



Geekbrains

**Создание прайс-Бота,
для оптимизации работы отдела по работе с клиентами
компании ART_Milling**

Программа: Программист
Специализация: Python-разработчик. Специалист
Васильева Екатерина Викторовна, гр.5046

Казань

2024



Содержание

Введение	4
1 Теоретико-аналитическая часть	
1.1 Постановка задачи	5
1.2 Выбор метода проектирования и разработки	10
1.3 Описание технического задания	14
2 Практическая часть	
2.1 Описание процесса разработки	14
2.2 Результат разработки	45
3 Заключение	47
4 Список использованных источников	50
Приложения	52



Тема проекта: Создание прайс бота, для оптимизации работы отдела по работе с клиентами компании ART_Milling.

Цель: Изучить особенности улучшения и ускорения работы менеджеров с помощью прайс бота

Какую проблему решает: освобождает менеджеров от ответов на вопросы клиентов о цене на продукции в зависимости от категории.

Задачи:

1. Изучить источники информации, а также литературу, касающуюся темы исследования.
2. Произвести анализ технологий реализации и обосновать выбор необходимых программных платформ.
3. В соответствии с техническим заданием провести разработку прайс бота.
4. Подготовить техническую и сопроводительную документацию
5. виды и методы ручного тестирования веб-приложений.
6. Составить план работы прайс бота с использованием баз данных SQL, выполняемых с течением времени во время тестирования в компании Art_milling.
7. Рассмотреть и выполнить основные виды и метода ручного тестирование бота
8. Разработать предложения по улучшению данного Прайс бота после тестирования в организации ART_milling

Инструменты: Python3, VScode, Aoigram, Git, SQLite, Asyncio, Draw.io

Состав команды: Васильева Екатерина Викторовна (Python-разработчик)



Введение

В современном мире пользование мобильными устройствами растет с каждым днем. Самыми распространёнными приложениями являются мессенджеры. В настоящее время для большинства пользователей основными предпочтениями для использования того или иного приложения является:

- минимум действий для решения задач;
- удобство взаимодействия с помощью мобильного устройства;
- отсутствие необходимости выходить из привычных приложений;
- Моментальная реакция на запросы

Соответствовать данным требованиям возможно при помощи чат-ботов.

В данной работе выбрана тема прайс-информационный чат-бот для производителей кухонь, заказывающих мебельные фасады, в компании ART_milling в Telegram.

Каждый раз, рассчитывая кухню, производители сталкиваются с проблемой расчета мебельных фасадов. Ценообразование зависит от многих факторов таких как толщина и тип материала из которого изготовлен фасад, пленка ПВХ разных производителей, разновидностей фрезеровки и т.д.

1 Теоретико-аналитическая часть

1.1 Постановка задачи

Чат-боты это виртуальные помощники, часть которых выполняет примитивные действия (напоминания, бронирование, запись информации, автоответчик), а другая построена на основе искусственного интеллекта. Основоположником можно считать Алана Тьюринга и его идею создания интеллектуальных машин. Технологии искусственного интеллекта, с помощью которых работают чат-боты, расширяют зоны своего присутствия. Они встроены в программы «умного дома», мобильные приложения, онлайн-банки и маркетплейсы.

Чат-бота создают, чтобы выстроить диалог с пользователем. Он похож на конструктор из сообщений и кнопок, имитирующий разговор между реальными людьми. Чат-боты могут однострочно отвечать на простой запрос или конструировать сложную беседу с высоким уровнем персонализации.

Чат-боты упрощают взаимодействие между клиентом и компанией и помогают экономить средства. Быстрый поиск ответа на вопрос, смена персональных данных, устранение мелких неполадок в приложении, оформление заявки на покупку продукта с этим чат-бот готов справиться. При необходимости он перенаправляет запрос вместе с полученными данными на нужного специалиста.



Рис. 1



Что чат-боты умеют:

- Отвечать на популярные вопросы. Бот расскажет о времени работы заведения, даст ссылки на сайт или каталог товаров, поможет разобраться с ценами.
- Принимать заказы. Чат-бот доступен круглосуточно, поэтому клиент может записаться на услугу или оформить заказ даже ночью, когда менеджеры уже не работают. Для этого нужно синхронизировать бота с CRM-системой. Информация о записях или заказах будет появляться в ней автоматически, а нагрузка на администратора снизится.
- Рассказывать о новых продуктах и акциях. Чат-бот может не только отвечать на вопросы пользователей, но и сам отправлять им сообщения. Используйте его как дополнительный канал связи и сообщайте клиентам, что запустили новую услугу или дарите им скидку.
- Уведомлять о статусе заказа. Чат-бот может информировать клиентов, что их заказ принят, оплачен, упакован или отправлен с курьером. В смс-расылках вы заплатите за одно сообщение около двух рублей, а с чат-ботом отправить уведомление можно в несколько раз дешевле или бесплатно. В конструкторе чат-ботов Unisender на бесплатном тарифе можно отправлять неограниченное количество сообщений, но только 500 клиентам.

Что чат-боты не умеют:

- Отвечать на нестандартные вопросы клиентов. Если у чат-бота нет доступа к каталогу товаров, он не может подсказать, есть ли товар на складе, или когда будет следующая поставка. Для этого к чат-боту нужно добавить кнопку или специальную команду, чтобы клиент мог позвать менеджера и обсудить этот вопрос с ним.
- Решать проблемные ситуации. Если клиент хочет узнать, почему его заказ долго едет, чат-бот не сможет ответить. В таких случаях бот может работать как посредник: он узнает номер заказа и другие детали, а потом передаст информацию менеджеру.



Преимущества чат-ботов

Круглосуточный доступ для пользователя

Чат-бот способен работать 24/7 и решать вопросы в моменте, поэтому клиенту не нужно звонить в офис компании и ждать на линии.

Самообслуживание

Клиент работает с ботом без общения с реальным человеком быстро, удобно, подходит для интровертов, упрощает процессы.

Экономия расходов

Бизнесу не нужно нанимать дополнительных сотрудников для общения с клиентами и обработки их запросов.

Сбор клиентских данных и отслеживание потребностей аудитории
Через чат-боты компания может изучать свою аудиторию, анализировать отзывы и поднимать продажи. Ранее клиенты с вопросами, проблемами или жалобами отправляли электронные письма или звонили в службу поддержки. Такой формат коммуникации сохраняется, но чат-боты позволяют оптимизировать количество входящих обращений и снизить нагрузку на персонал. Чат-боты допускают меньше ошибок при сборе данных и обрабатывают информацию быстрее, чем человек. Это позволяет сократить время контакта между первичным обращением клиента и получением услуги.

Эффективный инструмент продаж и вовлечения клиентов
Чат-боты дают клиенту точную информацию, упакованную под его потребности. Клиент чувствует, что его не перегружают информацией и не пытаются навязать много всего сразу. Он остаётся заинтересован в работе с компанией, получая в ответ бережное отношение к себе и рассылку с неагрессивной воронкой продаж.

Возможность подключения к чат-боту иностранных языков
Повышение лояльности к бренду и расширение своего присутствия на рынке через работу с клиентами за пределами страны. [1]

Чат-боты используются в связке с мессенджерами: VK, Facebook, Telegram

Рассмотрим понятие мессенджера, какие из них пользуются популярностью



у пользователей, а также для какого мессенджера лучше всего разрабатывать чат-бота.

Мессенджер – это программа для обмена быстрыми сообщениями. Приложение может быть установлено как на компьютер, так и на мобильное устройство.

Самыми популярными мессенджерами являются WhatsApp, Facebook Messenger, Viber, Telegram. Рассмотрим каждый из этих мессенджеров и их возможности по созданию чат-ботов.

Мессенджер WhatsApp является одним из самых массово используемых в России.

Для того чтобы создать чат-бота в WhatsApp необходимо:

- Иметь аккаунт в Facebook;
- Иметь бизнес-страницу и пройти верификацию через Facebook, что может занимать по времени примерно 1-2 месяца;
- Страница должна быть верифицирована на ту компанию, на которую планируется регистрация WhatsApp;
- Необходимо иметь свой сайт.

Данные пункты усложняют создание чат-бота на платформе WhatsApp. Кроме того, WhatsApp не предоставляет открытого API, что значительно усложняет разработку чат-бота. В данной ситуации можно использовать неофициальный API, но тогда в любой момент можно получить блокировку аккаунта. Также, в мессенджере имеется ограниченный инструмент для создания чат-бота и большинство функционала является платным.

Помимо всего вышеперечисленного, в данный момент на территории России Facebook заблокирован, что вызывает ещё больше трудностей для создания чат-бота в мессенджере WhatsApp. Поэтому платформа WhatsApp не подойдёт для создания чат-бота.

Facebook Messenger является менее популярным в России. Для создания чат-бота в Facebook Messenger необходимо:

- Необходимо иметь аккаунт Facebook;
- Иметь бизнес-страницу.

К одной странице может быть привязан только один чат-бот. Многие, кто пользуется социальной сетью Facebook, пользуется и мессенджером. [2]

Но у Facebook Messenger имеется та же проблема, что и мессенджера WhatsApp. Так как Facebook заблокирован на территории Российской Федерации, то данный мессенджер не подойдёт для создания на нём чат-бота.

Мессенджер Viber, также, как и WhatsApp, является одним из популярных мессенджеров, которым пользуются в России.

Чат-бот в мессенджере Viber привязывается к номеру телефона. Имеет ограниченный, но достаточно большой функционал для создания чат-ботов и постоянно развивается. Но у этого мессенджера не так много пользователей, поэтому писать для него чат бота тоже неправильно.

Из года в год статистика не меняется: ботами традиционно пользуется около 40% опрошенных, а больше половины общаются в групповых чатах. В то же время использование Telegram для корпоративного общения заметно растёт: в 2023 году уже каждый второй взаимодействует с коллегами по работе именно через этот мессенджер. Для общения с клиентами и подрядчиками его выбирают 26% пользователей. (Рис2) [3]



Рис2

Кроме мессенджеров, которые подходят для создания чат-ботов, рассмотрим социальную сеть ВКонтакте. В данной социальной сети есть возможность создания чат-ботов, а также она пользуется популярностью у молодёжи, но наш конечный продукт рассчитан на более возрастную аудиторию поэтому в контакте



нам не подходит.

1.2 Выбор метода проектирования и разработки

У мессенджера Telegram также много пользователей в России. Telegram стал самой первой платформой для создания чат-ботов.

Теперь нам нужно разобраться, что такое бот в Telegram.

Бот в приложении Telegram – это специальный аккаунт, созданный для того, чтобы автоматически обрабатывать и отправлять сообщения. Пользователи могут взаимодействовать с ботами при помощи сообщений, отправляемых через чат. Логика бота контролируется при помощи Bot API, который представляет из себя HTTP-интерфейс для работы с ботами в Telegram. [5]

Поэтому самым оптимальной платформой для реализации проекты является Telegram.

Чат-бот в мессенджере Telegram привязывается к номеру телефона. Имеет широкий функционал для создания чат-ботов. Чтобы создать чат-бота в мессенджере Telegram, необходимо:

- Написать пользователю @BotFather для создания бота в Telegram и следовать его инструкциям, чтобы создать своего бота;
- Получить токен от BotFather и использовать его для управления чат-ботом.

(Рис 2)

Для реализации прайс-бота потребуется:

- Выбрать среду проектирования, в которой будет определено, что влияет на данную задачу.
- Рассмотреть подходящие под задачу языки программирования, чтобы определиться на чём в дальнейшем разрабатывать чат-бота.
- Рассмотреть под выбранный язык программирования среду программирования и необходимые библиотеки.

В качестве демонстрации примерного интерфейса будет рассмотрено вебприложение draw.io.



Draw.io – это онлайн-инструмент для создания диаграмм, прототипов, блок-схем и других визуальных объектов. Разработчик данного приложения – компания JGraph Ltd. [4]

Draw.io имеет достаточно широкий выбор функций для визуализации задач пользователя. С помощью данного инструмента можно создавать различные диаграммы, графики, блок-схемы, эскизы программ, карты сайтов, ментальные карты и т.д. (Рис 3)

Особенности инструмента draw.io:

- множество различных шаблонов элементов и фигур;
- возможность совместной работы;
- поддержка горячих клавиш, задействованных в большинстве графических редакторов;
- мультиязычный интерфейс.

У данного веб-приложения простой и понятный интерфейс, при помощи которого легко можно за небольшой промежуток времени создать готовый проект.

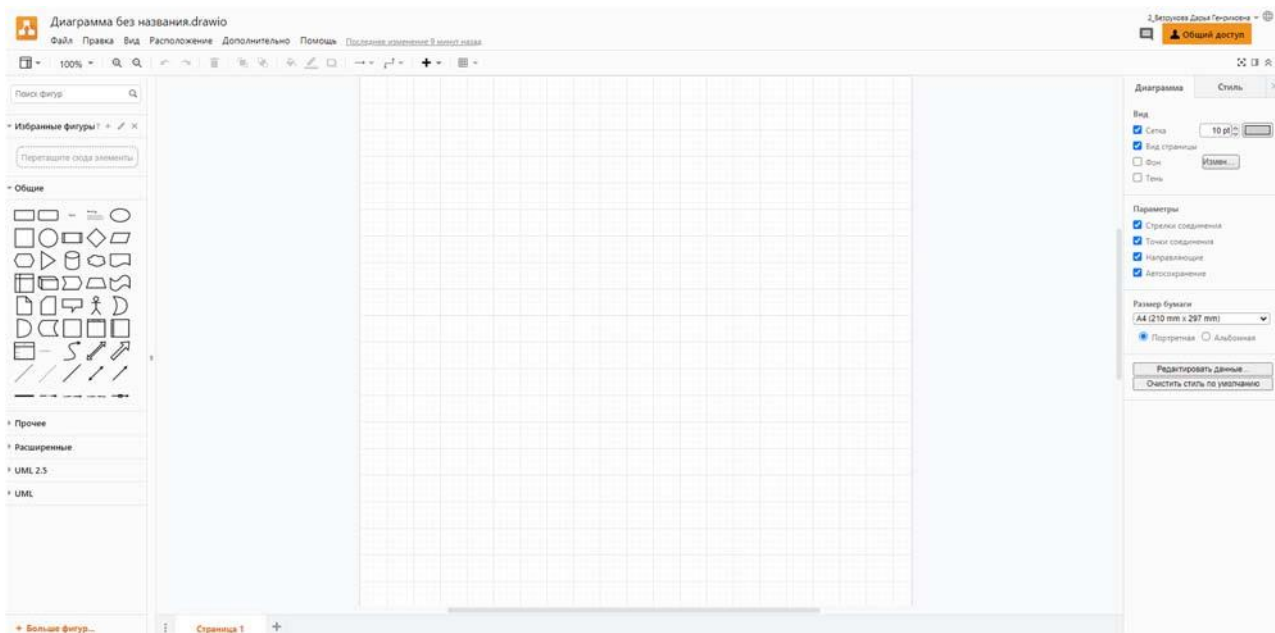


Рис. 3.

Написать чат-бота в Telegram можно используя почти любой язык программирования. Но я остановлюсь на Python.



Python – это высокоуровневый, интерпретируемый объектно-ориентированный язык программирования общего назначения, с помощью которого можно решать различные задачи. Включает в себя модули, исключения, динамическую типизацию, динамические типы данных и классы. Автором языка программирования Python является голландский программист Гвидо Ван Россум. [6]

Python является мультипарадигмальным языком программирования, который поддерживает императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование.

Рассмотрим библиотеки для языка программирования Python, такие как `pyTelegramBotAPI`, `python-telegram-bot`, `aiogram`, `rocketgram` для создания чатбота в Telegram.

`pyTelegramBotAPI` – это простая и лёгкая библиотека для создания ботов на языке программирования Python для мессенджера Telegram. Поддерживает версию Bot API 6.0. [7] Данная библиотека есть на GitHub.

`Python-telegram-bot` – это библиотека, которая имеет ряд классов высокого уровня, которые облегчают и упрощают разработку ботов в Telegram. Поддерживает Bot API 6.0. Совместим с Python 3.7 и выше. [7] Библиотека также есть на GitHub.

`Aiogram` – это довольно простая библиотека и полностью асинхронная, использует декораторы и содержит удобные инструменты для разработки. Помогает сделать бота быстрее и проще. [8] Эту библиотеку можно найти на GitHub. `Rocketgram` – это асинхронная библиотека для создания каркаса бота в Telegram. [9] Её так же можно найти на GitHub.

Python позволяет перегружать операторы, что даёт возможность использовать выражения близкие к естественным.

Касательно ООП в Python отсутствуют модификаторы доступа к полям и методам класса, атрибуты и поля у объектов могут создаваться в ходе исполнения программы, а все методы являются виртуальными. Скорость выполнения кода



на Python (как и других интерпретируемых языков) значительно ниже, чем скорость выполнения аналогичного кода на C++ и обычно ожидается ниже, чем в Java. Код на C++ получается производительнее Python, при этом занимает больше строк.

Кроме того, Python является самым популярным языком программирования, который используют для написания чат-ботов. Библиотеку для данной задачи можно выбрать любую, поэтому я выбрала Aiogram.

А теперь рассмотрим редакторы и среды для разработки, подходящие для языка программирования Python.

IDLE – это интегрированная среда разработки на языке Python. Данный редактор обычно поставляется вместе с Python. IDLE является хорошим редактором для начала программирования и понимания основ языка. [10] В нём есть оболочка Python — интерактивный интерпретатор. Его возможности обширны: автоматическое завершение кода, подсветка синтаксиса, подбор отступа и базовый встроенный отладчик.

Плюсами IDLE являются:

- легкость;
- подходит для начинающих.

Минусы:

- не подходит для сложных проектов;
- не хватает продвинутых функций.

Visual Studio Code (VS Code) — это кросс-платформенный редактор кода от компании Microsoft, разработанный на базе фреймворка Electron.

С его помощью можно разрабатывать кросс-платформенные десктопные приложения, используя веб-технологии. VS Code не привязан к определённому языку программирования, поэтому с его помощью можно создавать сайты, мобильные приложения, работать с базами данных и тестировать сервисы.

Возможности VS Code:

- подсветка синтаксиса;



- автоматическое дополнение;
- контроль версий;
- отладка;
- рефакторинг.

VS Code можно установить на Windows, macOS и Linux.[12]

Кроме вышеперечисленных редакторов существуют различные онлайн-редакторы. Плюсом таких редакторов является, что не нужно ничего дополнительно устанавливать и настраивать, и простой код без проблем можно запустить. Минусом является, что они не такие мощные, как IDE.

Для разработки прайс-бота выбрана среда разработки VSCode, её будет достаточно для реализации данной задачи.

В данной работе использовано:

- инструмент для демонстрации примерного интерфейса – Draw.io;
- язык программирования – Python;
- библиотека – Aiogram
- среда разработки VSCode

1.3 Описание технического задания на разработку прайс чат-бота «MDFBot»

Составлено на основе ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы». [11]

1. Общие сведения.

1.1. Название организации-заказчика.

Компания ART_milling

1.2. Название продукта разработки (проектирования).

«Прайс бот МДФ фасадов»

1.3. Назначение продукта.

Продукт предназначен для предоставления справочной информации о ценах на МДФ фасады клиентам компании



1.4. Плановые сроки начала и окончания работ.

В соответствии с планом выполнения Дипломного проекта (26.04.2024 – 13.05.2024).

1.5. Характеристика персонала (количество, квалификация, степень готовности)

Разработчик – написание кода на языке Python, работа с API Telegram. Пользователи – уверенный уровень владения ПК или мобильным устройством, а также мессенджерами.

2. Требования к продукту разработки.

2.1. Требования к продукту в целом.

Предполагается прайс чат-бот для ответа на вопросы клиентов о ценах на продукцию

2.2. Аппаратные требования.

Персональный компьютер или мобильный телефон.

2.3. Указание системного программного обеспечения (операционные системы, браузеры, программные платформы и т.п.).

iOS (версии 9 и выше), Android (версии 4.1 и выше), MacOS, Windows 7 и выше, Linux, либо веб-браузер.

2.4. Указание программного обеспечения, используемого для реализации. Python, Telegram, VScode.

2.5. Для сетевых систем – особенности реализации серверной и клиентской частей.

Серверная часть – приложение на языке программирования Python, осуществляющее работу пользовательских запросов.

Клиентская часть – бот в ПО Telegram.

2.6. Форматы входных и выходных данных

Формат входных данных – команды и текст запроса.

Формат выходных данных – сформированный ответ с текстом и ценами на выбранную категорию продукции.



2.7. Порядок взаимодействия с другими системами, возможности обмена информацией.

Возможность обмена сообщениями в ПО Telegram.

2.8. Меры защиты информации.

Автоматическое сквозное шифрование сообщений предусмотрено ПО Telegram.



2 Практическая часть

2.1 Описание процесса разработки

Для начала опишем процесс создания чат-бота в мессенджере Telegram.

Основная последовательность шагов при создании чат-бота на платформе Telegram обычно выглядит следующим образом:

1. Необходимо написать пользователю @BotFather.
2. При помощи команды /newbot создаём бота, даём ему имя и имя пользователя (Рис2).
3. @BotFather выдаёт нам токен, с которым дальше и буду работать.
4. Далее, при помощи команды /mybots можно отредактировать имя бота, добавить информацию о боте, прописать подсказки команд, изменить аватарку бота и добавить описание бота.
5. В программе VScode создаем виртуальное окружение и активируем его с помощью команд `python -m venv .venv` и активируем с помощью команды `venv\Scripts\activate`.

```
PS K:\MDFBot> python -m venv .venv
```

Данная операция создаст новое виртуальное окружение и позволит нам изолировать используемые модули в нашей программе.

```
• PS K:\MDFBot> .venv\Scripts\activate  
○ (.venv) PS K:\MDFBot> |
```

Активированное виртуальное окружение появится в скобках перед путем к проекту (.venv)

Если при активации виртуального окружения возникнет ошибка, то предварительно выполнить команду `Set-ExecutionPolicy -ExecutionPolicy AllSigned -Scope CurrentUser`

6. Скачиваем Python и устанавливаем необходимые библиотеки.



7. Предварительно смотрим какие библиотеки у нас установлены с помощью команды `pip list`. Так как мы создали и активировали виртуальное окружение и продолжили работу в нем, то все библиотеки которые мы устанавливаем будут устанавливаться в папку с виртуальным окружением

```
• (.venv) PS K:\MDFBot> pip list
Package      Version
-----
pip          23.2.1
rectpack     0.2.2
setuptools   39.0.1

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
○ (.venv) PS K:\MDFBot> 
```

Устанавливаем библиотеку `aiogram`

```
• (.venv) PS K:\MDFBot> pip install aiogram
```

`Aiogram` это библиотека Python, предназначенная для создания телеграмм-ботов. Современная библиотека, набирающая популярность: многие чат-боты написаны на ней. Библиотека реализует асинхронное выполнение кода, что позволяет не останавливать работу бота в ожидании ответа пользователя. Кроме того, у `Aiogram` есть подробная документация и большое русскоязычное комьюнити. [17]

Вот некоторые плюсы библиотеки `Aiogram`:

- Простой синтаксис: `aiogram` предлагает чистый и понятный синтаксис, что упрощает начало работы и разработку ботов.
- Поддержка асинхронности: Библиотека разработана с учетом асинхронного программирования, что позволяет эффективно обрабатывать множество запросов одновременно.
- Обработка сообщений: `aiogram` предоставляет мощные средства для обработки сообщений, команд и событий, включая текстовые сообщения, изображения, аудио и многие другие типы данных.



- Интерактивные клавиатуры: Вы можете легко создавать интерактивные клавиатуры для взаимодействия с пользователями вашего бота.
- Интеграция с сторонними сервисами: aiogram поддерживает интеграцию с базами данных, веб-сервисами и другими внешними ресурсами, что делает ее универсальным инструментом. [9]

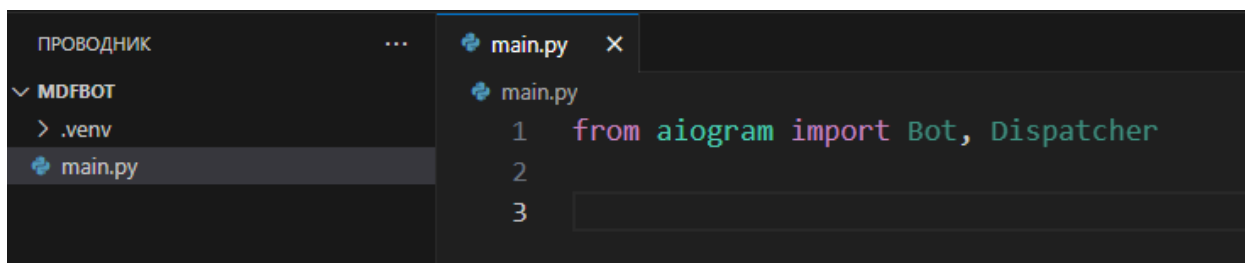
При установке aiogram плюсом устанавливается еще несколько библиотек необходимых для работы

Список библиотек после установки aiogram:

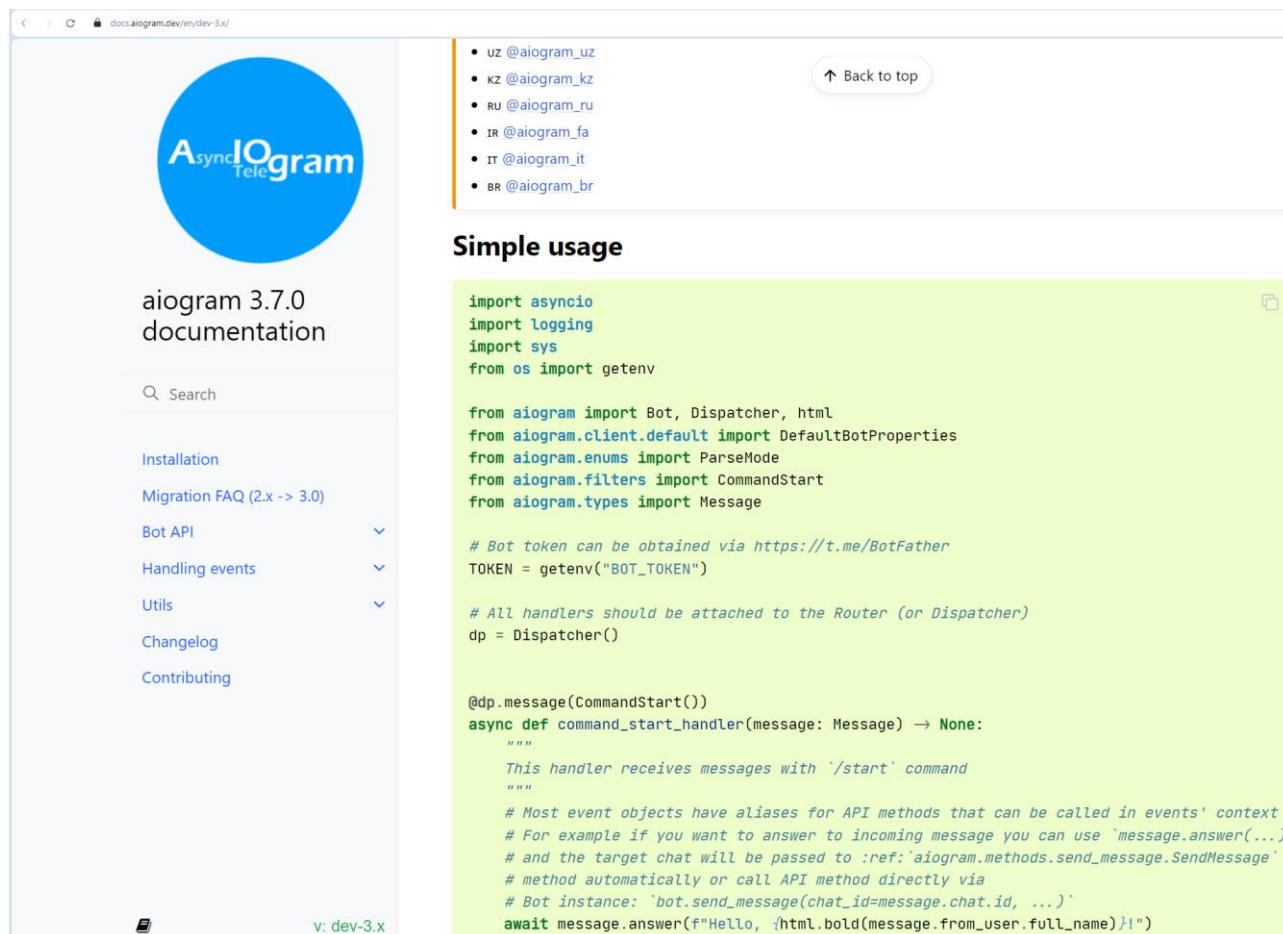
```
• (.venv) PS K:\MDFBot> pip list
Package                Version
-----
aiofiles                23.2.1
aiogram                 3.5.0
aiohttp                 3.9.5
aiosignal               1.3.1
annotated-types         0.6.0
attrs                   23.2.0
certifi                 2024.2.2
frozenlist              1.4.1
idna                    3.7
magic-filter            1.0.12
multidict               6.0.5
pip                     24.0
pydantic                2.7.1
pydantic_core           2.18.2
rectpack                0.2.2
setuptools              39.0.1
typing_extensions       4.11.0
yarl                    1.9.4
◦ (.venv) PS K:\MDFBot> |
```

8. Дальше приступаем к написанию кода. Для начала необходимо импортировать библиотеку и подключить токен нашего бота

В папке с ботом создаем файл main.py (это будет основной файл бота)



Для начала смотрив документацию по aiogram. С чего начать работу с этой библиотекой. Документацию можно посмотреть на сайте docs.aiogram.dev:



При работе с библиотекой я буду использовать следующие термины:

- ЛС — личные сообщения, в контексте бота это диалог один-на-один с пользователем, а не группа/канал.
- Чат — общее название для ЛС, групп, супергрупп и каналов.
- Апдейт — любое событие из этого списка: сообщение, редактирование сообщения, колбэк, инлайн-запрос, платёж, добавление бота в группу и т.д.



- Хэндлер — асинхронная функция, которая получает от диспетчера/роутера очередной апдейт и обрабатывает его.
- Диспетчер — объект, занимающийся получением апдейтов от Telegram с последующим выбором хэндлера для обработки принятого апдейта.
- Роутер — аналогично диспетчеру, но отвечает за подмножество множества хэндлеров. Можно сказать, что диспетчер — это корневой роутер.
- Фильтр — выражение, которое обычно возвращает True или False и влияет на то, будет вызван хэндлер или нет.
- Мидлварь — прослойка, которая вклинивается в обработку апдейтов.

После этого импортируем 2 класса это Bot и Dispatcher.

Создаем 2 экземпляра класса. первый объект это бот, второй объект это диспетчер. Диспетчер занимается хендлерами.

```
main.py x
main.py > ...
1 ~ import asyncio
2   from aiogram import Bot, Dispatcher # импорт 2х классов
3
4   #создание 2х экземпляров классов
5   bot = Bot(token='') # в этой переменной хранится бот
6   dp = Dispatcher() # то как диспетчер будет обрабатывать это сообщение
7
8
9 ~ async def main():
10     await dp.start.polling(bot)
11
12
13 ~ if __name__ == '__main__':
14     asyncio.run(main())
15
16
```

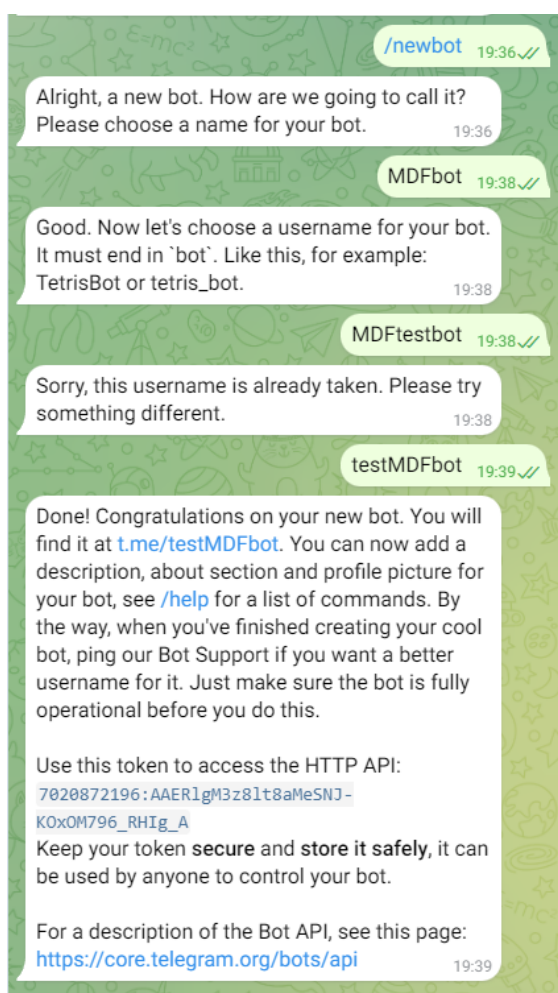
Также создаем функцию main() и в поллинг передаем бота. Файл main это основной файл бота. Далее создаем дополнительную конструкцию чтобы запускать вышесозданную функцию.



Конструкция `if __name__ == '__main__':` означает, то что внутри будет запускаться только в том случае если мы запустили именно этот файл, если мы будем его Далее заходим в телеграмм импортировать, то программа запускаться не будет и произойдет ошибка.

Заходим в телеграмм и с помощью официального бота @BotFather создаем нового бота используя функцию `/newbot`. Далее нас просят указать имя пользователя и название самого бота. После всех вышеперечисленных процедур нам сообщают что новый бот создан и дают его токен, который нам и требовался.

В объект Bot в его токен вставляем код полученный от @Botfather:



После всех действий наш бот готов, но он пока ничего не умеет делать, только подключаться к телеграмму.

9. Теперь при помощи языка программирования Python необходимо прописать обработчик, он будет принимать все сообщения. Прописали декоратор у него есть диспетчер и команда старт. Мы обрабатываем сообщение, на



вход асинхронной функции будет приходить сообщение и мы будем отвечать сообщением.

Выровем новый терминал и запустим бота для проверки. В терминале нужно прописать команду `python main.py`





Чтобы закрыть бот необходимо нажать на клавиатуре ctrl + C, появится окно с «ошибкой»:

```
File "C:\Users\User\AppData\Local\Programs\Python\Python312\Lib\asyncio\tasks.py", line 454, in wait
    return await _wait(fs, timeout, return_when, loop)
File "C:\Users\User\AppData\Local\Programs\Python\Python312\Lib\asyncio\tasks.py", line 540, in _wait
    await waiter
asyncio.exceptions.CancelledError

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "K:\MDFBot\main.py", line 26, in <module>
    File "C:\Users\User\AppData\Local\Programs\Python\Python312\Lib\asyncio\runners.py", line 194, in run
        return runner.run(main)
  File "C:\Users\User\AppData\Local\Programs\Python\Python312\Lib\asyncio\runners.py", line 123, in run
    raise KeyboardInterrupt
KeyboardInterrupt
(.venv) PS K:\MDFBot> |
```

Чтобы эту ошибку исправить и чтобы при закрытии бота на экране выходила надпись Бот выключен, необходимо прописать:

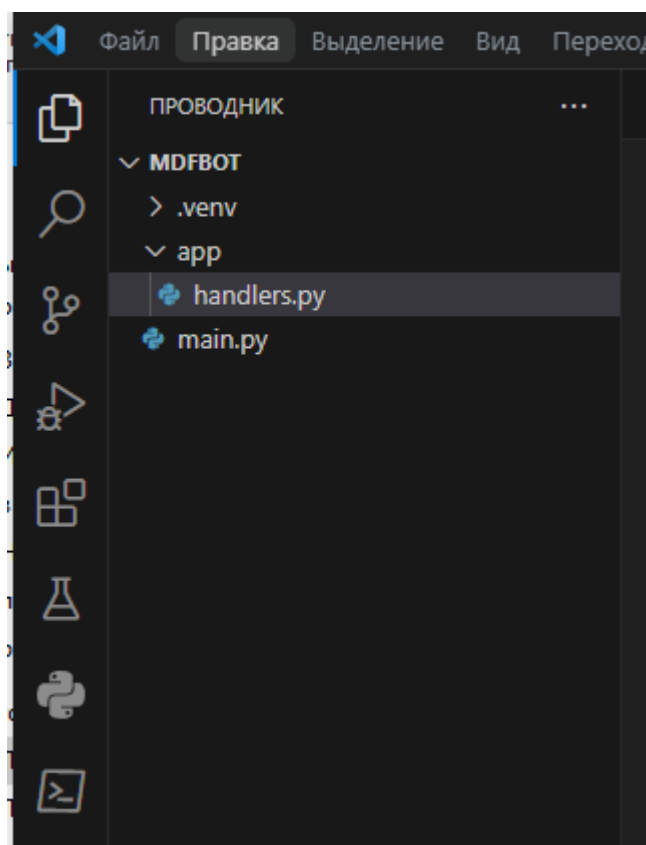
```
if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print('Бот выключен!')
```

```
• (.venv) PS K:\MDFBot> python main.py
Бот выключен!
○ (.venv) PS K:\MDFBot> |
```

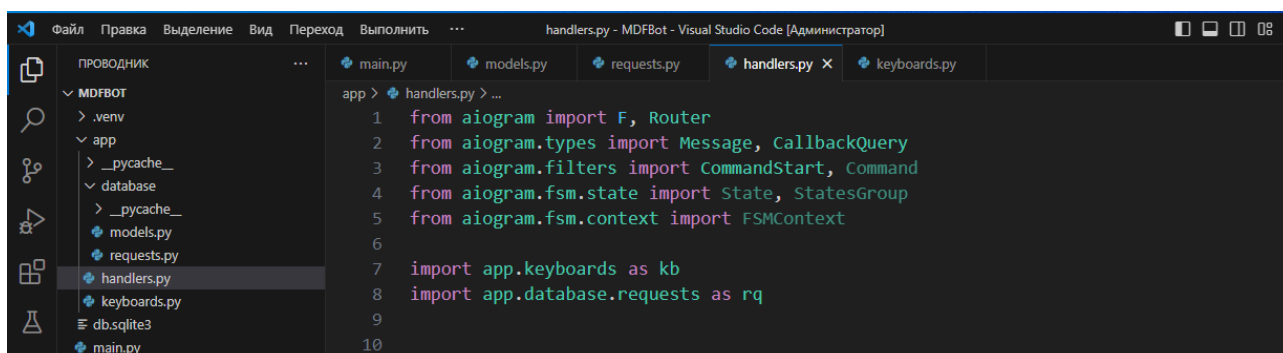



Создадим фильтр чтобы бот отвечал не на каждое сообщение, а на какое то определенное, для этого надо импортировать из aiogram `CommandStart`, `Command`, `F`.

Если в вашем проекте будет много обработчиков, то будет очень сложно разобраться какой и за что отвечает, поэтому для удобства все роутеры перенесем в отдельный файл `handlers.py`



Все `@dp.message` теперь будут храниться в этом файле в папке `app` находящейся в папке с проектом.



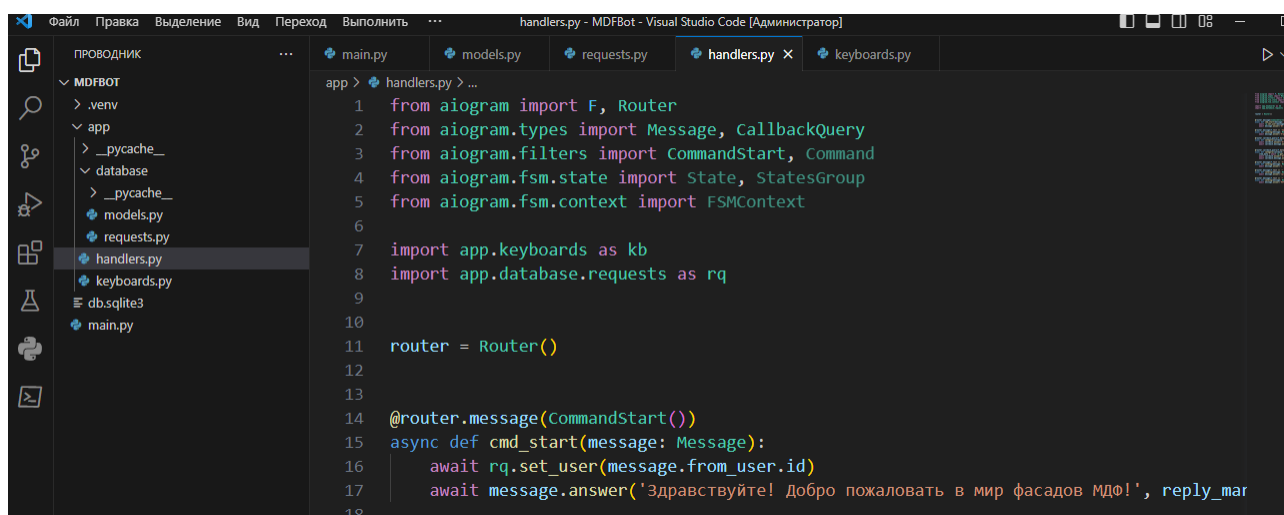


Затем надо импортировать класс `router` из `aiogram`? Потому что в противном случае будут ошибки. Есть роутеры которые будут обслуживать пользователей, какие то администратора в будущем.

Создаем объект от этого класса и назовем его также роутер

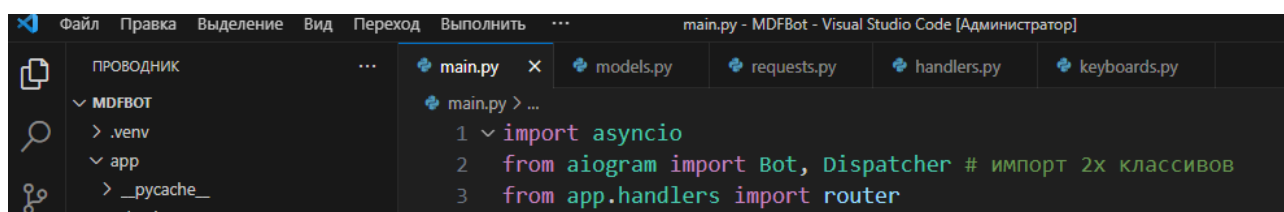
```
2 from aiogram import Bot, Dispatcher
3
4 from app.handlers import router
5
```

Теперь названия всех обработчиков будут начинаться с `@router`



```
1 from aiogram import F, Router
2 from aiogram.types import Message, CallbackQuery
3 from aiogram.filters import CommandStart, Command
4 from aiogram.fsm.state import State, StatesGroup
5 from aiogram.fsm.context import FSMContext
6
7 import app.keyboards as kb
8 import app.database.requests as rq
9
10
11 router = Router()
12
13
14 @router.message(CommandStart())
15 async def cmd_start(message: Message):
16     await rq.set_user(message.from_user.id)
17     await message.answer('Здравствуйте! Добро пожаловать в мир фасадов МДФ!', reply_mar
18
```

Но диспетчер и роутеры пока не связаны, надо это исправить, для этого дописывание строчку `from app.handlers import router`.



```
1 import asyncio
2 from aiogram import Bot, Dispatcher # импорт 2х классов
3 from app.handlers import router
```

Затем диспетчеру сообщаем о добавлении роутеров строчкой в файле `main.py` `dp.include_router(router)`

Так как у нас хендлер только один, мы можем его переместить внутрь функции:

```
async def main()
```



```
1 import asyncio
2 from aiogram import Bot, Dispatcher # импорт 2х классов
3 from app.handlers import router
4 from app.database.models import async_main
5
6
7 async def main():
8     await async_main()
9     bot = Bot(token='7020872196:AAERlgM3z8lt8aMeSNJ-K0x0M796_RHig_A') # в этой переменн
10    dp = Dispatcher()
11    dp.include_router(router)
```

Чтобы пользователю было удобно пользоваться ботом необходимо сделать клавиатуру с кнопками. Для этого импортируем ReplyKeyboardMarkup и KeyboardButton из aiogram

```
app > keyboards.py
1 from aiogram.types import ReplyKeyboardMarkup, KeyboardButton
2
3
4
5
```

Кнопки которые находятся в окне сообщений называют Inline кнопками, а те клавиши которые находятся снизу от окна ввода текста называются Reply клавиатурой. Такой название они получили потому что при нажатии отправляется определенное сообщение в чат. Inline кнопки сообщение не отправляют, они отправляют какой то запрос.

Создаем основную клавиатуру: создаем файл keyboards.py в папке app, который находится в папке с проектом. В этой клавиатуре 4 кнопки в 3 ряда. Первый ряд одна кнопка, второй ряд – одна, третий ряд 2 кнопки. ReplyKeyboardMarkup – это класс, main является экземпляром класса. В клавиатуру добавляем кнопки с помощью класса KeyboardButton/

```
9
10 main = ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text='Каталог')],
11                                       [KeyboardButton(text='О фасадах')],
12                                       [KeyboardButton(text='Контакты'),
13                                       KeyboardButton(text='О нас')]]],
```

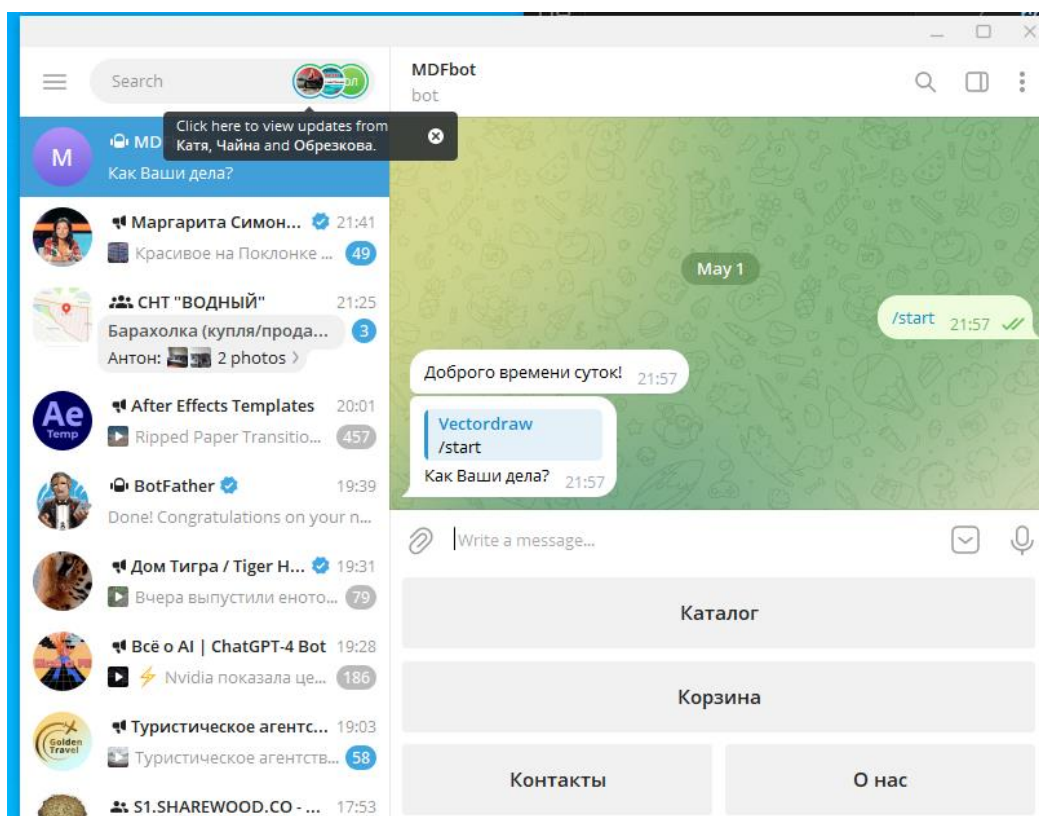


Клавиатура всегда связана с апдейтом который вы отправляете, т.е. сообщением.

```
14 @router.message(CommandStart())
15 async def cmd_start(message: Message):
16     await rq.set_user(message.from_user.id)
17     await message.answer('Здравствуйте! Добро пожаловать в мир фасадов МДФ!', reply_markup=kb.main)
18
```

Прописываем клавиатуру в файл handlers.py с помощью reply_markup.

Но после тестирования бота выяснилось ,что кнопки клавиатуры слишком большие:

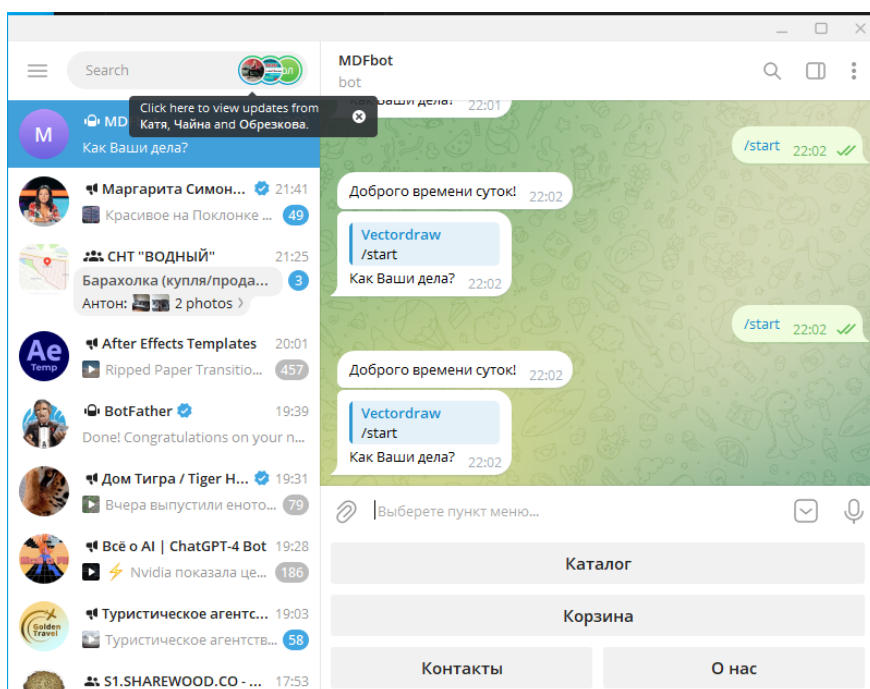


Для исправления надо прописать в клавиатуру еще два параметр `Resize_keyboard = True`, этот аргумент делает клавиатуру маленькой, а также добавляем `Place_holder` – это сообщение которое видно в окошке где нужно набирать текст, пока у нас там write a message (напишите сообщение), мы его изменим на выберите пункт меню:



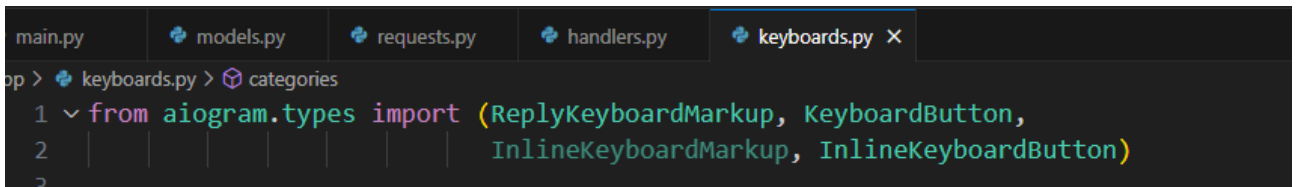
```
1 from aiogram.types import ReplyKeyboardMarkup, KeyboardButton
2
3 main = ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text='Каталог'),
4                                         KeyboardButton(text='Корзина'),
5                                         KeyboardButton(text='Контакты'),
6                                         KeyboardButton(text='О нас')]],
7                             resize_keyboard=True,
8                             input_field_placeholder='Выберете пункт меню...')
```

И после этого кнопки стали выглядеть гораздо лучше и строка в окне сообщений тоже поменялась:



Теперь создадим инлайн клавиатуру, которую впоследствии подключим к базе данных. При нажатии кнопки каталог будет появляться инлайн клавиатуру с некоторым списком.

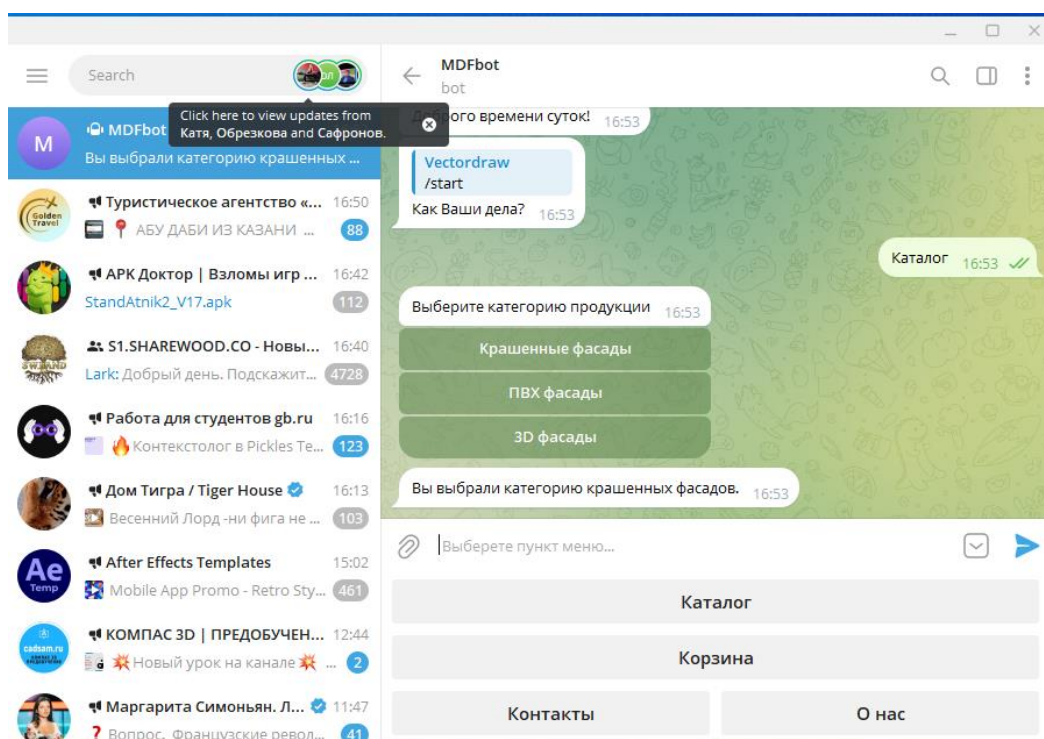
Для этого нужно снова импортировать несколько библиотек InlineKeyboardMarkup и InlineKeyboardButton



Visual Studio Code interface showing the file explorer on the left and the code editor on the right. The file explorer displays the project structure, including folders like .venv, app, and files like handlers.py, keyboards.py, db.sqlite3, and main.py. The code editor shows the content of keyboards.py, which includes imports from aiogram.types, a main function defining a keyboard with buttons for 'Каталог', 'Корзина', 'Контакты', and 'О нас', and a catalog function defining an inline keyboard with buttons for 'Крашенные фасады', 'ПВХ фасады', and '3D фасады'.



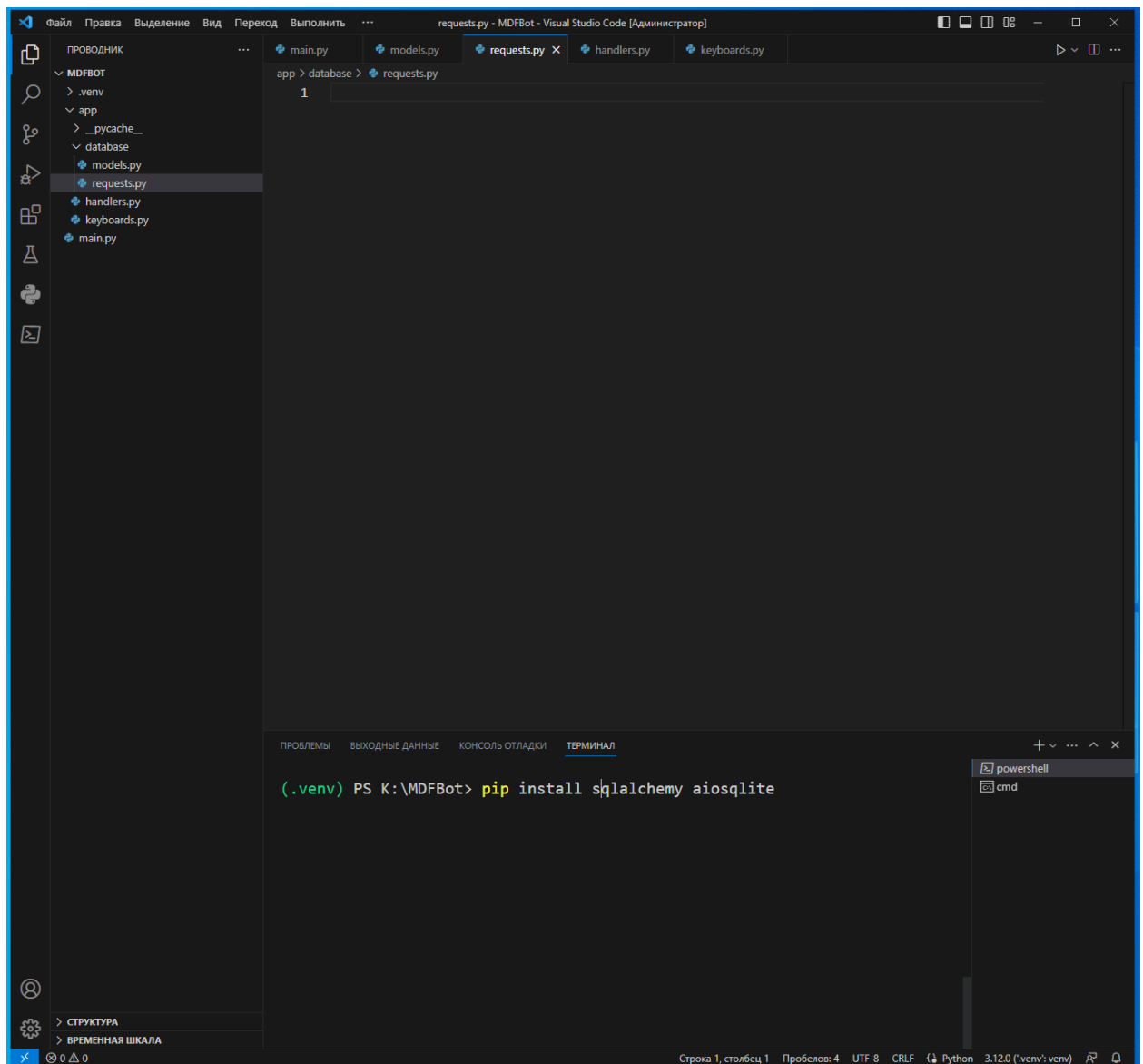
И после нажатия на крашенные фасады, будет появляться сообщение о том, что вы выбрали.



Чтобы наш бот мог работать с базами данных необходимо создать папку database в каталогу app. А в этой папке создадим 2 файла models.py и requests.py/

В файле models будут храниться модели, т.е. структура базы данных. В файле requests будут храниться запросы которые мы будем делать используя базу данных.

Теперь необходимо установить две библиотеки необходимые для работы с базами данных командой `pip install sqlalchemy aiomysql`



После этого заходим в файл `models` и начнем создавать базу данных товара:
Сначала пропишем импорты необходимых библиотек которые мы установили:

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column  
а также эту библиотеку  
from sqlalchemy.ext.asyncio import AsyncAttrs, async_sessionmaker, create_async_engine
```

Так как `aiogram` асинхронная библиотека, соответственно все другие фреймворки тоже должны быть асинхронными.



Для начала надо создать engine это подключение и создание БД, простыми словами «движок».

```
1 from sqlalchemy import BLOB, BigInteger, String, ForeignKey
2 from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
3 from sqlalchemy.ext.asyncio import AsyncAttrs, async_sessionmaker, crea
4
5
6
7 #подключаю и создаю базу
8 engine = create_async_engine(url='sqlite+aiosqlite:///db.sqlite3')
9
10
```

В engine прописывается какая база данных буде использоваться это sqlite и вместе с ней будет использоваться драйвер aiosqlite и прописываем название базы данных db.sqlite3

Следующим этапом нужно произвести подключение к базе данных командой через session maker в который передается ранее созданный engine.

```
11
12 async_session = async_sessionmaker(engine)
13
```

Мы должны реализовать класс Base который является дочерним по отношению AsyncAttrs и DeclarativeBase но при этом другие классы которые мы будем создавать будут дочерними по отношению к Base.

Base являясь основным классом дает возможность управлять всеми дочерними классами. А также мы сможем управлять всеми созданными таблицами с помощью Base.

Создаем класс user с таблицей, имеющей название users в которой будут храниться id user Telegram. Название класса в единственном числе а названия таблицы во множественном числе. Если обращаться к принципам объектно-ориентированного программирования, то мы описываем модель одного объекта, т.е. модель одного пользователя., что он в себе содержит id, telegram id и так далее могу быть различные данные. Но в таблице будут храниться уже много пользователей, поэтому таблица называется множественным числом.



Класс пользователь будет содержать id который будет записываться в таблицу базы данных. А также необходимо записывать telegram id чтобы можно было идентифицировать пользователя? Это значение является уникальным.

```
11
12 async_session = async_sessionmaker(engine)
13
14
15 class Base(AsyncAttrs, DeclarativeBase):
16     pass
17
18
19 class User(Base):
20     __tablename__ = 'users'
21
22     id: Mapped[int] = mapped_column(primary_key=True)
23     tg_id = mapped_column(BigInteger)
24
25
26
27 class Category(Base):
28     __tablename__ = 'categories'
29
30     id: Mapped[int] = mapped_column(primary_key=True)
31     name: Mapped[str] = mapped_column(String(25))
32
33
34
35 class Item(Base):
36     __tablename__ = 'items'
37
38     id: Mapped[int] = mapped_column(primary_key=True)
39     name: Mapped[str] = mapped_column(String(25))
40     description: Mapped[str] = mapped_column(String(150))
41     price: Mapped[int] = mapped_column()
42     category: Mapped[int] = mapped_column(ForeignKey('categories.id'))
43
```

```
PS K:\MDFBot> .venv\Scripts\activate
(.venv) PS K:\MDFBot> python main.py
```

Так же создадим еще 2 класса это Categories и Items. В которых буду храниться толщина МДФ которую используется компания, и соответственно рисунки которые она может сделать на том или другом МДФ. Везде прописывается id чтобы была связь между таблицами. В свойствах каждой строчке таблицы нужно указать ограничения по количеству символов.



Теперь нужно создать еще одну функцию `async def async_main()`. Через engine начинается новая сессия и создается новая переменная и делается синхронизация и создается база данных с таблицами.

В файле `main` сделаем еще один импорт, и пропишем функцию чтобы при запуске бота создались все эти таблицы.

```
main.py x models.py requests.py handlers.py app handlers.py KA...\MDFBot\... keyboards.py
main.py > ...
1 import asyncio
2 from aiogram import Bot, Dispatcher # импорт 2х классов
3 from app.handlers import router
4 from app.database.models import async_main
5
6
7 async def main():
8     await async_main()
9     bot = Bot(token='7020872196:AAERlgM3z8lt8aMeSNJ-K0xOM796_RHlg_A') # в этой переменной храни
10    dp = Dispatcher()
11    dp.include_router(router)
12    await dp.start_polling(bot)
13
14
15 if __name__ == '__main__':
16     try:
17         asyncio.run(main())
18     except KeyboardInterrupt:
19         print('Бот выключен!')
20
```

Результатом этих действий будет создание файла базы данных `db.sqlite3`

Для просмотра базы данных воспользуемся программой `SQLiteStudio`, которую можно скачать по адресу [15]/

При открытии файлы базы данных видно, что база данных есть, все таблицы которые мы прописали тоже есть, значит все работает правильно.



SQLiteStudio (3.4.4) - [categories (db)]

Database Structure View Tools Help

Databases

Filter by name

db (SQLite 3)

Tables (3)

categories

items

users

Views

Structure Data Constraints Indexes Triggers DDL

Table name: categories

WITHOUT ROWID STRICT

Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1 id	INTEGER	🔑				⚠️			NULL
2 name	VARCHAR (25)					⚠️			NULL

Type Name Details

1 PRIMARY KEY (id)

Status

[09:41:56] Database passed in command line parameters (K:\MDFBot\db.sqlite3) has been temporarily added to the list under name: db

users (db) items (db) categories (db)

SQLiteStudio (3.4.4) - [items (db)]

Database Structure View Tools Help

Databases

Filter by name

db (SQLite 3)

Tables (3)

categories

items

users

Views

Structure Data Constraints Indexes Triggers DDL

Table name: items

WITHOUT ROWID STRICT

Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1 id	INTEGER	🔑				⚠️			NULL
2 name	VARCHAR (25)					⚠️			NULL
3 description	VARCHAR (150)					⚠️			NULL
4 photo	BLOB					⚠️			NULL
5 price	INTEGER					⚠️			NULL
6 category	INTEGER		🔗			⚠️			NULL

Type Name Details

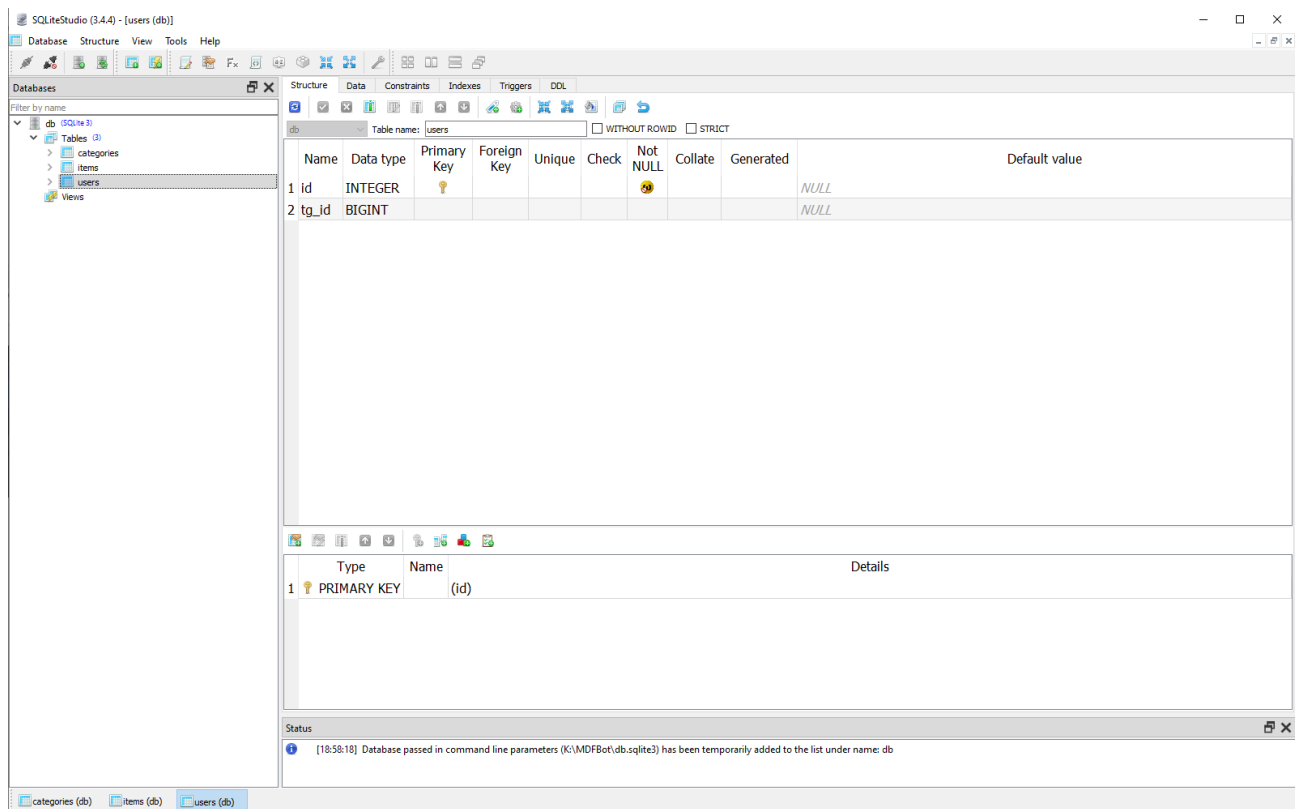
1 PRIMARY KEY (id)

2 FOREIGN KEY (category) REFERENCES categories (id)

Status

[18:58:18] Database passed in command line parameters (K:\MDFBot\db.sqlite3) has been temporarily added to the list under name: db

categories (db) items (db)



Теперь приступает к созданию requests базы данных. Первый раз при запуске бота пользователем он должен записываться в базу данных.

Из файла models импортируем функцию `async_session`, а также импортируем все модели.

Также импортируем операцию `select` из библиотеки `sqlalchemy`.

Создадим функцию `set_user` в которую будет передаваться телеграмм ID пользователя. Через тот же контекстный менеджер мы должны начать сессию создав переменную. Сессия должна открываться и закрываться. Когда функция контекстного менеджера заканчивает работу, сессия автоматически закрывается.

Создаем запрос в базу данных, результат функции запишем в переменную `user`.

```
user = await session.scalar(select(User).where(User.tg_id == tg_id))
```

Scalar функция которая выдает точные чистые данные, т.е. объект с полями.

Если мы не находим этого пользователя в базе данных ,то мы его туда добавляем используя конструкцию:



if not user:

```
session.add(User(tg_id=tg_id))
```

`await session.commit()` # это для того чтобы введенная информация сохранилась

Если пользователь в базе данных есть, то, естественно, это все не выполняется

```
1 from app.database.models import async_session
2 from app.database.models import User, Category, Item
3 from sqlalchemy import select
4
5 async def set_user(tg_id: int):
6     async with async_session() as session:
7         user = await session.scalar(select(User).where(User.tg_id == tg_id))
8
9         if not user:
10             session.add(User(tg_id=tg_id))
11             await session.commit()
12
13
14 async def get_categories():
15     async with async_session() as session:
16         return await session.scalars(select(Category))
17
18
19 async def get_category_item(category_id):
20     async with async_session() as session:
21         return await session.scalars(select(Item).where(Item.category == category_id))
22
23
24 async def get_item(item_id):
25     async with async_session() as session:
26         return await session.scalar(select(Item).where(Item.id == item_id))
```

```
(.venv) PS K:\MDFBot> python main.py
Бот выключен!
(.venv) PS K:\MDFBot> python main.py
```

В файле handlers делаем импорт файла requests. Далее обращаемся к функции `set_user` и передаем туда `id` пользователя.

По `id` пользователя ищем пользователя в базе данных, если он есть то ничего больше не делаем, если его нет то мы его туда добавляем. Это все делается на команде `Start`.



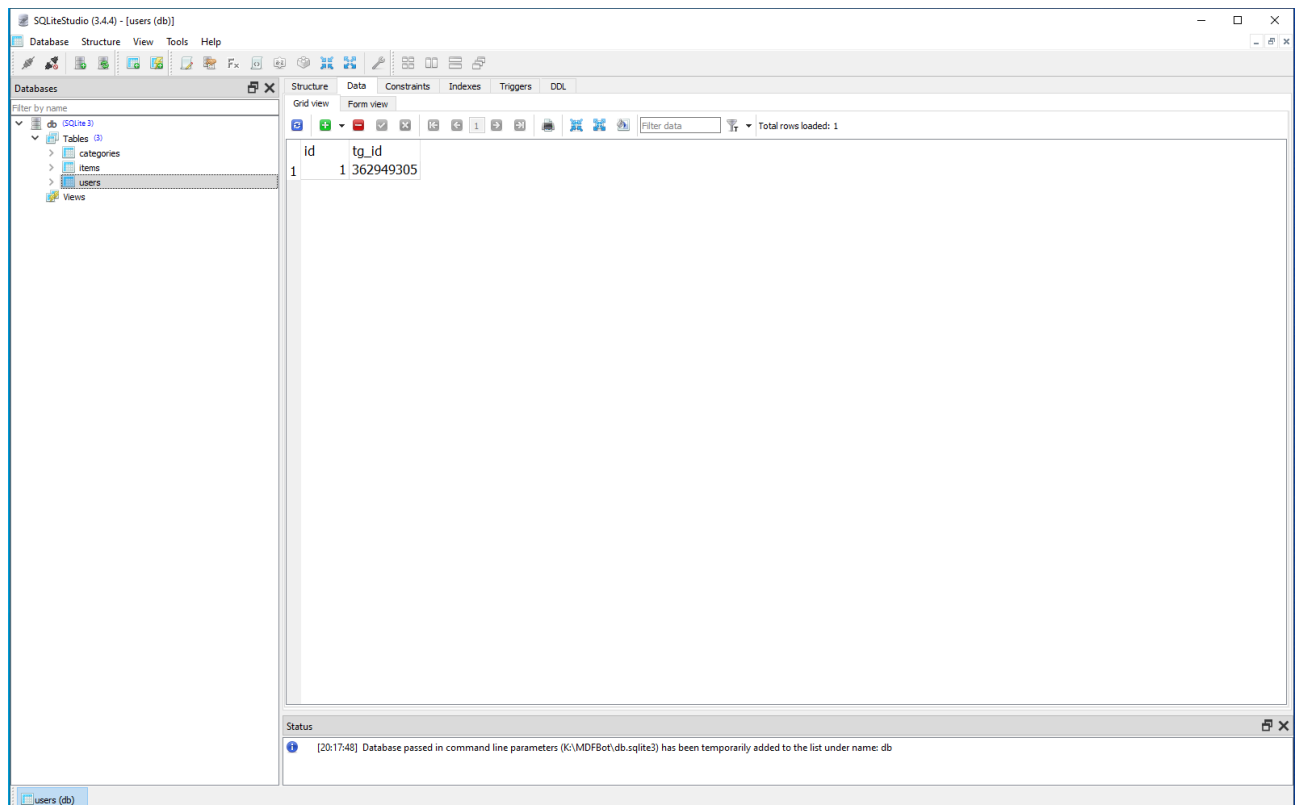
Проверяем работу нашего бота и базы данных, запускаем бота командой `python main.py`. В время разработки бота, нужно постоянно его запускать и проверять, на наличие ошибок, потому что в последствии будет довольно сложно найти ошибку если у вас бот получается большой и загруженный многими функциями:

The screenshot shows the Visual Studio Code editor with the `main.py` file open. The file contains the following code:

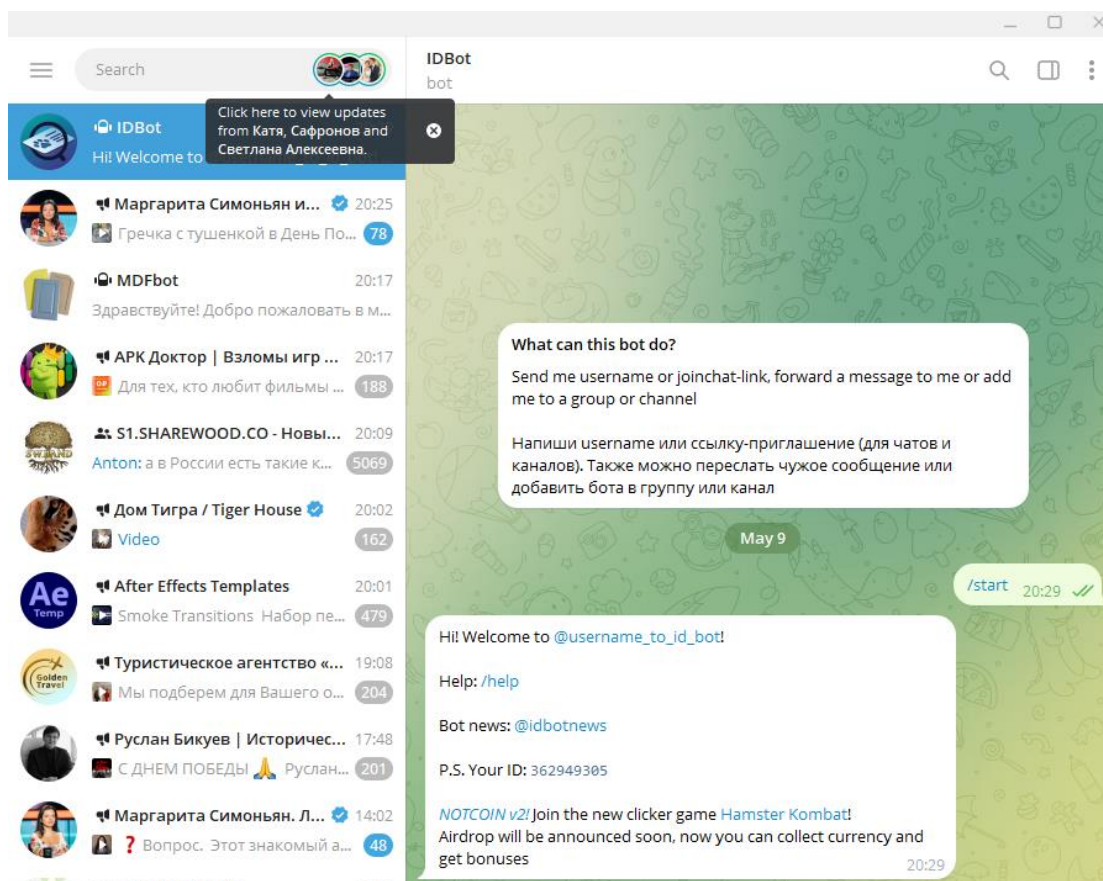
```
1 import asyncio
2 from aiogram import Bot, Dispatcher # импорт 2х классов
3 from app.handlers import router
4 from app.database.models import async_main
5
6
7 async def main():
8     await async_main()
9     bot = Bot(token='7020872196:AAERlgM3z8lt8aMeSNJ-K0xOM796_RHig_A') # в этой переменной храни
10    dp = Dispatcher()
11    dp.include_router(router)
12    await dp.start_polling(bot)
13
14
15 if __name__ == '__main__':
16     try:
17         asyncio.run(main())
18     except KeyboardInterrupt:
19         print("Бот выключен!")
20
21
22
23
```

The terminal window at the bottom shows the following output:

```
(.venv) PS K:\MDFBot> python main.py
Бот выключен!
(.venv) PS K:\MDFBot> python main.py
Бот выключен!
(.venv) PS K:\MDFBot> python main.py
|
```



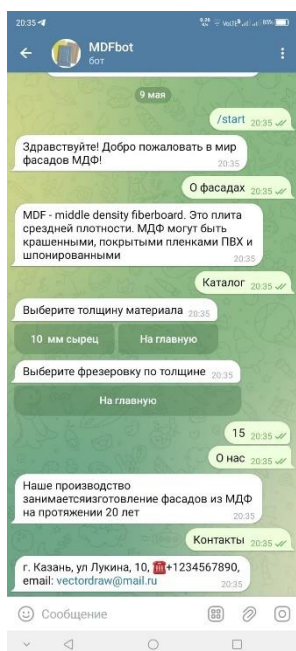
Наш Id добавился, пробуем сравнить его с id полученным другими средствами



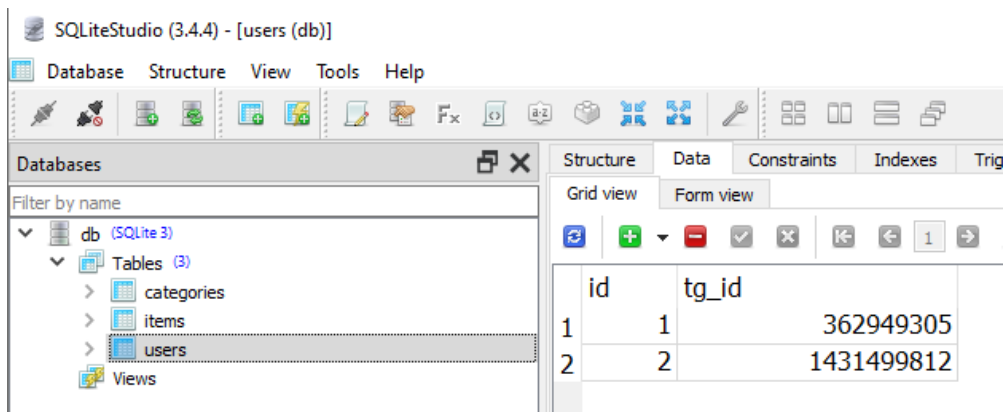


Они совпадают, значит наш бот и база данных работают отлично., данные пользователя записываются

Проверка работы бота с мобильного телефона другого пользователя



Тоже все работает хорошо, проверим базу данных:



Два пользователя, что нам и нужно.

Заполним базу данных исходя из наших потребностей, таблицы categories и items



id	name	description	photo	price	category
1	Палермо	сложная фрезеровка	◆◆◆◆	2500	4
2	Випорья	простая фрезеровка	◆◆◆◆	1500	6
3	Гранд	Простая фрезеровка	◆◆◆◆	1500	4
4	Галерея	Простая фрезеровка	◆◆◆◆	3500	6
5	Интегрированная ручка	ИНТЕГРО	◆◆◆◆	3000	8
6	Рио	Зд	◆◆◆◆	4000	9
7	Виктория	Сложная резервка	◆◆◆◆	2500	7
8	Медина	Сложная резервка	◆◆◆◆	2500	5
9	Орлеан	Сложная резервка	◆◆◆◆	2500	4
10	Римини	Простая фрезеровка	◆◆◆◆	1800	2
11	Тулуза	Простая фрезеровка	◆◆◆◆	3300	7
12	Тулуза	Простая фрезеровка	◆◆◆◆	1800	2
13	Чизано	Сложная резервка	◆◆◆◆	1800	1
14	Кальяри	Сложная фрезеровка	◆◆◆◆	2000	3
15	Гранд	простая фрезеровка	◆◆◆◆	3500	10
16	Прямые углы	Простая фрезеровка	◆◆◆◆	1800	2
17	Марсель	Зд	◆◆◆◆	4000	9

id	name
1	10 мм сырец
2	10 мм лам
3	16 мм сырец
4	16 мм лам
5	16 мм 2х
6	19 мм лам
7	19 мм сырец
8	19 мм 2х
9	22 мм сырец
10	22 мм лам

Теперь нужно связать созданные и заполненные таблицы с кнопками ,путем запроса в базу данных, для этого заходим в requests и прописываем следующее

```
13
14 async def get_categories():
15     async with async_session() as session:
16         return await session.scalars(select(Category))
17
18
19 async def get_category_item(category_id):
20     async with async_session() as session:
21         return await session.scalars(select(Item).where(Item.category == category_id))
22
23
24 async def get_item(item_id):
25     async with async_session() as session:
26         return await session.scalar(select(Item).where(Item.id == item_id))
```

Потом переходим в файл Keyboards и создаем инлайн клавиатуру с названиями толщин МДФ и фрезеровок.

Для этого создаем асинхронную функцию с названием `all_categories?` Сюда ничего не передается, а также в клавиатуру импортируем базу данных



```

5
6 from app.database.requests import get_categories, get_category_item
7

```

```

19 async def categories():
20     all_categories = await get_categories()
21     keyboard = InlineKeyboardBuilder()
22     for category in all_categories:
23         keyboard.add(InlineKeyboardButton(text=category.name, callback_data=f"category_{category.id}")
24     keyboard.add(InlineKeyboardButton(text='На главную', callback_data='to_main'))
25     return keyboard.adjust(2).as_markup()
26
27
28 async def items(category_id):
29     all_items = await get_category_item(category_id)
30     keyboard = InlineKeyboardBuilder()
31     for item in all_items:
32         keyboard.add(InlineKeyboardButton(text=item.name, callback_data=f"item_{item.id}")
33     keyboard.add(InlineKeyboardButton(text='На главную', callback_data='to_main'))
34     return keyboard.adjust(2).as_markup()

```

И после этого начинаем их итерировать категории во всех категориях. Создаем клавиатуру с помощью билдера ,импортируем эту библиотеку

```

4 from aiogram.utils.keyboard import InlineKeyboardBuilder
5

```

Здесь будут ловиться все callback в которых есть category и потом оттуда вытаскивать id. Тут получается своего рода фильтр а также получение нужного id.

Затем прописываем роутерв файле handlers ,чтобы можно было пользоваться каталогом из базы данных.

```

19 @router.message(F.text == 'Каталог')
20 async def catalog(message: Message):
21     await message.answer('Выберите толщину материала', reply_markup=await kb.categories())
22

```

Пользователь нажимает на категорию и мы ловим callback который начинается с category и после этого должны вывестись все фрезеровки с данной толщиной

```

23 @router.callback_query(F.data.startswith('category_'))
24 async def category(callback: CallbackQuery):
25     await callback.answer('Вы выбрали толщину ...')
26     await callback.message.answer('Выберите фрезеровку по толщине',
27     reply_markup=await kb.items(callback.data.split('_')[1]))
28

```

Затем создаем еще один запрос в файле requests



```
19 v async def get_category_item(category_id):
20 v     async with async_session() as session:
21 |         return await session.scalars(select(Item).where(Item.category == category_id))
22
23
```

Чтобы уже из выбранной категории показались все фрезеровки ,которые находятся в базе данных

И после этого делаем еще одну инайн клавиатуру уже с выбранными фрезеровками в данной толщине.

```
28 async def items(category_id):
29 |     all_items = await get_category_item(category_id)
30 |     keyboard = InlineKeyboardBuilder()
31 |     for item in all_items:
32 |         keyboard.add(InlineKeyboardButton(text=item.name, callback_data=f"item_{item.id}"))
33 |     keyboard.add(InlineKeyboardButton(text='На главную', callback_data='to_main'))
34 |     return keyboard.adjust(2).as_markup()
```

Прописываем handler lzk выбора сначала толщины потом фрезеровки:

```
@router.callback_query(F.data.startswith('category_'))
async def category(callback: CallbackQuery):
    await callback.answer('Вы выбрали толщину ...')
    await callback.message.answer('Выберите фрезеровку по толщине',
                                   reply_markup=await kb.items(callback.data.split('_')[1]))
```

И на экран будут выведены все товары которые удовлетворяют выборке.

Для этого прописываем роутеры и requests в соответствующих файлах и снова проверяем



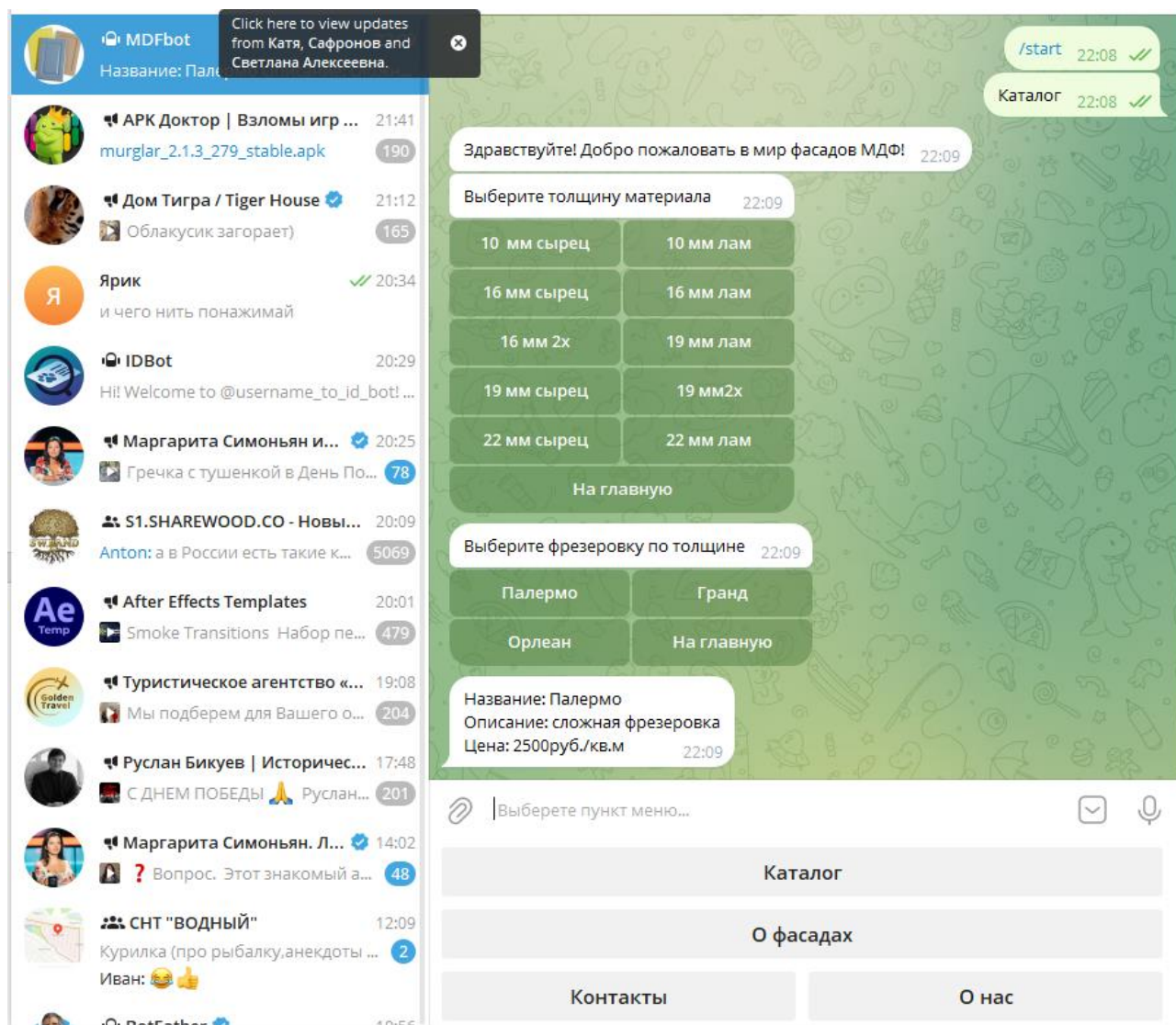
2.2 Результат разработки

Результат разработки представлен на нижеприведенном рисунке. Теперь разберём последовательность шагов работы пользователя со справочно-информационным чат-ботом.

Она выглядит следующим образом:

1. Пользователь вводит команду: /start
2. Чат-бот передаёт команду на сервер.
3. Получает и формирует ответ для пользователя.
4. Присылает готовый ответ пользователю.

Так выглядит работа прайс чат-бота, если в коде не имеется никаких ошибок. В дальнейшем можно усовершенствовать чат бот, добавив дополнительные материалы ,фрезеровки и их комбинации, а также выгрузку изображений той или иной фрезеровки, регистрацию пользователей, заявку на обратный звонок и т.д. Все дополнительные функции ограничиваются только фантазией.



Все работает, выходит цена за квадратный метр определенной фрезеровки на определенной толщине, что нам и требовалось сделать.

На этом вся работа закончена и если нет никаких ошибок, то наш прайс чат-бот должен работать.

Для ясности, ещё хотелось бы уточнить, что информация о боте – это текст, который будет виден в профиле бота, а описание бота – это текст, который пользователи будут видеть в начале диалога с ботом под заголовком «Что может делать этот бот?».



Заключение

Автоматизация работы производственной компании — это передача рутинных, простых задач службам автоматизации. Благодаря этому, принимать заказы, выставлять счета, отправлять товары и многое другое можно на автомате, без участия людей [12].

Чем автоматизация полезна для предприятия:

- Увеличивает прибыль. Автоматизация бизнес-процессов позволяет обслуживать больше клиентов с меньшим количеством сотрудников. Когда рутинная деятельность организована с использованием специальных программ, участие человека необходимо только в нестандартных ситуациях. Прибыль компании растет, а затраты на персонал - нет. Сотрудники могут получать более высокую заработную плату, поскольку им удастся обрабатывать больше клиентов.
- Экономит время. Некоторые задачи, например, разговор с заказчиками о цене на продукции вполне можно заменить автоматизированными системами. Автоматизация этих задач помогает сотрудникам делать больше вещей, требующих творчества или прибыли.
- Повышает эффективность и точность процесса. Есть процессы, где точность очень важна. Например, выставлять счета в службу доставки интернет-магазина. Лучше позволить пользователю выбрать желаемое почтовое отделение из списка, чем заставлять менеджера принимать заказы по телефону. Менеджер будет устал и не прав, посылка будет отправлена не туда, а клиент не получит заказ вовремя из-за ошибки магазина. [18, 19]

Чат-боты – это современные средства, позволяющие с помощью языка программирования, автоматизировать очень многие бизнес-процессы.

Функции чат-бота [19]:

- автоматизировать обработку запросов;
- осуществлять поиск и сбор данных;
- отправлять ответы и подтверждения;



- взаимодействовать с внешними системами;
- инициировать запросы к контрагентам.

В первой главе был более подробно представлен язык программирования и рассмотрены основные модули, функции, конструкции и т.д. для написания кода программы..

Во второй главе была описана практическая реализация сервиса в компании, а именно рассмотрен бизнес-процесс после внедрения чат-бота, был разработан ИТ-проект разработки чат-бота, далее приведена практическая реализация проекта, показаны фрагменты кода и интерфейс.

В ходе выполнения дипломной работы был разработан чат-бот, автоматизирующий работу менеджеров, и освобождающих их от ответов на вопросы о стоимости того или иного изделия, разработанного в мессенджере «Telegram» на языке программирования python.

В результате проведенной работы разработан справочный прайс-бот компании Art_milling на платформе социальной сети Telegram, обеспечивающий оперативный доступ к актуальным ценам на продукцию в зависимости от используемого материала и рисунка фрезеровки.

В процессе выполнения работы в рамках сформулированных задач было проделано следующее:

1. На основании проведенного анализа информационных источников, была выявлена и установлена актуальность использования данного бота в работе компании. Проанализированы существующие способы применения данного бота.

2. Был произведен анализ существующих технологий реализации чат-ботов, а также программных платформ, необходимых при разработке. Обоснован выбор языка программирования, среды разработки, платформы для интеграции разрабатываемой системы.

На основании произведенного анализа были сделаны следующие выводы:



- Для достижения поставленной цели наиболее подходящим языком программирования является Python;
- Наиболее подходящей средой разработки была избрана система VScode;
- Самой подходящей платформой для интеграции конечного продукта разработки является мессенджер Telegram.

3. В соответствии с техническим заданием произведена разработка прайс-бота компании ART_milling на базе мессенджера Telegram.

4. Подготовлено описание разработки, а также предложены дальнейшие улучшения модерации прайс бота, которые можно реализовать в будущем

Таким образом, следует считать, что результаты работы соответствуют всем требованиям технического задания, задачи выполнены, цель достигнута. Работа носит законченный характер



Список использованных источников

1. <https://practicum.yandex.ru/blog/что-такое-chat-bot/>(дата обращения 05.05.2024)
2. <https://tgstat.ru/research-2023>(дата обращения 05.05.2024)
3. Исследование аудитории Telegram // TGSTAT: [сайт]. – URL: <https://tgstat.ru/research-2021> (дата обращения 05.05.2024)
4. Welcome to pyTelegramBotAPI's documentation! // pyTelegramBotAPI Documentation: [сайт]. – URL: <https://pytba.readthedocs.io/en/latest/index.html> (дата обращения 29.04.2024)
5. <https://danshin.ms/Telegram-Bot-From-Scratch/> (дата обращения 08.05.2024)
6. <https://ru.wikipedia.org/wiki/Python> (дата обращения 29.04.2024)
7. <https://docs.python-telegram-bot.org/en/latest/> (дата обращения 29.04.2024)
8. <https://habr.com/ru/articles/543676/> (дата обращения 29.04.2024)
9. <https://github.com/rocketgram/rocketgram> (дата обращения 29.04.2024)
10. https://translated.turbopages.org/proxy_u/en-ru.ru.абсе344с-663daf4с-7d34656d-74722d776562/https/docs.python.org/3/library/idle.html (дата обращения 29.04.2024)
11. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы: межгосударственный стандарт: утв. и введ. в действие Постановлением Государственного комитета СССР по стандартам от 24 марта 1989 г. № 661: дата введ. 1990-01-01: переиздание июнь 2009 г. / разработан и внесен Государственным комитетом СССР по стандартам, Министерством приборостроения, средств автоматизации и систем управления СССР. –



М.: Стандарти-форм, 2009. – Текст : электронный // Электронный фонд правовых и нормативно-технических документов [сайт]. – URL: <https://docs.cntd.ru/document/1200006924> (Дата обращения: 07.05.2024)

12. <https://skillbox.ru/media/code/visual-studio-code-ustanovka-nastroyka-rusifikatsiya-i-spisok-goryachikh-klavish/> (Дата обращения: 07.05.2024)

13. <https://skillbox.ru/media/code/chatboty-v-telegram-na-python-chast-1/#stk-2> (Дата обращения: 07.05.2024)

14. <https://habr.com/ru/companies/otus/articles/771110/> (Дата обращения: 07.05.2024)

15. <https://sqlitestudio.pl> (Дата обращения: 07.05.2024)

16. Боты: информация для разработчиков // Telegram: [сайт]. – URL: <https://tlgrm.ru/docs/bots> (Дата обращения: 07.05.2024)

17. Welcome to aiogram's documentation! // Aiogram: [сайт]. – URL: <https://docs.aiogram.dev/en/latest/> (Дата обращения: 07.05.2024)

18. Что такое автоматизация бизнес-процессов [Электронный ресурс]. – URL: <https://new.unisender.com/ru/support/about/glossary/chto-takoe-avtomatizacija-business/> (Дата обращения: 07.05.2024).

19. Роботы для борьбы с рутиной [Электронный ресурс]. – URL: https://terralink.ru/rpa/?utm_source=google&utm_medium=cpc&utm_term=автоматизация%20бизнес%20процессов&utm_content=automatizatoin_message2&utm_campaign=RPA_RU_LP_RPA_TRG_GA&gclid=CjwKC Ajw8J32BRBCEiwApQEKgeKRmP9f9lunlC5mY4MKg0hddoZFakNzSA GV8l-VRxZwYzWmYewZRhoCSkgQAvD_BwE (Дата обращения: 07.05.2024).



Приложение

Для работы и проектирования данного бота необходимо:

ПО:

- Python v3.12
- VScode
- SQLite

Библиотеки :

- Aiogram
- SQLAlchemy
- aisqlite