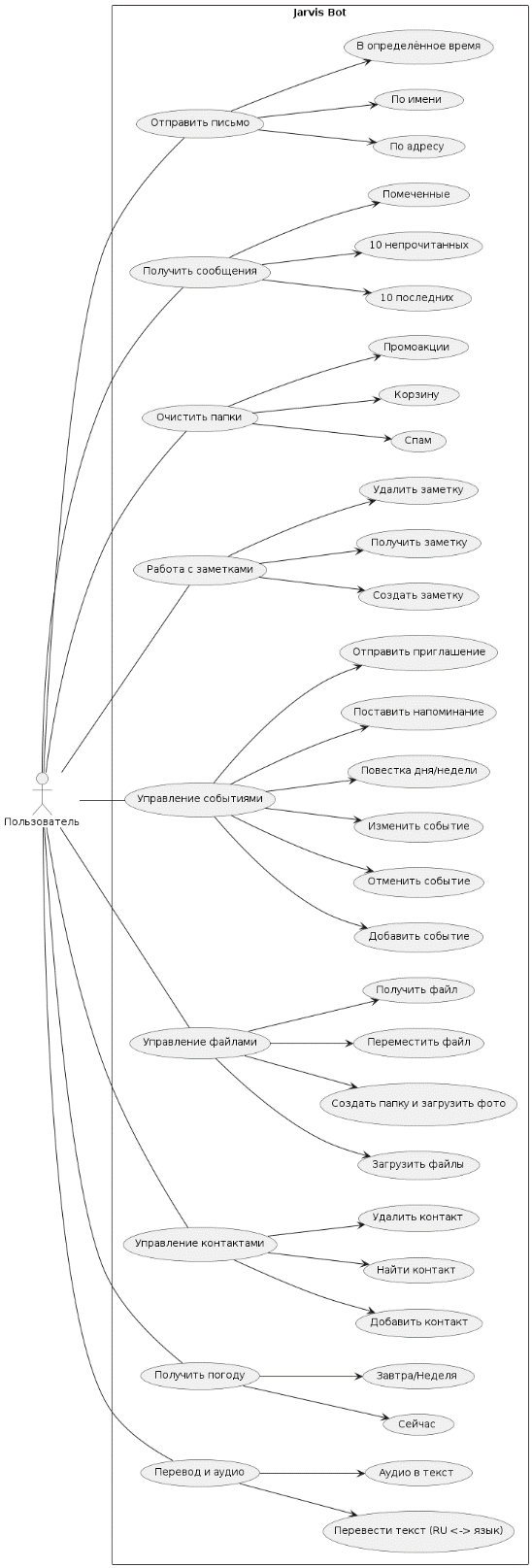
|  |  |  |
| --- | --- | --- |
| № | Термин или понятие | Описание |
| 1. | Telegram-бот | Программный робот, функционирующий на платформе Telegram, способный взаимодействовать с пользователями посредством команд. |
| 2. | API | Application Programming Interface — интерфейс, позволяющий программам взаимодействовать между собой |
| 3. | OAuth 2.0 | Протокол авторизации, используемый для безопасного предоставления доступа к ресурсам пользователя (например, Google-аккаунту). |
| 4. | Google API | Набор инструментов и интерфейсов программирования для интеграции с сервисами Google. |
| 5. | LLM | Large Language Model — крупная языковая модель, обученная на больших объёмах текстовых данных для понимания и генерации текста. |
| 6. | Whisper | Библиотека от OpenAI для распознавания речи, используемая в проекте для преобразования голосовых сообщений в текст. |
| 7. | JSON | JavaScript Object Notation — формат хранения и обмена данными, используемый для структурированной передачи команд и параметров |
| 8. | Оркестратор | Центральный модуль, который получает распознанные команды от нейросети и распределяет их исполнение между соответствующими агентами. |
| 9. | Агент | Логический модуль, отвечающий за выполнение конкретных функций (например, работа с Gmail, Calendar, Contacts, Drive, погода и т.д.). |
| 10. | Интент (Intent) | Намерение пользователя, извлекаемое из его запроса, определяющее, какое действие необходимо выполнить (например, «поиск контакта», «отправка письма»). |
| 11. | STT | Speech-to-Text — технология преобразования голосового сообщения в текст. В проекте реализуется при помощи Whisper. |

2.3 Модель прецендентов

Telegram-бот «Jarvis» имеет следующий функционал:

* Регистрация и авторизация пользователя:
  + Авторизация осуществляется через Google-аккаунт с использованием протокола OAuth 2.0.
  + Сохранение учетных данных пользователя в локальной базе данных для последующего взаимодействия.
* Обработка команд в свободной форме:
  + Прием текстовых и голосовых сообщений от пользователя.
  + Преобразование голосовых команд в текст с помощью модуля Whisper STT
  + Использование локальной нейросети (LLM) для интерпретации запросов
* Интеграция с сервисами Google:
  + Gmail: отправка писем, поиск и управление письмами, работа с черновиками и папками.
  + Google Calendar: создание, редактирование и удаление событий, формирование повестки дня, напоминания и отправка приглашений.
  + Google Drive: загрузка файлов и фото, создание папок, перемещение файлов, получение файлов и ссылок
  + Google Contacts: поиск, добавление и удаление контактов, вывод дополнительной информации (телефоны, email, дни рождения)
* Дополнительные возможности:
  + Получение актуальной информации о погоде и прогнозов (интеграция с сервисами Яндекс-погода)
* Вывод итогового результата:
  + Информирование пользователя о выполненных операциях через Telegram-чат.
  + Представление результатов в удобном и структурированном виде (например, подробная информация о событии, контактах, письмах и файлах).

Для удобства разработали диаграмму прецедентов (см. Приложение Рис.1).

  
Рис.2.1 Диаграмма прецедентов

3. Разработка

3.1 Стек технологий

Ниже приведён стек технологий, используемых в проекте «Telegram-бот “Jarvis”»:

* Языки программирования:
  + *Python:* Основной язык разработки, обеспечивающий широкий выбор библиотек для интеграции с внешними сервисами, обработки естественного языка и построения телеграм-ботов.
* Работа с Telegram:
  + *python-telegram-bot:* Главная библиотека для создания и управления ботом в Telegram.
* Работа с нейросетями и обработка речи:
  + *openai-whisper:* Библиотека для преобразования речи в текст.
  + *pymorphy2:* Библиотека для морфологического анализа русского языка (нормализация имён, согласование слов и т.д).
  + *Ollama*: Платформа для локального развёртывания нейросетей. Используется для запуска модели deepseek‑r1:14b, которая интерпретирует естественно-языковые запросы и генерирует структурированные ответы, обеспечивая высокую скорость и эффективность обработки запросов

3.2 Архитектура проекта

Ниже приведено текстовое описание архитектуры проекта «Telegram-бот “Jarvis”»:

* TelegramBot — Главный класс, который отвечает за взаимодействие с пользователем в мессенджере Telegram. Обрабатывает входящие сообщения (текстовые и голосовые), маршрутизирует их к соответствующим агентам или сервисам и формирует ответы пользователю
* Агенты (Agent-классы) - Каждому направлению функциональности соответствует свой агент: работа с почтой, календарём, документами, контактами, переводами, погодными сервисами и т.д. TelegramBot напрямую вызывает методы агентов в зависимости от «интента» (намерения), распознанного в сообщении пользователя

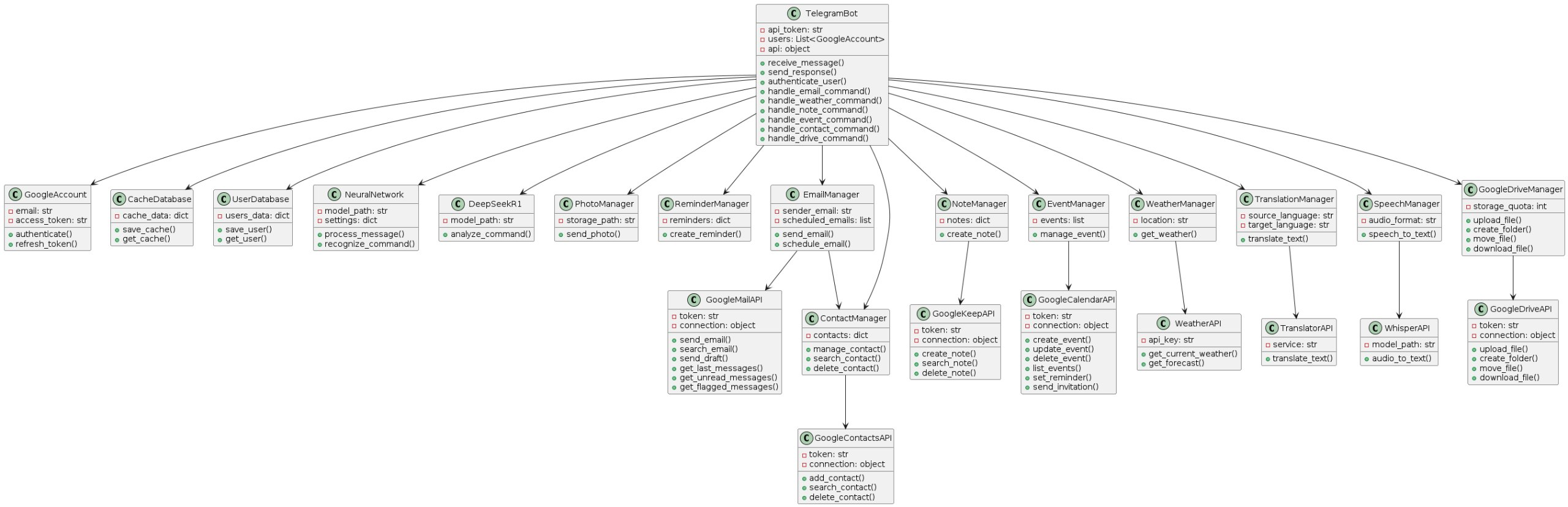


Рис. 3.1 Диаграмма классов

3.3 Основные модули

Ниже кратко перечислены основные файлы и их назначение (см. Приложение Рис.3):

* Корневые файлы:
  + telegram\_bot.py — Главный скрипт для запуска бота. Содержит логику инициализации python-telegram-bot, обработки входящих сообщений и маршрутизации к соответствующим модулям (оркестратору, нейросети и т.д.).
  + orchestrator.py — Центральный модуль, который после определения «интента» (намерения пользователя) передаёт задачу соответствующему агенту. Здесь происходит «сборка» всех компонентов (GmailAgent, CalendarAgent, ContactsAgent и др.) в единую систему.
* Пакет llm:
  + llm\_prompts.py — Содержит набор промптов (few-shot examples) для корректного распознавания и структурирования команд пользователя в формате JSON.
  + local\_llm.py — Реализует логику вызова локально развёрнутой языковой модели.
  + normalize\_russian\_text.py - Дополнительные функции для нормализации русского текста
* Пакет stt:
  + whisper\_stt.py — Модуль, использующий библиотеку openai-whisper для распознавания голосовых сообщений. Содержит логику загрузки аудиофайла, его предварительной обработки и получения итоговой текстовой расшифровки

3.5 Фрагменты кода

* local\_llm.py (Рис.1.1): В этом файле реализованы классы для работы с локальной языковой моделью (LLM) посредством отправки HTTP-запросов на локальный сервер, предоставляющий генерацию текста. DeepseekClient отправляет POST-запросы на указанный URL с переданным промптом, моделью. LocalLLM формирует итоговый промпт, объединяя заранее заданные few-shot примеры (из файла llm\_prompts.py) с текстом запроса. Полученный ответ обрабатывается: сначала пытается распарсить JSON-структуру, затем с помощью метода extract\_json извлекается корректный JSON-блок. Если парсинг успешен, возвращается словарь с определённым «интентом» и параметрами, иначе – возвращается ответ с intent "unknown"

import re  
import json  
import requests  
from config import DEEPSEEKS\_BASE\_URL, MODEL\_NAME, TEMPERATURE  
from llm.llm\_prompts import FEW\_SHOT\_EXAMPLES  
  
class DeepseekClient:  
 def \_\_init\_\_(self, base\_url: str = "http://127.0.0.1:11434/v1/completions",  
 model: str = "deepseek-r1:14b", temperature: float = 0.2):  
 self.base\_url = base\_url  
 self.model = model  
 self.temperature = temperature  
  
 def generate(self, prompt: str) -> str:  
 payload = {  
 "model": self.model,  
 "prompt": prompt,  
 "temperature": self.temperature  
 }  
 response = requests.post(self.base\_url, json=payload)  
 response.raise\_for\_status()  
 return response.text  
  
class LocalLLM:  
 def \_\_init\_\_(self):  
 self.client = DeepseekClient(  
 base\_url=DEEPSEEKS\_BASE\_URL,  
 model=MODEL\_NAME,  
 temperature=TEMPERATURE  
 )  
  
 def analyze\_request(self, user\_text: str) -> dict:  
 final\_prompt = FEW\_SHOT\_EXAMPLES + f"\nПользователь: \"{user\_text}\"\nОтвет:"  
 generated\_text = self.client.generate(final\_prompt).strip()  
  
 try:  
 full\_response = json.loads(generated\_text)  
 choice\_text = full\_response.get("choices", [{}])[0].get("text", "")  
 except json.JSONDecodeError:  
 choice\_text = generated\_text  
  
 json\_text = self.extract\_json(choice\_text)  
 try:  
 data = json.loads(json\_text)  
 return data  
 except json.JSONDecodeError as e:  
 print("Ошибка JSONDecode:", e)  
 print("Сгенерированный текст для парсинга:", choice\_text)  
 return {"intent": "unknown", "parameters": {}}  
  
 def extract\_json(self, text: str) -> str:  
 match = re.search(r"```json(.\*?)```", text, re.DOTALL)  
 if match:  
 return match.group(1).strip()  
 start = text.find('{')  
 end = text.rfind('}')  
 if start != -1 and end != -1 and end > start:  
 return text[start:end+1]  
 return text

Рис. 3.2 local\_llm.py

* whisper\_stt.py (Рис.1.2): Метод transcribe\_audio принимает путь к аудиофайлу и дополнительные параметры декодирования. Полученный аудиосигнал передается в модель для транскрипции, после чего возвращается распознанный текст.

import whisper  
import numpy as np  
  
class WhisperSTT:  
 def \_\_init\_\_(self, model\_name="large"):  
 self.model = whisper.load\_model(model\_name)  
  
 def transcribe\_audio(self, audio\_file\_path: str,  
 temperature: float = 0.0,  
 beam\_size: int = 5,  
 best\_of: int = 5,  
 apply\_preprocessing: bool = True) -> str:  
 if apply\_preprocessing:  
 # Загрузка аудио и его подготовка  
 audio = whisper.load\_audio(audio\_file\_path)  
 audio = whisper.pad\_or\_trim(audio)  
 # Пример нормализации громкости (опционально, можно расширить обработку)  
 if np.max(np.abs(audio)) > 0:  
 audio = audio / np.max(np.abs(audio))  
 # Передаем аудио в модель  
 result = self.model.transcribe(  
 audio,  
 language="ru",  
 temperature=temperature,  
 beam\_size=beam\_size,  
 best\_of=best\_of  
 )  
 else:  
 # Если предварительная обработка не требуется, передаем путь к файлу напрямую  
 result = self.model.transcribe(  
 audio\_file\_path,  
 language="ru",  
 temperature=temperature,  
 beam\_size=beam\_size,  
 best\_of=best\_of  
 )  
 return result["text"]

Рис. 3.3 whisper\_stt.py

* llm\_prompts.py: Информировать LLM о формате и правилах генерации корректного JSON-ответа. Примеры охватывают различные сценарии (поиск контактов, отправка писем, управление событиями и т.д.), что позволяет модели более точно интерпретировать запросы пользователя

FEW\_SHOT\_EXAMPLES = r"""  
Ты — ассистент для обработки голосовых запросов. Твоя задача — анализировать входящий запрос на русском языке и возвращать строго структурированный ответ в формате JSON, без каких-либо дополнительных комментариев или пояснений.  
  
Правила:  
1. Всегда возвращай корректный JSON.  
2. Поддерживаемые значения поля "intent":  
 - "search\_contact" — для поиска контактов.  
 - "send\_email" — для отправки писем.  
 - "search\_document" — для поиска документов.  
 - "show\_messages" — для показа последних сообщений от заданного отправителя.  
 - "week\_forecast" — для получения прогноза погоды на следующую неделю в указанном городе.  
 - "unknown" — если запрос не распознан.  
3. Для intent "search\_contact" объект "parameters" должен содержать:  
 - "contact\_name": строка (например, "Антон").  
 - "company": строка или null.  
 - "requested\_field": строка или массив строк ("phone", "email", "birthday").  
4. Для intent "send\_email" объект "parameters" должен содержать:  
 - Либо "to\_address": строка, либо "contact\_name" и опционально "company".  
 - "message\_content": строка.  
 - Опционально "scheduled\_day": число.  
16. Для intent "list\_events\_period" объект "parameters" может быть пустым или содержать параметры периода, если необходимо.  
17. Для intent "create\_meeting" объект "parameters" должен содержать:  
 - "contact\_name": строка.  
 - "datetime": строка (например, "завтра в 15:00").  
18. Для intent "cancel\_meeting" объект "parameters" должен содержать:  
 - "contact\_name": строка.  
19. Для intent "current\_weather" объект "parameters" должен содержать:  
 - "city": строка (например, "Москва").  
22. Если запрос не соответствует ни одному известному шаблону, возвращай:  
 {  
 "intent": "unknown",  
 "parameters": {}  
 }  
23. Если имя задано в склонённой форме, нормализуй его до именительного падеже.  
  
Примеры:  
  
[Пример 1]  
Пользователь: "Дай мне номер Антохи"  
Ответ:  
{  
 "intent": "search\_contact",  
 "parameters": {  
 "contact\_name": "Антон",  
 "company": null,  
 "requested\_field": "phone"  
 }  
}  
  
[Пример 2]  
Пользователь: "Дай мне почту и номер телефона Антохи"  
Ответ:  
{  
 "intent": "search\_contact",  
 "parameters": {  
 "contact\_name": "Антон",  
 "company": null,  
 "requested\_field": ["email", "phone"]  
 }  
}  
  
[Пример 3]  
Пользователь: "Когда день рождения у Витали?"  
Ответ:  
{  
 "intent": "search\_contact",  
 "parameters": {  
 "contact\_name": "Виталий",  
 "company": null,  
 "requested\_field": "birthday"  
 }  
}  
  
[Пример 4]  
Пользователь: "Добавь контакт Глеб из ЧелГУ +79126973674"  
Ответ:  
{  
 "intent": "add\_contact",  
 "parameters": {  
 "contact\_name": "Глеб",  
 "company": "ЧелГУ",  
 "phone": "+79126973674"  
 }  
}  
  
[Пример 5]  
Пользователь: "Отправь на адрес anton@csu.ru письмо, скажи, что все будет сделано завтра вечером"  
Ответ:  
{  
 "intent": "send\_email",  
 "parameters": {  
 "to\_address": "anton@csu.ru",  
 "message\_content": "все будет сделано завтра вечером"  
 }  
}  
  
Пожалуйста, выдай ответ строго в формате JSON без каких-либо пояснений.  
"""

Рис. 3.4 llm\_prompts.py

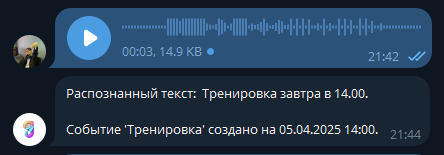
* normalize\_russian\_text.py (Рис. 1.4): Из входящего текста с помощью регулярного выражения извлекаются все слова. С помощью библиотеки pymorphy2 каждое слово приводится к его нормальной форме (лемматизация). Результатом работы функции является строка, содержащая нормализованные (лемматизированные) слова, что упрощает дальнейший анализ и сопоставление запросов.

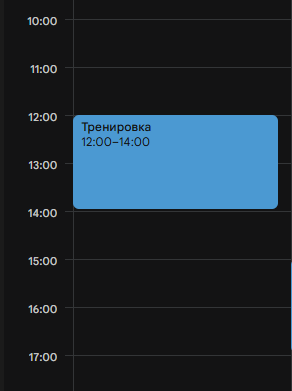
import inspect  
if not hasattr(inspect, 'getargspec'):  
 def getargspec(func):  
 fas = inspect.getfullargspec(func)  
 return (fas.args, fas.varargs, fas.varkw, fas.defaults)  
 inspect.getargspec = getargspec  
  
import re  
import pymorphy2  
  
morph = pymorphy2.MorphAnalyzer()  
  
def normalize\_russian\_text(text: str) -> str:  
 words = re.findall(r"[А-Яа-яA-Za-zёЁ]+", text)  
 normalized = []  
 for w in words:  
 parsed = morph.parse(w)[0]  
 normalized.append(parsed.normal\_form)  
 return " ".join(normalized)

Рис. 3.5 normalize\_russian\_text.py

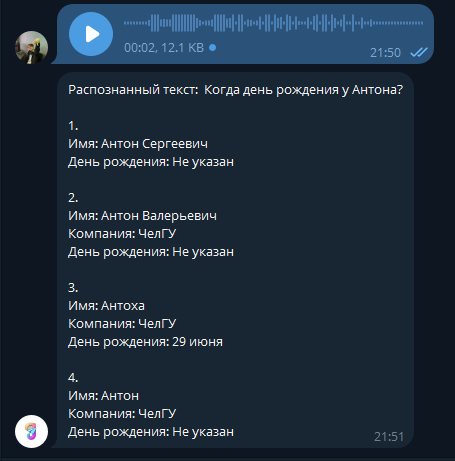
4. Тестирование

1. Проверка команды по добавлению события в календарь:

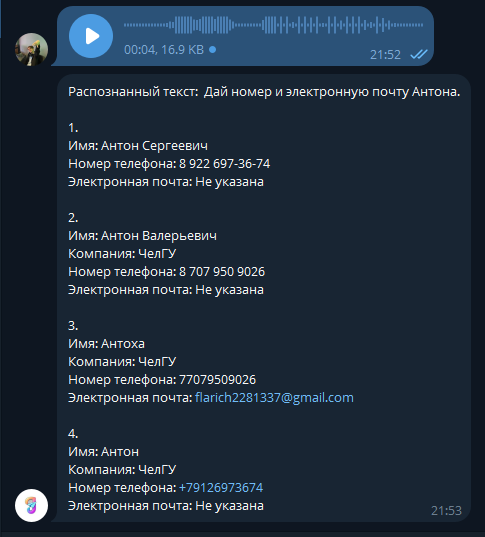




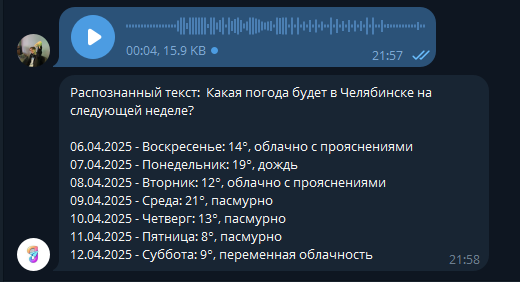
2. Проверка команды выводу дня Рождения определенного контакта:



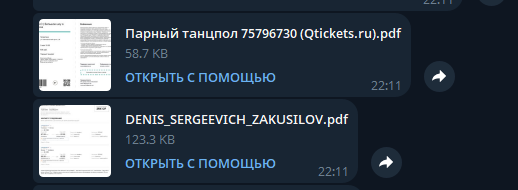
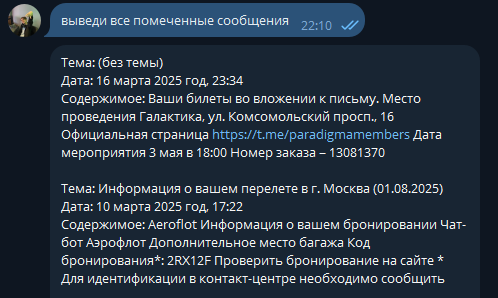
3. Проверка команды по выводу и телефона и электронной почты:



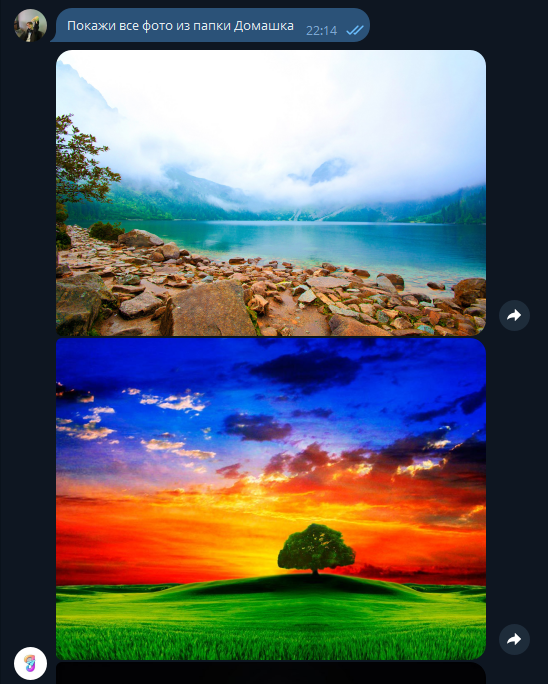
4. Проверка команды по выводу погоды:



5. Проверка команды введенная текстом по выводу писем на почте:



6. Проверка команды введенная текстом по выводу фото из определенной папки на Google Drive:



5. Заключение

В ходе разработки проекта «Telegram-бот “Jarvis”» была реализована комплексная система, способная значительно упростить выполнение рутинных задач за счет автоматизации и интеграции с основными сервисами Google. Модульная архитектура, основанная на четком разделении ответственности между компонентами — от обработки входящих сообщений до взаимодействия с внешними API — обеспечивает гибкость, масштабируемость и удобство поддержки системы.

Использование современных технологий, таких как нейросети для анализа естественного языка, библиотека openai-whisper для преобразования речи в текст и платформа Ollama для локального развёртывания моделей, позволяет ботy интерпретировать команды пользователей даже в свободной форме.