**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ**

**МЕЖДУНАРОДНЫХ ОТНОШЕНИЙ (УНИВЕРСИТЕТ)»**

**МИНИСТЕРСТВА ИНОСТРАННЫХ ДЕЛ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ОДИНЦОВСКИЙ ФИЛИАЛ**

**ФАКУЛЬТЕТ ФИНАНСОВОЙ ЭКОНОМИКИ**

**КАФЕДРА МАТЕМАТИЧЕСКИХ МЕТОДОВ**

**И БИЗНЕС-ИНФОРМАТИКИ**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

по направлению подготовки

**38.03.05 Бизнес-информатика**

Направленность (профиль) подготовки

**Информационные технологии в международном бизнесе**

Тема работы:

**Разработка программного агента для автоматизации процесса оформления заказов тортов**

|  |  |
| --- | --- |
|  | Выполнила:  студентка Чебыкина Стефания Андреевна  факультет ФЭ, гр. ИТБ(б)-О-21/1  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента)  Научный руководитель:  Кавин Юрий Александрович  канд. техн. наук, доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись научного руководителя) |
| **Выпускная**  **квалификационная работа**  **защищена**  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Секретарь ГЭК \_\_\_\_\_\_\_\_\_\_ |

**Одинцово 2025**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc188883718)

[ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 10](#_Toc188883719)

[1.1 Основные сведения об индивидуальном предпринимателе / кондитерском бизнесе 10](#_Toc188883720)

[1.1.1 История и профиль деятельности 10](#_Toc188883721)

[1.1.2 Миссия, функции и структура 12](#_Toc188883722)

[1.1.3 Текущее программное обеспечение 13](#_Toc188883723)

[1.2 Модель бизнес-процесса «AS-IS» 15](#_Toc188883724)

[1.2.1 Описание существующего процесса оформления заказа 15](#_Toc188883725)

[1.2.2 Диаграмма последовательности (AS-IS) 17](#_Toc188883726)

[1.3 Проблемный анализ текущего бизнес-процесса 21](#_Toc188883727)

[1.3.1 Недостатки ручной обработки 21](#_Toc188883728)

[1.3.2 Риски для бизнеса 23](#_Toc188883729)

[1.4 Анализ лучших практик в области чат-ботов для бизнеса 25](#_Toc188883730)

[1.4.1 Обзор современных решений 26](#_Toc188883731)

[1.4.2 Сравнение и выбор подхода 29](#_Toc188883732)

[1.5 Обоснование выбора направления и технологии автоматизации 33](#_Toc188883733)

[1.5.1 Критерии выбора 33](#_Toc188883734)

[1.5.2 Описание ожидаемых результатов 35](#_Toc188883735)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ ТЕЛЕГРАМ-БОТА ДЛЯ АВТОМАТИЗАЦИИ ЗАКАЗОВ 38](#_Toc188883736)

[2.1 Анализ требований к программному модулю 38](#_Toc188883737)

[2.1.1 Характеристики задачи автоматизации 38](#_Toc188883738)

[2.1.2 Описание функциональных требований 40](#_Toc188883739)

[2.1.3 Требования к техническому, программному и информационному обеспечению 43](#_Toc188883740)

[2.2 Разработка чат-бота и сценарии его работы 46](#_Toc188883741)

[2.2.1 Структура и логика работы бота 46](#_Toc188883742)

[2.2.2 Программная реализация 49](#_Toc188883743)

[2.2.3 Сценарии взаимодействия 59](#_Toc188883744)

[ГЛАВА 3. ОЦЕНКА ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ПРОЕКТА 73](#_Toc188883745)

[3.1 Расчёт фактических затрат на реализацию 73](#_Toc188883746)

[3.1.1 Календарно-ресурсное планирование 73](#_Toc188883747)

[3.1.2 Анализ затрат 75](#_Toc188883748)

[3.2 Оценка целесообразности проекта 76](#_Toc188883749)

[ЗАКЛЮЧЕНИЕ 80](#_Toc188883750)

[ПРИЛОЖЕНИЯ 86](#_Toc188883751)

[Приложение 1 86](#_Toc188883752)

# ВВЕДЕНИЕ

В современных условиях быстро растущей конкуренции в сфере услуг и торговли, включая кондитерское производство, эффективность и скорость обслуживания клиентов играют решающую роль. Ручное взаимодействие с покупателями через телефон, электронную почту или мессенджеры часто приводит к ряду проблем:

1. **Высокая нагрузка на менеджера или владельца**
   * Нужно вручную отвечать на каждое сообщение или звонок.
   * Невозможность оперативно обрабатывать заказы в часы пик или в ночное время.
2. **Риск потери потенциальных клиентов**
   * Долгое ожидание ответа снижает лояльность.
   * Часть клиентов уходит к конкурентам, имеющим более удобные каналы взаимодействия.
3. **Вероятность ошибок при оформлении заказов**
   * Человеческий фактор: неправильное понимание пожеланий клиента, путаница в датах, ошибочный расчёт стоимости.
   * Трудности систематизации большого объёма информации о заказах.

В свете этих проблем автоматизация процесса приёма заказов на торты посредством чат-бота в Telegram приобретает особую актуальность. Telegram, как один из самых популярных мессенджеров, предоставляет широкие возможности для быстрого и удобного взаимодействия с клиентами. Преимущества использования такого решения включают:

* **Круглосуточная доступность**: Чат-бот продолжает принимать заказы и отвечать на типовые вопросы 24/7 без участия человека.
* **Сокращение издержек**: Уменьшается необходимость в дополнительном персонале, отвечающем на звонки и сообщения.
* **Улучшение качества обслуживания**: Чёткая структура диалога и мгновенная реакция повышают удовлетворённость клиентов.
* **Минимизация ошибок**: Предусмотренные сценарии диалога, валидация вводимых данных и автоматический расчёт стоимости снижают вероятность неправильного оформления заказа.

Таким образом, разработка и внедрение Telegram-бота способствует росту эффективности бизнеса, улучшает клиентский сервис и упрощает процессы внутри компании. Данная дипломная работа посвящена именно этому — детальному анализу, моделированию и реализации системы, способной взять на себя значительную часть рутины при взаимодействии с покупателями.

**Главная цель** настоящей выпускной квалификационной работы (ВКР) — разработка и внедрение программного модуля (Telegram-бота) для автоматизации процесса приёма и обработки заказов тортов. Для достижения поставленной цели необходимо решить следующий комплекс задач:

1. **Изучить предметную область и выявить основные требования**
   * Изучение специфики малого кондитерского бизнеса.
   * Определение ключевых функций и ограничений, исходя из существующей практики приёма заказов.
2. **Разработать бизнес-модель «как есть» (AS-IS) и «как должно быть» (TO-BE) при помощи IDF0/BPMN 2.0**
   * Зафиксировать текущие бизнес-процессы, выявить узкие места и потенциальные точки улучшения.
   * Предложить оптимизированную модель, интегрирующую автоматизированное решение.
3. **Сформировать технические требования к системе**
   * Описать необходимые функциональные, технические и пользовательские требования.
   * Определить платформу, инструменты и технологии, обеспечивающие реализацию системы.
4. **Спроектировать архитектуру и создать прототип чат-бота**
   * Разработать архитектурную схему, описывающую компоненты и взаимодействие между ними (бот, база данных, API и пр.).
   * Подготовить и протестировать работающий прототип чат-бота с базовыми функциями: приёмом заказов, хранением информации и интеграцией с системой уведомлений.
5. **Провести оценку экономической целесообразности проекта**
   * Оценить затраты на разработку и внедрение решения.
   * Сопоставить выгоды (сокращение ручного труда, рост лояльности клиентов, увеличение числа заказов) с затратами.
   * Рассчитать ключевые экономические показатели (ROI, период окупаемости).

**Объект исследования** — процесс оформления заказов на торты в рамках небольшого кондитерского бизнеса, который взаимодействует с клиентами преимущественно через телефон и социальные сети.

**Предмет исследования** — разработка Telegram-бота и сопутствующей инфраструктуры для автоматизации и оптимизации данного процесса. В сфере малого и среднего бизнеса такая автоматизация позволяет одновременно снизить нагрузку на сотрудников и повысить удовлетворённость клиентов.

Для решения поставленных задач используются следующие методы:

1. **Анализ предметной области**
   * Сбор и систематизация данных о существующих способах приёма заказов в малом бизнесе.
   * Изучение кейсов, где уже применяются похожие решения.
2. **Сравнение существующих решений**
   * Исследование конструкторов чат-ботов, готовых платформ, а также способов самостоятельной разработки.
   * Сравнительный анализ по критериям стоимости, функционала, гибкости и простоты внедрения.
3. **Моделирование бизнес-процессов в IDF0/BPMN 2.0**
   * Построение модели «AS-IS» (описание текущего процесса).
   * Проектирование модели «TO-BE», отражающей автоматизированное решение.
   * Выявление узких мест и путей оптимизации.
4. **Проектирование программных систем**
   * Выбор фреймворков для разработки бота (Python + aiogram, Node.js + Telegraf и т.д.).
   * Создание схемы взаимодействия модулей, формирование технической и логической архитектуры приложения.
5. **Экономический анализ (оценка затрат и выгод)**
   * Расчёт затрат на разработку и внедрение чат-бота.
   * Анализ потенциального роста прибыли за счёт экономии времени и улучшения качества обслуживания.
   * Прогнозирование периода окупаемости решения.

Практическая ценность результатов данной ВКР заключается в их широком применении в малом и среднем бизнесе, занимающемся изготовлением или продажей товаров на заказ. Основные выгоды от использования Telegram-бота в кондитерском бизнесе включают:

1. **Увеличение скорости обслуживания**
   * Заказчики получают ответ на вопросы и подтверждение оформления заказа практически мгновенно.
   * Нет необходимости ждать, пока сотрудник освободится от текущих дел.
2. **Снижение ошибок и временных затрат**
   * Структурированный диалог в боте, и автоматическая проверка вводимых данных уменьшают вероятность путаницы и некорректных записей.
   * Освобождается время владельца или менеджера, которое можно направить на расширение ассортимента или повышение качества продукции.
3. **Повышение конкурентоспособности**
   * Возможность круглосуточного приёма заказов без увеличения численности персонала.
   * Формирование современного имиджа предприятия, идущего в ногу с технологическими трендами.

Таким образом, в ходе данной работы будут представлены конкретные инструменты и методологии, позволяющие организовать автоматизированный приём и обработку заказов, что особенно актуально в условиях стремительного роста конкуренции и возрастающих ожиданий клиентов по качеству сервиса.

# ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Основные сведения об индивидуальном предпринимателе / кондитерском бизнесе

### 1.1.1 История и профиль деятельности

Индивидуальный предприниматель (ИП) специализируется на изготовлении тортов на заказ в рамках небольшого кондитерского производства. Его деятельность началась как домашнее хобби: сначала владелец готовил торты для друзей и семьи, затем количество клиентов постепенно расширялось за счёт рекомендаций и отзывов в социальных сетях.

На сегодняшний день бизнес сохраняет **индивидуальный подход** к каждому заказу, позволяя клиентам выбирать:

* Вкус и размер торта,
* Тип крема или глазури,
* Декор (фигурки, надписи, тематическое оформление),
* Дополнительные опции (безглютеновая основа, сахарозаменители и др.).

Ключевыми **каналами сбыта** являются:

1. **Социальные сети**:
   * Instagram — основная площадка для демонстрации работ, фото тортов и общения с потенциальными покупателями.
   * VK — альтернативный канал для продвижения и поиска клиентов.
2. **Личные контакты**:
   * Рекомендации от довольных клиентов, которые рассказывают о продукции друзьям и знакомым.
3. **Телефон**:
   * Прямые звонки и сообщения (SMS, мессенджеры).
   * Основной способ для уточнения деталей и быстрого обратного звонка.

|  |  |  |
| --- | --- | --- |
| **Канал** | **Преимущества** | **Недостатки** |
| Instagram | - Широкая аудитория - Красивое визуальное оформление | - Зависимость от алгоритмов соц.сети - Не все потенциальные клиенты активны в Instagram |
| VK | - Возможность таргетированной рекламы - Группы по интересам | - Менее популярная платформа для кулинарных заказов (по сравнению с Instagram) |
| Личные контакты | - Высокий уровень доверия - Прямая рекомендация | - Ограниченный охват аудитории |
| Телефон (звонки/СМС) | - Оперативное уточнение деталей - Удобно для старшего поколения | - Не всегда удобно для занятых клиентов - Нет записей переписки (если это звонок) |

Таблица - Сравнительный обзор каналов сбыта

В совокупности эти каналы позволяют бизнесу формировать устойчивый поток заказчиков. Однако, по мере роста числа заказов и разнообразия запросов встала необходимость оптимизации и автоматизации процесса, чтобы не терять клиентов и сократить время на рутинное общение.

### 1.1.2 Миссия, функции и структура

Основная миссия данного кондитерского предприятия — **предоставлять клиентам возможность приобретать высококачественные, красивые и вкусные кондитерские изделия, сделанные с учётом индивидуальных предпочтений и ограничений**. При этом важен не только результат — готовый торт, но и весь процесс взаимодействия с клиентом, начиная от первичной консультации и заканчивая доставкой и обратной связью.

**Основные функции**

1. **Приём заказов**
   * Сбор информации о клиенте и его предпочтениях по дизайну, составу, дате и времени готовности.
   * Согласование цены и способа оплаты.
2. **Изготовление тортов**
   * Подготовка необходимых ингредиентов и материалов.
   * Непосредственный процесс выпечки, сборки и декорирования торта.
3. **Cамовывоз**
   * Возможность клиенту самостоятельно забрать заказ из кондитерской.

**Организационная структура**

В большинстве случаев бизнес ведётся одним человеком — владельцем ИП, который совмещает роль шеф-кондитера, администратора и менеджера по работе с клиентами. Однако при повышенной загрузке владелец привлекает помощников и курьеров. Ниже приведена упрощённая схема структуры:

* **Владелец (ИП)** — осуществляет общее руководство, контроль качества, финансовое планирование, продвигает бизнес в соцсетях.Приём заказов как правило, ведётся самим владельцем. Он отвечает на сообщения, фиксирует пожелания клиентов и оформляет внутренние документы (или записи).
* **Помощники / Кондитерские работники** — помогают в процессе приготовления и декорирования. Могут быть наняты на сезонные периоды (например, во время праздников и повышенного спроса).
* **Курьеры** — занимаются доставкой тортов к клиентам либо в точки самовывоза, если бизнес таковые предусматривает.

Такая структура довольно гибка, но при увеличении количества заказов может возникнуть перегрузка владельца, который вынужден одновременно выполнять кулинарные, организационные и коммуникационные функции.

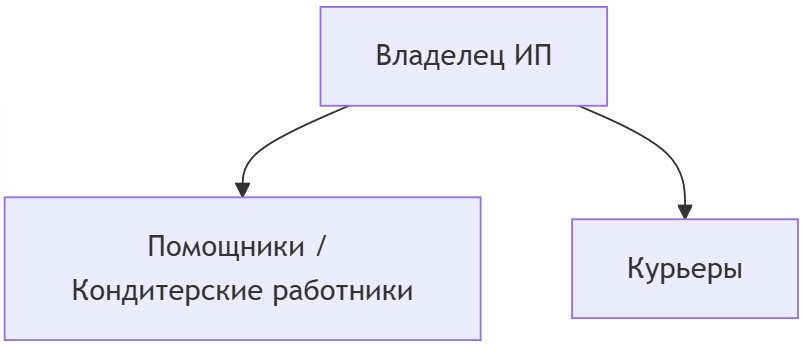


Рисунок - Упрощенная схема структуры компании

### 1.1.3 Текущее программное обеспечение

В настоящее время в бизнесе используется в основном стандартный набор цифровых инструментов:

1. **Социальные сети (Instagram, VK)**
   * Основной способ привлечения клиентов через фотографии, сторис и обратную связь в комментариях.
   * Размещение контактной информации (номер телефона, ссылка на мессенджеры).
2. **Мессенджеры (WhatsApp, Telegram)**
   * Способ оперативной коммуникации.
   * Клиенты присылают изображения желаемого декора или примеры похожих работ.
   * Бывают ситуации, когда часть переписки находится в WhatsApp, а другая — в Telegram, что затрудняет систематизацию.
3. **Телефонные звонки**
   * Звонки принимаются на личный номер владельца.
   * Часть важных уточнений может теряться, если не записывать их сразу или вести «блокнот заказов» вручную.
4. **Отсутствие централизованного CRM**
   * Нет единой базы данных, где хранится информация о клиентах, истории заказов и предпочтениях.
   * Иногда ведётся простой Excel-файл или Google-таблица, однако он редко актуализируется и не интегрирован напрямую с каналами коммуникации.
5. **Отсутствие чат-бота**
   * Все ответы на вопросы и детали заказа владелец даёт самостоятельно.
   * В случае большой загруженности или временного отсутствия владельца (например, во время закупок или доставки) клиент может долго ждать ответа.

Подобная ситуация приводит к тому, что значительная часть времени уходит на ручную обработку однотипных запросов, поиск нужной переписки и продублированных сообщений в разных мессенджерах. Кроме того, при резком увеличении количества заказов (праздники, свадебный сезон, корпоративные мероприятия) возрастает риск пропуска или неправильного оформления заказа.

## 1.2 Модель бизнес-процесса «AS-IS»

### 1.2.1 Описание существующего процесса оформления заказа

В текущих условиях приёма заказов на торты бизнес-процесс представляет собой цепочку из нескольких шагов, начиная от момента, когда потенциальный клиент решает оставить заявку на торт, и заканчивая получением готового изделия. Ниже приводится детальное описание основных этапов.

1. **Инициирование заказа**
   * **Способ контакта**: Клиент может написать сообщение в социальной сети (Instagram, VK), в мессенджере (WhatsApp, Telegram) или позвонить по номеру телефона.
   * **Цель**: Сообщить свои пожелания по поводу торта, уточнить детали и узнать цену.
2. **Уточнение деталей заказа**
   * **Сбор ключевой информации**: Вкус торта, желаемая форма, количество порций, декор (надпись, фигурки, шоколад, ягоды и т.д.), дата и время, к которому торт нужен.
   * **Дополнительные запросы**: Аллергические ограничения, использование сахарозаменителей, варианты теста и крема.
   * **Расчёт предварительной стоимости**: Исходя из состава и сложности декора, а также дополнительных пожеланий.
3. **Подтверждение заказа**
   * **Согласование цены и сроков**: После консультации клиент подтверждает готовность оплатить стоимость.
   * **Фиксация данных**: Кондитер (или владелец ИП) записывает всю информацию в электронный блокнот, Excel-таблицу или даже на бумажный носитель (в зависимости от предпочтений).
   * **Выбор способа оплаты**: Возможен наличный расчёт при получении или перевод на карту (например, через Сбербанк, Тинькофф, перевод по номеру телефона).
4. **Оплата**
   * **Предоплата или полная оплата**: Некоторые клиенты вносят 50% предоплаты для резервирования даты, другие оплачивают всю сумму сразу. Есть и клиенты, предпочитающие оплачивать при получении наличными.
   * **Подтверждение платежа**: Кондитер проверяет поступление средств (если это онлайн-перевод).
5. **Изготовление торта**
   * **Закупка ингредиентов** (при необходимости): Если заказ отличается от стандартного, либо требуется специфический декор.
   * **Приготовление теста, выпечка коржей, приготовление крема**: Процесс строго регламентируется рецептурой.
   * **Сборка и оформление торта**: Декорирование согласно пожеланиям клиента.
6. **Доставка или самовывоз**
   * **Варианты**:
     1. Самовывоз — клиент приезжает самостоятельно в условленное время.
     2. Доставка курьером — владелец либо приглашённый курьер довозит торт по указанному адресу.
   * **Время и условия**: Предварительно согласовываются; в случае доставки — указывается примерный интервал прибытия.
7. **Завершение**
   * **Получение торта клиентом**: Проверка соответствия ожиданиям, расчёт при необходимости.
   * **Обратная связь**: Клиент может оставить отзыв, поделиться фотографиями и впечатлениями в социальных сетях, тем самым стимулируя сарафанное радио.

Таким образом, в текущей («AS-IS») модели процесс во многом опирается на ручную коммуникацию (телефон, мессенджеры) и не использует централизованную систему учёта заказов, что нередко приводит к дублированию информации, рискам пропуска важных деталей и задержкам в ответах.

### 1.2.2 Диаграмма последовательности (AS-IS)

Для наглядной иллюстрации взаимодействия основных участников процесса в текущей модели можно воспользоваться диаграммой последовательности (Sequence Diagram). Ниже представлена диаграмма, отражающая типовой сценарий приёма заказа, уточнения деталей, изготовления и доставки:

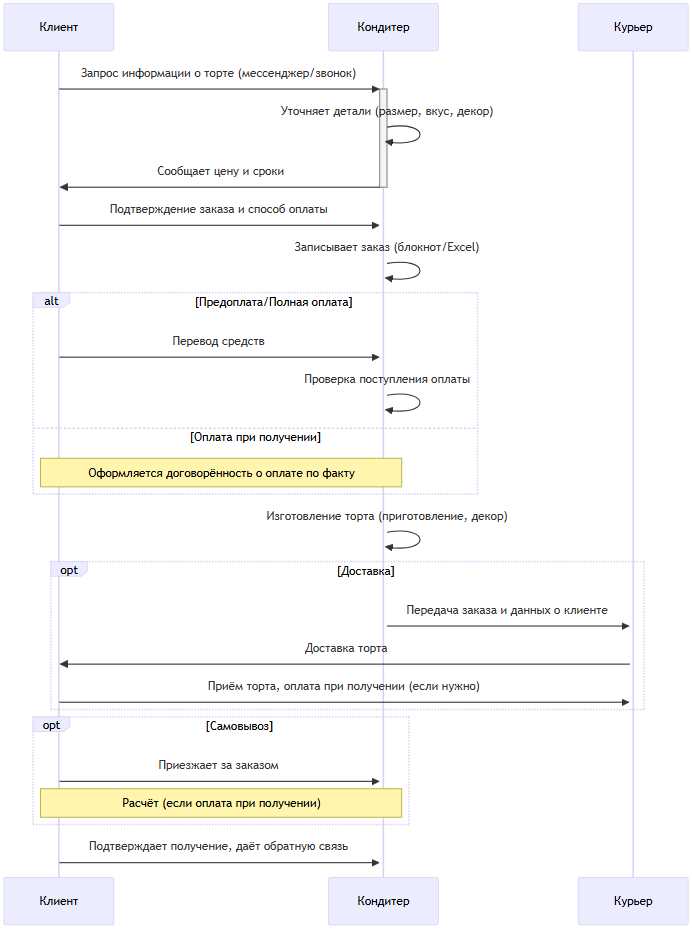


Рисунок - Диаграмма последовательности бизнес-процесса

**Расшифровка основных взаимодействий**

1. **Клиент ==> Кондитер** — клиент связывается через удобный канал (мессенджер или звонок) и излагает свои пожелания.
2. **Кондитер ==> Кондитер** — внутренний процесс уточнения технических деталей (требуемые ингредиенты, сложность исполнения).
3. **Кондитер ==> Клиент** — возврат с информацией о цене, сроках изготовления и возможных вариантах оплаты.
4. **Клиент ==> Кондитер** — подтверждение намерения оформить заказ и выбор метода оплаты (предоплата/полная оплата или при получении).
5. **Кондитер ==> Кондитер** — фиксация заказа в блокноте или таблице, что при большом объёме заказов может приводить к дублированию и ошибкам.
6. **Оплата** — при переводе средств или оплате при получении: проверка транзакции, согласование деталей.
7. **Изготовление** — процесс выпечки, сборки, декора.
8. **Доставка/Самовывоз** — зависимо от пожеланий клиента и возможностей кондитера (наличие курьера).
9. **Завершение** — клиент забирает или получает заказ, даёт обратную связь.

**Ключевые участники (Actors)**

* **Клиент**: любой человек, желающий приобрести торт на заказ.
* **Кондитер**: владелец или ответственный за изготовление и приём заказов. Часто совмещает несколько ролей в небольшом бизнесе.
* **Курьер**: привлекается при необходимости доставки, может быть отдельным сотрудником или самим владельцем, если он решит отвезти торт лично.

**Потоки данных**

* **Информация о заказе**: детали торта, предпочтения клиента, время и место доставки.
* **Оплата**: денежный перевод (онлайн) или наличные при получении.
* **Уведомление о готовности**: сообщение или звонок клиенту, что торт готов к выдаче/доставке.

## 1.3 Проблемный анализ текущего бизнес-процесса

### 1.3.1 Недостатки ручной обработки

Применение ручных методов приёма, учёта и сопровождения заказов — будь то телефонные звонки, переписка в социальных сетях или мессенджерах — ведёт к нескольким системным проблемам.

**Ошибки и путаница при большом количестве заказов**

В периоды высокого спроса (сезонные праздники, корпоративные мероприятия, акции) увеличивается объём одновременно поступающих заявок. Кондитер или владелец ИП:

* Получает множество сообщений в разных каналах (Instagram, VK, WhatsApp, Telegram, телефонные звонки).
* Записывает детали заказа в блокнот, Excel или простые электронные заметки.

Без централизованного инструмента и общей базы данных **возникает риск**:

1. **Дублирования заказов** — заказ может быть зафиксирован в разных заметках несколько раз.
2. **Потери контактов** — при ручном копировании контактов из одного чата в другой.
3. **Путаницы в датах и времени** — возможна ошибка в сроках исполнения, особенно при резком увеличении заявок.

**Сложность контроля статусов**

На каждом этапе (принятие, подтверждение, оплата, изготовление, доставка) **отсутствует единый инструмент мониторинга**, поэтому владелец ИП зачастую полагается на личные заметки и память. Следствия:

* **Сложность проверки текущего положения заказа** — в случае, если клиент позвонит, чтобы узнать статус.
* **Затруднённая координация с курьером** — нет автоматического уведомления, приходится вручную передавать адрес, время, контакт клиента и детали доставки.

**Отсутствие автоматических уведомлений**

Многие клиенты ожидают оперативной информации о ходе выполнения заказа. В ручном процессе:

* **Ответственность за отправку уведомлений** лежит на владельце, что не всегда происходит своевременно.
* **Постоянная необходимость отвечать на однотипные вопросы** (“Готов ли мой торт?”, “Когда могу забрать?”) приводит к лишним трудозатратам.

В результате, клиенты могут терять ощущение «личного подхода» и контроля, когда своевременные уведомления не приходят. Это снижает уровень доверия и лояльности.

|  |  |  |
| --- | --- | --- |
| **Недостаток** | **Проявления** | **Последствия для бизнеса** |
| Ошибки и путаница при большом количестве заказов | - Разрозненная информация - Сложность учёта и контроля | - Пропущенные или неверно зафиксированные заказы - Недовольные клиенты |
| Сложность контроля статусов | - Нет общего реестра заказа - Неизвестно, на каком этапе находится торт | - Рост времени на ручную проверку - Задержки и ошибки в выполнении заказов |
| Отсутствие автоматических уведомлений | - Нет централизованного механизма уведомления о статусах - Владелец пишет вручную | - Увеличение времени ответов - Недовольство клиентов из-за неполучения своевременных ответов |

Таблица - Недостатки ручной обработки и их влияние на бизнес

### 1.3.2 Риски для бизнеса

Сохранение традиционной ручной модели без модернизации способно привести к ряду неблагоприятных последствий, связанных как с финансовыми, так и с репутационными потерями.

**Потеря потенциальных клиентов из-за несвоевременных ответов**

Современные клиенты привыкли к быстрому сервису. Если заказчик не может оперативно получить информацию о цене, сроках и других деталях, то:

* **Высок риск ухода к конкурентам** — быстрая реакция может быть решающим фактором при выборе поставщика.
* **Снижение конверсии** — часть потенциальных клиентов не дожидается ответа и не оформляет заказ.

При ручной обработке сообщений владелец ИП не всегда успевает отвечать вовремя, особенно когда параллельно занимается приготовлением тортов, ведением соцсетей и другими делами.

**Перегрузка владельца рутинными задачами**

Владельцу, совмещающему несколько ролей — от шеф-кондитера до маркетолога, — зачастую приходится:

* Тратить значительное время на консультации и уточнение деталей.
* Сортировать заявки, проверять статусы, передавать информацию курьеру.
* Вести учёт оплат, взаимодействовать с банком или платёжными системами.

Из-за отсутствия автоматизации:

1. **Снижается эффективность** — кондитер не может полностью сконцентрироваться на изготовлении продукции и повышении её качества.
2. **Сложнее масштабировать бизнес** — невозможно нанять дополнительного сотрудника, который быстро разберётся во всех хаотичных записях и чатах.

|  |  |  |
| --- | --- | --- |
| **Риск** | **Описание** | **Возможное влияние на бизнес** |
| Потеря потенциальных клиентов | - Долгое ожидание ответа - Недостаточная информированность пользователей | - Снижение объёма продаж - Ухудшение репутации |
| Перегрузка владельца рутинными задачами | - Множество обязанностей: приём заказов, консультации, изготовление, доставка | - Снижение качества продукции из-за нехватки времени - Задержки в ответах клиентам |

Таблица - Риски для бизнеса при сохранении «AS-IS» модели

**Итоговые замечания**

В совокупности выявленные недостатки и риски указывают на необходимость **оптимизации** и **автоматизации** основных этапов бизнес-процесса. Выбор решения в пользу централизованной системы (например, чат-бота в Telegram с интеграцией в базу заказов) способен:

* Свести к минимуму человеческий фактор, который порождает путаницу и ошибки.
* Обеспечить быстрое реагирование на запросы клиентов и предоставление статусов заказа.
* Уменьшить нагрузку на владельца, освободив его время для ключевых функций — разработка новых рецептов, контроль качества, продвижение бренда.

Таким образом, проведение проблемного анализа «AS-IS» бизнес-процесса подтверждает высокую актуальность и значимость внедрения автоматизированных инструментов, о которых пойдёт речь в следующих главах работы.

## 1.4 Анализ лучших практик в области чат-ботов для бизнеса

### 1.4.1 Обзор современных решений

**Существующие платформы для создания ботов**

На рынке существует множество инструментов и сервисов, позволяющих создавать чат-ботов без значительных познаний в программировании. Ниже перечислены некоторые из самых популярных:

1. **ManyChat**
   * **Описание**: Платформа с упором на продажи и маркетинг. Изначально ориентирована на Facebook Messenger, однако также поддерживает Instagram, WhatsApp и другие каналы.
   * **Особенности**:
     + Визуальный конструктор ботов.
     + Множество готовых шаблонов и интеграций (CRM, системы email-рассылок).
     + Поддержка сегментирования аудитории и инструментов для ретаргетинга.
   * **Ограничения**: Фокус на e-commerce и маркетинговые воронки; дополнительные каналы (например, Telegram) могут быть интегрированы, но не всегда так же полнофункциональны, как родные решения.
2. **Botmother**
   * **Описание**: Российская платформа, ориентированная в первую очередь на создание ботов для Telegram, ВКонтакте и Facebook.
   * **Особенности**:
     + Блок-схемы диалогов, упрощённая логика без необходимости глубокого программирования.
     + Поддержка встроенных платежей в Telegram.
     + Возможность подключения к сторонним сервисам через webhook.
   * **Ограничения**: Для комплексных сценариев взаимодействия и интеграции с кастомными базами данных могут потребоваться дополнительные разработки или использование API.
3. **Botpress**
   * **Описание**: Open-source платформа на Node.js, позволяющая создавать и размещать ботов на собственных серверах.
   * **Особенности**:
     + Расширяемая архитектура, доступен исходный код.
     + Возможность подключения модулей (Natural Language Understanding, управление контекстом диалога).
     + Активное сообщество, свободная лицензия для базовой функциональности.
   * **Ограничения**: Требует определённой технической компетенции для настройки и развёртывания. Возможно избыточно для простых сценариев.
4. **Другие решения** (Chatfuel, Landbot, Flow XO, Диалоговые системы на базе Dialogflow)
   * Предлагают различные варианты визуальных конструкторов, NLP-модулей и интеграций.
   * Могут работать в разных мессенджерах и соцсетях.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Платформа** | **Основной язык/технология** | **Упор на маркетинг/продажи** | **Упрощённая разработка** | **Гибкость для сложных интеграций** | **Поддержка Telegram** |
| ManyChat | Закрытая платформа | Да | Да | Ограничена шаблонами | Частичная |
| Botmother | Закрытая платформа | Частично | Да | Средняя | Хорошая |
| Botpress | Node.js (Open-source) | Нет (больше технический) | Нет (нужны навыки) | Высокая | Хорошая |
| Dialogflow | Google Cloud (NLP) | Нет (больше технический) | Для NLP-сценариев — да | Высокая | Хорошая |

Таблица - Краткое сравнение популярных платформ для создания чат-ботов

**Преимущества интеграции с Telegram**

Несмотря на существование множества платформ и каналов, для российского и ближнего зарубежного рынка часто ключевым выбором становится **Telegram**. К основным плюсам данной среды относятся:

1. **Широкая аудитория**
   * Telegram популярен среди разных возрастных групп.
   * Простая установка и удобный интерфейс, активно используемый в бизнесе.
2. **Встроенные платёжные механизмы**
   * Поддержка Telegram Payments, позволяющая организовать приём оплаты внутри приложения.
   * Возможность взаимодействовать с локальными платёжными провайдерами (например, Яндекс.Касса, Stripe, и другие), в зависимости от страны.
3. **Удобные инструменты разработчика**
   * Открытый API и обширные библиотеки на различных языках (Python, Node.js, PHP и т.д.).
   * Возможность создавать кастомные клавиатуры, инлайн-кнопки, меню, что упрощает пользовательский путь в боте.
4. **Безопасность и надёжность**
   * Данные шифруются, процесс обмена сообщениями хорошо защищён.
   * Сервисы Telegram обычно не сталкиваются с массовыми перебоями.

Таким образом, Telegram является оптимальным решением для чат-бота, ориентированного на российский рынок и ближнее зарубежье, особенно если важны простота взаимодействия, оперативность и возможность встраивать платежи.

### 1.4.2 Сравнение и выбор подхода

**Сравнительный анализ: готовые конструкторы vs. индивидуальная разработка**

Перед началом проектирования системы автоматизации встаёт вопрос о том, использовать ли готовую платформу (конструктор) для чат-ботов или создавать решение «с нуля» (или почти с нуля, опираясь на основные библиотеки). Рассмотрим ключевые критерии выбора:

1. **Уровень технической компетенции**
   * Готовый конструктор: Большинство рутинных моментов скрыто. Подходит, если нет штата разработчиков или времени на изучение тонкостей программирования.
   * Индивидуальная разработка: Требует от разработчика понимания API Telegram, навыков программирования на Python или Node.js, знания баз данных и т.д.
2. **Гибкость и кастомизация**
   * Готовый конструктор: Существует набор типовых шаблонов и решений. Однако при необходимости сложной логики, интеграции с CRM или платёжными системами могут возникнуть ограничения.
   * Индивидуальная разработка: Позволяет создавать нестандартные сценарии, интегрировать любые модули, базы данных, платёжные шлюзы.
3. **Стоимость внедрения и обслуживания**
   * Готовый конструктор: Обычно имеет модель подписки — бесплатный базовый тариф или платные тарифы с расширенным функционалом. При большом числе пользователей стоимость может расти.
   * Индивидуальная разработка: Изначальные затраты (время и средства на создание) могут быть выше, однако отсутствие абонентской платы и возможность развивать проект по мере необходимости.
4. **Поддержка и обновления**
   * Готовый конструктор: Обновляется автоматически, ответственность за масштабирование и исправление ошибок лежит на платформе.
   * Индивидуальная разработка: Все изменения и обновления производятся самостоятельно, требуется мониторинг стабильности системы и своевременное устранение уязвимостей.

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Готовый конструктор** | **Индивидуальная разработка** |
| Техническая компетенция | Требуется минимальная | Нужны навыки программирования, опыт работы с API Telegram |
| Масштабируемость и гибкость | Ограничена функционалом конструктора | Высокая (произвольная логика, любые интеграции) |
| Стоимость | Подписка + потенциальные тарифные ограничения | Первоначальные вложения в разработку, далее контроль затрат |
| Контроль за данными | Часто размещение на серверах провайдера | Полный контроль (собственный сервер, своя БД) |
| Скорость внедрения | Быстро (готовые шаблоны, визуальный интерфейс) | Дольше (проектирование, написание кода, тестирование) |
| Обновления и поддержка | Автоматические со стороны платформы | Полная ответственность на разработчике/владельце |

Таблица - Сравнение подходов к разработке чат-бота

**Обоснование индивидуальной разработки**

В контексте **малого кондитерского бизнеса**, где необходима автоматизация уникальных процессов (например, учёт специфических ингредиентов, пошаговый выбор декора, учёт праздничных дат), **индивидуальная разработка** видится более привлекательным вариантом по ряду причин:

1. **Уникальные сценарии диалога и ветвления**
   * Некоторые платформы ограничены базовыми схемами (заказ — уточнение — оплата). Если нужно внедрять индивидуальные вопросы по декору, ограничения по ингредиентам, выбор предоплаты или полной оплаты, готовые конструкты могут не дать всей необходимой гибкости.
2. **Интеграция с локальными системами**
   * Может потребоваться соединение с локальным Google-таблицами (если владелец предпочитает их) или собственной небольшой CRM. С помощью индивидуальной разработки можно точно настроить обмен данными и их структуру.
3. **Расширение функционала в будущем**
   * Возможно добавлять модули для автоматической генерации накладных, систем аналитики, интеграции с сервисами рассылок и многого другого. Готовые платформы часто имеют строгие лимиты на число функций или типов интеграций.
4. **Контроль над расходами и безопасностью**
   * При использовании конструктора есть вероятность повышения тарифов, а данные хранятся на стороне сервиса. Индивидуальное решение даёт полный контроль над тем, где и как будут храниться данные о клиентах и заказах.

Таким образом, для бизнеса, желающего выстроить **долгосрочную стратегию** с возможностью масштабирования и глубокой кастомизации, индивидуальная разработка чат-бота в Telegram — наиболее оптимальный выбор. Этот подход позволяет учесть все нюансы и специфичные требования, минимизируя риски и создавая устойчивый фундамент для развития.

## 1.5 Обоснование выбора направления и технологии автоматизации

### 1.5.1 Критерии выбора

При выборе направления и технологии автоматизации важно учитывать, как бизнес-требования, так и возможности конкретного инструментария. Ниже представлены ключевые критерии:

1. **Удобство для пользователя (Telegram)**
   * **Распространённость платформы**: Клиентам не нужно устанавливать дополнительное ПО, так как Telegram уже широко используется.
   * **Интуитивный интерфейс**: Бот в Telegram может предлагать кнопки, меню и уведомления, упрощающие взаимодействие.
   * **Мультиплатформенность**: Telegram доступен на мобильных платформах (iOS, Android), а также в веб-версии и десктопном приложении.
2. **Низкий порог входа для бизнеса (достаточно простая реализация)**
   * **Отсутствие необходимости сложных инфраструктур**: Для старта может хватить недорогого хостинга или облачного сервиса.
   * **Простая настройка**: Используя готовые библиотеки (например, aiogram для Python или Telegraf для Node.js), можно относительно быстро создать базовый функционал.
   * **Этапность внедрения**: Возможность запустить MVP (минимально жизнеспособный продукт) за короткое время и постепенно расширять функционал.
3. **Возможность расширения: интеграция с CRM, платёжными системами и др.**
   * **Гибкая архитектура**: Индивидуальная разработка через Telegram API или специализированные фреймворки позволяет добавлять новые модули (CRM, оповещения, аналитику).
   * **Наличие встроенного механизма платежей**: Telegram Payments даёт возможность организовать удобную систему предоплаты или полной оплаты без выхода из мессенджера.
   * **База данных и аналитика**: В перспективе бот может собирать статистику заказов, проводить сегментацию клиентов и интегрироваться с рекомендательными сервисами.

|  |  |  |
| --- | --- | --- |
| **Критерий выбора** | **Обоснование** | **Выгоды для бизнеса** |
| Удобство (Telegram) | - Широкая аудитория - Простота взаимодействия - Возможность отправки мультимедиа, клавиатур, уведомлений | - Лёгкое привлечение клиентов - Высокий отклик и конверсия |
| Низкий порог входа | - Быстрый старт - Достаточно простой хостинг - Готовые библиотеки (aiogram, Telegraf) | - Минимизация затрат на внедрение - Сокращение сроков разработки |
| Возможность расширения | - Интеграции (CRM, платёжные шлюзы) - API Telegram - Масштабирование при росте бизнеса | - Долгосрочная перспектива - Адаптация к меняющимся условиям |

Таблица - Критерии выбора технологии автоматизации

### 1.5.2 Описание ожидаемых результатов

Ожидается, что разработка и внедрение Telegram-бота в процессы малого кондитерского бизнеса позволит достичь целого ряда положительных эффектов.

1. **Ускорение процесса оформления заказа**
   * **Сокращение времени** на уточнение стандартных вопросов (цена, наличие, сроки) за счёт шаблонных сценариев диалога.
   * **Круглосуточная доступность** бота: клиенты могут оставлять заявки и получать оперативную информацию даже в нерабочие часы.
   * **Автоматизированное взаимодействие**: быстрая проверка свободных дат и формирование запроса без ручного ввода менеджером.
2. **Уменьшение вероятности ошибок**
   * **Структурированный ввод данных**: чётко заданный формат для выбора даты, размера торта, декоративных элементов, что снижает риск путаницы.
   * **Единая база заказов**: все обращения аккумулируются в одном месте, исключая проблему «данные разбросаны по разным чатам и записям».
   * **Контроль статусов**: благодаря автоматическим уведомлениям и внутренней системе трекинга можно избежать забытых или потерянных заказов.
3. **Повышение удовлетворённости клиентов за счёт быстрого ответа**
   * **Мгновенная реакция на запрос**: бот отвечает немедленно, отправляет информацию о цене и условиях.
   * **Прозрачность статусов**: клиент может получить уведомление о том, что торт взят в работу, готов или отправлен на доставку.
   * **Оценка качества**: в конце диалога можно предложить клиенту оставить отзыв, что укрепляет доверие и помогает улучшать сервис.

|  |  |  |
| --- | --- | --- |
| **Результат** | **Механизмы достижения** | **Выгоды для бизнеса** |
| Ускорение оформления заказа | - Шаблонные сценарии - Круглосуточный приём заявок | - Экономия времени - Рост числа обрабатываемых заказов |
| Уменьшение ошибок | - Формализованные данные - Единая база клиентов и заказов | - Сокращение случаев пропуска заказов - Снижение недовольства клиентов |
| Повышение удовлетворённости клиентов | - Быстрые ответы - Автоматические уведомления о статусах | - Улучшение репутации - Лояльность и повторные заказы |

Таблица - Ожидаемые результаты от внедрения чат-бота

Таким образом, реализация чат-бота в Telegram отвечает потребностям малого кондитерского бизнеса как в краткосрочной, так и в долгосрочной перспективе. С одной стороны, достигается оперативная оптимизация процесса приёма и учёта заказов; с другой — появляется гибкая основа для последующих улучшений, например, подключения рекомендательных систем или более сложных механизмов аналитики.

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ ТЕЛЕГРАМ-БОТА ДЛЯ АВТОМАТИЗАЦИИ ЗАКАЗОВ

## 2.1 Анализ требований к программному модулю

### 2.1.1 Характеристики задачи автоматизации

Для автоматизации процесса оформления заказов на торты было принято решение использовать телеграм-бот, взаимодействующий с Google Sheets. Данная задача имеет следующие ключевые элементы:

1. **Пользователи**
   * **Рядовые пользователи (заказчики)**: оформляют заказы на торты (выбор вкуса, размера, декора и т.д.).
   * **Администраторы (владельцы/менеджеры)**: осуществляют подтверждение заказов, обновление статуса, просмотр текущих заявок.
2. **Входная информация**
   * От пользователя:
     + Наименование желаемого торта (из каталога).
     + Параметры заказа (вкус, размер, декор).
     + Идентификатор пользователя (определяется автоматически через Telegram ID).
   * От администратора:
     + Команда на просмотр заказов (не «доставленных»).
     + OrderID и новый статус для обновления.
   * Из Google Sheets (лист cakes):
     + Данные каталога тортов (название, цена, описание, ссылка на фото).
   * Из Google Sheets (лист orders):
     + Данные о каждом заказе (OrderID, пользователь, статус и т.д.).
3. **Выходная информация**
   * Для пользователя:
     + Отображение списка доступных тортов (название, цена, описание, фото).
     + Сообщение о принятии заказа (номер заказа).
     + Информация об изменении статуса заказа.
     + Отображение списка заказов пользователя.
   * Для администратора:
     + Список заказов, которые ещё не имеют статус «доставлен».
     + Сообщение об успешном обновлении статуса.
   * Для Google Sheets:
     + Новая строка в листе orders при оформлении заказа.
     + Обновлённый статус заказа по OrderID.
4. **Информационные процессы**
   * **Создание заказа**:
5. Сбор введённых пользователем параметров (вкус, размер, декор).
6. Формирование уникального OrderID и запись в Google Sheets.
7. Уведомление администратора о новом заказе.
   * **Обновление статуса**:
8. Администратор выбирает заказ (OrderID) и вводит новый статус.
9. Статус в Google Sheets обновляется.
10. Пользователь получает уведомление об изменении статуса.
    * **Просмотр заказов**:
11. Пользователь видит только свои заказы (фильтрация по Telegram ID).
12. Администратор видит все заказы, кроме «доставленных».
13. **Показатели**

* **Статус заказа** (ожидается подтверждение администратора / изготовление / готов / доставлен).
* **Количество заказов** для каждого пользователя.
* **Количество заказов** в статусе «ожидается подтверждение администратора» (для администратора).

1. **Методы расчёта**

* **Уникальный OrderID**: вычисляется путём определения последней строки в листе orders и увеличения её номера на 1.
* **Фильтрация заказов** по Telegram ID и текущему статусу: простые операции поиска и сравнения строк в Google Sheets.
* **Передача уведомлений** пользователю или администратору: методы бота (например, bot.send\_message), основанные на Telegram ID/чате.

Таким образом, основной смысл автоматизации — дать пользователю удобный интерфейс (чат-бот), где он может круглосуточно оформить заказ на торт, а администратору — инструменты для просмотра и управления заказами без сложных систем, используя Google Sheets как базу данных.

### 2.1.2 Описание функциональных требований

Функциональные требования к телеграм-боту для автоматизации заказов можно разделить на несколько блоков:

1. **Авторизация пользователей по Telegram ID**
   * Не требуется дополнительная регистрация.
   * Бот автоматически определяет, является ли пользователь администратором, сверяя его ID со списком, хранимым в .env (переменная ADMIN\_IDS).
2. **Круглосуточный приём заказов**
   * Бот должен постоянно прослушивать входящие сообщения (polling или webhook).
   * Пользователи могут в любое время суток сделать заказ, выбрав торт из каталога.
3. **Каталог тортов**
   * Содержит название, цену, описание, а также ссылку на фото (при наличии).
   * Данные о каталоге берутся из листа cakes в Google Sheets.
   * Бот выводит список тортов (название, цена, описание, фото) и даёт возможность выбрать необходимый.
4. **Пошаговое формирование заказа** (FSM: ChoosingCake, ChoosingTaste, ChoosingSize, ChoosingDecor, Confirming)
   * Переход между состояниями происходит по нажатию кнопок (или вводе текста), контролируя корректность ввода (например, проверка, что количество персон — число).
   * При подтверждении заказа (Confirming) формируется запись в Google Sheets (лист orders) с уникальным ID, статусом «ожидается подтверждение администратора», фиксацией даты и прочими параметрами.
5. **Отслеживание статуса**
   * Пользователь в любой момент может нажать кнопку «Статус заказов» и увидеть свои заказы.
   * Заказы сортируются по дате (от нового к старому).
   * Статус заказа изменяется администратором (например, на «изготовление», «готов», «доставлен»).
6. **Режим администратора**
   * Администратор при старте видит другое меню (с кнопками «Просмотреть заказы» и «Обновить статус заказа»).
   * **Просмотр заказов**: выводит только заказы, статус которых отличается от «доставлен».
   * **Обновление статуса заказа**: вводит OrderID и новый статус, бот обновляет запись в Google Sheets, уведомляя соответствующего пользователя.
7. **Уведомления**
   * При создании нового заказа администратор получает сообщение: «Новый заказ №X от пользователя Y».
   * При обновлении статуса бот отправляет пользователю сообщение о том, что статус заказа №X изменён на «new\_status».

Таким образом, бот обеспечивает все основные шаги по оформлению, хранению и управлению заказами тортов: от выбора параметров до уведомления о доставке.

### 2.1.3 Требования к техническому, программному и информационному обеспечению

**1. Техническое обеспечение**

* **Серверная инфраструктура**
  + Запуск Python-скрипта должен осуществляться на постоянно работающем сервере или виртуальной машине (VPS / Docker-контейнер / облачный сервис).
  + Требования к ресурсам минимальны (Python, интернет-соединение).
  + Возможна интеграция с любым сервисом, поддерживающим постоянный фоновый процесс, например Heroku, AWS, DigitalOcean, local server.
* **Надёжное интернет-соединение**
  + Телеграм-бот для приёма заказов должен быть всегда онлайн, чтобы мгновенно реагировать на сообщения пользователей и отправлять уведомления администраторам.

**2. Программное обеспечение**

* **Язык разработки**
  + Python (версия 3.7+), с использованием фреймворка aiogram (версия 3.x), обеспечивающего удобную реализацию FSM и прослушивание сообщений из Telegram API.
* **Библиотеки и зависимости**
  + gspread\_asyncio и oauth2client для работы с Google Sheets.
  + python-dotenv для безопасного хранения токена и прочих приватных данных в .env.
  + logging для логирования (уровень INFO или выше для отладки).
* **Внешние сервисы**
  + **Google Sheets**: хранение списка тортов (лист cakes) и заказов (лист orders).
  + **Telegram API**: отправка и приём сообщений, кнопок, оповещений.
* **Функциональные требования к ПО**
  + **Отвечать пользователям быстро**, без длительных задержек.
  + **Обеспечивать понятный интерфейс** (кнопки для создания заказа, просмотра статуса и т.д.).
  + **Защита от несанкционированного доступа**:
    - Администраторские функции доступны только Telegram ID, прописанным в ADMIN\_IDS.
    - Данные о заказах не разглашаются другим пользователям (фильтрация по Telegram ID).

**3. Информационное обеспечение**

* **Данные для работы бота**
  1. **.env файл**
     + Токен бота (BOT\_TOKEN).
     + Список администраторов (ADMIN\_IDS).
     + Названия Google Sheets (SPREADSHEET\_NAME, листы ORDERS\_SHEET\_NAME, CAKES\_SHEET\_NAME).
  2. **Учетные данные сервисного аккаунта** (credentials.json)
     + Доступ к Google Sheets: сервисному аккаунту выданы права Editor.
  3. **Таблицы в Google Sheets**
     + cakes (столбцы: name, price, photo, description) — каталог тортов.
     + orders (столбцы: OrderID, user\_id, user\_name, cake\_name, price, taste, size, decor, status, date) — учёт заказов.
* **Способ хранения**
  1. Информация о тортах хранится в листе cakes. При изменении ассортимента администратор редактирует вручную Google Sheets.
  2. Информация о заказах хранится в листе orders. Каждая новая строка соответствует одному заказу.
  3. При обновлении статуса той же строки обновляется ячейка в столбце status.
* **Объём и структура**
  1. Для малого или среднего бизнеса Google Sheets подходит как простая база данных.
  2. В случае роста заказов (до нескольких тысяч записей) Google Sheets остаётся рабочим решением, но может потребоваться переход на более эффективное решение (реляционная БД).

Таким образом, для корректной работы системы требуются:

1. **Физическое или виртуальное устройство** с Python, подключённое к сети 24/7.
2. **Настроенная учётная запись Google** с доступом к Google Sheets и добавленный сервисный аккаунт.
3. **Указанный токен бота** и список администраторов в .env.

Все эти требования образуют **техническое, программное и информационное обеспечение**, без которого автоматизированный приём заказов в телеграм-боте невозможен.

## 2.2 Разработка чат-бота и сценарии его работы

### 2.2.1 Структура и логика работы бота

**Объединённый бот**

**Основная идея** проекта заключается в том, чтобы один и тот же Телеграм-бот обслуживал сразу две различные роли:

1. **Пользователь** (клиент, оформляющий заказ):
   * Может просматривать каталог тортов и оформлять заказы.
   * Имеет доступ к пользовательскому меню.
   * Может просматривать статусы собственных заказов.
2. **Администратор**:
   * Имеет доступ к специальному администраторскому меню.
   * Может просматривать все заказы или только те, что ожидают подтверждения.
   * Может изменять статус заказа (например, «ожидается подтверждение», «изготовление», «доставлен» и т.д.).

**Распознавание роли** происходит при первом обращении к боту (команда /start или нажатие «Запустить»). Бот сверяется со списком ADMIN\_IDS, хранящемся в конфигурации (например, в файле .env), и если user\_id пользователя присутствует в этом списке, то пользователю показывается администраторское меню; если нет — пользовательское.

**Ветвление меню**

Вместо использования набора текстовых команд, в боте реализовано **кнопочное меню** (ReplyKeyboardMarkup). После /start пользователю показывается одна из **двух клавиатур**:

1. **Меню для пользователя**:
   * **«Сделать заказ»** Вызывает сценарий оформления заказа. Включает последовательные вопросы о выборе торта, предпочтительном вкусе, декоре и т.д.
   * **«Статус заказов»** Отображает список оформленных заказов текущего пользователя. Данные загружаются из Google Sheets, сортируются по дате и отправляются в чат.
   * **«Отмена»** Позволяет прервать текущий сценарий и вернуться к главному пользовательскому меню.
2. **Меню для администратора**:
   * **«Просмотреть заказы»** Позволяет увидеть все заказы (или только те, которые не имеют статус «Доставлен»). Используется для определения, какие заказы нуждаются в подтверждении.
   * **«Обновить статус заказа»** Запрашивает у администратора номер заказа (OrderID) и новый статус, после чего в таблице orders обновляется соответствующая запись, а пользователь получает уведомление.
   * **«Отмена»** Выполняет сброс всех текущих действий и возвращает админа в главное админ-меню.

Таким образом, **ветвление** на пользовательское или администраторское меню предоставляет удобный способ разделить функциональность, не используя «сложные» текстовые команды.

**Состояния (FSM)**

Для **оформления заказа** бот использует механизмы конечных автоматов (Finite State Machine, FSM) из библиотеки **aiogram**. Последовательность шагов отражена в **пяти состояниях**:

1. **ChoosingCake** (выбор торта) Пользователь вводит/вводит название торта или выбирает из каталога.
2. **ChoosingTaste** (выбор вкуса) Запрос текстового ввода о желаемом вкусе.
3. **ChoosingSize** (выбор количества персон) Запрос числового ввода, который указывает, на сколько персон требуется торт.
4. **ChoosingDecor** (уточнение декора) Пользователь описывает желаемый декор (ягоды, фигурки, надписи и т.д.).
5. **Confirming** (подтверждение) Отображается итог заказа (название торта, вкус, декор, цена и т.д.) и пользователю предлагают подтвердить или отменить заказ.

Всякий раз, когда пользователь нажимает «Отмена», бот сбрасывает текущее состояние и возвращает его в главное пользовательское меню.

**Google Sheets**

Для **хранения и обновления** данных о заказах используется **онлайн-таблица** (Google Sheets).

1. **Лист cakes**
   * Содержит каталог тортов с полями: name, price, photo, description.
   * При выводе каталога бот динамически загружает эти данные, чтобы отправить в чат информацию обо всех доступных тортах.
2. **Лист orders**
   * Хранит оформленные заказы.
   * Каждая строка содержит:
     + OrderID (уникальный номер заказа),
     + user\_id (Telegram ID пользователя),
     + user\_name (имя пользователя или юзернейм),
     + cake\_name,
     + taste,
     + size,
     + decor,
     + status,
     + date (дата и время оформления).

**Процесс оформления заказа**:

* Когда пользователь подтверждает заказ на последнем шаге (состояние Confirming), бот **добавляет** новую запись в лист orders:
* Статус по умолчанию устанавливается как **«ожидается подтверждение администратора»**.

**Обновление статуса** (администратор):

* При нажатии на кнопку «Обновить статус заказа» администратор вводит OrderID и желаемый статус.
* Бот находит соответствующую строку в листе orders и изменяет поле status.
* После обновления бот отправляет уведомление пользователю, чтобы он мог увидеть изменения в своем заказе.

### 2.2.2 Программная реализация

С полным кодом телеграмм бота можно ознакомиться в Приложении 1. В данном разделе будет рассматриваться общая структура.

Программная часть бота выполнена с использованием **Python** и библиотеки **aiogram** (v3), что обеспечивает асинхронную работу и гибкое управление состояниями (FSM). Для хранения заказов и каталога тортов используется **Google Sheets** через асинхронную библиотеку **gspread\_asyncio**.

**Общая структура кода**

1. **Импорт библиотек и инициализаци** В начале файла combined\_bot.py импортируются необходимые библиотеки: logging, asyncio, datetime, gspread\_asyncio, aiogram и др. Затем загружаются переменные из .env (токен бота, список администраторов, данные для Google Sheets).
2. **Инициализация бота** Создаётся объект Bot с токеном BOT\_TOKEN, и объект Dispatcher с MemoryStorage для FSM.

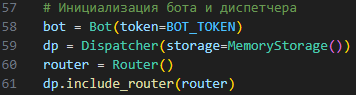


Рисунок - Инициализация бота

1. **Описание классов состояний (FSM)** Объявляются два класса состояний — OrderStates (для сценария оформления заказа) и AdminStates (для сценария обновления статуса заказа).

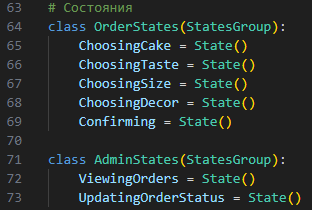


Рисунок - Описание классов состояний (FSM)

1. **Объявление меню** Два набора кнопок (ReplyKeyboardMarkup) — для **пользователей** (user\_menu) и **администраторов** (admin\_menu).

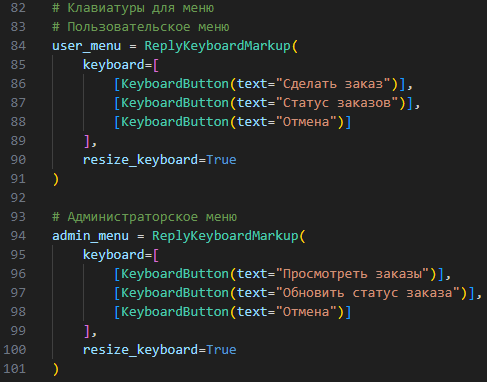


Рисунок - Объявление меню

**Ключевые функции и связь с Google Sheets**

**1. Инициализация соединения с Google Sheets**

Для связи с Google Sheets используется библиотека gspread\_asyncio. Определяется функция get\_gspread\_client, в которой создаётся AsyncioGspreadClientManager, и затем бот авторизуется на основе credentials.json.

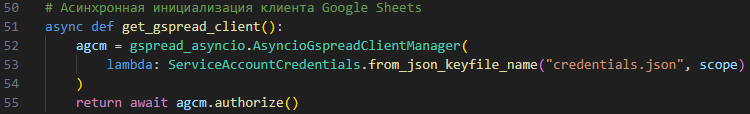


Рисунок - Инициализация соединения с Google Sheets

После этого в main() сохраняется глобальная переменная gc (Google Client), с помощью которой в дальнейшем открываются таблицы и обрабатываются листы.

**2. Чтение каталога тортов из листа cakes**

Функция get\_catalog\_of\_cakes отвечает за чтение каталога. Здесь открывается нужная таблица SPREADSHEET\_NAME, выбирается лист CAKES\_SHEET\_NAME, и вызывается метод get\_all\_records(). Результат возвращается в виде списка словарей.

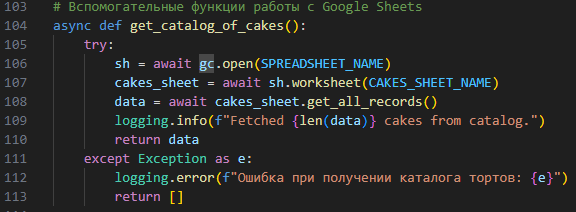


Рисунок - Чтение каталога тортов из листа cakes

В рамках программы каталог нужен для отображения тортов в чате: бот перебирает записи и отправляет пользователю описание и фото.

**3. Чтение и запись заказов (лист orders)**

* **Чтение заказов** (get\_all\_orders, get\_all\_orders\_by\_user):
  + Открывается та же таблица, но лист ORDERS\_SHEET\_NAME.
  + Вызывается get\_all\_records(), и на его основе формируется список всех заказов или фильтруется по user\_id (в случае get\_all\_orders\_by\_user).

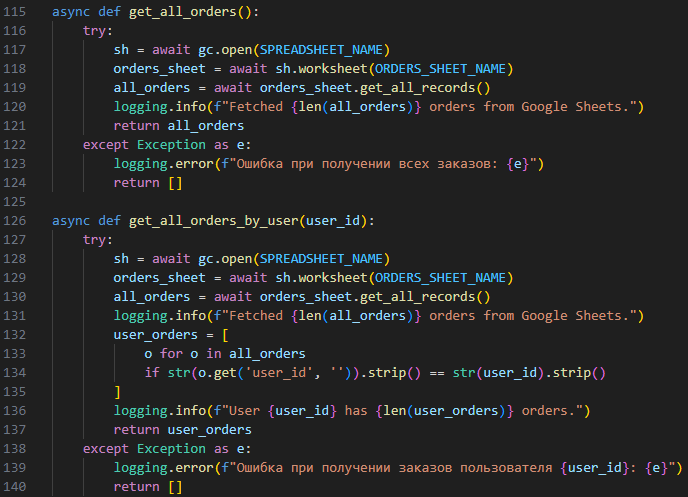


Рисунок - Чтение заказов

* **Создание нового заказа** (create\_new\_order):
  + Прежде чем добавить новую строку, функция получает список всех значений (get\_all\_values()) для вычисления нового OrderID.
  + Подготовленные данные (OrderID, user\_id, user\_name, cake\_name и т.д.) дополняются полем status (по умолчанию «ожидается подтверждение администратора») и полем даты (current\_date).
  + Далее вызывается append\_row([...]), чтобы записать новую строку в таблицу.

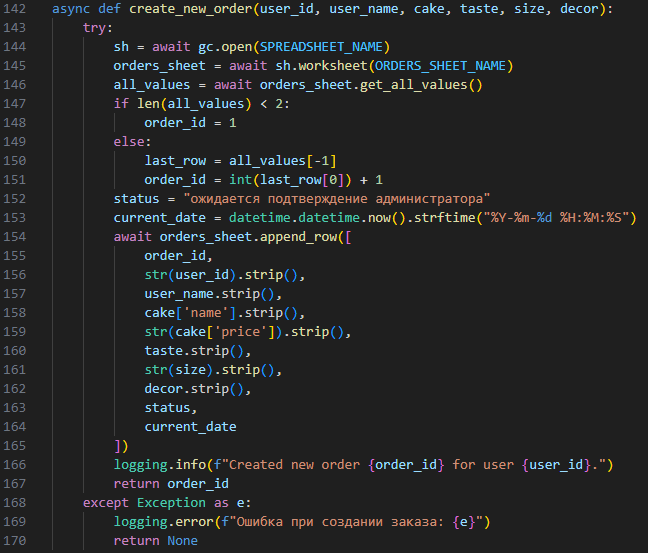


Рисунок - Создание нового заказа

* **Обновление статуса** (update\_order\_status):
  + Функция считывает все заказы, ищет нужный OrderID и по его индексу в таблице вызывает update\_cell, меняя поле status.
  + После успешного обновления логируется запись об изменении, а бот может уведомить пользователя.

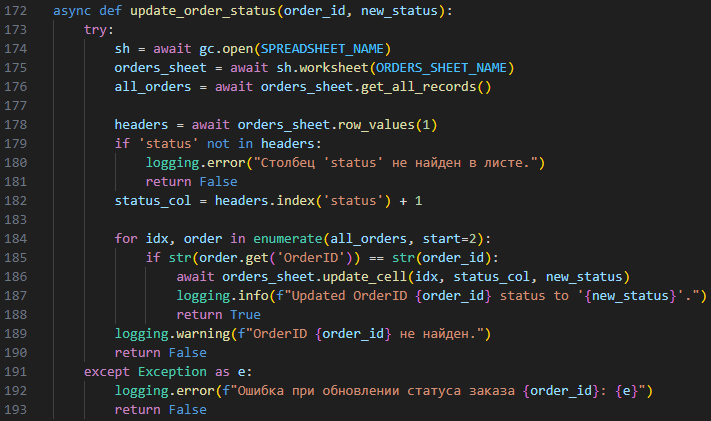


Рисунок - Обновление статуса



Рисунок – Таблица OrdersForCakes лист cakes

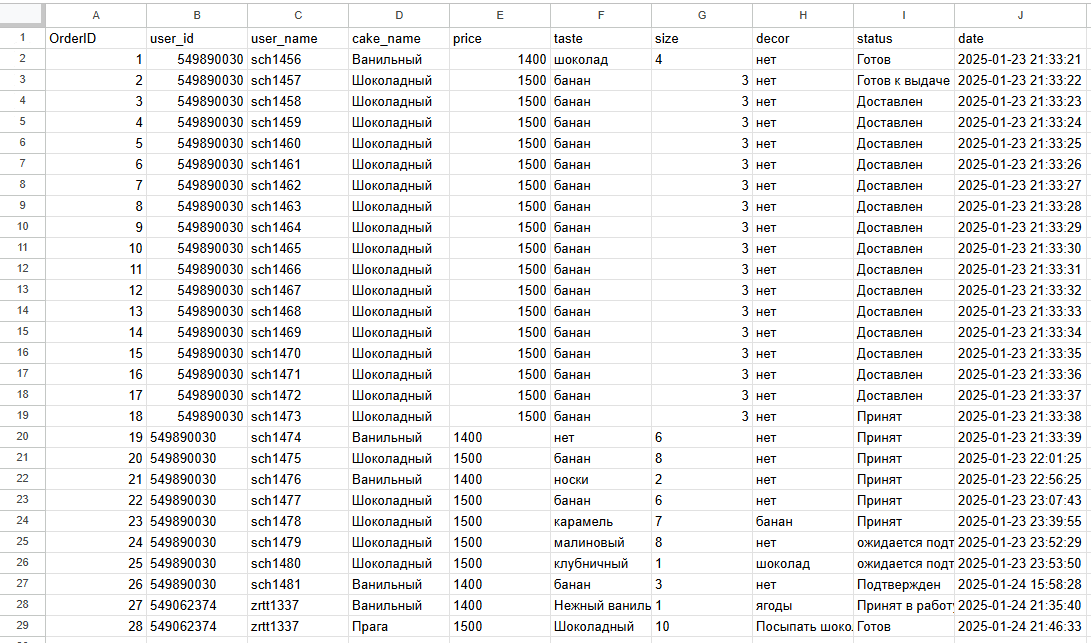


Рисунок - Таблица OrdersForCakes лист orders

Таким образом, **Google Sheets** используется как своеобразная «база данных», но в удобном и привычном для небольшого бизнеса формате онлайн-таблицы.

**Обработчики событий (Router)**

**1. handle\_start (обработка /start)**

После вызова команды /start бот:

1. Очищает текущее состояние FSM (await state.clear()).
2. Проверяет is\_admin(user\_id).
3. Если админ, показывает admin\_menu; иначе — user\_menu.

Это позволяет одной и той же точке входа (/start) обслуживать разные сценарии в зависимости от роли пользователя.

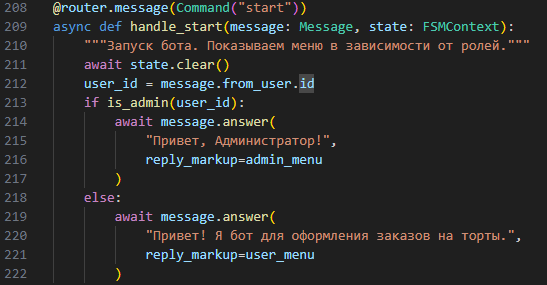


Рисунок - Обработка /start

**2. Обработка кнопки «Отмена»**

Во всех сценариях, если пользователь нажимает **«Отмена»**, бот сбрасывает текущее состояние и возвращает пользователя в главное меню (пользовательское или администраторское).

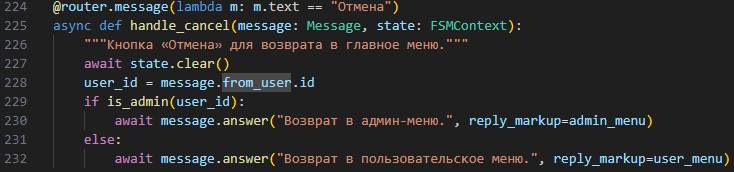


Рисунок - Обработка кнопки «Отмена»

**3. Пользовательский сценарий: «Сделать заказ»**

* **Сделать заказ** — пользователь видит список тортов (название, цена, описание, фото).
* Далее бот поэтапно спрашивает о вкусе, размере, декоре. Каждый вопрос обрабатывается отдельной функцией в соответствующем состоянии FSM: ChoosingCake, ChoosingTaste, ChoosingSize, ChoosingDecor.
* В конце (Confirming) бот предлагает ответить **«Да»** (подтвердить) или **«Нет»** (отменить).
* Если ответ **«Да»**, вызывается create\_new\_order; если успех — выводится сообщение, а администратор получает уведомление.

**4. Пользовательский сценарий: «Статус заказов»**

По кнопке **«Статус заказов»** вызывается функция, которая:

1. Считывает заказы пользователя через get\_all\_orders\_by\_user(user\_id).
2. Сортирует их по дате и отправляет в чат, разбивая на страницы, чтобы избежать слишком длинного сообщения.

**5. Администраторские сценарии**

* **«Просмотреть заказы»** — фильтрация заказов со статусом, отличным от «Доставлен» (ожидающие подтверждения и т.д.).
* **«Обновить статус заказа»** — бот просит ввести OrderID и новый статус; после ввода обновляет Google Sheets через update\_order\_status. Далее отправляется уведомление пользователю (send\_status\_update).

**Итоги реализации**

Таким образом, **программная часть** чат-бота состоит из:

1. **Главного файла** combined\_bot.py, где:
   * Импортируются библиотеки и настраивается логирование.
   * Инициируется **Bot**, **Dispatcher** и **Router**.
   * Объявляются **состояния** (FSM) и **меню** (ReplyKeyboardMarkup).
   * Описываются **обработчики** событий (сообщений, состояний) для пользователя и администратора.
2. **Файла** credentials.json (конфиденциального) с ключом для доступа к Google Sheets.
3. **Файла** .env, содержащего токен бота и список админов, а также имена листов Google Sheets.

Все взаимодействие с Google Sheets выполнено в **асинхронном** стиле, что позволяет обрабатывать множество запросов параллельно, не блокируя других пользователей. Применение конечных автоматов (FSM) облегчает поддержание логики пошаговых сценариев (оформление заказа), что обеспечивает удобство использования и сводит к минимуму возможность ошибок или путаницы во вводе данных.

### 2.2.3 Сценарии взаимодействия

В целях удобства описания сценарии разделены на две части:

1. **Действия пользователя** (клиента, оформляющего заказ).
2. **Действия администратора** (просмотр и обновление заказов).

**1. Действия пользователя**

**1.1 Запуск бота (команда /start)**

1. Пользователь открывает чат с ботом и вводит (или нажимает) команду /start.
2. Бот выполняет проверку: **если** user\_id пользователя не входит в список администраторов, **отправляет** приветственное сообщение и показывает **пользовательское меню** (кнопки: «Сделать заказ», «Статус заказов», «Отмена»).

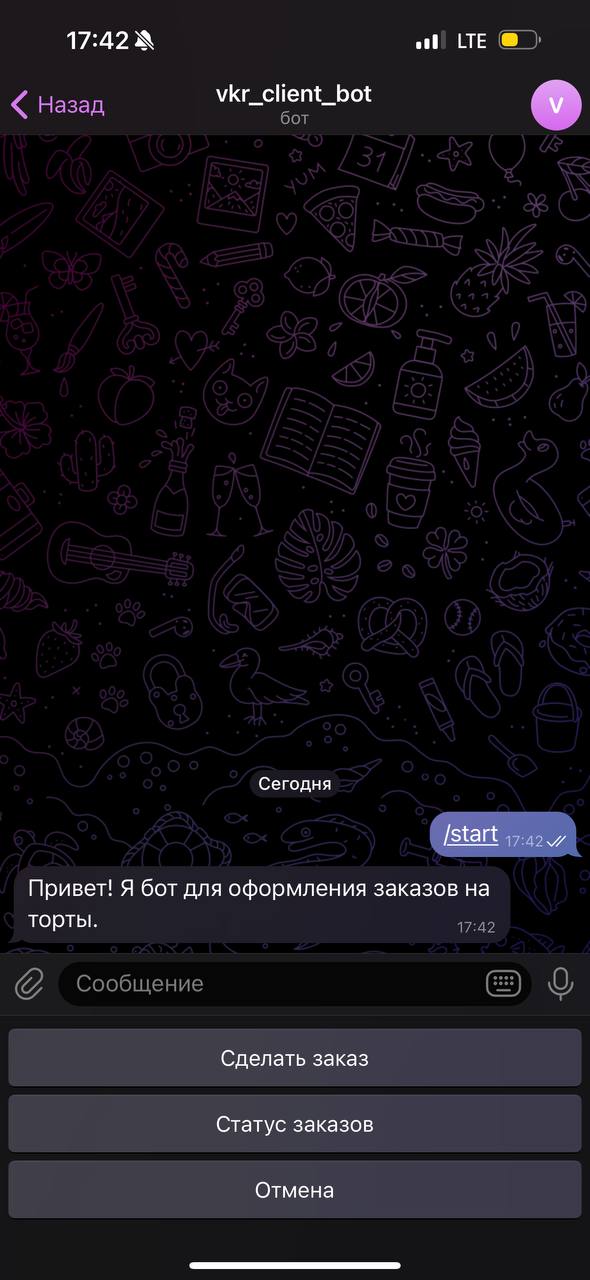


Рисунок - Запуск бота

**1.2 «Сделать заказ»**

1. Пользователь нажимает кнопку **«Сделать заказ»**.
2. Бот загружает каталог тортов из Google Sheets (лист cakes), итеративно отправляет сообщения с описанием и фото (если указано в таблице).
3. После отправки каталога бот спрашивает: **«Введите название торта»** (клавиатура с одной кнопкой «Отмена» для прерывания процесса).

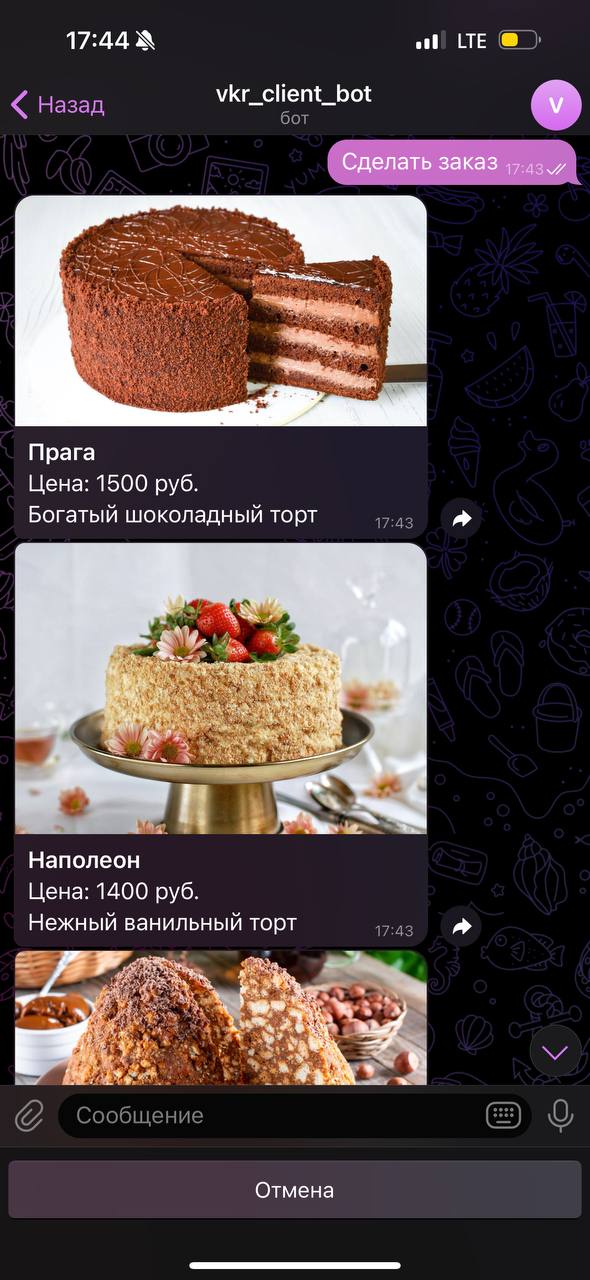


Рисунок - Отображение каталога

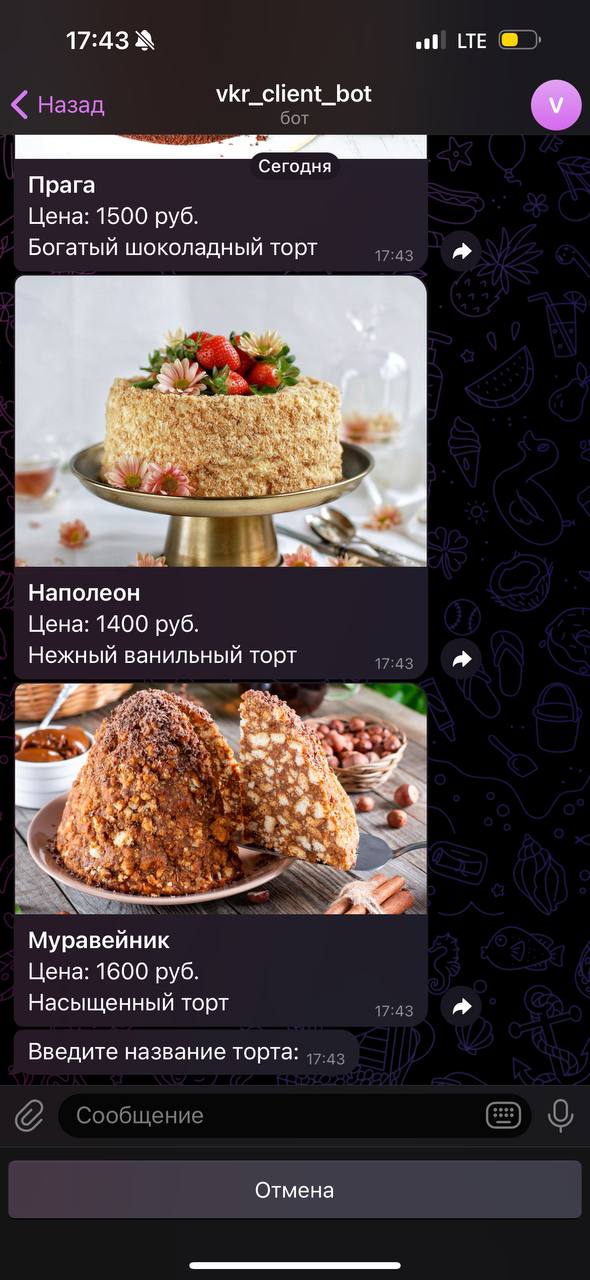


Рисунок - Сообщение бота: «Введите название торта»

**1.2.1 Выбор торта (состояние ChoosingCake)**

1. Пользователь **вводит название** нужного торта (например, «Шоколадный»).
2. Бот проверяет, есть ли такой торт в каталоге. **Если** нет, отвечает «Такого торта нет…». **Если** да, переходит к следующему вопросу: **«Какой вкус вы предпочитаете?»**.

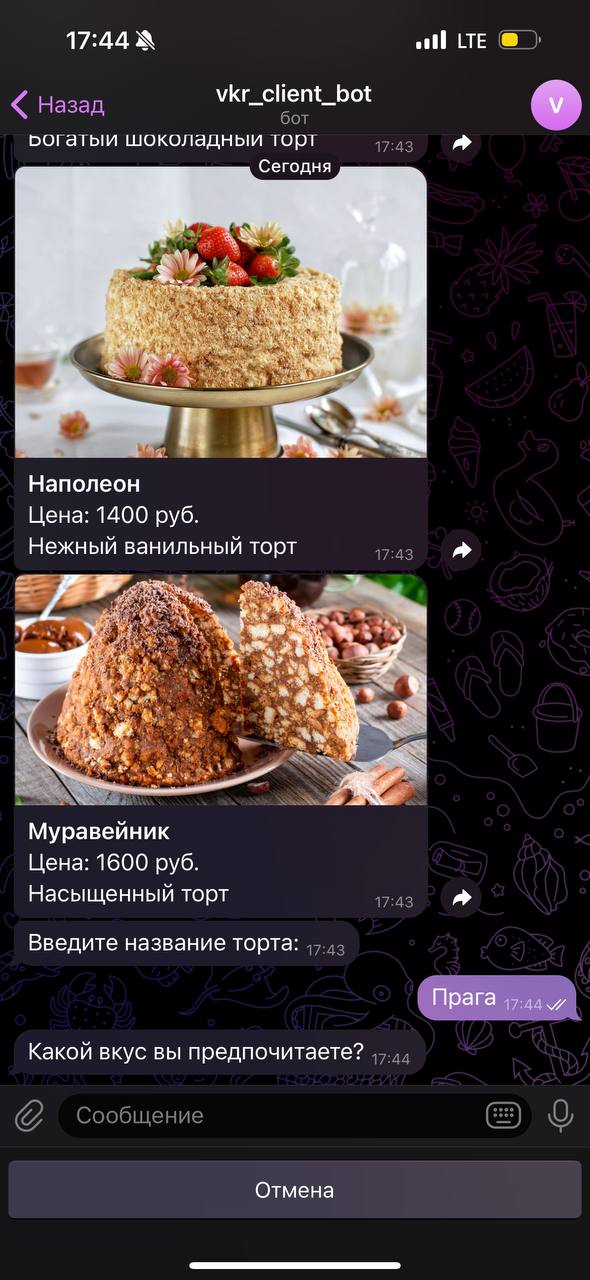


Рисунок - Пользователь ввёл «Шоколадный», бот переходит к вопросу о вкусе

**1.2.2 Выбор вкуса (состояние ChoosingTaste)**

1. Пользователь вводит, например, «клубника».
2. Бот сохраняет эту строку во временном хранилище FSM и спрашивает **«На сколько персон?»**.

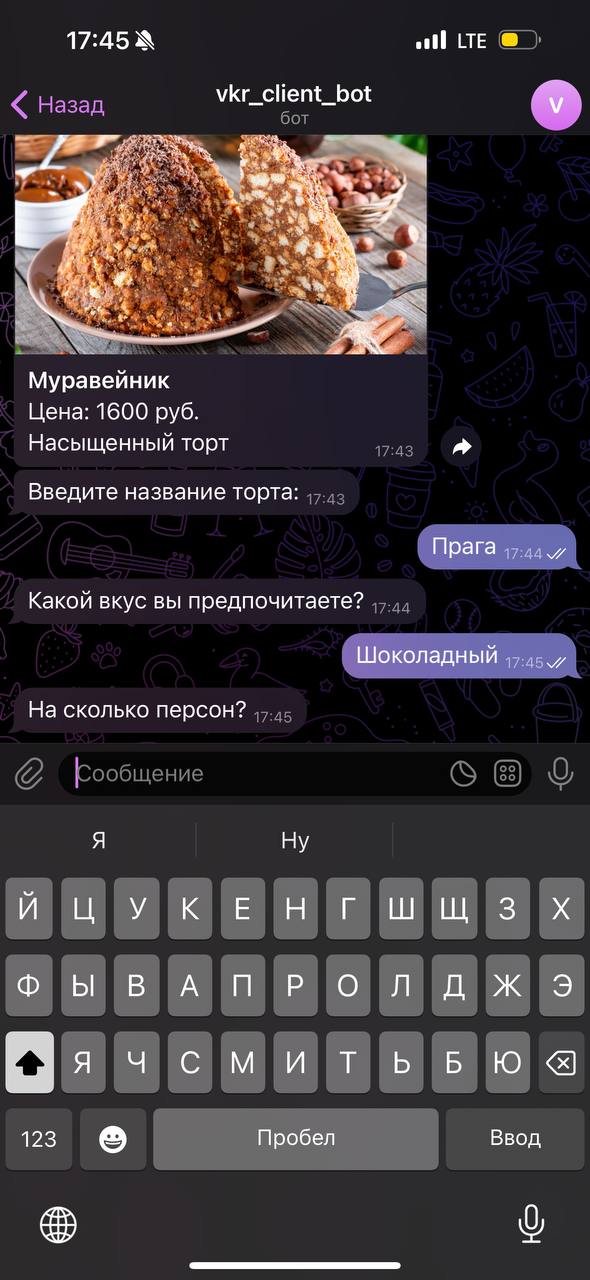


Рисунок - Пользователь получил сообщение: «На сколько персон?».

**1.2.3 Выбор количества персон (состояние ChoosingSize)**

1. Пользователь вводит число (например, «10»).
2. Бот валидирует (проверяет, действительно ли это число). **Если** не число, просит ввести заново.
3. Переходит к вопросу о декоре: **«Какой декор?»**.

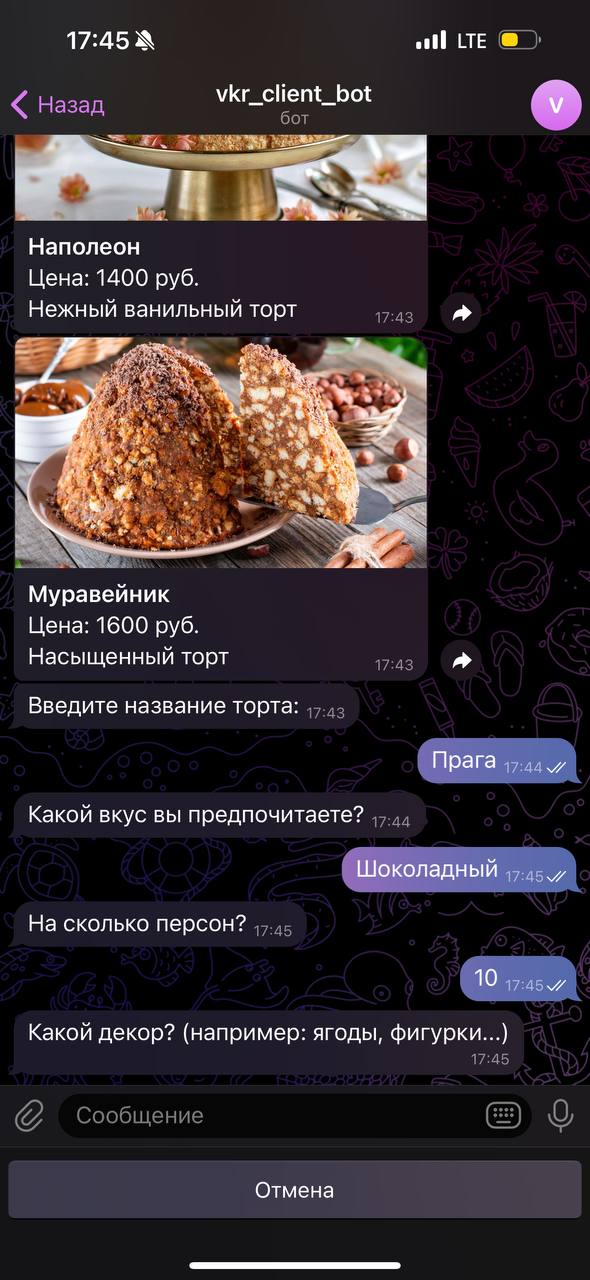


Рисунок - Пользователь ввёл количество персон, бот спрашивает про декор

**1.2.4 Выбор декора (состояние ChoosingDecor)**

1. Пользователь вводит произвольную строку (например: «Ягоды и надпись 'С Днём Рождения'»).
2. Бот сохраняет это и выводит **итог**: название торта, вкус, количество персон, декор, и спрашивает: «Отправьте 'Да' для подтверждения или 'Нет' для отмены».

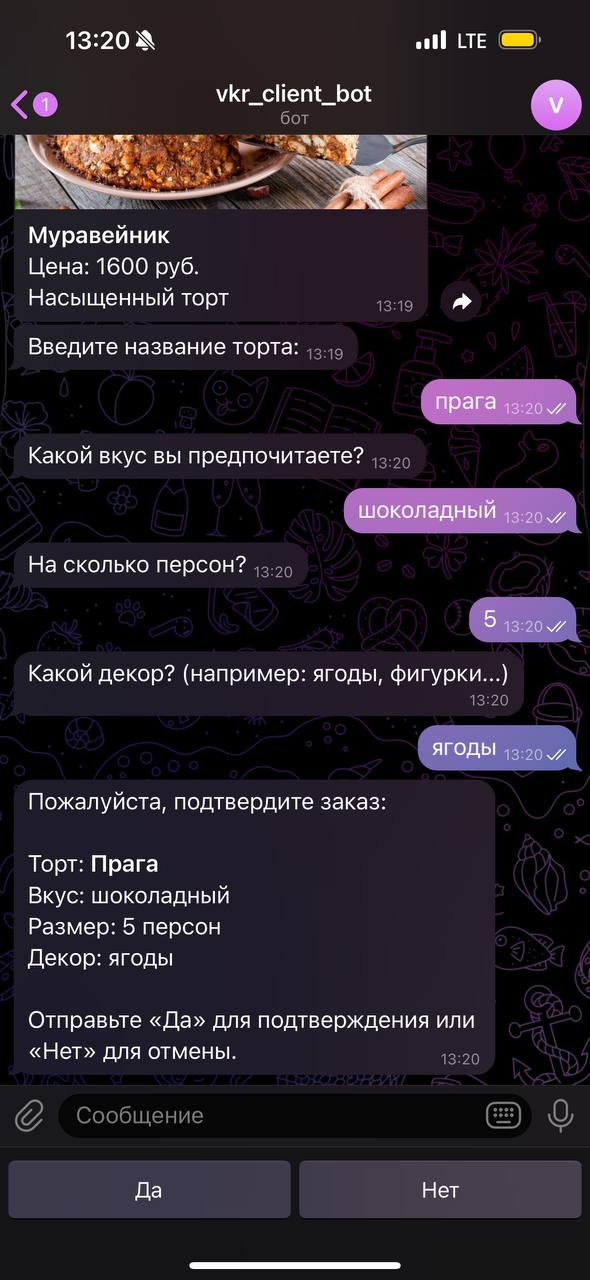


Рисунок - Сообщение бота с данными заказа и кнопками «Да» / «Нет»

**1.2.5 Подтверждение (состояние Confirming)**

1. Пользователь нажимает **«Да»**.
2. Бот вызывает функцию create\_new\_order, записывает заказ в Google Sheets с **статусом** «ожидается подтверждение администратора».
3. Бот отправляет пользователю финальное сообщение: **«Спасибо! Заказ #<номер> оформлен»** + «Ожидается подтверждение администратора».
4. Дополнительно бот уведомляет всех администраторов через функцию bot.send\_message (в цикле по ADMIN\_IDS).

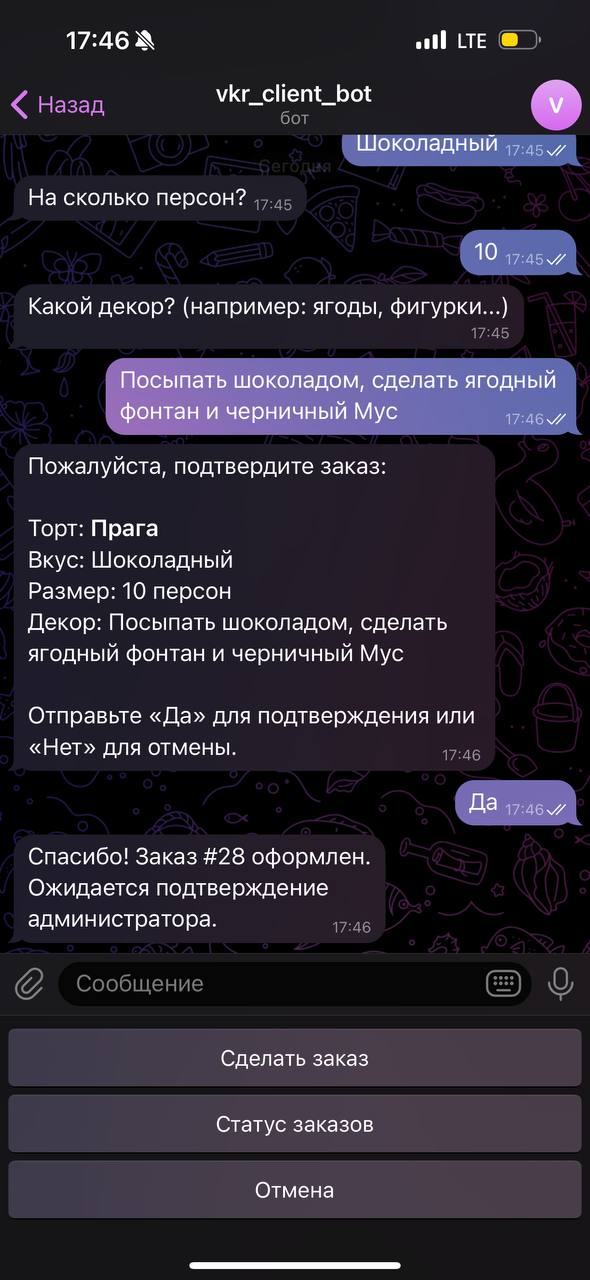


Рисунок - Пользователь видит сообщение с номером заказа.

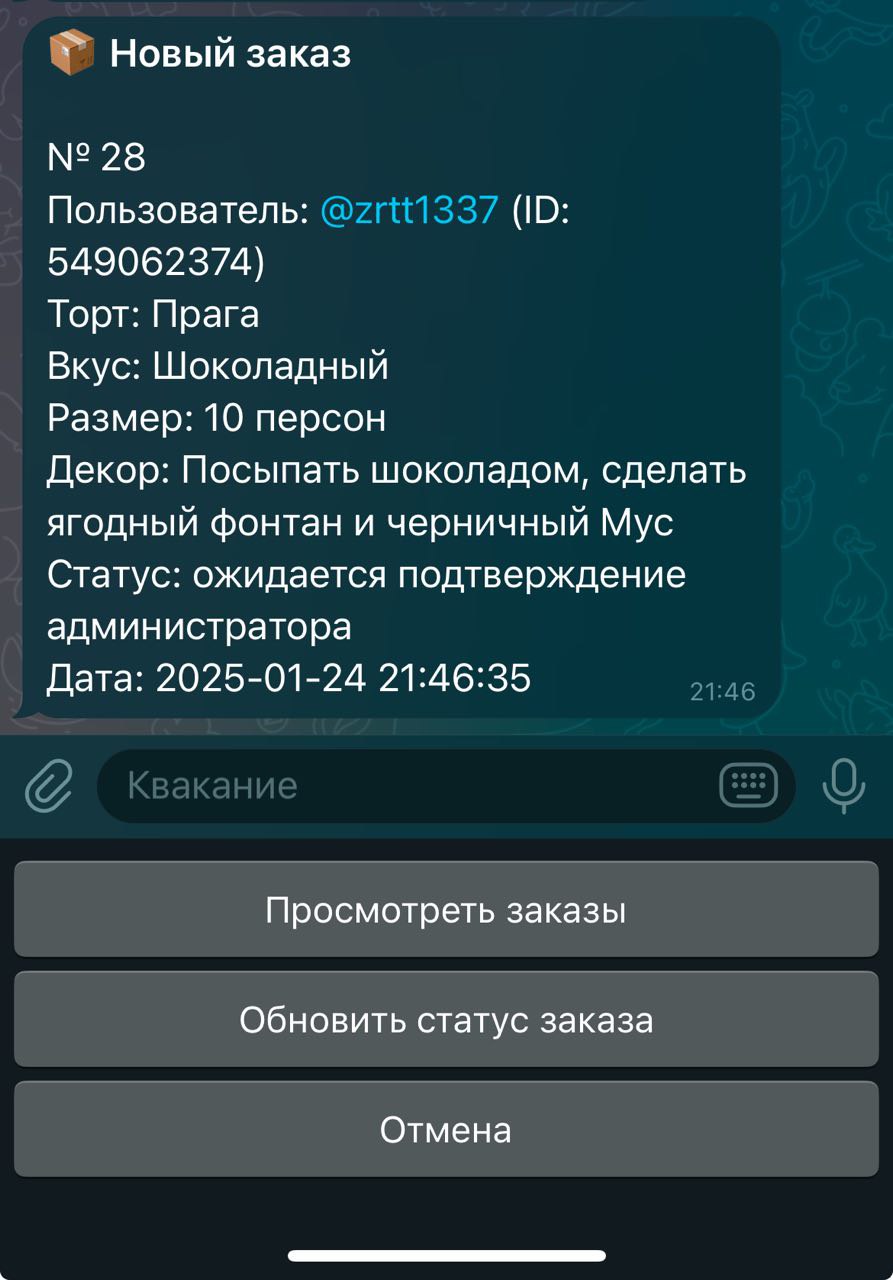


Рисунок - Пример сообщения, которое пришло администратору о новом заказе.

Если пользователь нажимает **«Нет»**, бот отвечает: «Заказ отменён» и возвращает пользователя в главное меню.

**1.3 «Статус заказов»**

1. Пользователь нажимает кнопку **«Статус заказов»**.
2. Бот загружает все заказы этого пользователя (через get\_all\_orders\_by\_user(user\_id)), сортирует по дате, **разбивает** на страницы по 5 штук и отправляет по частям.
3. Если заказов нет, бот отвечает: «У вас ещё нет заказов».

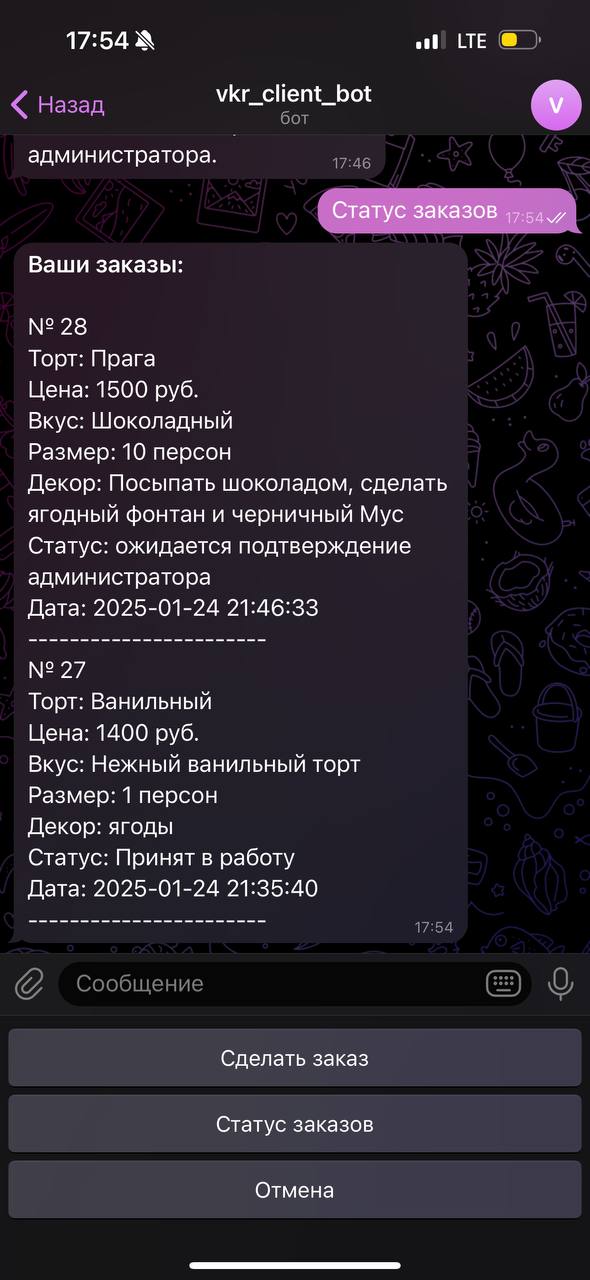


Рисунок - Пример страницы со списком заказов (№, торт, статус, дата)

**2. Действия администратора**

**2.1 Запуск бота (команда /start)**

1. Администратор вводит /start в чате с ботом.
2. Поскольку администраторский ID есть в ADMIN\_IDS, бот приветствует «Привет, Администратор!» и показывает **админ-меню**: «Просмотреть заказы», «Обновить статус заказа», «Отмена».

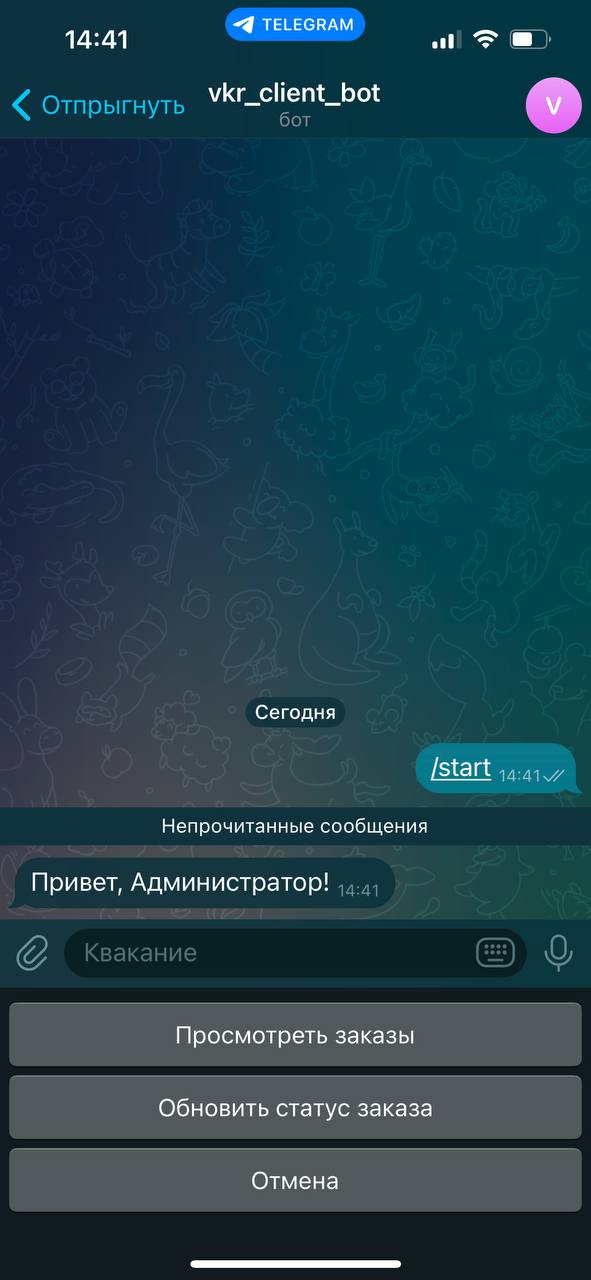


Рисунок - Отображение админ-меню в Telegram

**2.2 «Просмотреть заказы»**

1. Администратор нажимает **«Просмотреть заказы»**.
2. Бот читает лист orders в Google Sheets, фильтрует, исключая заказы со статусом «Доставлен» (предположим, что так организована логика, чтобы показать только ожидающие подтверждения/изготовления).
3. Сортирует заказы по дате (от самых новых) и разбивает на страницы по 10 заказов.
4. Администратору отправляются сообщения со списком заказов: **OrderID**, **Пользователь**, **Торт**, **Статус**, **Дата** и т.д.

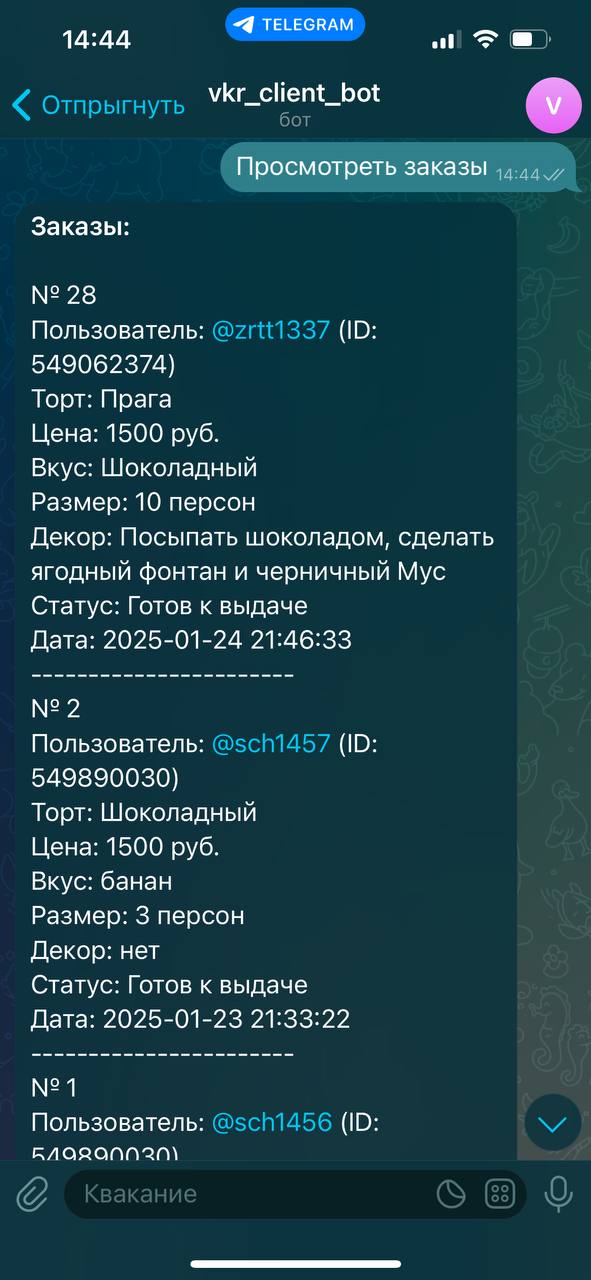


Рисунок - Администратор видит список заказов

**2.3 «Обновить статус заказа»**

1. Администратор нажимает **«Обновить статус заказа»**.
2. Бот просит: **«Введите OrderID и новый статус через пробел. Пример: 1 Доставлен»**. Клавиатура отображается только с кнопкой «Отмена».
3. Администратор вводит, например, «2 Доставлен».
4. Бот парсит ввод: разделяет 2 и Доставлен, вызывает update\_order\_status(2, "Доставлен").
5. Если заказ с OrderID=2 найден, меняется поле status. Если успешно — отправляется уведомление пользователю.
6. Администратор видит сообщение: **«Статус заказа №2 обновлён на 'Доставлен'. Уведомление пользователю отправлено.»**.

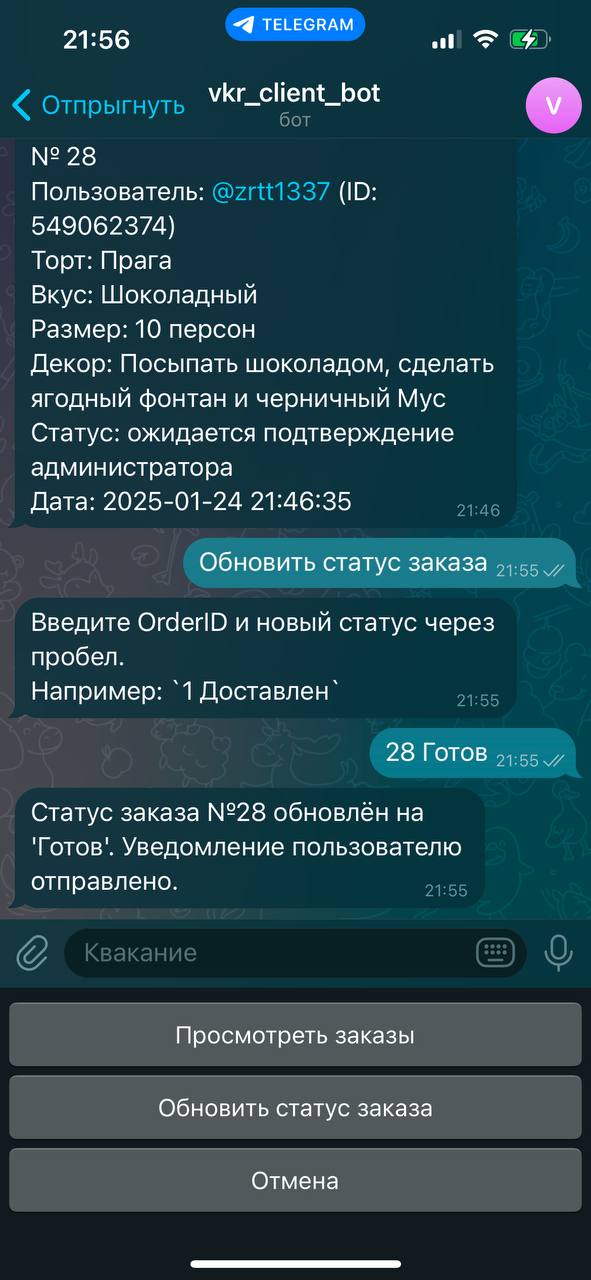


Рисунок - Администратор вводит данные для обновления, бот сообщает о результате

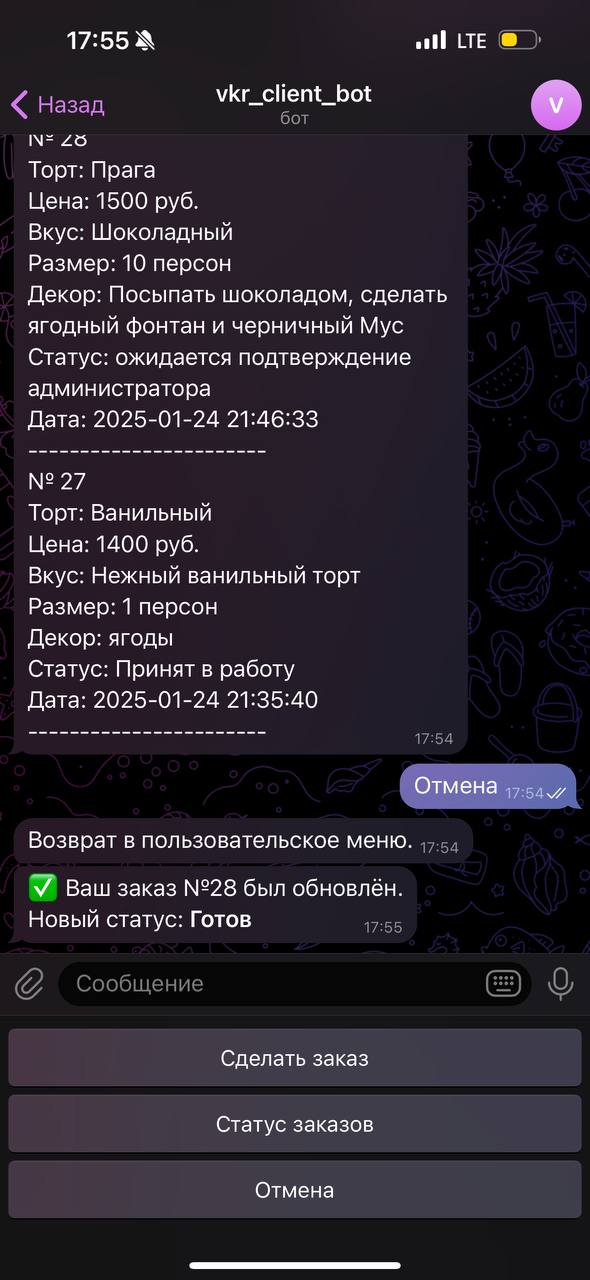


Рисунок - Пользователь (ID=...) получает уведомление «Ваш заказ №2 обновлен, новый статус: Доставлен»

**2.4 «Отмена» (администратор)**

В любой момент администратор может нажать **«Отмена»**, бот сбрасывает состояние (если было), и возвращает в главное админ-меню.

**Таким образом**, сценарии работы бота наглядно демонстрируют, что:

1. **Пользователь** легко оформляет заказ, видя «дружественные» кнопки, и может отслеживать свой статус.
2. **Администратор** оперативно видит новые заказы, меняет статусы и при необходимости общается с пользователем.
3. Google Sheets используется как **централизованный** регистр, в котором хранятся все данные о заказах, что упрощает аналитику и контроль.

# ГЛАВА 3. ОЦЕНКА ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ПРОЕКТА

## 3.1 Расчёт фактических затрат на реализацию

### 3.1.1 Календарно-ресурсное планирование

Разработка чат-бота для автоматизации заказов на торты осуществляется командой из одного разработчика и одного тестировщика. Проект планируется на период **1 недели**, в течение которой выполняются следующие задачи:

1. **Сбор требований (1 день)**
   * Анализ текущих бизнес-процессов оформления заказов.
   * Определение функциональных и нефункциональных требований к боту.
   * Согласование требований с заинтересованными сторонами.
2. **Разработка структуры бота и интеграция с Google Sheets (3 дня)**
   * Создание базовой структуры проекта на Python с использованием библиотеки aiogram.
   * Реализация системы состояний (FSM) для обработки последовательных шагов оформления заказа.
   * Интеграция с Google Sheets через библиотеку gspread\_asyncio для хранения и управления данными заказов и каталога тортов.
3. **Тестирование и отладка (2 дня)**
   * Проведение функционального тестирования всех сценариев работы бота.
   * Выявление и устранение ошибок.
   * Оптимизация кода для повышения производительности и устойчивости.
4. **Подготовка документации (1 день)**
   * Написание технической документации по установке и настройке бота.
   * Создание пользовательских инструкций для администраторов и конечных пользователей.
   * Разработка руководства по эксплуатации и обновлению бота.

**Ресурсное планирование**

Для успешной реализации проекта были выделены следующие ресурсы:

* **Разработчик**:
  + **Трудоёмкость**: 40 человеко-часов
  + **Роль**: Разработка кода бота, интеграция с Google Sheets, реализация FSM, тестирование и отладка.
* **Тестировщик**:
  + **Трудоёмкость**: 10 человеко-часов
  + **Роль**: Проведение тестирования, выявление и документирование ошибок, проверка функциональности.

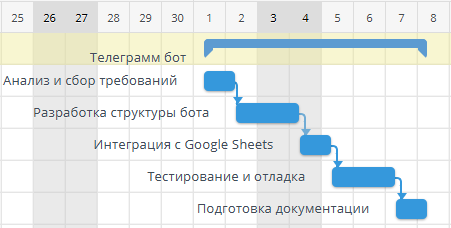


Рисунок - Диаграмма Ганта для календарно-ресурсного планирования проекта чат-бота

### 3.1.2 Анализ затрат

**Фонд оплаты труда**

**Расчёт исходя из трудоёмкости и ставки разработчика:**

* **Разработчик**:
  + **Трудоёмкость**: 40 ч
  + **Ставка**: 1 200 руб./ч
  + **Расчёт**: 40 ч \* 1 200 руб./ч = 48 000 руб.
* **Тестировщик**:
  + **Трудоёмкость**: 10 ч
  + **Ставка**: 800 руб./ч
  + **Расчёт**: 10 ч \* 800 руб./ч = 8 000 руб.

**Итого**: 56 000 рублей.

|  |  |  |  |
| --- | --- | --- | --- |
| **Ресурс** | **Трудоёмкость (ч)** | **Ставка (руб./ч)** | **Общие затраты (руб.)** |
| Разработчик | 40 | 1 200 | 48 000 |
| Тестировщик | 10 | 800 | 8 000 |
| **Итого** |  |  | **56 000** |

Таблица - Расчёт затрат на оплату труда

**Затраты на инфраструктуру**

1. **VPS или локальный сервер**:
   * **VPS**: 2 500 руб./мес.
   * **Локальный сервер**: уже имеется, дополнительных затрат нет.
2. **Интернет-соединение**:
   * Ежемесячные затраты: 1 200 руб.

|  |  |  |  |
| --- | --- | --- | --- |
| **Статья расходов** | **Ед. стоимость (руб.)** | **Количество** | **Общие затраты (руб.)** |
| VPS (1 неделя) | 2 500 | 0.25 | 625 |
| Интернет-соединение (1 неделя) | 1 200 | 0.25 | 300 |
| **Итого** |  |  | **925** |

Таблица - Расчёт затрат на инфраструктуру

## 3.2 Оценка целесообразности проекта

Оценка **целесообразности** проекта основывается на сравнении затрат и выгод, а также на анализе экономической эффективности реализации чат-бота для автоматизации заказов на торты.

**Сравнение «до» и «после»**

1. **Время, которое раньше тратилось на приём заказов вручную**:
   * **До**: Оператор принимал заказы по телефону или через личные сообщения в Telegram, что требовало постоянного присутствия и вручную вводило данные в систему.
   * **После**: Чат-бот автоматизирует процесс приёма заказов, позволяя пользователям самостоятельно оформлять заказы в любое время, без участия оператора.
2. **Уменьшение количества потерянных заказов из-за отсутствия оперативного ответа**:
   * **До**: Возможны были случаи, когда оператор не успевал ответить вовремя, что приводило к потере заказов.
   * **После**: Чат-бот обеспечивает мгновенный ответ на запросы пользователей, что значительно снижает риск потери заказов.
3. **Сокращение ошибок при передаче данных**:
   * **До**: Ввод данных оператором мог приводить к ошибкам при записи информации о заказе.
   * **После**: Автоматизированный процесс минимизирует вероятность ошибок, так как данные вводятся пользователем непосредственно в бот.

**Срок окупаемости**

1. **Оценка ежемесячной экономии**:
   * **Снижение затрат на ручной труд**: Освобождение времени оператора позволяет использовать его ресурсы более эффективно или сократить количество сотрудников.
   * **Увеличение числа заказов**: Благодаря круглосуточной доступности бота, количество потенциальных заказов увеличивается.
2. **Оценка дополнительных заказов (рост прибыли)**:
   * Автоматизация процессов способствует увеличению количества обработанных заказов, что напрямую влияет на рост прибыли компании.

**Пример расчёта срока окупаемости**

**Исходные данные**:

* **Общие затраты**:
  + Фонд оплаты труда: 56 000 руб.
  + Инфраструктура: 925 руб.
  + Техническое обеспечение: 625 руб.
  + **Итого**: 57 550 руб.
* **Ежемесячная экономия (рассматриваем сценарий, когда количество заказов осталось таким же)**:
  + Снижение затрат на оператора: 30 000 руб./мес.
  + **Итого**: 30 000 руб./мес.

**Расчёт срока окупаемости**:

|  |  |
| --- | --- |
| **Статья расходов** | **Сумма (руб.)** |
| Фонд оплаты труда | 56 000 |
| Инфраструктура | 925 |
| Техническое обеспечение | 625 |
| **Итого** | **57 550** |

Таблица - Общие затраты

|  |  |
| --- | --- |
| **Статья экономии** | **Сумма (руб./мес.)** |
| Снижение затрат на оператора | 30 000 |
| **Итого** | **30 000** |

Таблица - Ежемесячная экономия

я

|  |  |
| --- | --- |
| **Показатель** | **Значение** |
| Общие затраты | 57 550 руб. |
| Ежемесячная экономия | 30 000 руб./мес. |
| **Срок окупаемости** | **1,91 месяца (примерно 2 месяца)** |

Таблица - Расчёт срока окупаемости проекта

**Вывод о целесообразности**

На основании проведённого анализа видно, что проект **чрезвычайно целесообразен**. Срок окупаемости составляет примерно **1,91 месяца (примерно 2 месяца)**, что является весьма привлекательным показателем. Дополнительные преимущества включают:

* **Увеличение эффективности работы**: Автоматизация процессов позволяет сосредоточить усилия сотрудников на более важных задачах.
* **Повышение уровня обслуживания клиентов**: Круглосуточная доступность бота улучшает клиентский опыт и способствует лояльности.
* **Снижение операционных рисков**: Автоматизация минимизирует человеческий фактор, что уменьшает вероятность ошибок и потери заказов.

Таким образом, инвестиции в разработку и внедрение чат-бота окупаются в короткие сроки и приносят значительные экономические выгоды компании. Проект способствует повышению общей эффективности бизнес-процессов, улучшению клиентского опыта и снижению операционных затрат.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была разработана и внедрена система автоматизации заказов на торты посредством чат-бота в мессенджере Telegram. Проект включал в себя анализ требований, проектирование и программную реализацию бота, а также оценку экономической эффективности его внедрения. В данном разделе подводятся итоги проделанной работы, оценивается достижение поставленных целей и задач, а также рассматривается потенциал дальнейшего развития и масштабирования проекта.

**Выводы по итогам разработки чат-бота и моделирования бизнес-процессов**

Разработка чат-бота для автоматизации заказов на торты позволила оптимизировать ключевые бизнес-процессы компании, связанные с приёмом и обработкой заказов. Были выполнены следующие основные этапы:

1. **Анализ требований и моделирование бизнес-процессов**: На этапе анализа были выявлены основные потребности как со стороны пользователей (клиентов), так и со стороны администраторов. Была проведена моделировка бизнес-процессов, включающая приём заказов, их обработку, хранение данных и уведомление клиентов о статусе заказов. Это позволило чётко определить функциональные и нефункциональные требования к системе.
2. **Проектирование чат-бота**: Разработана архитектура бота, предусматривающая поддержку двух ролей пользователей — клиентов и администраторов. Использование конечных автоматов (FSM) обеспечило последовательное прохождение пользователем через этапы оформления заказа, что повысило удобство и интуитивность взаимодействия.
3. **Программная реализация**: Бот был реализован с использованием языка программирования Python и библиотеки aiogram для создания Telegram-бота. Для хранения данных о заказах и каталоге тортов использовалась Google Sheets, интеграция с которыми осуществлялась через библиотеку gspread\_asyncio. Были реализованы основные функции: оформление заказа, просмотр статусов заказов, просмотр и обновление заказов администратора, а также уведомления пользователей и администраторов о событиях в системе.
4. **Тестирование и отладка**: Проведено функциональное тестирование всех сценариев взаимодействия пользователей и администраторов с ботом. Были выявлены и устранены ошибки, обеспечена стабильная работа системы при различных сценариях использования.
5. **Оценка экономической эффективности**: Выполнен анализ затрат на разработку и внедрение чат-бота, а также оценена экономия времени и ресурсов, достигаемая за счёт автоматизации процессов. Расчёт показал высокую рентабельность проекта с быстрым сроком окупаемости.

**Достижение целей и решение поставленных задач**

В ходе выполнения проекта были достигнуты все поставленные цели и решены задачи, сформулированные в начале исследования:

1. **Автоматизация процесса приёма заказов**: Чат-бот успешно реализует функционал приёма заказов от пользователей, позволяя им выбирать торт, указывать предпочтения и подтверждать заказ без участия оператора. Это значительно ускорило процесс оформления заказов и снизило нагрузку на персонал.
2. **Улучшение клиентского опыта**: Благодаря интуитивно понятному интерфейсу и последовательным шагам оформления заказа, пользователи могут легко и быстро оформить заказ в любое удобное время, что повышает удовлетворённость клиентов и способствует их лояльности.
3. **Эффективное управление заказами администраторами**: Администраторы получают удобные инструменты для просмотра всех заказов, фильтрации по статусу и обновления статусов заказов. Это позволяет оперативно реагировать на новые заказы и контролировать процесс их выполнения.
4. **Снижение операционных затрат**: Автоматизация процессов позволила сократить время и ресурсы, затрачиваемые на ручную обработку заказов, что напрямую отразилось на снижении общих операционных затрат компании.
5. **Обеспечение точности и уменьшение ошибок**: Автоматизированный ввод данных уменьшил вероятность ошибок при записи заказов, что повысило общую точность и надёжность системы управления заказами.

**Потенциал масштабирования и дальнейшего развития**

Разработанный чат-бот обладает значительным потенциалом для масштабирования и дальнейшего развития, что позволит расширить функциональные возможности системы и интегрировать её с другими инструментами для повышения эффективности бизнеса. Рассмотрим основные направления развития:

1. **Интеграция с CRM-системами**:
   * **Преимущества**: Интеграция чат-бота с CRM-системой позволит централизовать управление клиентскими данными, отслеживать историю взаимодействий и улучшить анализ клиентских предпочтений.
   * **Возможные решения**: Использование API CRM-систем для передачи данных о заказах и клиентах в реальном времени, автоматизация последующих маркетинговых и сервисных мероприятий.
2. **Расширенные маркетинговые инструменты**:
   * **Преимущества**: Возможность проводить таргетированные маркетинговые кампании, отправлять персонализированные предложения и уведомления, повышая тем самым конверсию и удержание клиентов.
   * **Возможные решения**: Внедрение системы рассылок, интеграция с платформами для email-маркетинга и социальными сетями, использование аналитических инструментов для оценки эффективности кампаний.
3. **Добавление функционала аналитики**:
   * **Преимущества**: Получение детализированных данных о продажах, предпочтениях клиентов и эффективности маркетинговых стратегий.
   * **Возможные решения**: Создание дашбордов для визуализации ключевых метрик, автоматизация отчётности и прогнозирования спроса на различные виды тортов.
4. **Поддержка мультиязычности**:
   * **Преимущества**: Расширение аудитории за счёт поддержки нескольких языков, что особенно актуально для компаний с международными клиентами.
   * **Возможные решения**: Реализация многоязычного интерфейса, автоматический перевод сообщений и запросов, адаптация сценариев взаимодействия под различные языковые группы.
5. **Внедрение дополнительных каналов взаимодействия**:
   * **Преимущества**: Повышение доступности и удобства использования чат-бота через различные платформы и устройства.
   * **Возможные решения**: Разработка версии бота для других мессенджеров (WhatsApp, Viber), интеграция с веб-сайтами и мобильными приложениями компании.
6. **Автоматизация доставки и логистики**:
   * **Преимущества**: Полная автоматизация процесса от оформления заказа до его доставки клиенту, что позволяет минимизировать временные задержки и улучшить контроль над процессом.
   * **Возможные решения**: Интеграция с системами управления логистикой, автоматическое назначение курьеров, отслеживание статуса доставки в реальном времени.

**Заключение**

Разработанный чат-бот для автоматизации заказов на торты успешно достиг всех поставленных целей и решил основные задачи, связанные с оптимизацией бизнес-процессов компании. Автоматизация процессов позволила значительно повысить эффективность работы, улучшить клиентский опыт, снизить операционные затраты и уменьшить количество ошибок при обработке заказов. Проведённая оценка экономической эффективности показала высокую рентабельность проекта с быстрым сроком окупаемости.

Потенциал масштабирования и дальнейшего развития чат-бота предоставляет широкие возможности для интеграции с CRM-системами, внедрения расширенных маркетинговых инструментов, добавления функционала аналитики, поддержки мультиязычности и автоматизации логистики. Эти направления развития позволят не только удерживать текущих клиентов, но и привлекать новых, а также обеспечат устойчивый рост бизнеса в долгосрочной перспективе.

Таким образом, проект разработки и внедрения чат-бота для автоматизации заказов на торты является не только эффективным решением для текущих задач компании, но и фундаментом для дальнейшего развития и инноваций в сфере обслуживания клиентов и управления бизнес-процессами.

# **ПРИЛОЖЕНИЯ**

## Приложение 1

Программный код телеграмм бота

import logging

import asyncio

import datetime

import gspread\_asyncio

from oauth2client.service\_account import ServiceAccountCredentials

from aiogram import Bot, Dispatcher, types

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton, ReplyKeyboardRemove

from aiogram.filters import Command

from aiogram.fsm.context import FSMContext

from aiogram.fsm.state import StatesGroup, State

from aiogram.fsm.storage.memory import MemoryStorage

from aiogram import Router

from aiogram.types import Message

from aiogram.filters.command import CommandObject

from dotenv import load\_dotenv

import os

# Загрузка переменных окружения из .env файла

load\_dotenv()

# Настройка логирования

logging.basicConfig(level=logging.INFO)

# Получение токена и админ ID из переменных окружения

BOT\_TOKEN = os.getenv("BOT\_TOKEN")

ADMIN\_IDS = os.getenv("ADMIN\_IDS", "")

# Преобразуем строку с ID в список целых чисел

try:

    ADMIN\_IDS = [int(id\_.strip()) for id\_ in ADMIN\_IDS.split(",") if id\_.strip().isdigit()]

except ValueError:

    ADMIN\_IDS = []

    logging.error("Ошибка при парсинге ADMIN\_IDS. Убедитесь, что они являются числами и разделены запятыми.")

if not ADMIN\_IDS:

    logging.warning("Список ADMIN\_IDS пуст. Административные функции будут недоступны.")

SPREADSHEET\_NAME = os.getenv("SPREADSHEET\_NAME", "OrdersForCakes")

ORDERS\_SHEET\_NAME = os.getenv("ORDERS\_SHEET\_NAME", "orders")

CAKES\_SHEET\_NAME = os.getenv("CAKES\_SHEET\_NAME", "cakes")

scope = [

    "https://spreadsheets.google.com/feeds",

    "https://www.googleapis.com/auth/spreadsheets",

    "https://www.googleapis.com/auth/drive.file",

    "https://www.googleapis.com/auth/drive"

]

# Асинхронная инициализация клиента Google Sheets

async def get\_gspread\_client():

    agcm = gspread\_asyncio.AsyncioGspreadClientManager(

        lambda: ServiceAccountCredentials.from\_json\_keyfile\_name("credentials.json", scope)

    )

    return await agcm.authorize()

# Инициализация бота и диспетчера

bot = Bot(token=BOT\_TOKEN)

dp = Dispatcher(storage=MemoryStorage())

router = Router()

dp.include\_router(router)

# Состояния

class OrderStates(StatesGroup):

    ChoosingCake = State()

    ChoosingTaste = State()

    ChoosingSize = State()

    ChoosingDecor = State()

    Confirming = State()

class AdminStates(StatesGroup):

    ViewingOrders = State()

    UpdatingOrderStatus = State()

# Глобальная переменная для Google Sheets

gc = None

# Проверка на администратора

def is\_admin(user\_id: int) -> bool:

    return user\_id in ADMIN\_IDS

# Клавиатуры для меню

# Пользовательское меню

user\_menu = ReplyKeyboardMarkup(

    keyboard=[

        [KeyboardButton(text="Сделать заказ")],

        [KeyboardButton(text="Статус заказов")],

        [KeyboardButton(text="Отмена")]

    ],

    resize\_keyboard=True

)

# Администраторское меню

admin\_menu = ReplyKeyboardMarkup(

    keyboard=[

        [KeyboardButton(text="Просмотреть заказы")],

        [KeyboardButton(text="Обновить статус заказа")],

        [KeyboardButton(text="Отмена")]

    ],

    resize\_keyboard=True

)

# Вспомогательные функции работы с Google Sheets

async def get\_catalog\_of\_cakes():

    try:

        sh = await gc.open(SPREADSHEET\_NAME)

        cakes\_sheet = await sh.worksheet(CAKES\_SHEET\_NAME)

        data = await cakes\_sheet.get\_all\_records()

        logging.info(f"Fetched {len(data)} cakes from catalog.")

        return data

    except Exception as e:

        logging.error(f"Ошибка при получении каталога тортов: {e}")

        return []

async def get\_all\_orders():

    try:

        sh = await gc.open(SPREADSHEET\_NAME)

        orders\_sheet = await sh.worksheet(ORDERS\_SHEET\_NAME)

        all\_orders = await orders\_sheet.get\_all\_records()

        logging.info(f"Fetched {len(all\_orders)} orders from Google Sheets.")

        return all\_orders

    except Exception as e:

        logging.error(f"Ошибка при получении всех заказов: {e}")

        return []

async def get\_all\_orders\_by\_user(user\_id):

    try:

        sh = await gc.open(SPREADSHEET\_NAME)

        orders\_sheet = await sh.worksheet(ORDERS\_SHEET\_NAME)

        all\_orders = await orders\_sheet.get\_all\_records()

        logging.info(f"Fetched {len(all\_orders)} orders from Google Sheets.")

        user\_orders = [

            o for o in all\_orders

            if str(o.get('user\_id', '')).strip() == str(user\_id).strip()

        ]

        logging.info(f"User {user\_id} has {len(user\_orders)} orders.")

        return user\_orders

    except Exception as e:

        logging.error(f"Ошибка при получении заказов пользователя {user\_id}: {e}")

        return []

async def create\_new\_order(user\_id, user\_name, cake, taste, size, decor):

    try:

        sh = await gc.open(SPREADSHEET\_NAME)

        orders\_sheet = await sh.worksheet(ORDERS\_SHEET\_NAME)

        all\_values = await orders\_sheet.get\_all\_values()

        if len(all\_values) < 2:

            order\_id = 1

        else:

            last\_row = all\_values[-1]

            order\_id = int(last\_row[0]) + 1

        status = "ожидается подтверждение администратора"

        current\_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        await orders\_sheet.append\_row([

            order\_id,

            str(user\_id).strip(),

            user\_name.strip(),

            cake['name'].strip(),

            str(cake['price']).strip(),

            taste.strip(),

            str(size).strip(),

            decor.strip(),

            status,

            current\_date

        ])

        logging.info(f"Created new order {order\_id} for user {user\_id}.")

        return order\_id

    except Exception as e:

        logging.error(f"Ошибка при создании заказа: {e}")

        return None

async def update\_order\_status(order\_id, new\_status):

    try:

        sh = await gc.open(SPREADSHEET\_NAME)

        orders\_sheet = await sh.worksheet(ORDERS\_SHEET\_NAME)

        all\_orders = await orders\_sheet.get\_all\_records()

        headers = await orders\_sheet.row\_values(1)

        if 'status' not in headers:

            logging.error("Столбец 'status' не найден в листе.")

            return False

        status\_col = headers.index('status') + 1

        for idx, order in enumerate(all\_orders, start=2):

            if str(order.get('OrderID')) == str(order\_id):

                await orders\_sheet.update\_cell(idx, status\_col, new\_status)

                logging.info(f"Updated OrderID {order\_id} status to '{new\_status}'.")

                return True

        logging.warning(f"OrderID {order\_id} не найден.")

        return False

    except Exception as e:

        logging.error(f"Ошибка при обновлении статуса заказа {order\_id}: {e}")

        return False

async def send\_status\_update(user\_id, order\_id, new\_status):

    try:

        await bot.send\_message(

            chat\_id=user\_id,

            text=f"✅ Ваш заказ №{order\_id} был обновлён.\nНовый статус: <b>{new\_status}</b>",

            parse\_mode='HTML'

        )

        logging.info(f"Sent status update to user {user\_id} for order {order\_id}.")

    except Exception as e:

        logging.error(f"Не удалось отправить уведомление пользователю {user\_id}: {e}")

# === Обработчики ===

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

async def handle\_start(message: Message, state: FSMContext):

    """Запуск бота. Показываем меню в зависимости от ролей."""

    await state.clear()

    user\_id = message.from\_user.id

    if is\_admin(user\_id):

        await message.answer(

            "Привет, Администратор!",

            reply\_markup=admin\_menu

        )

    else:

        await message.answer(

            "Привет! Я бот для оформления заказов на торты.",

            reply\_markup=user\_menu

        )

@router.message(lambda m: m.text == "Отмена")

async def handle\_cancel(message: Message, state: FSMContext):

    """Кнопка «Отмена» для возврата в главное меню."""

    await state.clear()

    user\_id = message.from\_user.id

    if is\_admin(user\_id):

        await message.answer("Возврат в админ-меню.", reply\_markup=admin\_menu)

    else:

        await message.answer("Возврат в пользовательское меню.", reply\_markup=user\_menu)

# === Пользовательское меню ===

@router.message(lambda m: m.text == "Сделать заказ")

async def user\_make\_order(message: Message, state: FSMContext):

    """Начало оформления заказа."""

    user\_id = message.from\_user.id

    if is\_admin(user\_id):

        await message.answer("Администратор не может использовать этот функционал.")

        #return

    catalog = await get\_catalog\_of\_cakes()

    if not catalog:

        await message.answer("Каталог тортов пока пуст.")

        return

    # Выводим список тортов

    for cake in catalog:

        text = f"<b>{cake['name']}</b>\nЦена: {cake['price']} руб.\n{cake['description']}"

        photo = cake.get('photo')

        if photo:

            await message.answer\_photo(photo=photo, caption=text, parse\_mode='HTML')

        else:

            await message.answer(text, parse\_mode='HTML')

    await message.answer(

        "Введите название торта:",

        reply\_markup=ReplyKeyboardMarkup(

            keyboard=[

                [KeyboardButton(text="Отмена")]

            ],

            resize\_keyboard=True

        )

    )

    await state.set\_state(OrderStates.ChoosingCake)

@router.message(lambda m: m.text == "Статус заказов")

async def user\_check\_status(message: Message, state: FSMContext):

    """Проверка статусов заказов пользователя."""

    user\_id = message.from\_user.id

    if is\_admin(user\_id):

        await message.answer("Администратор не может использовать этот функционал.")

        #return

    orders = await get\_all\_orders\_by\_user(user\_id)

    if not orders:

        await message.answer("У вас ещё нет заказов.", reply\_markup=user\_menu)

        return

    # Сортировка по дате

    try:

        sorted\_orders = sorted(

            orders,

            key=lambda x: datetime.datetime.strptime(x['date'], "%Y-%m-%d %H:%M:%S"),

            reverse=True

        )

    except Exception as e:

        logging.error(f"Ошибка при сортировке заказов: {e}")

        sorted\_orders = orders

    # Вывод

    page\_size = 5

    total = len(sorted\_orders)

    pages = (total + page\_size - 1) // page\_size

    for page in range(pages):

        start = page \* page\_size

        end = start + page\_size

        chunk = sorted\_orders[start:end]

        text = "<b>Ваши заказы:</b>\n\n"

        for o in chunk:

            text += (

                f"№ {o['OrderID']}\n"

                f"Торт: {o['cake\_name']}\n"

                f"Цена: {o['price']} руб.\n"

                f"Вкус: {o['taste']}\n"

                f"Размер: {o['size']} персон\n"

                f"Декор: {o['decor']}\n"

                f"Статус: {o['status']}\n"

                f"Дата: {o['date']}\n"

                "-----------------------\n"

            )

        await message.answer(text, parse\_mode='HTML', reply\_markup=user\_menu)

# === Обработчики состояний оформления заказа (OrderStates) ===

@router.message(OrderStates.ChoosingCake)

async def user\_choosing\_cake(message: Message, state: FSMContext):

    if message.text == "Отмена":

        await handle\_cancel(message, state)

        return

    user\_id = message.from\_user.id

    if is\_admin(user\_id):

        await message.answer("Администратор не может использовать этот функционал.")

        #return

    chosen\_cake\_name = message.text.strip()

    catalog = await get\_catalog\_of\_cakes()

    chosen\_cake = next((c for c in catalog if c['name'].lower() == chosen\_cake\_name.lower()), None)

    if not chosen\_cake:

        await message.answer("Такого торта нет в каталоге. Попробуйте ещё раз или нажмите Отмена.")

        return

    await state.update\_data(chosen\_cake=chosen\_cake)

    await message.answer(

        "Какой вкус вы предпочитаете?",

        reply\_markup=ReplyKeyboardMarkup(

            keyboard=[

                [KeyboardButton(text="Отмена")]

            ],

            resize\_keyboard=True

        )

    )

    await state.set\_state(OrderStates.ChoosingTaste)

@router.message(OrderStates.ChoosingTaste)

async def user\_choosing\_taste(message: Message, state: FSMContext):

    if message.text == "Отмена":

        await handle\_cancel(message, state)

        return

    await state.update\_data(taste=message.text.strip())

    await message.answer(

        "На сколько персон?",

        reply\_markup=ReplyKeyboardMarkup(

            keyboard=[

                [KeyboardButton(text="Отмена")]

            ],

            resize\_keyboard=True

        )

    )

    await state.set\_state(OrderStates.ChoosingSize)

@router.message(OrderStates.ChoosingSize)

async def user\_choosing\_size(message: Message, state: FSMContext):

    if message.text == "Отмена":

        await handle\_cancel(message, state)

        return

    size = message.text.strip()

    if not size.isdigit():

        await message.answer("Число, пожалуйста. Или нажмите Отмена.")

        return

    await state.update\_data(size=size)

    await message.answer(

        "Какой декор? (например: ягоды, фигурки...)",

        reply\_markup=ReplyKeyboardMarkup(

            keyboard=[

                [KeyboardButton(text="Отмена")]

            ],

            resize\_keyboard=True

        )

    )

    await state.set\_state(OrderStates.ChoosingDecor)

@router.message(OrderStates.ChoosingDecor)

async def user\_choosing\_decor(message: Message, state: FSMContext):

    if message.text == "Отмена":

        await handle\_cancel(message, state)

        return

    await state.update\_data(decor=message.text.strip())

    data = await state.get\_data()

    cake = data['chosen\_cake']

    taste = data['taste']

    size = data['size']

    decor = data['decor']

    confirmation\_text = (

        f"Пожалуйста, подтвердите заказ:\n\n"

        f"Торт: <b>{cake['name']}</b>\n"

        f"Вкус: {taste}\n"

        f"Размер: {size} персон\n"

        f"Декор: {decor}\n\n"

        "Отправьте «Да» для подтверждения или «Нет» для отмены."

    )

    await message.answer(

        confirmation\_text,

        parse\_mode='HTML',

        reply\_markup=ReplyKeyboardMarkup(

            keyboard=[

                [KeyboardButton(text="Да"), KeyboardButton(text="Нет")]

            ],

            resize\_keyboard=True

        )

    )

    await state.set\_state(OrderStates.Confirming)

@router.message(OrderStates.Confirming)

async def user\_confirming\_order(message: Message, state: FSMContext):

    response = message.text.lower()

    if response == "отмена":

        await handle\_cancel(message, state)

        return

    data = await state.get\_data()

    user\_id = message.from\_user.id

    user\_name = message.from\_user.username or message.from\_user.full\_name

    if response == "да":

        order\_id = await create\_new\_order(

            user\_id=user\_id,

            user\_name=user\_name,

            cake=data['chosen\_cake'],

            taste=data['taste'],

            size=data['size'],

            decor=data['decor']

        )

        if order\_id is not None:

            await message.answer(

                f"Спасибо! Заказ #{order\_id} оформлен.\n"

                "Ожидается подтверждение администратора.",

                reply\_markup=user\_menu

            )

            # Уведомление администраторов

            current\_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

            for admin\_id in ADMIN\_IDS:

                try:

                    await bot.send\_message(

                        chat\_id=admin\_id,

                        text=(

                            f"📦 <b>Новый заказ</b>\n\n"

                            f"№ {order\_id}\n"

                            f"Пользователь: @{user\_name} (ID: {user\_id})\n"

                            f"Торт: {data['chosen\_cake']['name']}\n"

                            f"Вкус: {data['taste']}\n"

                            f"Размер: {data['size']} персон\n"

                            f"Декор: {data['decor']}\n"

                            f"Статус: ожидается подтверждение администратора\n"

                            f"Дата: {current\_date}"

                        ),

                        parse\_mode='HTML'

                    )

                except Exception as e:

                    logging.error(f"Не удалось отправить уведомление администратору {admin\_id}: {e}")

        else:

            await message.answer("Произошла ошибка при оформлении заказа.", reply\_markup=user\_menu)

        await state.clear()

    elif response == "нет":

        await message.answer("Заказ отменён.", reply\_markup=user\_menu)

        await state.clear()

    else:

        await message.answer("Введите «Да» или «Нет».")

@router.message(lambda m: m.text == "Просмотреть заказы")

async def admin\_view\_orders\_menu(message: Message, state: FSMContext):

    """Обработчик нажатия на кнопку «Просмотреть заказы»."""

    user\_id = message.from\_user.id

    if not is\_admin(user\_id):

        await message.answer("У вас нет доступа к этому боту.")

        return

    all\_orders = await get\_all\_orders()

    if not all\_orders:

        await message.answer("Нет доступных заказов.", reply\_markup=admin\_menu)

        return

    # Фильтрация: Статус != "Доставлен"

    filtered\_orders = [o for o in all\_orders if o.get('status') != "Доставлен"]

    if not filtered\_orders:

        await message.answer("Нет заказов, ожидающих подтверждения.", reply\_markup=admin\_menu)

        return

    try:

        sorted\_orders = sorted(

            filtered\_orders,

            key=lambda x: datetime.datetime.strptime(x['date'], "%Y-%m-%d %H:%M:%S"),

            reverse=True

        )

    except Exception as e:

        logging.error(f"Ошибка при сортировке заказов: {e}")

        sorted\_orders = filtered\_orders

    page\_size = 10

    total\_orders = len(sorted\_orders)

    pages = (total\_orders + page\_size - 1) // page\_size

    for page in range(pages):

        start = page \* page\_size

        end = start + page\_size

        chunk = sorted\_orders[start:end]

        text = "<b>Заказы:</b>\n\n"

        for o in chunk:

            text += (

                f"№ {o['OrderID']}\n"

                f"Пользователь: @{o['user\_name']} (ID: {o['user\_id']})\n"

                f"Торт: {o['cake\_name']}\n"

                f"Цена: {o['price']} руб.\n"

                f"Вкус: {o['taste']}\n"

                f"Размер: {o['size']} персон\n"

                f"Декор: {o['decor']}\n"

                f"Статус: {o['status']}\n"

                f"Дата: {o['date']}\n"

                "-----------------------\n"

            )

        await message.answer(text, parse\_mode='HTML', reply\_markup=admin\_menu)

@router.message(lambda m: m.text == "Обновить статус заказа")

async def admin\_update\_status\_menu(message: Message, state: FSMContext):

    """Начинаем процесс обновления статуса: просим ввести OrderID и новый статус."""

    user\_id = message.from\_user.id

    if not is\_admin(user\_id):

        await message.answer("У вас нет доступа к этому боту.")

        return

    await message.answer(

        "Введите OrderID и новый статус через пробел.\nНапример: `1 Доставлен`",

        reply\_markup=ReplyKeyboardMarkup(

            keyboard=[

                [KeyboardButton(text="Отмена")]

            ],

            resize\_keyboard=True

        )

    )

    await state.set\_state(AdminStates.UpdatingOrderStatus)

@router.message(AdminStates.UpdatingOrderStatus)

async def admin\_process\_update\_status(message: Message, state: FSMContext):

    """Собственно обновляем статус."""

    if message.text == "Отмена":

        await handle\_cancel(message, state)

        return

    user\_id = message.from\_user.id

    if not is\_admin(user\_id):

        await message.answer("У вас нет доступа к этому боту.")

        return

    parts = message.text.strip().split(maxsplit=1)

    if len(parts) != 2:

        await message.answer(

            "Неверный формат. Введите OrderID и новый статус через пробел.\nНапример: `1 Доставлен`",

            reply\_markup=ReplyKeyboardMarkup(

                keyboard=[

                    [KeyboardButton(text="Отмена")]

                ],

                resize\_keyboard=True

            )

        )

        return

    order\_id, new\_status = parts

    if not order\_id.isdigit():

        await message.answer(

            "OrderID должен быть числом.",

            reply\_markup=ReplyKeyboardMarkup(

                keyboard=[

                    [KeyboardButton(text="Отмена")]

                ],

                resize\_keyboard=True

            )

        )

        return

    success = await update\_order\_status(order\_id, new\_status)

    if success:

        # Отправляем уведомление пользователю

        all\_orders = await get\_all\_orders()

        order = next((o for o in all\_orders if str(o['OrderID']) == str(order\_id)), None)

        if order:

            user\_id\_to\_notify = int(order['user\_id'])

            await send\_status\_update(user\_id\_to\_notify, order\_id, new\_status)

            await message.answer(

                f"Статус заказа №{order\_id} обновлён на '{new\_status}'. Уведомление пользователю отправлено.",

                reply\_markup=admin\_menu

            )

        else:

            await message.answer(

                f"Статус заказа №{order\_id} обновлён, но не удалось найти заказ для уведомления пользователя.",

                reply\_markup=admin\_menu

            )

    else:

        await message.answer(

            "Не удалось обновить статус. Проверьте OrderID.",

            reply\_markup=admin\_menu

        )

    await state.clear()

# ===== Запуск бота =====

async def main():

    global gc

    gc = await get\_gspread\_client()

    await dp.start\_polling(bot, skip\_updates=True)

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

    asyncio.run(main())