МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Благовещенский государственный педагогический университет»

Физико-математический факультет

Кафедра информатики и методики преподавания информатики

КУРСОВАЯ РАБОТА

на тему: Разработка Telegram-бота для обработки заказов

по дисциплине: Современные технологии программирования

Исполнитель:

Студент ФМФ, группы 3 «Б» \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ И.С. Жуковец

дата подпись

Руководитель:

ст. преподаватель А.А. Ягелло

дата подпись

Защита состоялась « »\_\_\_\_\_\_\_\_ 202 г. Оценка

Благовещенск 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc152160214)

[1 Описание предметной области 5](#_Toc152160215)

[1.1 Основные понятия 5](#_Toc152160216)

[1**.**2 Выбор СУБД 6](#_Toc152160217)

[1.3 Выбор среды разработки 8](#_Toc152160218)

[1.4 Асинхронность 9](#_Toc152160219)

[2 Проектирование Telegram-бота 12](#_Toc152160220)

[2.1 Описание функционала 12](#_Toc152160221)

[2.1 Первоначальная настройка 12](#_Toc152160222)

[2.2 Первоначальная настройка 15](#_Toc152160223)

[2.3 Cоздание базы данных 17](#_Toc152160224)

[3 Создание Telegtam-бота 21](#_Toc152160225)

[3.1 Библиотеки 21](#_Toc152160226)

[3.2 Хэндлеры 22](#_Toc152160227)

[3.3 Состояния 23](#_Toc152160228)

[3.4 Функции 24](#_Toc152160229)

[3.5 Обзор функционала 26](#_Toc152160230)

[ЗАКЛЮЧЕНИЕ 29](#_Toc152160231)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc152160232)

[ПРИЛОЖЕНИЯ 31](#_Toc152160233)

# **ВВЕДЕНИЕ**

**В настоящее время люди активно используют ботов в мессенджере «Telegram», так как они просты в использовании и всегда находятся под рукой, поэтому к ним можно быстро получить доступ без необходимости устанавливать отдельное приложение. Они начали появляться в 2015 году и с того времени развиваются всё больше и могут выполнять множество различных функций. Даже начинающие программисты могут освоить технологии создания Telegram-ботов. В том числе многие компании и предприниматели используют мессенджеры для коммуникации с клиентами и приёма заказов. Простота и удобство оформления заказов через мессенджер повышает рейтинг организации и делает её более популярной среди пользователей.**

**Актуальность выбранной темы обусловлена тем, что боты в сфере бизнеса используются всё чаще, в настоящее время обычный пользователь предпочтет сделать заказ за пару кликов в мессенджере, чем устанавливать отдельное приложение или делать звонок. Подобные боты способны облегчить и автоматизировать работу, они делают процесс заказов удобным и для пользователя, и для производителя.**

**Клиент сразу видит все возможности, нет необходимости разбираться в новом интерфейсе, так же человек может делать выбор, не испытывая дискомфорта при общении с производителем напрямую, а производитель в свою очередь — экономит время на переписке с пользователем и оформлении заказа. Соответственно, такой инструмент поможет оптимизировать процесс приема заказов и улучшить качество обслуживания клиентов.**

**В связи с этим была сформулирована цель работы: Создание Telegram-бота для обработки заказов средствами языка программирования «Python».**

**Для достижения цели были поставлены следующие задачи:**

* **провести анализ предметной области, отметить преимущества Telegram-ботов;**
* **выбрать язык программирования для разработки Telegram-бота;**
* **выбрать среду разработки и СУБД, рассмотреть их преимущества и недостатки;**
* **изучить все варианты существующих библиотек для написания современных Telegram-ботов;**
* **разработать и реализовать алгоритмы работы.**

**Предметом исследования является процесс создания удобного функционала для Telegram-бота.**

**Объектом исследования является Telegram-бот для обработки заказов в пекарне.**

# **1 Описание предметной области**

**1.1 Основные понятия**

**Telegram является популярной кроссплатформенной системой мгновенного обмена сообщениями. С 2015 года в данный мессенджер был добавлен собственный API (Application Programming Interface) для разработки пользователями своих ботов.**

API так же называется интерфейсом программирования приложений*.* Это совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой. Это помогает увеличить функционал своего продукта, а следовательно, сделать приложение более удобным для пользователей.

Бот – это специальный аккаунт, позволяющий пользователям выполнять различные задачи прямо в мессенджере. Среди преимуществ Telegram-ботов можно выделить следующее [3]:

* **взаимодействие между пользователем и ботом происходит посредством обмена сообщений по принципу «вопрос-ответ», так что использование ботов будет интуитивно понятно не только продвинутым пользователям;**
* **круглосуточная работа;**
* **отсутствие необходимости в установке дополнительного программного обеспечения;**
* **быстрое получение ответа;**
* **для разработчиков боты стали способом автоматизации работы и удержания клиентов;**
* **разнообразие ботов позволяет найти бота под любой запрос, от развлекательных ботов до новостных.**

**Разработка Telegram-бота осуществлялась на языке программирования «Python». Этот язык программирования отличается простотой в использовании, понятным синтаксисом, а также обладает богатыми библиотеками, которые значительно упрощают разработку ботов. «Python» является наиболее подходящим вариантом для начинающих разработчиков.**

**Для бота в сфере обработки заказов необходимо внедрить базу данных, где будет храниться вся информация. СУБД является совокупностью программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.**

## **1.2 Выбор СУБД**

**При выборе СУБД для создания Telegram-бота обычно рассматривается три варианта: SQLite, MySQL, PostgreSQL. Их краткое сравнение представлено в таблице 1.**

**Таблица 1** – **Сравнение SQLite, MySQL, PostgreSQL**

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **SQLite** | **MySQL** | **PostgreSQL** |
| **1** | **2** | **3** | **4** |
| **Основное преимущество** | **Мощность, встраиваемость** | **Популярность** | **Гибкость, продвинутость** |
| **Поддерживаемые**  **типы данных (количество)** | **5** | **21** | **36** |
| **Есть многопользовательский доступ?** | – | **+** | **+** |
| **Простота использования** | **+** | – | – |

**Продолжение таблицы 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |
| **Где используется:** | **Встроенные приложения, система доступа к дисковой памяти, тестирование** | **Веб-сайты и приложения, специфичный продукт с необходимостью широкой настройки** | **Случаи, где БД должна выполнять сложные процедуры.**  **Случаи, когда всю базу необходимо переносить** |

**По итогу сравнения качестве базы данных выла выбрана SQLite, которая лучше всего подходила к поставленной задаче и была проста в освоении. Одним из основных критериев стало то, что SQLite не имеет сервера. Это значит, что все данные программное обеспечение хранит на одном устройстве. СУБД встраивается в приложение и работает как его составная часть. Если установить на компьютер программу, использующую SQLite, то база данных тоже будет храниться на нем же. Формат базы** – **один текстовый файл, который можно прочитать на любой платформе. Такой подход повышает производительность и скорость работы [5]. Так же среди преимуществ можно выделить следующее:**

* Эта СУБД, представленная модулем sqlite3 входит в стандартную библиотеку языка. Для работы с SQLite из Python-кода достаточно лишь импортировать модуль sqlite3 и приступить к его использованию в программе [8];
* **Эффективность: СУБД использует минимальные ресурсы системы;**
* **Высокая скорость. Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами;**
* **Минимализм. Создатели SQLite пользуются принципом «минимального полного набора». Из всех возможностей SQL в ней есть наиболее нужные. Поэтому SQLite отличают малый размер, простота решений и легкость администрирования. Для повышения базовой функциональности можно использовать стороннее программное обеспечение и расширения.**

**Среди недостатков SQLite обычно выделяют два основных:**

* **Невозможность дополнительной настройки: SQLite нельзя сделать более производительной, изменив настройки**
* **Отсутствие пользовательского управления: продвинутые БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет.**

**Но так как в рамках данной работы эти минусы оказались несущественными, то была выбрана именно эта СУБД.**

## **1.3 Выбор среды разработки**

**В рамках данной курсовой работы, написание всего программного кода бота осуществляется в программе «PyCharm». «PyCharm»** – **это среда программирования для языка «Python», или IDE. Средами называют программы, в которых можно писать, запускать и отлаживать код, устанавливать новые расширения и дополнительные модули [9].**

**Данная среда разработки пользуется популярностью по следующим причинам:**

* **Удобство: редактор кода содержит большое количество функций, которые упрощают работу, например автодополнение кода и проверку синтаксиса;**
* **Встроенный отладчик позволяет быстро находишь ошибку в коде и устранить её;**
* **Сообщество пользователей регулярно создаёт уроки и блоги с обучением по «PyCharm», что позволяет быстро освоить функционал;**
* **Возможность интеграции с такими инструментами, как базы данных, тестировщики, отладчиками и т.д.;**
* **Кроссплатформенность.**

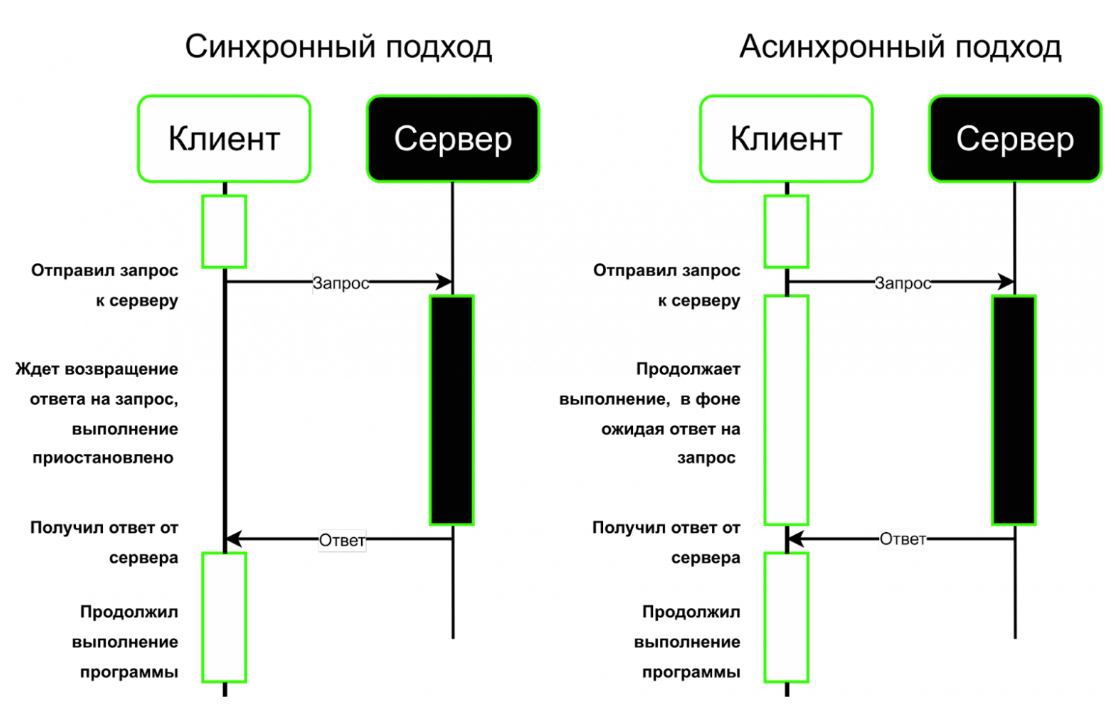
**Среди основных недостатков выделяют:**

* **Стоимость. «PyCharm» является платным инструментом, если пользователь хочет использовать полный набор функций. Однако существует и бесплатная версия «PyCharm CE» с ограниченным функционалом;**
* **Ресурсоёмкость;**
* **Сложность в освоении для начинающих.**

**Преимущества этой среды разработки перекрывают недостатки, даже сложность в освоении не является существенным минусом, ведь из-за поддержки сообщества и большого количества видеоуроков даже начинающий пользователь сможет освоить «PyCharm».**

## **1.4 Асинхронность**

**Асинхронное программирование** – **это подход к написанию кода, при котором несколько задач выполняются параллельно и независимо друг от друга. Следовательно, созданным Telegram-ботом сможет пользоваться несколько человек одновременно, и для каждого клиента у бота своё состояние, разные сессии. Разница синхронного и асинхронного подходов представлена на рисунке 1.**



**Рисунок 1** – **Принцип синхронного и асинхронного подходов**

**Слева изображён синхронный подход. Синхронная программа отправляет запрос. Пока она ожидает результатов выполнения этого запроса, она не может выполнять другую полезную работу. Программа сможет продолжить работу только тогда, когда получит ответ от внешнего сервера, так как дальнейший ход ее действий зависит от внешних данных.**

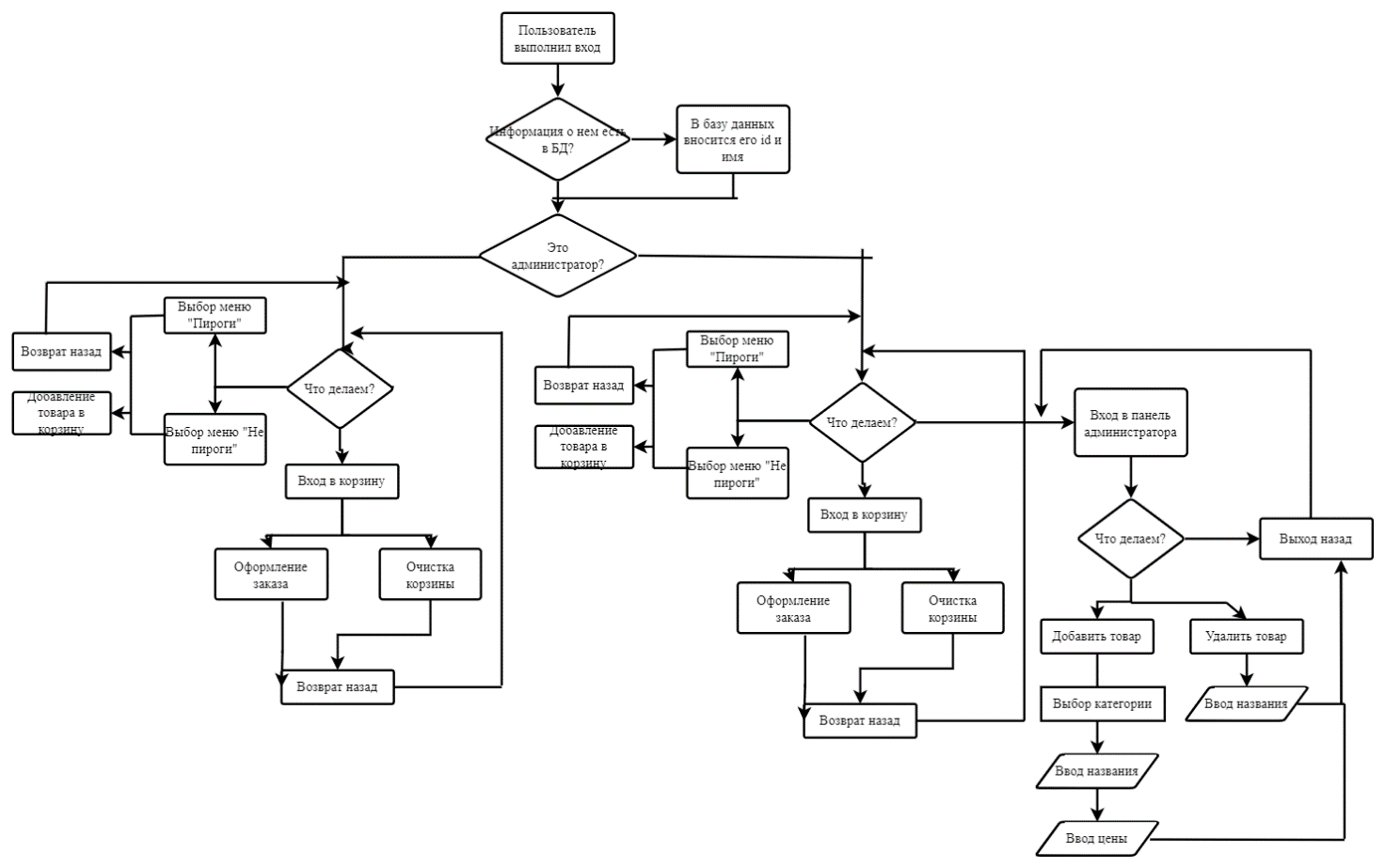
**Справа изображен асинхронный подход. Асинхронная программа сначала ведет себя также как и синхронная** – **она посылает запрос на сервер. Но затем ее поведение кардинально отличается** – **она не блокируется, ожидая результатов, а делает какие-то другие действия, которые не требуют ответа внешнего сервера. Только после того, как сервером будет возвращён ответ, она сможет вернуться к дальнейшей обработке полученных данных [7].**

**Асинхронное программирование нацелено на то, чтобы программа как можно меньше времени бездействовала в ожидании какого-то внешнего события. Для написания асинхронного кода с использованием синтаксиса async/await создан модуль «asyncio», async используется для объявления асинхронной функции. Это означает, что функция будет возвращать объект coroutine, который можно выполнить асинхронно. Await используется внутри асинхронной функции для ожидания результата другой асинхронной операции.**

# **2 Проектирование Telegram-бота**

## **2.1 Описание функционала**

**Перед началом создания определимся, какие функции будет выполнять бот. Основная его функция – приём заказов от клиентов. В базе данных будет храниться информация о клиентах, о товарах в меню и о товарах в корзине у пользователей. Администратору в свою очередь доступна специальная панель, с которой он может удалять и добавлять продукты в меню. На рисунке 2. представлена диаграмма, иллюстрирующая принцип работы бота.**

****

**Рисунок 2** – **Диаграмма, иллюстрирующая принцип работы**

## **2.1 Первоначальная настройка**

**Процесс создания** **Telegram**-**бота не так сложен, для начала нужно открыть само приложение «Telegram», этот этап можно выполнять на любой удобной платформе, в том числе и на телефоне. В поисковой строке необходимо набрать «botfather», вы увидите его первым в списке, рядом с названием вы так же увидите галочку верификации, как показано на рисунке 3.**

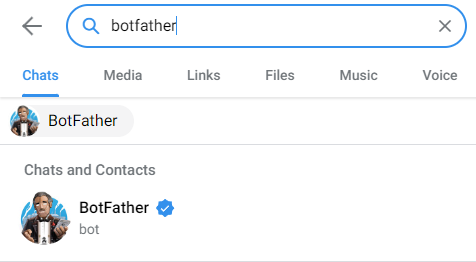


Рисунок 3 – Поиск бота botfather

Botfather – это «бот над всеми ботами», именно с его помощью пользователи создают новых ботов и могут управлять существующими. «Общение» с ботом осуществляется посредством ввода сообщений-команд. Взаимодействие начинается с команды /start, как на рисунке 4. Эта команда необходима для запуска сервиса.

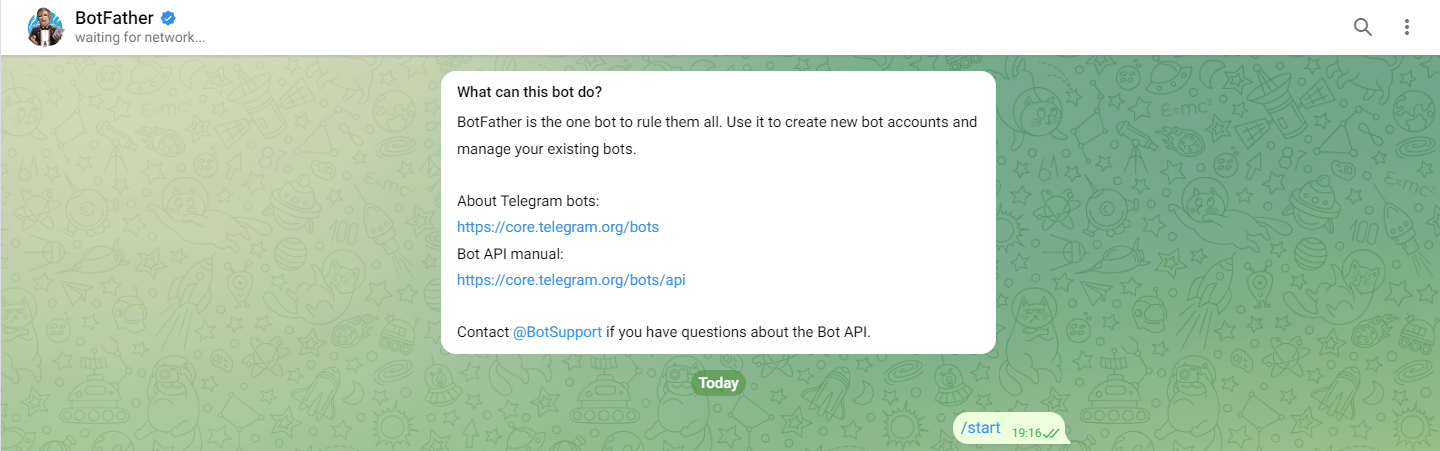


Рисунок 4 – Бот Botfather

Далее система выведет список доступных команд, представленный на рисунке 5, среди которых есть настройка, создание и изменение ботов, создание игры и так далее. Чтобы создать бота, нужно ввести команду /newbot.

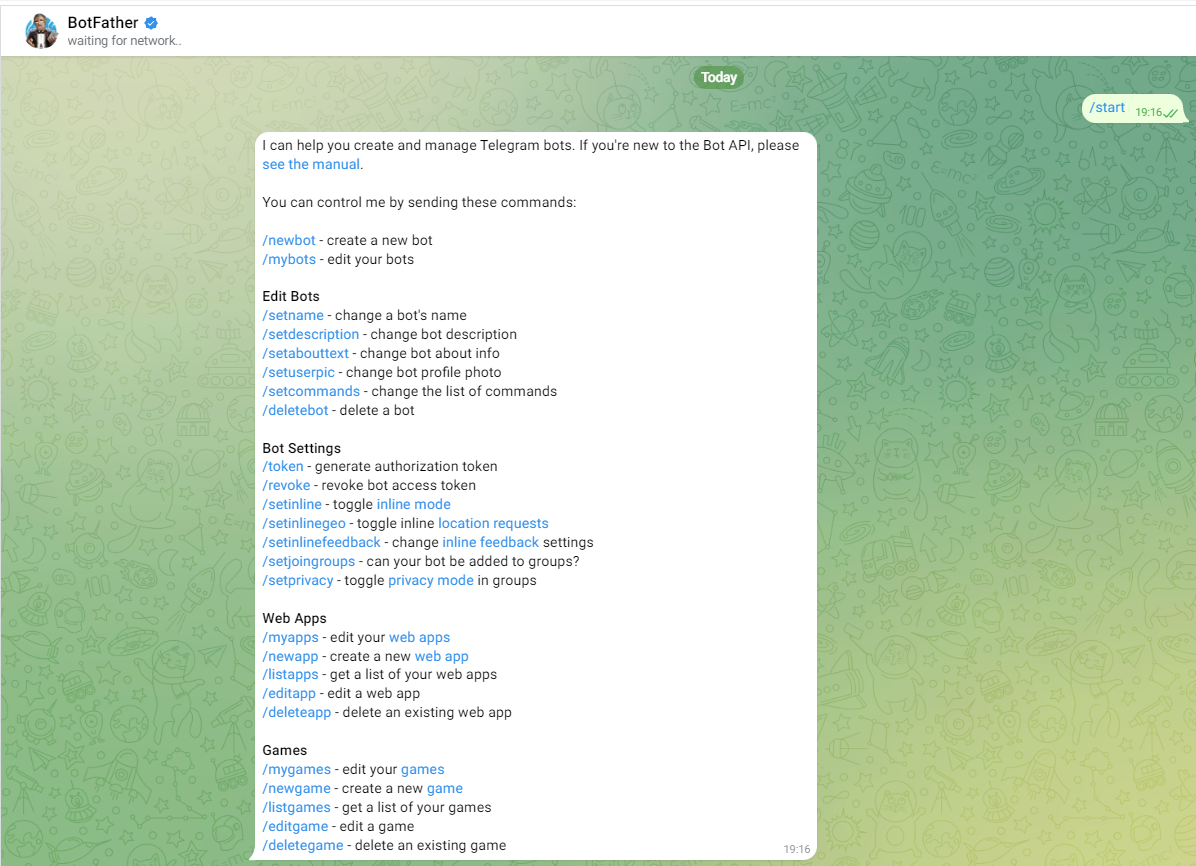


Рисунок 5 – Список команд Botfather

После этого, что показано на рисунке 6, система попросит указать имя бота, то, под каким названием он будет отображаться в списке чата и как его будут видеть пользователи.

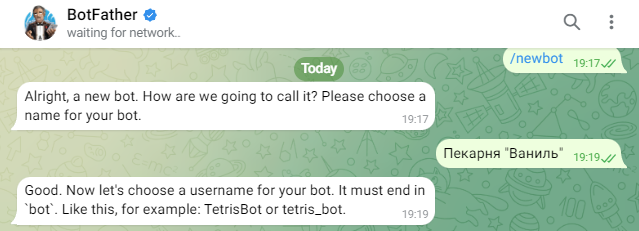


Рисунок 6 – Сообщение о создании нового бота

Далее необходимо выбрать username, то, какой будет прямая ссылка на бота в рамках телеграмм. Botfather сразу уведомляет пользователя о том, что username должен оканчиваться слово bot. Если введённый username уже существует, бот выдаст сообщение об ошибке и предложит ввести другой вариант. Последним этапом бот выдаст сообщение, показанное на рисунке 7, где выдаст ссылку на созданного бота и его API.

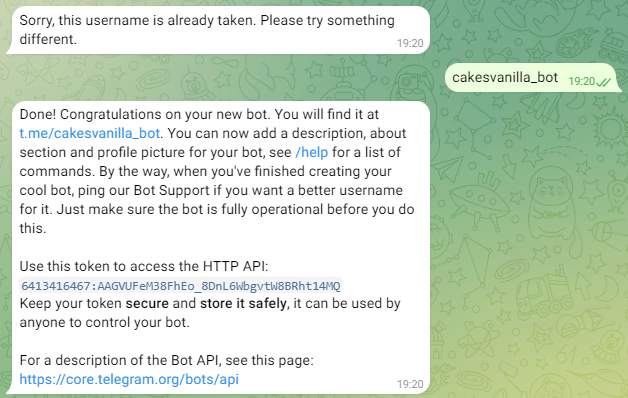


Рисунок 7 – Завершающий этап создания в botfather

В заключении Botfather присылает сообщение, в котором говорится об успешном создании бота, а также выдаётся справочная информация о том, как редактировать бота. Вместе с этим в сообщении пользователь получает секретный token бота и ссылку на него. Token нужен для того, чтобы скрипт, который будет отвечать за работу бота, мог получить к нему доступ

## 2.2 Первоначальная настройка

Сразу после создания бота он будет пустым. Пока что он не только не реагирует на команды, но и не имеет описания. Его необходимо настроить и выполнить работу над элементами, которые будут видеть пользователи, так как в боте для продажи услуг визуальное оформление играет большую роль. Нам необходимо указать название, справочная информация и описание того, для чего предназначен бот. Для этого пользователь выбирает необходимые команды в «Botfather», среди которых:

* /setuserpic – команда для установки фото профиля;
* /setdescription – команда для ввода описания, которое видно в профиле бота. Описание не должно превышать 120 символов;
* /setabouttext – команда для ввода текста, который появляется в блоке «What can this bot do?».

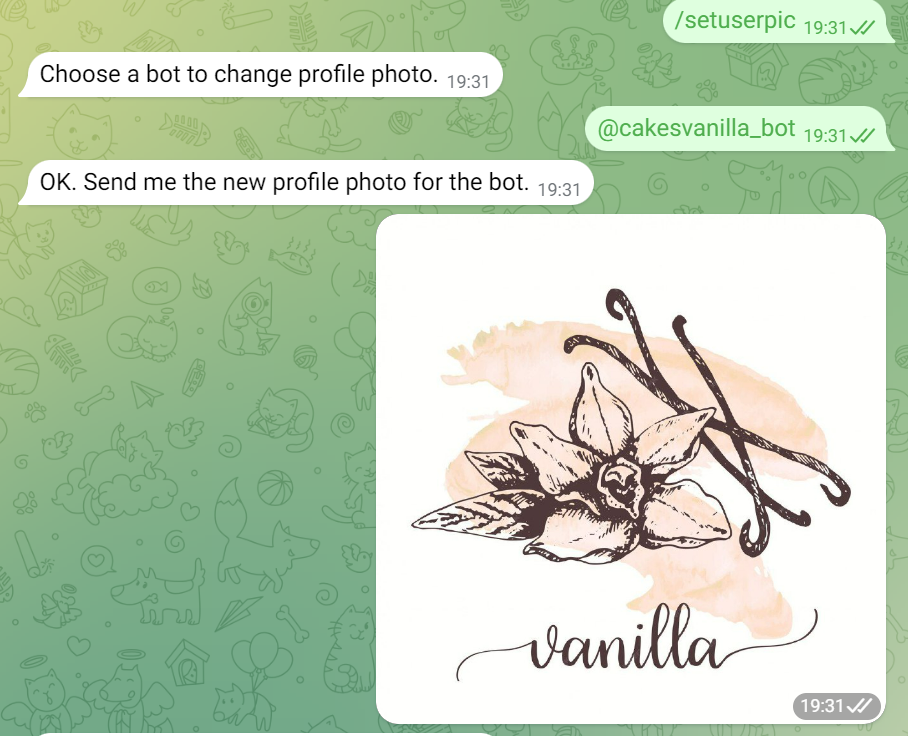


Рисунок 8 – Настройка фото профиля

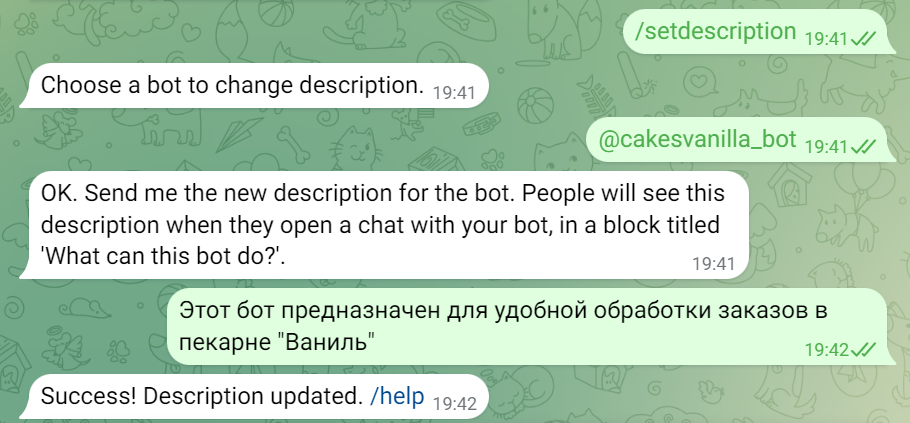


Рисунок 9 – Настройка описания возможностей бота

## 2.3 Cоздание базы данных

Созданный бот должен где-то хранить информацию о товарах, пользователях и корзине. Для начала создадим файл под названием bakery.db, позже, благодаря тому, что эта база данных, представленная модулем sqlite3, входит в стандартную библиотеку языка «Python», мы будем ссылаться на этот файл в коде.

Сначала создадим таблицу «Products», где будут отображаться все товары. Она будет состоять из уникального номера, названия, цены и «флага» под названием «is\_pirogi». В данном боте будет две категории с товарами – «Пироги» и «Не пироги». Вместо того, чтобы делать две отдельные таблицы с категориями – был создан такой столбец. Стоит учесть, что SQLite не имеет значений типа Boolean, но данная проблема решается нулём и единицей. Следовательно, столбец «is\_pirogi» показывает, к какой категории относится товар. Это так же помогает сократить количество функций в два раза. Структура таблицы представлена на рисунке 10.

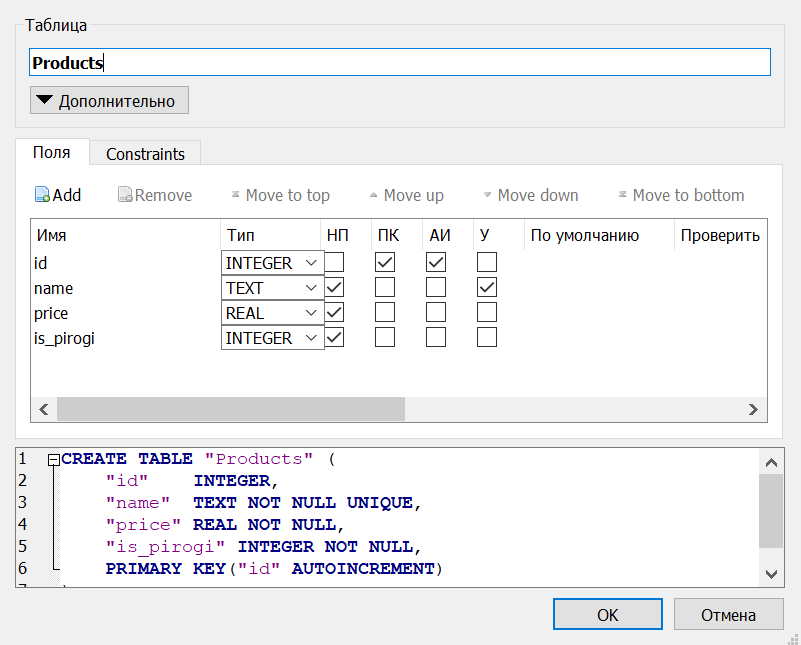


Рисунок 10 – Структура таблицы «Products»

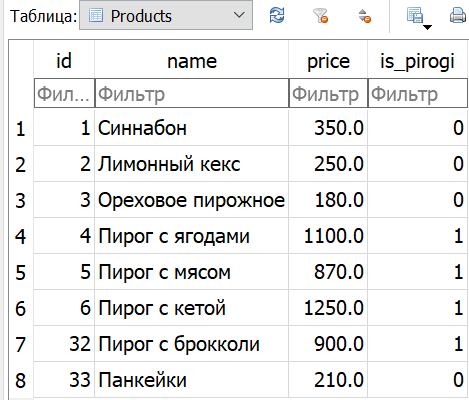


Рисунок 11 – Содержимое таблицы «Products»

Далее создадим таблицу «Users», в которой будут храниться уникальные номера и имена пользователей в рамках Telegram. Структура таблицы представлена на рисунке 12.

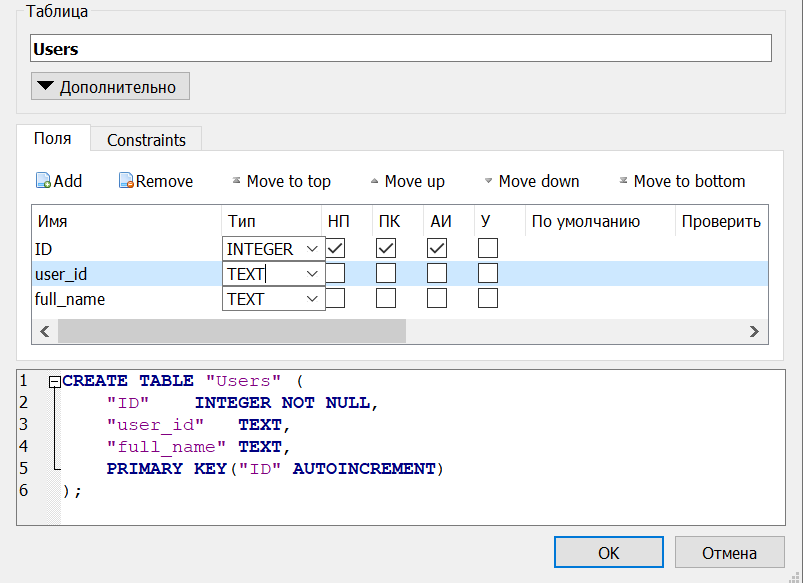


Рисунок 12 – Структура таблицы «Users»

Уникальные номера нужны для того, чтобы никто из посторонних пользователей не мог получить доступ к панели администратора. Эта панель доступна только в том случае, если бот видит, что сообщение отправлено именно администратором, он получает номер пользователя и сравнивает его с тем, что указан в коде. Администраторов может быть несколько. Поле «user\_id» также используется при приветственном сообщении. Когда бот видит, что информации о пользователе ещё нет в базе данных – он заносит его уникальный номер и имя в таблицу и выводит приветственное сообщение. У пользователей, которые есть в базе данных – сообщение отличается, что видно на рисунке 13.

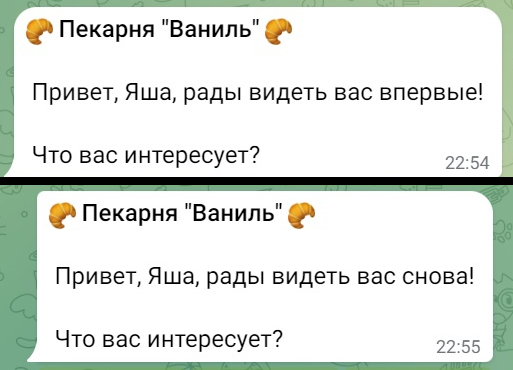


Рисунок 13 – Пример приветственных сообщений

Создадим последнюю таблицы «Cart». Структура таблицы представлена на рисунке 14.

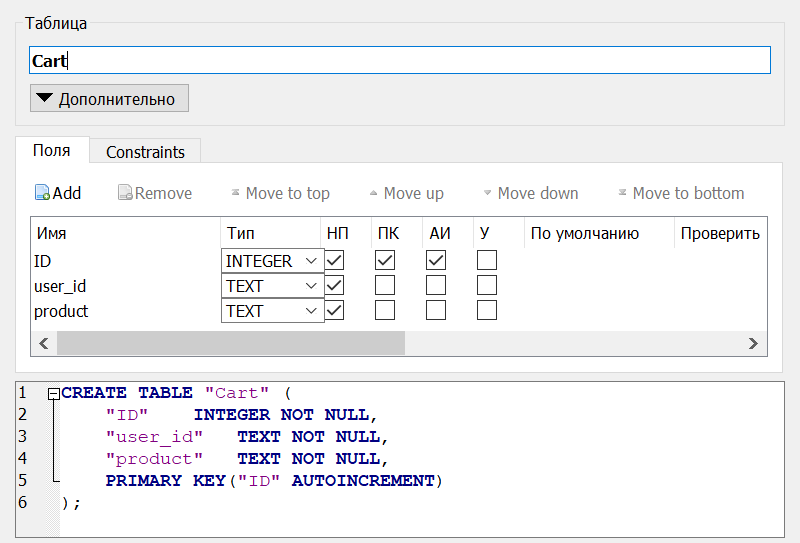


Рисунок 14 – Структура таблицы «Cart»

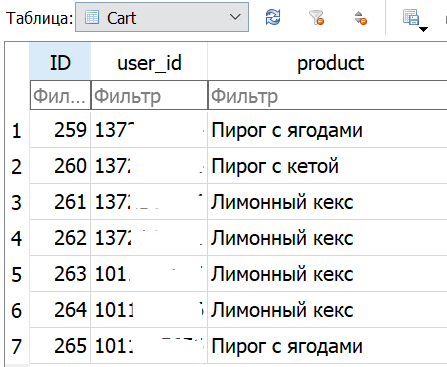


Рисунок 15 – Содержимое таблицы «Cart»

В таблице «Cart» хранится информация о клиентах и товарах, которые находятся у них в корзинах. После того, как пользователь совершает заказ, и его корзина очищается – данные удаляются из таблицы.

# 3 **Создание Telegtam-бота**

## 3.1 Библиотеки

Для создания ботов для «Telegram» на «Python» существует несколько библиотек, которые предоставляют различные функции и возможности. Наиболее популярные представлены в таблице 2.

Таблица 2 – Описание библиотек

|  |  |
| --- | --- |
| Название библиотеки | Описание |
| 1 | 2 |
| python-telegram-bot | Простая в использовании библиотека, которая предоставляет все необходимые инструменты для создания ботов на «Python». Она поддерживает большое количество API, имеет обширную документацию. |
| telebot | Обладает хорошей функциональностью и простотой в использовании. Отличительной особенностью telebot является поддержка встроенного логгера. |
| aiogram | Эта библиотека предоставляет возможность создания асинхронных ботов на «Python». Она основана на asyncio и является отличным выбором для разработчиков, которые предпочитают работать с асинхронными функциями. |

Проанализировав представленные варианты, была выбрана библиотека «aiogram», так как создаваемый бот является асинхронным, а среди пользователей этой библиотеки можно найти много обучающих материалов.

Стоит учитывать, что «aiogram» – асинхронная библиотека, поэтому хэндлеры тоже должны быть асинхронными, а перед вызовами методов API нужно ставить ключевое слово await, т.к. эти вызовы возвращают корутины – блоки кода, которые работают асинхронно, то есть по очереди [6].

Перед написанием программного кода были импортированы библиотеки, указанные в таблице 3.

Таблица 3 – Список импортированных библиотек

|  |  |
| --- | --- |
| Библиотека | Предназначение |
| 1 | 2 |
| asyncio | Позволяет выполнять функции асинхронно |
| sqlite3 | Библиотека базы данных |
| logging | Логирование работы программы |
| aiogram | Библиотеки и модули для работы с телеграм |

Библиотека «aiogram» импортировалась не полностью, а только необходимые функции, чтобы уменьшить нагрузку.

## 3.2 Хэндлеры

Перед тем, как приступить к написанию основной части кода, необходимо разобрать, что такое хэндлеры, так как далее это понятие будет часто использоваться. Хендлеры - асинхронные функции, которые вызываются при получении определенного события от пользователя [2]. На рисунке 16 показан пример использования хэндлера.

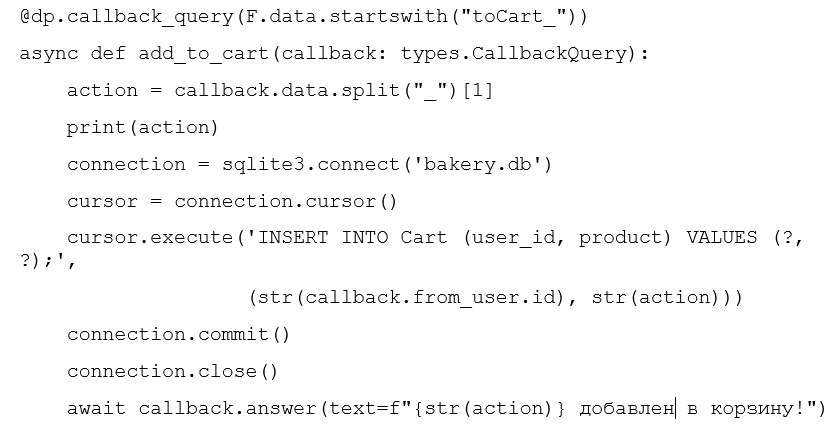


Рисунок 16 – Пример использования хэндлера

В данном случае хэндлер «отлавливает», реагирует на определенное событие, начинающееся с «ToCart\_». После того, как событие произошло, то есть пользователь положил товар в корзину – срабатывает соответствующий ряд команд.

## 3.3 Состояния

У Telegram-ботов присутствует система диалогов. В разрабатываемом боте она используется, когда администратор добавляет в корзину новый товар, указывает его название и цену. Рассмотрим пошаговый ввод данных, где в начале бот находится в режиме ожидания определённой информации от администратора, а далее на каждом этапе проверяет вводимые данные. Схема процесса представлена на рисунке 17.

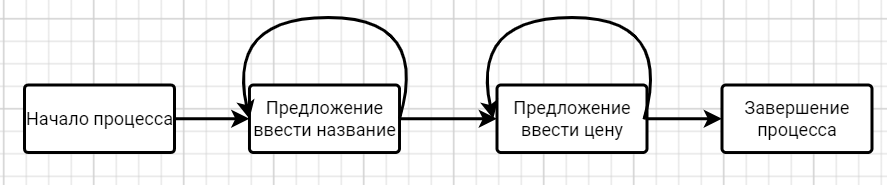


Рисунок 17 – Схема работы системы диалогов

Прямыми стрелками обозначается процесс перехода по шагам без ошибок. Стрелки в виде кривых означают сохранение текущего состояния и ожидание повторного ввода [6]. На рисунке 18 представлен класс «AdminStates», внутри которого созданы переменные и им присвоены экземпляры класса «State».

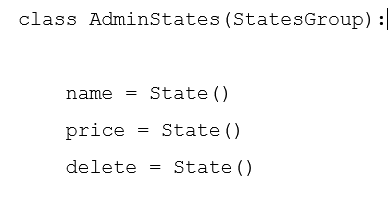


Рисунок 18 – Класс состояний

## 3.4 Функции

После импорта необходимых библиотек и подключения токена бота можно приступать к написанию основного кода. Важным его элементом будут функции. Функцию можно определить при помощи ключевого слова def, за которым должно следовать название функции и список её формальных параметров в круглых скобках. Сама по себе функция является последовательностью инструкций, возвращающая некое значение [1]. В таблице 4 описаны основные функции, используемые Telegram-ботом.

Таблица 4 – Описание функций

|  |  |
| --- | --- |
| Функция | Назначение |
| 1 | 2 |
| get\_keyboard\_cart() | Более простой способ размещения кнопок используется в корзине, так как в этот раздел новые кнопки не добавляются динамически. |

Продолжение таблицы 4

|  |  |
| --- | --- |
| 1 | 2 |
| get\_keyboard\_products  (array) | Функция для размещения кнопок по две в ряд в меню. Она нужна. т.к. продукты в список добавляются динамически и положение кнопок так же динамически изменяется. |
| cmd\_start(message: types.Message) | Функция обработки команды /start. Здесь создаются кнопки для первого сообщения. Определяется, вошел администратор или обычный пользователь. Если пользователь новый – информация о нём заносится в базу данных |
| menu\_ne\_pirogi(  callback: types.CallbackQuery) | Появление меню после нажатия кнопки «Не пироги» или «Пироги». Из базы данных выводится информация о продуктах, для которых значение поля «is pirogi» - ноль или единица. Всё зависит от того, нажатие на какую кнопку «отловил» хэндлер |
| cart(callback: types.CallbackQuery) | Функция для корзины. Из базы данных берется информация с учетом уникального номера пользователя. Через цикл выводится список продуктов и их цен. |
| add\_to\_cart(callback: types.CallbackQuery) | Функция добавления в корзину. После того, как хэндлер «отлавливает» событие, начинающееся с «ToCart\_», это событие разделяется и мы получает часть с названием товара. В базу данных добавляется соответствующая информация. |
| add\_to\_menu(callback: types.CallbackQuery, state: FSMContext): | Функция для того, чтобы администратор добавлял в список новые товары. Устанавливается состояние ввода имени нового товара. |

Продолжение таблицы 4

|  |  |
| --- | --- |
| 1 | 2 |
| new\_name(message: types.Message,  state: FSMContext) | Функция для того, чтобы администратор указывал название. Сообщения «отлавливаются» только из состояния name. |
| new\_price(message: types.Message,  state: FSMContext) | Функция для того, чтобы администратор указывал цену. Сообщения «отлавливаются» только из состояния price. Здесь же происходит проверка типов данных, если они соответствуют необходимым – информация добавляется в базу данных. |
| deleter(callback: types.CallbackQuery, state: FSMContext) | Функция для получения названия товара, который нужно удалить. Здесь устанавливается состояние удаления. |
| deletebyname(message: types.Message,  state: FSMContext) | Функция, удаляющая товар из базы данных, если введённое название присутствует в ней. |
| make\_order(callback: types.CallbackQuery) | Функция создания заказа. Если он создан - администратору отправляется сообщение с указанием имени пользователя и товаров из его корзины |

## 3.5 Обзор функционала

После того, как программный код написан и бот готов к эксплуатации, можно протестировать его. На рисунке 19 представлены доступные функции. Кнопка «Админ-панель» доступна только администраторам, обычный пользователь её не видит. Но даже если при ошибке кнопка отобразиться – в коде это учтено и при проверке уникального номера к доступу к функциям администратора будет отказано. При нажатии кнопки «Меню» клиент увидит картинку со списком блюд.

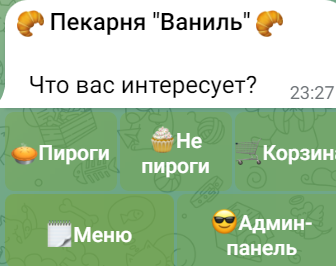


Рисунок 19 – Доступный функционал

На рисунке 20 показано, как выглядит меню раздела «Пироги», тот же принцип используется в меню «Не пироги». Кнопки и названия блюд добавляются, когда администратор редактирует этот список через панель администратора или через базу данных. При нажатии блюдо добавляется в корзину, а пользователь видит соответствующее сообщение.

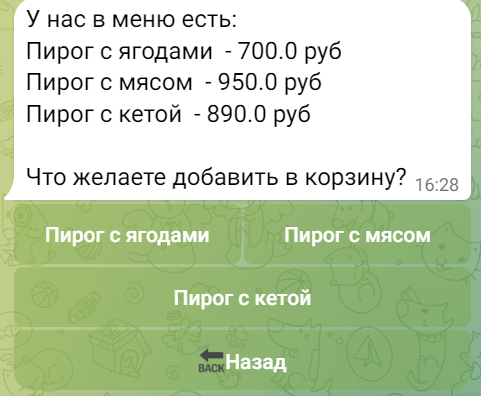


Рисунок 20 – Меню

На рисунке 21 показано, что после оформления заказа через корзину администратору приходит сообщение с указанием имени пользователя для обратной связи и списка товаров, которые он заказал.

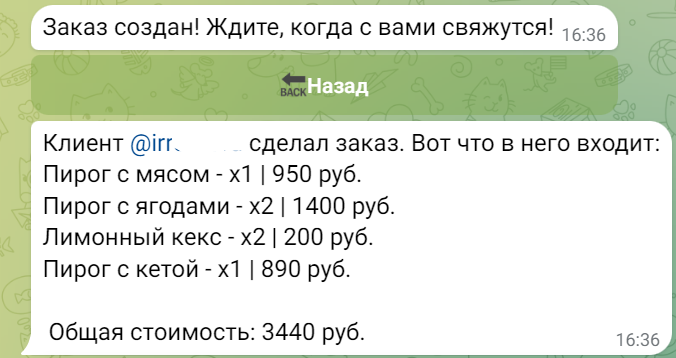


Рисунок 21 – Сообщение администратору

На рисунке 22 показаны функции, доступные с панели администратора. При добавлении бот поэтапно запросит название и цену товара. При удалении – попросит ввести название с учетом регистра.

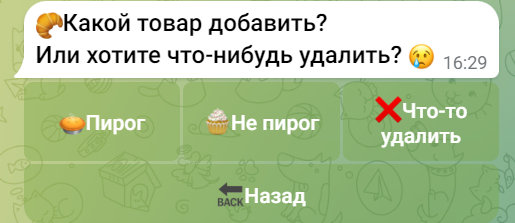


Рисунок 22 – Панель администратора

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы на платформе мессенджера «Telegram» был разработан бот, который умеет обрабатывать заказы клиентов пекарни. Помимо просмотра меню и добавления товара в корзину, были добавлены функции администрирования, а именно динамическое удаление и добавление новых пунктов меню и получение уведомлений о поступивших заказах. В ходе работы над проектом были выполнены следующие задачи:

* был проведён анализ мессенджера «Telegram», его преимуществ, которые привлекают пользователей. На основе полученных данных был определен функционал;
* в процессе исследования были изучены различные варианты баз данных, сред разработки и языков программирования. Благодаря анализу были выбраны те варианты, которые являются наиболее удобными и наибольшим образом удовлетворяют требованиям;
* были рассмотрены варианты существующих библиотек для написания современных Telgram-ботов, за счёт требования к асинхронности была выбрана библиотека «aiogram»;
* бот был протестирован, весь запланированный функционал удалось реализовать.

Работа над данным проектом позволила приобрести ценный опыт в области разработки Telegram-ботов, улучшить навыки программирования на языке «Python» и развить навыки в анализе требований пользователей. Проект можно считать успешно завершенным, а приложение - перспективным продуктом.

В ходе выполнения данной курсовой работы было проведено исследование такой актуальной темы, как создание современного и удобного Telegram-бота, который делает функцию обработки заказов простой даже для неопытных пользователей, а также автоматизирует работу для производителя.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Златопольский, Д.М. Основы программирования на языке Python. – М. : «ДМК Пресс», 2017. – 284 с.
2. Лутц, М. Программирование на Python. Т. 2 / М. Лутц. – М. : «Символ», 2016. - 992 c
3. Боты в Telegram: какими они бывают, как работают и как сделать своего [Электрон. ресурс]. – Режим доступа : https://skillbox.ru/media/marketing/boty-v-telegram-kakimi-oni-byvayut-kak-rabotayut-i-kak-sdelat-svoego/ – 20.11.2023
4. Всё, о чём должен знать разработчик Телеграм-ботов [Электрон. ресурс]. – Режим доступа : https://habr.com/ru/post/543676/ – 20.11.2023
5. SQLite, MySQL и PostgreSQL: сравниваем популярные реляционные СУБД [Электрон. ресурс]. – Режим доступа : https://tproger.ru/translations/sqlite-mysql-postgresql-comparison – 25.11.2023
6. Пишем Telegram-ботов с aiogram 3.x [Электрон. ресурс]. – Режим доступа : https://mastergroosha.github.io/aiogram-3-guide/ – 18.11.2023
7. Разбираемся в асинхронности: где полезно, а где – нет? [Электрон. ресурс]. – Режим доступа : https://habr.com/ru/companies/kts/articles/713084/ – 27.11.2023
8. Работа с SQLite в Python [Электрон. ресурс]. – Режим доступа : https://habr.com/ru/articles/754400/ – 18.11.2023
9. Возможности PyCharm [Электрон. ресурс]. – Режим доступа : https://www.jetbrains.com/ru-ru/pycharm/features/ – 20.11.2023

# ПРИЛОЖЕНИЯ

Приложение А

Листинг А.1 – Telegram-бот для обработки заказов

# импорт библиотек

import asyncio # библиотека позволяющая выполнять функции ассинхронно

import sqlite3 # библиотека базы данных

import logging # логирование работы программы

from aiogram import Bot, Dispatcher, types, F # библиотеки и модули для работы с телеграм

from aiogram.filters import Command

from aiogram.enums import ParseMode

from aiogram.types import Message, FSInputFile, URLInputFile, BufferedInputFile

from aiogram.utils.keyboard import InlineKeyboardBuilder

from aiogram.fsm.storage.memory import MemoryStorage

from aiogram.fsm.state import State, StatesGroup

from aiogram.fsm.context import FSMContext

from config import token

# Включаем логирование, чтобы не пропустить важные сообщения

logging.basicConfig(level=logging.INFO)

# Объект бота

bot = Bot(token=token, parse\_mode="HTML")

# Диспетчер

dp = Dispatcher() # Диспетчер - объект, занимающийся получением апдейтов от Telegram с последующим выбором хэндлера для обработки принятого апдейта

class AdminStates(StatesGroup): # Класс состояний

name = State()

price = State()

delete = State()

def get\_keyboard\_products(array):

buttons = []

prebuttons = []

counter = 0

for [i] in array:

counter += 1

if counter == 1:

first = types.InlineKeyboardButton(text=i, callback\_data='toCart\_'+i)

prebuttons.append(first)

else:

counter = 0

second = types.InlineKeyboardButton(text=i, callback\_data='toCart\_'+i)

prebuttons.append(second)

buttons.append(prebuttons)

prebuttons = []

if len(prebuttons) != 0:

buttons.append(prebuttons)

buttons.append([types.InlineKeyboardButton(text="🔙Назад", callback\_data="back")])

keyboard = types.InlineKeyboardMarkup(inline\_keyboard=buttons)

return keyboard

def get\_keyboard\_cart():

buttons = [

[

types.InlineKeyboardButton(text="✅Создать заказ", callback\_data="make\_order"),

types.InlineKeyboardButton(text="❌Очистить корзину", callback\_data="delete\_cart")],

[types.InlineKeyboardButton(text="🔙Назад", callback\_data="back")]]

keyboard = types.InlineKeyboardMarkup(inline\_keyboard=buttons)

return keyboard

async def empty\_cart(message: types.Message):

await message.edit\_text(

f"Корзина пуста. Добавьте что-нибудь!",

reply\_markup=get\_keyboard\_cart()

)

# Хэндлер на команду /start

@dp.message(Command("start"))

async def cmd\_start(message: types.Message):

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor() #объект "курсор" для выполнения SQL-запросов и операций с базой данных

cursor.execute('SELECT \* FROM Users WHERE user\_id=?', (message.from\_user.id,))

entry = cursor.fetchone()

# создание кнопок

builder = InlineKeyboardBuilder()

builder.add(types.InlineKeyboardButton(

text="🥧Пироги",

callback\_data="Pirogi")

)

builder.add(types.InlineKeyboardButton(

text="🧁Не пироги",

callback\_data="Ne pirogi")

)

builder.add(types.InlineKeyboardButton(

text="🛒Корзина",

callback\_data="Cart")

)

builder.add(types.InlineKeyboardButton(

text="🗒️Меню",

callback\_data="Menu")

)

if message.from\_user.id == 1372145704:

builder.add(types.InlineKeyboardButton(

text="😎Админ-панель",

callback\_data="AdminPanel")

)

builder.adjust(3) # сколько кнопок в ряд

if entry is None:

cursor.execute('INSERT INTO Users (user\_id, full\_name) VALUES (?, ?);',

(message.from\_user.id, message.from\_user.full\_name))

# Сохраняем изменения и закрываем соединение

connection.commit()

connection.close()

await message.answer(

f'🥐 <strong>Пекарня "Ваниль"</strong> 🥐\n\n Привет, {message.from\_user.full\_name}, рады видеть вас впервые! \n\n Что вас интересует?',

reply\_markup=builder.as\_markup()) # reply\_markup - это кнопки, которые будут добавлены к сообщению.

# delete users message

await message.delete()

else:

await message.answer(

f'🥐 <strong>Пекарня "Ваниль"</strong> 🥐\n\n Привет, {message.from\_user.full\_name}, рады видеть вас снова! \n\n Что вас интересует?',

reply\_markup=builder.as\_markup())

await message.delete()

@dp.message(Command('images'))

async def upload\_photo(message: Message):

file\_ids = []

image\_from\_url = URLInputFile("https://avatars.akamai.steamstatic.com/87c4bb7bbd24b73048b123cf183bde83a7a1ce11\_full.jpg")

result = await message.answer\_photo(

image\_from\_url,

caption="Изображение по ссылке"

)

file\_ids.append(result.photo[-1].file\_id)

await message.answer("Отправленные файлы:\n" + "\n".join(file\_ids))

@dp.message(Command('images2'))

async def upload\_photo(message: Message):

await message.answer\_photo('AgACAgIAAxkDAANZZVc2mvTbLYp25NfPhfahftHMgK0AAvXRMRs-n8FKrDsVxDzwZ8oBAAMCAANtAAMzBA')

@dp.message(Command('link'))

async def upload\_photo(message: Message):

await message.answer('<a href = "https://steamcommunity.com/profiles/76561198367341207">LINK</a>', disable\_web\_page\_preview=True)

@dp.callback\_query(F.data == "Ne pirogi") #Хэндлер на нажатие кнопки Не пироги. За события после нажатия кнопки отвечают @dp.callback\_query, а за сообщения @dp.message

async def menu\_ne\_pirogi(callback: types.CallbackQuery):

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute('SELECT name FROM Products WHERE is\_pirogi = "0"')

names = cursor.fetchall()

await bot.delete\_message(callback.message.chat.id, callback.message.message\_id)

menu = 'У нас в меню есть:\n'

for [i] in names: # Цикл в питоне. Перебор всех элементов

cursor.execute(f'SELECT price FROM Products WHERE name="{i}" AND is\_pirogi = "0"')

price = str(cursor.fetchone()[0])

menu += str(i) + f' - {price} руб\n'

menu += '\nЧто желаете добавить в корзину?'

connection.close()

await bot.send\_message(callback.from\_user.id, menu, reply\_markup=get\_keyboard\_products(names))

await callback.answer()

@dp.callback\_query(F.data == "Pirogi")

async def menu\_ne\_pirogi(callback: types.CallbackQuery):

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute('SELECT name FROM Products WHERE is\_pirogi = "1"')

names = cursor.fetchall()

await bot.delete\_message(callback.message.chat.id, callback.message.message\_id)

menu = 'У нас в меню есть:\n'

for [i] in names:

cursor.execute(f'SELECT price FROM Products WHERE name="{i}" AND is\_pirogi = "1"')

price = str(cursor.fetchone()[0])

menu += str(i) + f' - {price} руб\n'

menu += '\nЧто желаете добавить в корзину?'

connection.close()

await bot.send\_message(callback.from\_user.id, menu, reply\_markup=get\_keyboard\_products(names))

await callback.answer()

@dp.callback\_query(F.data == "Cart")

async def cart(callback: types.CallbackQuery):

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute(f'SELECT product FROM Cart WHERE user\_id = "{callback.message.chat.id}"')

products = cursor.fetchall()

if len(products) != 0:

print(products)

unique = set(products)

await bot.delete\_message(callback.message.chat.id, callback.message.message\_id)

cartMessage = 'У вас в корзине:\n'

fullcartprice = 0

# menu = 'У нас в меню есть:\n'

for [i] in unique:

count = products.count((f'{i}',)) # cколько предметов с названием i в корзине

cursor.execute(f'SELECT price FROM Products WHERE name = "{i}"')

price = cursor.fetchone()[0]

priceforproduct = count \* int(price)

fullcartprice += priceforproduct

cartMessage += i + ' - x' + str(count) + ' | ' + str(priceforproduct) + ' руб.\n'

cartMessage += f'\n Общая стоимость: {fullcartprice} руб. Создать заказ?'

connection.close()

await bot.send\_message(callback.from\_user.id, cartMessage, reply\_markup=get\_keyboard\_cart())

await callback.answer()

else:

await empty\_cart(callback.message)

await callback.answer()

@dp.callback\_query(F.data.startswith("toCart\_")) # хэндлер отлова события, начинающегося с toCart\_

async def add\_to\_cart(callback: types.CallbackQuery):

action = callback.data.split("\_")[1]

print(action)

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute('INSERT INTO Cart (user\_id, product) VALUES (?, ?);',

(str(callback.from\_user.id), str(action)))

connection.commit()

connection.close()

await callback.answer(text=f"{str(action)} добавлен в корзину!")

@dp.callback\_query(F.data.startswith("add\_"))

async def add\_to\_menu(callback: types.CallbackQuery, state: FSMContext):

#user\_value = user\_data.get(callback.from\_user.id, 0)

action = callback.data.split("\_")[1]

if action == 'pirog':

await state.update\_data(product\_status=1)

else:

await state.update\_data(product\_status=0)

await callback.message.edit\_text('Введите название нового товара:')

await state.set\_state(AdminStates.name) #Установили состояние ввода имени нового товара

@dp.message(AdminStates.name) # хэндлер отлова сообщений, но только из состояния name

async def new\_name(message: types.Message, state: FSMContext) -> None:

await state.update\_data(product\_name=message.text)

await message.reply('Принято! Какая стоимость товара?')

await state.set\_state(AdminStates.price)

@dp.message(AdminStates.price) # хэндлер отлова сообщений, но только из состояния price

async def new\_price(message: types.Message, state: FSMContext) -> None:

builder = InlineKeyboardBuilder()

builder.add(types.InlineKeyboardButton(

text="В главное меню",

callback\_data="back")

)

await state.update\_data(product\_price=message.text)

data = await state.get\_data() # получить всю информацию из хранилища в состояниях

if data['product\_price'].isdigit() and data['product\_name'].isdigit() is False:

await message.reply('Товар добавлен в меню!', reply\_markup=builder.as\_markup())

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute('INSERT INTO Products (name, price, is\_pirogi) VALUES (?, ?, ?);',

(data['product\_name'], int(data['product\_price']), data['product\_status']))

# Сохраняем изменения и закрываем соединение

connection.commit()

connection.close()

else:

await message.reply('Типы данных не совпадают, пересоздайте товар заново!', reply\_markup=builder.as\_markup())

await state.clear()

@dp.callback\_query(F.data == "delete by name")

async def deleter(callback: types.CallbackQuery, state: FSMContext):

await callback.message.edit\_text('Введите полное название товара, который вы хотите удалить (с учетом регистра):')

await state.set\_state(AdminStates.delete) #Установили состояние удаления

@dp.message(AdminStates.delete)

async def deletebyname(message: types.Message, state: FSMContext) -> None:

builder = InlineKeyboardBuilder()

builder.add(types.InlineKeyboardButton(

text="В главное меню",

callback\_data="back")

)

product\_name = message.text

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute(f'SELECT name FROM Products WHERE name = "{product\_name}"')

if cursor.fetchone() is None:

await message.reply('Такого товара нет и удалять нечего!', reply\_markup=builder.as\_markup())

else:

cursor.execute(f'DELETE FROM Products WHERE name = "{product\_name}"')

await message.reply('Товар удален!', reply\_markup=builder.as\_markup())

# Сохраняем изменения и закрываем соединение

connection.commit()

connection.close()

await state.clear()

@dp.callback\_query(F.data == "back")

async def process\_back(callback: types.CallbackQuery):

builder = InlineKeyboardBuilder()

builder.add(types.InlineKeyboardButton(

text="🥧Пироги",

callback\_data="Pirogi")

)

builder.add(types.InlineKeyboardButton(

text="🧁Не пироги",

callback\_data="Ne pirogi")

)

builder.add(types.InlineKeyboardButton(

text="🛒Корзина",

callback\_data="Cart")

)

builder.add(types.InlineKeyboardButton(

text="🗒️Меню",

callback\_data="Menu")

)

if callback.from\_user.id == 1372145704:

builder.add(types.InlineKeyboardButton(

text="😎Админ-панель",

callback\_data="AdminPanel")

)

builder.adjust(3)

await callback.message.answer(f'🥐 <strong>Пекарня "Ваниль"</strong> 🥐\n\n Что вас интересует?',reply\_markup=builder.as\_markup())

await callback.message.delete()

@dp.callback\_query(F.data == "Menu")

async def send\_menu(callback: types.CallbackQuery):

await callback.message.answer\_photo('AgACAgIAAxkDAAIIBGVlxWe\_SUyWKBz39MSDlpNHgy-TAAIh0TEbiqgxSwVCmP7S5VPuAQADAgADdwADMwQ')

@dp.callback\_query(F.data == "delete\_cart")

async def delete\_cart(callback: types.CallbackQuery):

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute(f'DELETE FROM Cart WHERE user\_id = "{callback.from\_user.id}"')

# Сохраняем изменения и закрываем соединение

connection.commit()

connection.close()

await callback.answer(text="Корзина очищена!")

await empty\_cart(callback.message)

@dp.callback\_query(F.data == "make\_order")

async def make\_order(callback: types.CallbackQuery):

if callback.from\_user.username is None:

await callback.answer(text="Невозможно создать заказ - установите ваш Telegram Username, чтобы с вами можно было связаться!", show\_alert=True)

else:

connection = sqlite3.connect('bakery.db')

cursor = connection.cursor()

cursor.execute(f'SELECT product FROM Cart WHERE user\_id = "{callback.message.chat.id}"')

products = cursor.fetchall()

if len(products) != 0:

unique = set(products)

#await bot.delete\_message(callback.message.chat.id, callback.message.message\_id)

messageforclient = 'Заказ создан! Ждите, когда с вами свяжутся!\n'

messageforadmin = f'Клиент @{callback.from\_user.username} сделал заказ. Вот что в него входит:\n'

fullcartprice = 0

for [i] in unique:

count = products.count((f'{i}',))

cursor.execute(f'SELECT price FROM Products WHERE name = "{i}"')

price = cursor.fetchone()[0]

priceforproduct = count \* int(price)

fullcartprice += priceforproduct

messageforadmin += i + ' - x' + str(count) + ' | ' + str(priceforproduct) + ' руб.\n'

messageforadmin += f'\n Общая стоимость: {fullcartprice} руб.'

cursor.execute(f'DELETE FROM Cart WHERE user\_id = "{callback.from\_user.id}"')

# Сохраняем изменения и закрываем соединение

connection.commit()

connection.close()

await callback.answer(text=messageforclient, show\_alert=True)

builder = InlineKeyboardBuilder()

builder.add(types.InlineKeyboardButton(

text="🔙Назад",

callback\_data="back")

)

await callback.message.edit\_text(messageforclient, reply\_markup=builder.as\_markup())

await bot.send\_message('1372145704', messageforadmin)

else:

await callback.answer(text="Невозможно создать заказ - ваша корзина пуста!", show\_alert=True)

@dp.callback\_query(F.data == "AdminPanel")

async def admin\_panel(callback: types.CallbackQuery):

if callback.from\_user.id == 1372145704:

builder = InlineKeyboardBuilder()

builder.add(types.InlineKeyboardButton(

text="🥧Пирог",

callback\_data="add\_pirog")

)

builder.add(types.InlineKeyboardButton(

text="🧁Не пирог",

callback\_data="add\_ne pirog")

)

builder.add(types.InlineKeyboardButton(

text="❌Что-то удалить",

callback\_data="delete by name")

)

builder.add(types.InlineKeyboardButton(

text="🔙Назад",

callback\_data="back")

)

builder.adjust(3)

await callback.message.edit\_text(f'🥐<strong>Какой товар добавить? \nИли хотите что-нибудь удалить?</strong> 😢', reply\_markup=builder.as\_markup())

# Запуск процесса поллинга новых апдейтов

async def main():

await bot.delete\_webhook(drop\_pending\_updates=True)

await dp.start\_polling(bot)

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())