

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

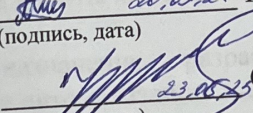
на тему

**ЧАТ-БОТ ДЛЯ ЭЛЕКТРОННОЙ
КОММЕРЦИИ**

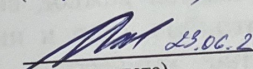
Студент

 20.05.25 Миликов Артём Русланович
(подпись, дата)

Руководитель

 23.05.25 Слепцов Николай Владимирович
(подпись, дата)

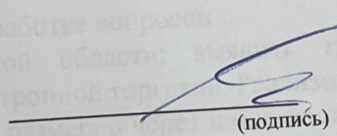
Нормоконтролёр

 23.05.25 Бычков Андрей Станиславович
(подпись, дата)

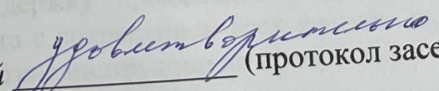
Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. №

16)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от

24 июня 2025 г. № 44)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» 05 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Миликов Артём Русланович гр.21ВВП2 факультета вычислительной техники

2. Тема работы: Чат-бот для электронной коммерции.

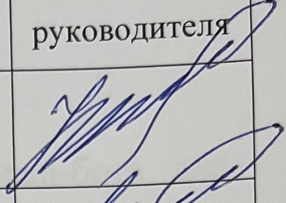
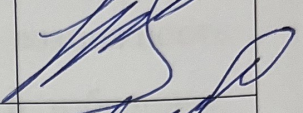
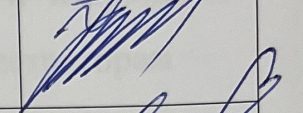
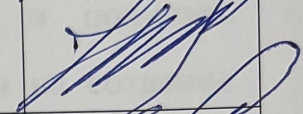
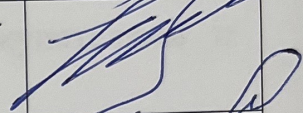
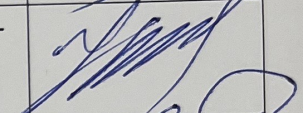
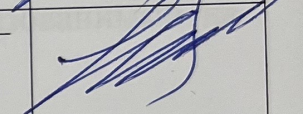
Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Слепцов Н.В.

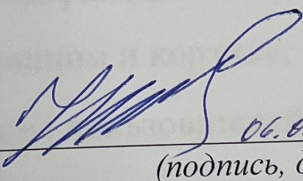
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать Telegram-бот для автоматизации электронной коммерции в сфере розничной торговли одеждой. Система должна позволять пользователям: просматривать каталог товаров с учётом категории и пола, выбирать размер, добавлять товары в корзину и оформлять заказы через встроенный интерфейс. Администратор должен иметь возможность управлять товарами и категориями, подтверждать заказы и получать уведомления о действиях пользователей.

5. Перечень подлежащих разработке вопросов
Провести анализ предметной области: выявить требования к Telegram-боту как альтернативному каналу электронной торговли. Реализовать систему отображения каталога с изображениями и выбором размеров через inline-кнопки. Разработать логику корзины и оформления заказа, включая поддержку промокодов и расчёт стоимости доставки. Обеспечить стабильную работу бота с сохранением всех данных в базе и возможностью масштабирования. Подготовить краткие инструкции по использованию системы для ролей: пользователь (покупатель) и администратор.

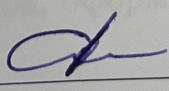
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	6.05.2025 - 28.05.2025	
2	Анализ требований	32	7.05.2025 - 30.05.2025	
3	Создание макета графического интерфейса	30	9.05.2025 - 11.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	40	11.05.2025 - 13.05.2025	
5	Реализация графического интерфейса	66	14.05.2025 - 18.05.2025	
6	Разработка основной части программы	70	16.05.2025 - 23.05.2025	
7	Оформление пояснительной записки	54	24.05.2025 - 30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Слепцов Н.В.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Миликов А.Р.
(подпись, дата)

Работу к защите допустить
Декан факультета  24.06.2025 Фионова Л.Р.

Реферат

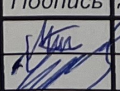
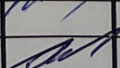
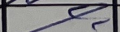
Бакалаврская работа содержит 117 листов, 32 рисунка, 7 таблиц, 8 источников и 3 приложения.

TELEGRAM-БОТ, TELEGRAM API, PYTHON, БАЗА ДАННЫХ

Цель работы - разработка полнофункционального Telegram-бота для электронной коммерции, предоставляющего пользователям возможность просматривать каталог товаров, добавлять товары в корзину и избранное, оформлять заказы с применением промокодов, а также администраторам — управлять категориями и товарами, подтверждать заказы и получать уведомления о действиях пользователей. Проект ориентирован на создание удобного и интуитивного канала взаимодействия между покупателями и управляющим персоналом через мессенджер Telegram.

Объектами разработки являются:

- Telegram-бот с обработчиками команд и кнопок, реализованными на Python (python-telegram-bot);
- база данных SQLite, содержащая информацию о пользователях, товарах, заказах, категориях, избранном и корзине;
- автоматическая регистрация пользователей при первом запуске бота;
- каталог товаров с фильтрацией по категориям и отображением подробной информации (цена, размер, фото);
- корзина с выбором размера, оформление заказа с расчётом стоимости доставки и учётом скидок;
- промокоды и история заказов;
- панель администратора с возможностью добавления, редактирования и удаления товаров и категорий;
- система уведомлений администраторам при новых заказах.

					ПГУ 1.090301.36.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Чат-бот для электронной коммерции	Лит.	Лист	Листов	
Разраб.		Миликов А.Р.					2	114	
Провер.		Слепцов Н.В.							
									
Н. Контр.		Бычков А. С.							
Утверд.		Митрохин М.А.							
						ФВТ, каф. ВТ гр. 21ВВП2			

В результате проделанной работы реализован Telegram-бот с полной пользовательской и административной логикой.

Реализованы:

- автоматическая регистрация пользователей по chat_id, без необходимости ввода логина и пароля;
- добавление товаров в корзину с выбором размера и количества;
- оформление заказов с возможностью применения промокодов и расчётом скидок на доставку;
- подтверждение заказов администраторами через админ-панель внутри Telegram;
- управление товарами и категориями: добавление, редактирование, удаление;
- система уведомлений для администраторов о новых заказах и действиях пользователей;
- адаптивность интерфейса за счёт нативной поддержки Telegram-клиентов на различных устройствах.

Проект демонстрирует возможность создания удобного и расширяемого Telegram-бота для электронной коммерции, ориентированного на продажу одежды. Реализация охватывает весь цикл взаимодействия с пользователем — от просмотра каталога и добавления товаров в корзину до оформления заказа с учётом промокодов и уведомлений. Полученные результаты могут быть использованы как в учебных целях, так и при разработке реальных коммерческих решений в сфере онлайн-торговли.

Содержание

Введение.....	7
1 Анализ предметной области	8
1.1 Анализ актуальности темы.....	8
1.2 Клиент-серверная архитектура	9
1.3 Анализ аналогов чат-ботов для продажи одежды и товаров.....	10
1.4 Постановка задачи.....	15
2 Выбор технологий и средств разработки Telegram-бота	18
2.1 Python 3.8.10.....	18
2.2 Библиотека python-telegram-bot	19
2.3 SQLite	19
2.4 Aiosqlite + JSON	20
2.5 Visual Studio Code.....	20
3 Архитектура чат-бота	21
3.1 Контроллеры (Handlers).....	22
3.2 Сервисы (Логика).....	22
3.3 Модели (Models / ORM)	23
3.4 Представление (View).....	23
3.5 Интеграция технологий	24
4 Реализация бота для электронной коммерции	25
4.1 Описание работы приложения.....	25
4.2 Структура базы данных	25
4.3 Функционал приложения	29
4.4 Основные компоненты	30
4.5 Основные алгоритмы взаимодействия клиента и сервера.....	31
5 Тестирование приложения	34
5.1 Регистрация пользователя.....	34
5.2 Добавление товара в корзину	36
5.3 Оформление заказа	38

5.4 Подтверждение и управление заказами администратором	40
5.5 Редактирование товаров администратором	42
6 Эргономические требования	45
6.1 Организация рабочего места разработчика	45
6.2 Правовое соответствие	46
6.3 Требования к тестированию и качеству	46
Заключение	48
Список использованных источников	49
Приложение А	50
Приложение Б	55
Приложение В.....	114

Перечень сокращений и обозначений

Фреймворк – программная основа, упрощающая и ускоряющая разработку приложения

API (Application Programming Interface) – интерфейс программирования приложений, набор инструментов для взаимодействия между компонентами программного обеспечения

chat_id / Telegram ID – уникальный идентификатор пользователя или чата в Telegram, используемый для обращения к конкретному пользователю

CRUD (Create, Read, Update, Delete) – базовые операции для работы с данными: создание, чтение, обновление и удаление

IDE (Integrated Development Environment) – интегрированная среда разработки, объединяющая инструменты написания, компиляции и отладки кода

InlineKeyboard – встроенные клавиатуры Telegram-бота с кнопками внутри чата.

JSON (JavaScript Object Notation) – лёгкий формат обмена данными, читаемый человеком и легко обрабатываемый машинами

MVC (Model View Controller) – архитектурный шаблон, разделяющий приложение на три логических компонента: модель, представление и контроллер

ORM (Object-Relational Mapping) – технология, связывающая базы данных с объектно-ориентированным кодом, позволяющая работать с таблицами как с объектами

SQLite – встроенная реляционная система управления базами данных, не требующая отдельного сервера

Telegram-бот – автоматизированная программа, взаимодействующая с пользователями через интерфейс Telegram

Введение

Современный мир становится всё более зависимым от цифровых технологий, и торговля не является исключением. Всё больше людей совершают покупки через интернет, предпочитая удобные, быстрые и интуитивные решения. В этом контексте Telegram-боты приобретают особую популярность благодаря своей доступности, простоте и интеграции с привычной средой общения.

Telegram-бот для интернет-магазина одежды представляет собой удобный и эффективный инструмент для покупателей: он позволяет просматривать каталог товаров, выбирать размеры, добавлять вещи в корзину и оформлять заказы — всё это прямо в мессенджере, без перехода на сторонние сайты или приложения.

Для владельцев бизнеса и администраторов такой бот также упрощает управление: они могут добавлять и редактировать товары, отслеживать заказы, взаимодействовать с клиентами, а также получать уведомления о действиях пользователей. Это делает систему более гибкой и лёгкой в сопровождении.

Таким образом, разработка Telegram-бота для электронной коммерции является актуальной задачей, способствующей автоматизации онлайн-продаж, расширению клиентской базы и улучшению пользовательского опыта.

1 Анализ предметной области

Начало разработки любого цифрового продукта начинается с анализа предметной области. Необходимо понимать, кто будет целевой аудиторией, какие задачи необходимо решить, какие технологии актуальны для реализации проекта и как можно обеспечить его устойчивость и поддержку в будущем. В этом разделе рассматриваются ключевые теоретические вопросы, связанные с созданием Telegram-бота для электронной коммерции: актуальность идеи, архитектура системы, существующие аналоги и формулировка конкретной задачи, на основе которой будет вестись проектирование.

1.1 Анализ актуальности темы

В современном мире развитие электронной коммерции идёт стремительными темпами. Всё больше пользователей отдают предпочтение онлайн-покупкам, особенно в удобном и простом формате - без необходимости устанавливать отдельные приложения или искать сайты в поисковиках. Telegram, как один из самых популярных мессенджеров в мире и особенно в странах СНГ, становится новой площадкой для взаимодействия между продавцами и покупателями.

Появление Telegram-ботов открыло новые возможности для бизнеса: автоматизация заказов, отображение каталога товаров, оформление покупок, интеграция с платёжными системами и обработка отзывов - всё это можно реализовать в рамках одного чат-бота. Благодаря привычному интерфейсу Telegram, пользователь чувствует себя уверенно и комфортно: всё происходит прямо в диалоге, как при обычной переписке.

Для малого и среднего бизнеса создание Telegram-бота является экономически выгодным решением. Он дешевле, чем полноценный сайт или мобильное приложение, быстрее в разработке и удобен в поддержке. Кроме того, бот не требует от пользователя регистрации - достаточно просто начать

чат. Всё это делает Telegram-ботов крайне актуальными для розничной торговли и прямых продаж.

Таким образом, создание Telegram-бота для интернет-магазина - это своевременная и практически востребованная задача, которая позволяет сделать процесс покупки максимально быстрым и удобным, а бизнес - более эффективным и автоматизированным.

1.2 Клиент-серверная архитектура

Telegram-бот построен на классической клиент-серверной архитектуре. Пользователь (клиент) взаимодействует с ботом через приложение Telegram, отправляя команды и выбирая кнопки. На сервере располагается логика обработки этих действий: получение запроса, обработка, работа с базой данных, формирование ответа и отправка его обратно пользователю.

Преимущества такой архитектуры:

- централизованное управление данными (товары, заказы, пользователи)
- простота в обслуживании и масштабировании
- возможность работы с тысячами пользователей одновременно
- разграничение прав доступа (обычные пользователи и администратор)

Недостатки:

- высокая нагрузка на сервер при большом количестве заказов
- критическая зависимость от стабильности серверной части
- необходимость обеспечивать надёжную защиту пользовательских данных

Telegram Bot API работает по протоколу HTTPS и поддерживает REST-ориентированный подход. Все данные (например, информация о товаре, заказе или пользователе) воспринимаются как ресурсы, с которыми осуществляется обмен через стандартные команды или callback-запросы. Такой стиль делает архитектуру бота гибкой, расширяемой и легко интегрируемой с другими сервисами (например, с внешней системой оплаты или CRM).[1]

1.3 Анализ аналогов чат-ботов для продажи одежды и товаров

Telegram становится всё более популярной площадкой для электронной коммерции. В мессенджере активно развиваются боты-магазины, предлагающие пользователю каталог товаров, возможность оформить заказ, выбрать способ оплаты и доставки. Рассмотрим наиболее интересные и распространённые решения в этой области.

Для того чтобы реализовать Telegram-бота, предназначенного для продаж товаров, необходимо провести анализ существующих решений. Это позволит выявить сильные и слабые стороны популярных Telegram-магазинов, определить лучшие практики, а также сформировать набор требований к собственному приложению.

В данном разделе будут рассмотрены три конкретных Telegram-бота, активно используемых в сфере электронной коммерции и демонстрирующих различные подходы к продаже товаров через мессенджер:

- @american_shopbot - каталог товаров с ручной и автоматической обработкой;
- @ShopBot - демонстрационный бот Telegram с поддержкой встроенных платежей;
- @Dodologic_bot - бот по расчету доставки вещей и аксессуаров с популярных площадок.

Анализ этих решений позволит глубже понять, какие функции действительно важны для конечного пользователя, какие элементы интерфейса наиболее удобны, и какие компоненты необходимо реализовать для создания конкурентоспособного Telegram-магазина.

1.3.1 Telegram-бот @american_shopbot

Этот бот представляет собой каталог американской одежды, преимущественно streetwear и casual-стиля. Он ориентирован на русскоязычную аудиторию, желающую заказывать вещи из США.

Функционал:

- отдельное меню для оформления заказа;
- каждому товару прикреплены изображения, описание и стоимость;
- команда “Заказать” - отправляет запрос в чат с менеджером.

Преимущества:

- простой и понятный интерфейс;
- возможность заказа редких товаров с зарубежных площадок;
- хорошо оформленные карточки товаров.

Недостатки:

- нет полноценной корзины;
- заказы обрабатываются вручную;
- отсутствует встроенная история покупок.

На рисунке 1 изображен интерфейс чат-бота @american_shopbot.

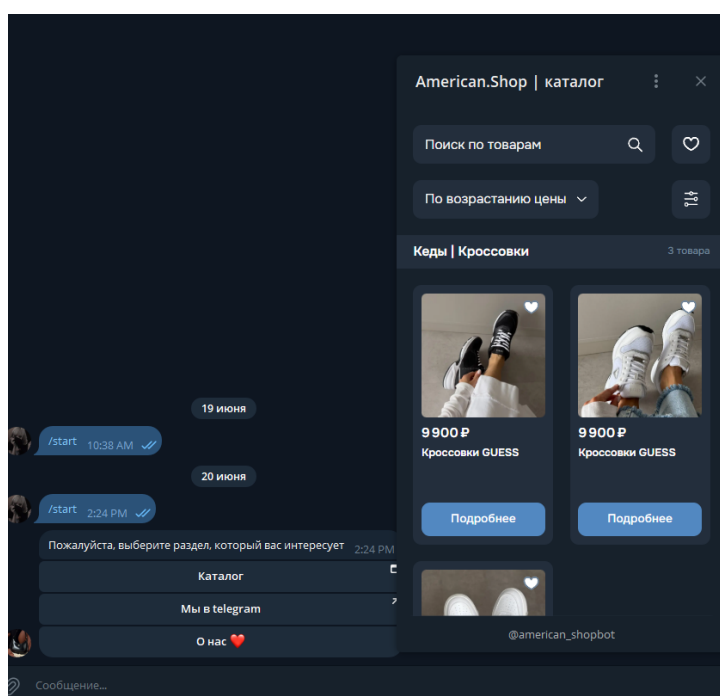


Рисунок 1– Интерфейс бота @american_shopbot

1.3.2 Telegram-бот @ShopBot

Это официальный демонстрационный бот Telegram, иллюстрирующий встроенные платежные возможности и оформление заказов через API. Хотя он

не специализируется на одежде, структура применима к электронной коммерции любого типа.

Функционал:

- добавление товаров в корзину;
- пошаговое оформление заказа;
- поддержка встроенных Telegram Payments.

Преимущества:

- пример качественной реализации UI внутри Telegram;
- безопасные и мгновенные оплаты;
- гибкая интеграция с внешними API.

Недостатки:

- демо-версия, не привязана к реальному магазину;
- не хранит заказы пользователя;
- нет функции избранного или фильтров по категориям.

На рисунке 2 представлен интерфейс чат-бота @Shopbot.

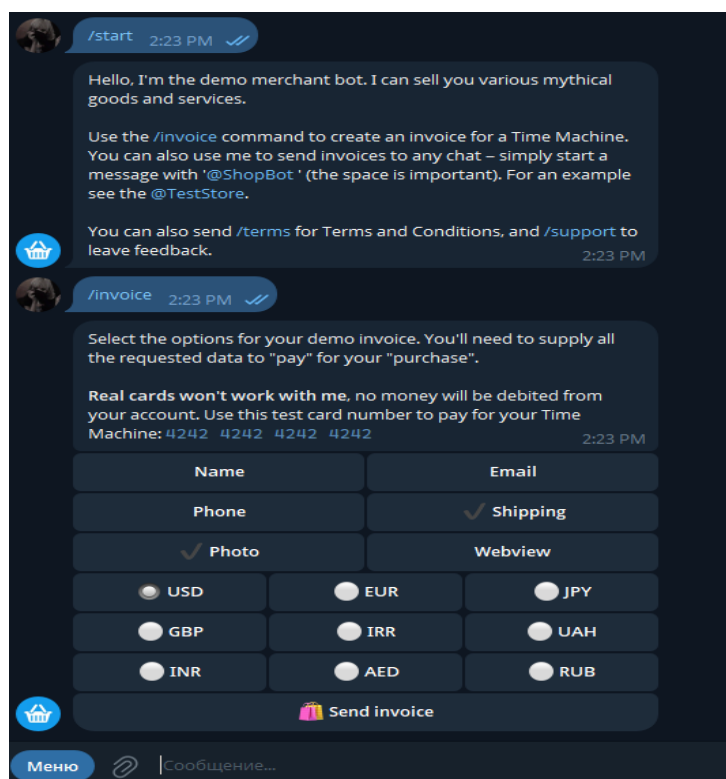


Рисунок 2– Интерфейс бота @Shopbot

1.3.3 Telegram-бот @Dodologic_bot

Бот работает как полноценный мини-магазин с доставкой одежды и аксессуаров. Ориентирован на международные бренды, товары заказываются через посредников с доставкой в СНГ.

Функционал:

- каталог товаров с фото, ценой и размерной сеткой;
- поддержка выбора размера и оформления заявки;
- автоматическая генерация информации для доставки;
- проверка статуса заказа и обратная связь.

Преимущества:

- частично автоматизирован процесс;
- реализована логика обработки заказа;
- присутствует функция статуса и трекинга.

Недостатки:

- некоторые действия требуют подтверждения оператором;
- отсутствует встроенная система оплаты через Telegram.

На рисунке 3 изображен интерфейс чат-бота @Dodologic_bot.

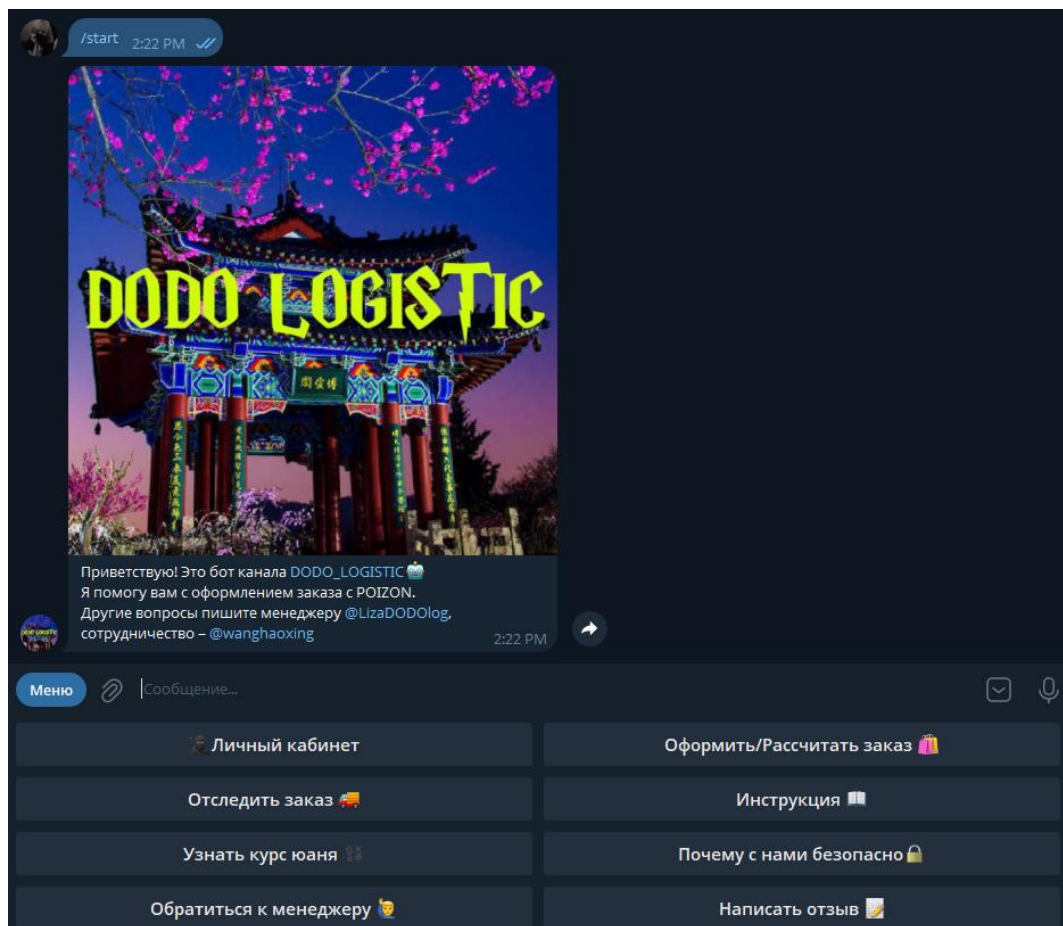


Рисунок 3– Интерфейс бота @dodologistic_bot

Вывод:

Каждый из представленных ботов показывает разные подходы к реализации Telegram-магазинов.

– @american_shopbot - простой визуальный каталог с заказом через менеджера;

– @ShopBot - образец реализации платежей и оформления через Telegram API;

– @Dodologistic_bot - рабочий магазин с оформлением и логикой доставки, но без встроенных платёжных функций.

Ниже будет приведена таблица достоинств и недостатков каждого бота.

Таблица 1—Достоинства и недостатки чат-ботов

Функционал	@american_shopbot	@ShopBot	@Dodologicistic_bot
Каталог одежды	+	+	+
Фотографии и описания	+	+	+
Добавление в корзину	-	+	+
Выбор размера	+	-	+
Оформление заказа	+	+	+
Встроенная оплата	-	+	-
Статус заказа	-	-	+
Автоматическая обработка	-	+	-
История заказов	-	-	+

1.4 Постановка задачи

Telegram-бот для электронной коммерции (продажа одежды) должен обладать следующими функциональными возможностями, обеспечивающими удобство для пользователей и эффективность работы для администратора магазина.

Со стороны сервера (внутренняя логика и база данных):

- хранение и предоставление данных о пользователях, товарах, категориях, заказах и корзине;
- проверка корректности вводимых данных (например, проверка промокодов, существующих размеров и наличия товара);

– добавление, редактирование и удаление информации о товарах, ценах, изображениях и доступных размерах;

– разграничение прав доступа: только авторизованный администратор может управлять каталогом и получать уведомления о заказах;

– автоматическая очистка корзины после оформления заказа;

– сохранение истории заказов каждого пользователя;

– безопасное хранение данных о пользователях (ID, username, предпочтения и заказы).

Со стороны клиента (взаимодействие через Telegram):

– стартовое меню и регистрация/идентификация пользователя по Telegram ID;

– удобная навигация по категориям одежды (верхняя, нижняя, аксессуары и т.п.);

– просмотр товаров с изображением, описанием, ценой и доступными размерами;

– добавление товаров в корзину с выбором размера;

– возможность ввода промокода перед оформлением заказа;

– оформление заказа с подтверждением и расчётом итоговой суммы с учётом доставки;

– просмотр истории заказов;

– получение уведомлений о статусе заказа;

– доступ к поддержке через отдельную кнопку (связь с администратором).

Минимальные технические требования для работы пользователя:

– установленное приложение Telegram (на Android, iOS или ПК);

– стабильное интернет-соединение со скоростью от 2–3 Мбит/сек;

– устройство с минимум 1 ГБ оперативной памяти (рекомендуется 2 ГБ и выше);

– наличие свободного места на устройстве для хранения кэшированных изображений товаров .

					<i>ПГУ 1.090301.36.001 ПЗ</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		<i>17</i>

2 Выбор технологий и средств разработки Telegram-бота

В данном разделе рассмотрены программные средства, технологии и инструменты, использованные при создании Telegram-бота для электронной коммерции. Основной задачей является разработка простого и удобного средства взаимодействия между пользователями и онлайн-магазином одежды через привычный интерфейс мессенджера Telegram.

Технологии и инструменты:

- Python 3.8.10;
- библиотека python-telegram-bot;
- SQLite (для хранения данных);
- JSON (резервное хранение/отладка);
- библиотеки aiosqlite, Pillow, requests;
- сервер на Python с использованием asyncio;
- IDE: Visual Studio Code;

2.1 Python 3.8.10

Язык Python был выбран как основной язык разработки благодаря простоте, богатой экосистеме и широкому сообществу. Версия 3.8.10 обеспечивает стабильность и поддержку большинства актуальных библиотек. Python прекрасно подходит для создания Telegram-ботов, особенно при использовании библиотеки python-telegram-bot.

Ключевые преимущества:

- простой синтаксис, читаемый даже начинающими разработчиками;
- встроенный сборщик мусора (Garbage Collector);
- поддержка многозадачности через asyncio;
- богатая стандартная библиотека и доступ к сторонним модулям;
- возможность подключения C-расширений (например, NumPy, Pillow).

2.2 Библиотека python-telegram-bot

Python-telegram-bot - это официальная и одна из самых популярных библиотек для создания ботов в Telegram . Она предоставляет удобные классы для работы с API Telegram, в том числе:

- обновления (updates), сообщения, callback-кнопки;
- команды и фильтры;
- FSM (finite state machine) для отслеживания состояния пользователя;
- инлайн-кнопки, фото, медиа, документы и корзина.

Плюсы библиотеки:

- активная поддержка и документация;
- работает как с polling, так и через webhook;
- интеграция с асинхронным кодом (async def).[2]

2.3 SQLite

Для хранения данных используется встроенная СУБД SQLite. Она не требует установки и отлично подходит для небольших и средних проектов.

Используется для хранения:

- информации о пользователях (ID, username, заказы);
- каталога товаров (название, цена, фото, категории, размеры);
- заказов (состав, статус);
- корзины.

Преимущества:

- лёгкость в установке и использовании;
- быстрое выполнение запросов;
- поддержка транзакций и индексов;
- достаточна для одного Telegram-бота без нагрузки в десятки тысяч пользователей.[3]

2.4 Aiosqlite + JSON

Aiosqlite - это асинхронный драйвер для работы с SQLite, позволяющий выполнять SQL-запросы без блокировки основного потока. Также частично используются JSON-файлы для отладки и хранения статичных данных (например, шаблоны сообщений, настройки).

2.5 Visual Studio Code

Для разработки проекта была использована среда Visual Studio Code - мощный и гибкий редактор с поддержкой Python, интеграцией с Git и терминалом. В нём удобно управлять проектной структурой, выполнять отладку и синхронизировать проект с репозиторием.

Плюсы:

- большое количество расширений;
- встроенная поддержка venv, pip, pylint;
- терминал, Git, отладка в одном окне.

					ПГУ 1.090301.36.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

3 Архитектура чат-бота

В этом разделе рассматривается архитектура Telegram-бота и взаимодействие между компонентами внутри системы. Архитектурным паттерном, положенным в основу данного проекта, стала модель MVC (Model-View-Controller), адаптированная под Telegram-среду. Такой подход позволяет логически разделить работу приложения на несколько уровней, что упрощает разработку, отладку и последующую доработку проекта.

Модель MVC (Model-View-Controller) - это один из самых распространённых и понятных способов организации архитектуры приложения. В случае Telegram-бота, эта модель интерпретируется следующим образом:

- модель (Model) - отвечает за структуру и хранение данных: пользователей, товаров, корзины, заказов. Это уровень, взаимодействующий с базой данных (в данном проекте используется SQLite);

- контроллер (Controller) - отвечает за обработку сообщений от пользователя. Telegram-команды, callback-кнопки, навигация по меню - всё это проходит через контроллеры (или хендлеры);

- представление (View) - в контексте Telegram - это не HTML-страницы, а текстовые сообщения, кнопки, изображения, уведомления. То есть вся информация, которую получает пользователь от бота в окне чата, формируется в слое View.

На рисунке 4 представлена архитектура приложения.

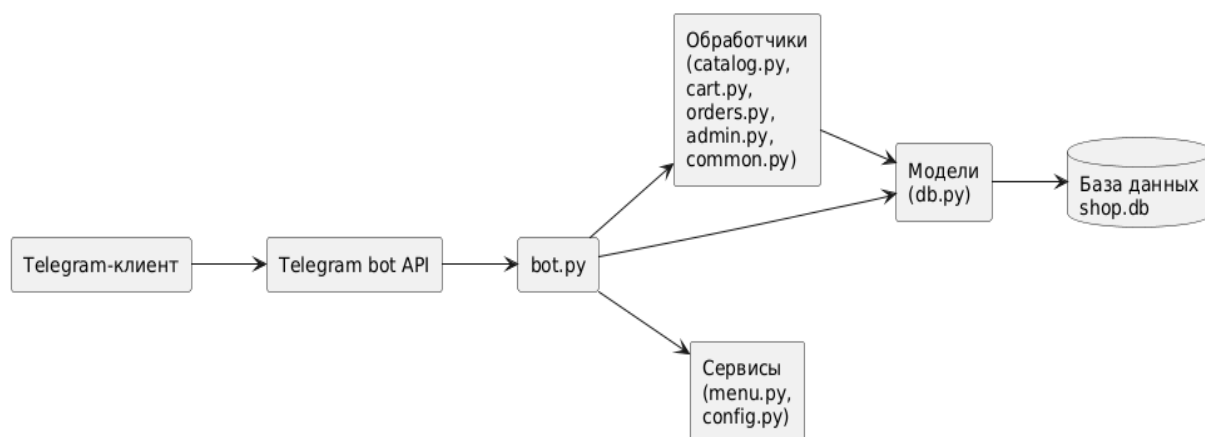


Рисунок 4–Архитектура приложения

3.1 Контроллеры (Handlers)

Контроллеры - это Python-файлы, в которых обрабатываются действия пользователя: нажатия кнопок, команды типа /start, /catalog, /cart, а также любые текстовые сообщения. Каждый контроллер отвечает за отдельный блок функциональности:

- catalog.py - вывод категорий одежды, просмотр товаров, кнопки "Добавить в корзину";
- cart.py - просмотр текущей корзины, удаление товаров, переход к оформлению;
- order.py - оформление заказа, применение промокодов, подтверждение;
- admin.py - админ-панель, управление товарами, просмотр заказов, редактирование информации;
- Контроллеры являются связующим звеном между пользователем и логикой сервиса. Они принимают запрос, проверяют его формат и передают дальше в соответствующий сервис.

3.2 Сервисы (Логика)

Сервисный слой - это центральный блок бизнес-логики. Он обрабатывает данные, управляет состоянием пользователя и взаимодействует с базой данных

через модели:

- product.py - отвечает за фильтрацию, отображение и поиск товаров;
- cart.py - добавление, удаление товаров из корзины, подсчёт итоговой стоимости;
- user.py - регистрация новых пользователей, проверка наличия, сохранение предпочтений;
- order.py - оформление заказа, расчёт скидок, генерация ID заказа, фиксация времени.

Сервисы делают код чистым, легко поддерживаемым и позволяют переиспользовать логику из разных хендлеров.

3.3 Модели (Models / ORM)

Модельный уровень реализован на SQLite с использованием простого ORM-подхода. Каждая модель соответствует таблице в базе данных. Все основные сущности описаны здесь:

- UserModel - хранит Chat_id, username, историю заказов;
- ProductModel - содержит название, категорию, цену, размеры, фото;
- CartModel - список выбранных пользователем товаров, привязанных к chat_id;
- OrderModel - итоговый заказ с составом, суммой, временем, статусом (новый, оформлен, отправлен и т.д.).

SQLite выбран как надёжная, лёгкая и не требующая отдельного сервера СУБД. Он отлично подходит для Telegram-бота с малой и средней нагрузкой.

3.4 Представление (View)

В отличие от веб-приложений, представление в Telegram-боте реализовано через:

- текстовые сообщения;
- инлайн-кнопки (InlineKeyboardMarkup);

- фото товаров;
- уведомления (например, “Товар добавлен в корзину”);
- формы подтверждения (например, “Вы уверены, что хотите оформить заказ?”).

Пользователь взаимодействует только с этим уровнем, не подозревая о логике, скрытой за ним.

3.5 Интеграция технологий

- клиентская часть - Telegram-приложение на Android, iOS или Web;
- сервер - Python-приложение на базе библиотеки python-telegram-bot;
- база данных - SQLite (через aiosqlite);
- IDE - Visual Studio Code.

4 Реализация бота для электронной коммерции

В этом разделе описывается реализация Telegram-бота, предназначенного для удобной покупки одежды. Проект был разработан как клиент-серверное приложение, в котором Telegram используется как клиентская платформа, а серверная часть реализована с помощью Python и базы данных. Здесь будут рассмотрены архитектура, структура базы данных, ключевые функции, программные компоненты и алгоритмы взаимодействия между пользователем и системой [4].

4.1 Описание работы приложения

Принцип работы действия бота изображен на рисунке 5. При любом взаимодействии (нажатие кнопки, отправка команды, выбор товара и т.п.) Telegram передаёт информацию серверу бота. Далее бот определяет, что делать с этим событием: показать меню, добавить товар в корзину, обработать заказ и т.д.

Если при выполнении действий возникает ошибка, бот уведомляет об этом пользователя. Если всё прошло успешно - пользователь видит соответствующее сообщение или получает запрашиваемую информацию [5].

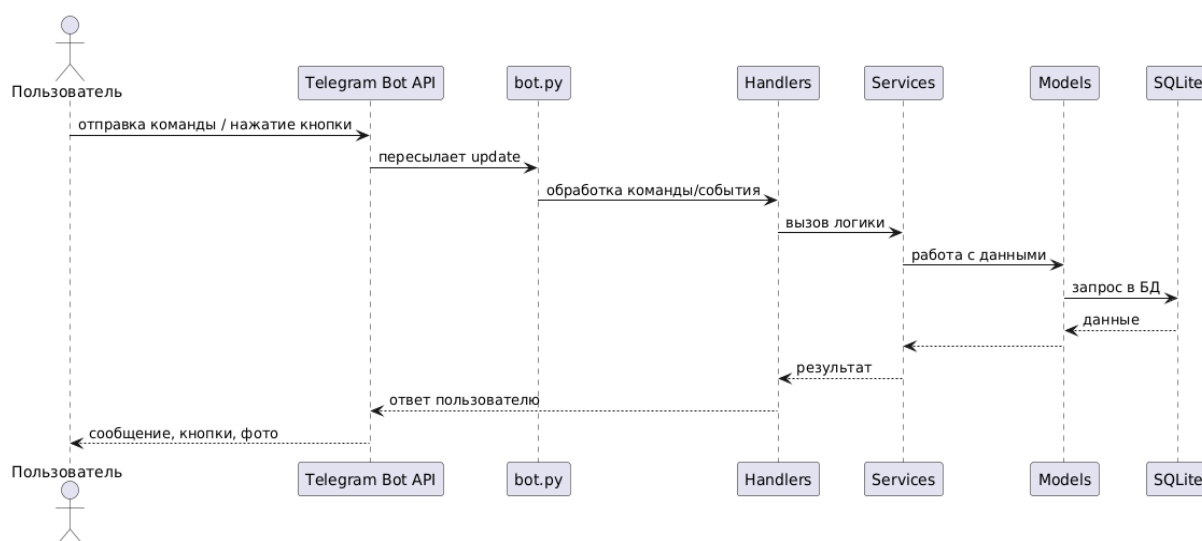


Рисунок 5 - Диаграмма последовательности обработки команды Telegram

4.2 Структура базы данных

Серверная часть бота работает с базой данных SQLite, которая обеспечивает хранение информации о пользователях, товарах, заказах и корзинах. Структура ключевых сущностей БД представлена в таблицах 2-7 [6].

Таблица 2 – Users

Поле	Тип данных	Описание
Chat_id	Integer	Идентификатор пользователя
address	Text	Адрес доставки

Таблица 3 – Categories

Поле	Тип данных	Описание
id	Integer	Идентификатор категории
name	Text	Название категории
gender	Text	Пол категории

Таблица 4 – Products

Поле	Тип данных	Описание
id	Integer	Идентификатор товара
name	Text	Название товара
Price	Integer	Цена
Category	Text	Название категории товара

Image	Text	Ссылка на изображение
Gender	Text	Пол для товара

Таблица 5 – Cart

Поле	Тип данных	Описание
Id	Integer	Идентификатор записи в корзине
chat_id	Integer	Ссылка на пользователя(внешний ключ)
Product	Text	Название товара
size	Text	Размер

Таблица 6 – Orders

Поле	Тип данных	Описание
Id	integer	Идентификатор заказа
chat_id	Integer	Ссылка на пользователя(внешний ключ)
Username	Text	Имя пользователя
Items	Text	Содержимое заказа
discounted_price	Integer	Итоговая сумма заказа
Delivery	Integer	Итоговая сумма

		доставки
Promocode	Text	Примененный промокод(если есть)
Status	Text	Статус заказа

Таблица 7 – Favorites

Поле	Тип данных	Описание
Id	Integer	Идентификатор записи
User_id	Integer	Ссылка на пользователя(Внешний ключ)
Product	Text	Название товара
size	Text	Размер, добавленный в избранное

На рисунке 6 изображена схема структуры базы данных.

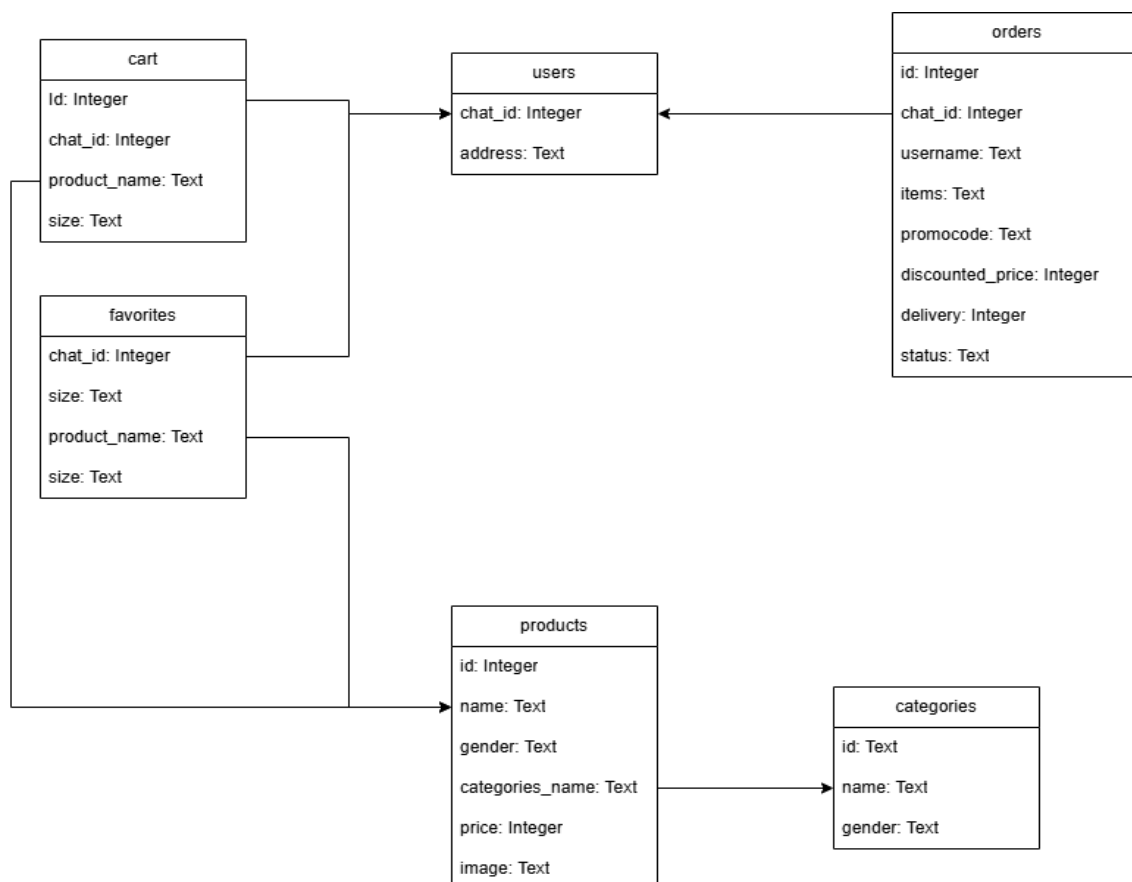


Рисунок 6 - Схема структуры базы данных Telegram-бота

4.3 Функционал приложения

Функции Telegram-бота можно условно разделить на две группы: для обычного пользователя и для администратора.

Пользователь может:

- начать работу с ботом командой /start;
- просматривать каталог одежды, фильтровать по категориям;
- получать информацию о товаре (название, цена, фото, доступные размеры);
- добавлять товары в корзину с выбором размера;
- применять промокод перед оформлением заказа;
- получать итоговую сумму с учётом скидок и доставки;
- подтверждать заказ;
- просматривать историю заказов.

Администратор может:

- открывать панель администратора;
- добавлять новые товары (включая фото и цену);
- редактировать или удалять существующие товары;
- управлять категориями (добавлять, переименовывать, удалять);
- просматривать и подтверждать заказы;
- получать уведомления о новых действиях пользователей.

На рисунках 7 и 8 представлены диаграммы вариантов использования для пользователя и администратора. [7]

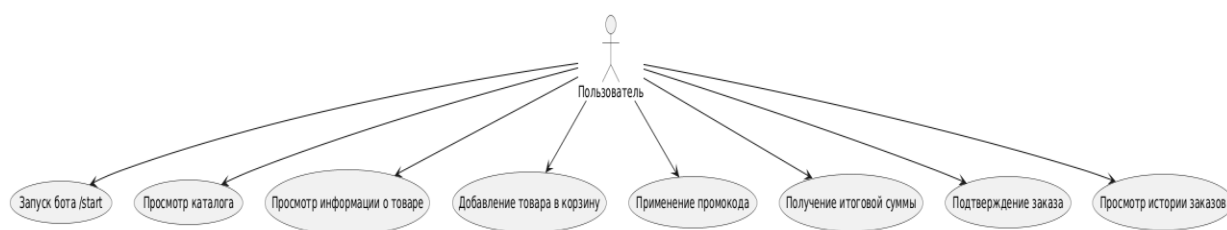


Рисунок 7 - Диаграмма вариантов использования (пользователь)



Рисунок 8 - Диаграмма вариантов использования (администратор)

4.4 Основные компоненты

Приложение построено по логике MVC и делится на следующие программные блоки:

- controller - это хендлеры (обработчики команд и кнопок). Именно они реагируют на команды, вызывают нужные действия и возвращают результат пользователю;
- service - логика, которая управляет корзиной, заказами, промокодами,

фильтрацией и т.д. Контроллеры обращаются к сервису, а тот уже работает с БД;

– model - это структура данных: User, Product, Order, Cart и другие. Именно они отражают структуру таблиц в базе данных.

Также отдельно реализованы:

– utils - вспомогательные функции (форматирование текста, подсчёт доставки и т.д.);

– filters - кастомные фильтры состояний, например, для добавления товаров.

4.5 Основные алгоритмы взаимодействия клиента и сервера

В системе реализованы несколько ключевых алгоритмов, которые формируют поведение Telegram-бота [5].

4.5.1 Регистрация и авторизация

Telegram автоматически передаёт chat_id и username, которые сохраняются в базе при первом использовании бота. Таким образом, дополнительная регистрация не требуется.

На рисунке 9 представлена диаграмма последовательности регистрации и авторизации.

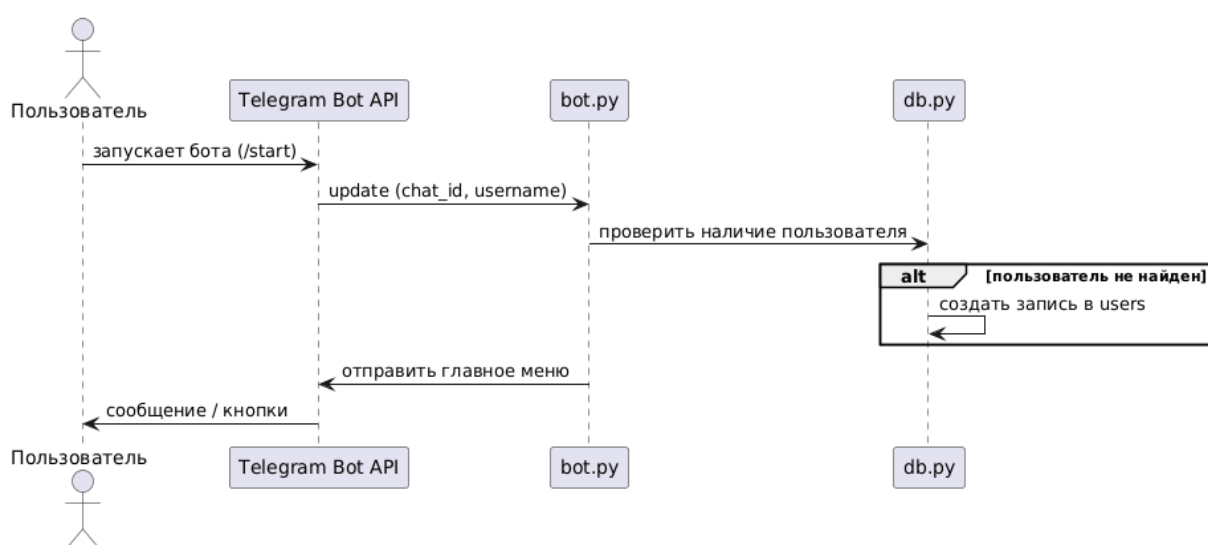


Рисунок 9 – Диаграмма последовательности регистрации и авторизации

4.5.2 Добавление товаров в корзину

На рисунке 10 представлена диаграмма последовательности добавления товара в корзину.

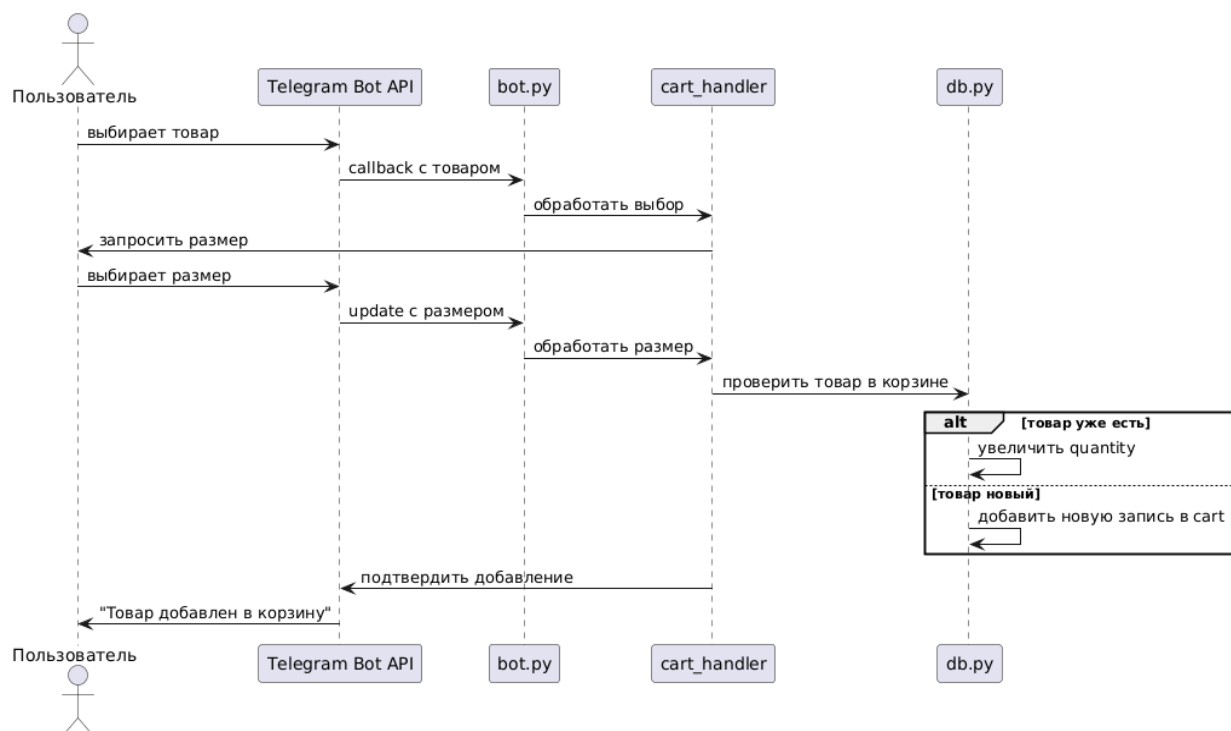


Рисунок 10 - Диаграмма последовательности добавления в корзину

4.5.3 Оформление заказа

Пользователь нажимает кнопку «Оформить заказ», бот показывает итог (включая доставку и скидку), предлагает ввести промокод. После подтверждения заказ сохраняется, корзина очищается.

На рисунке 11 представлена диаграмма оформления заказа.

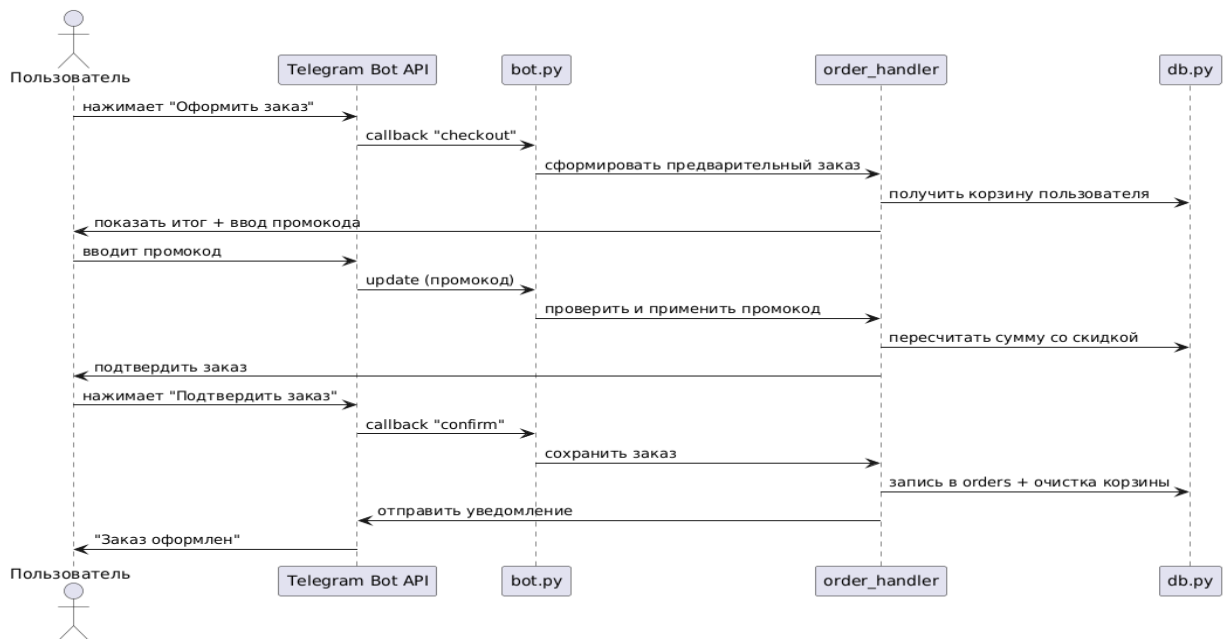


Рисунок 11 - Диаграмма последовательности оформления заказа

4.5.4 Подтверждение заказов администратором

Администратор видит список новых заказов, может подтвердить или отклонить. Статус заказа обновляется, пользователь получает уведомление.

Ниже представлена диаграмма последовательности подтверждения заказа.

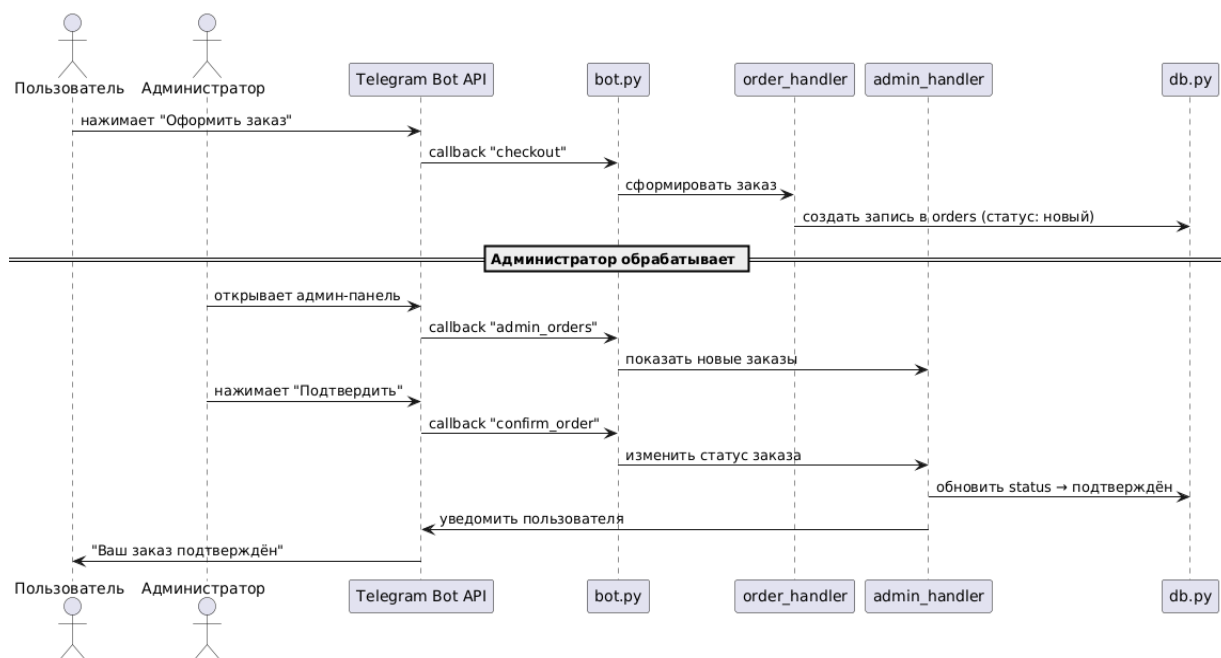


Рисунок 12 - Диаграмма последовательности подтверждения заказа

5 Тестирование приложения

Тестирование Telegram-бота проводилось по основным пользовательским сценариям, включая взаимодействие с каталогом товаров, оформление заказов, проверку прав доступа и действия администратора. Целью тестирования было убедиться в стабильной и корректной работе всех ключевых функций бота.

Тестируемые сценарии:

- автоматическая регистрация при первом запуске бота;
- добавление товаров в корзину;
- оформление и подтверждение заказа;
- просмотр и управление заказами в админ-панели;
- удаление товаров администратором;

5.1 Регистрация пользователя

В боте отсутствует ручная регистрация - она осуществляется автоматически при первом взаимодействии пользователя с ботом (ввод команды /start). После запуска команда фиксирует chat_id, username и создает запись в базе данных.

На рисунке 11 представлено состояние таблицы до запуска бота и добавления адреса.

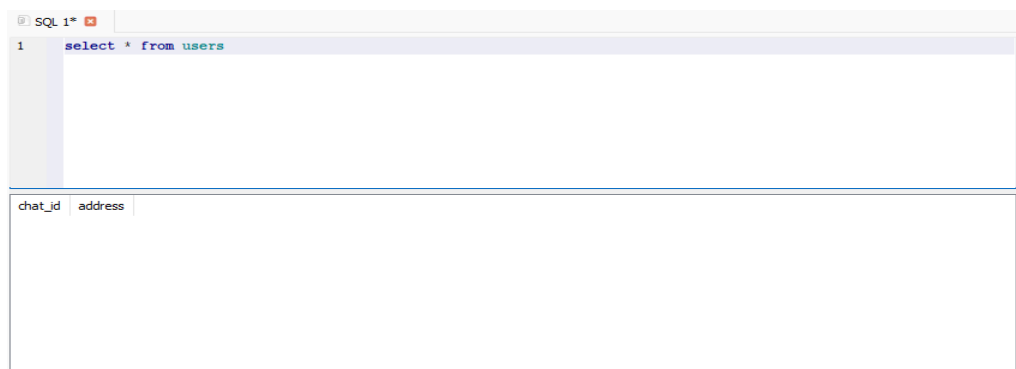


Рисунок 11– Состояние таблицы users до начала работы

На рисунке 12 представлено начало работы с ботом, ввод команды /start.

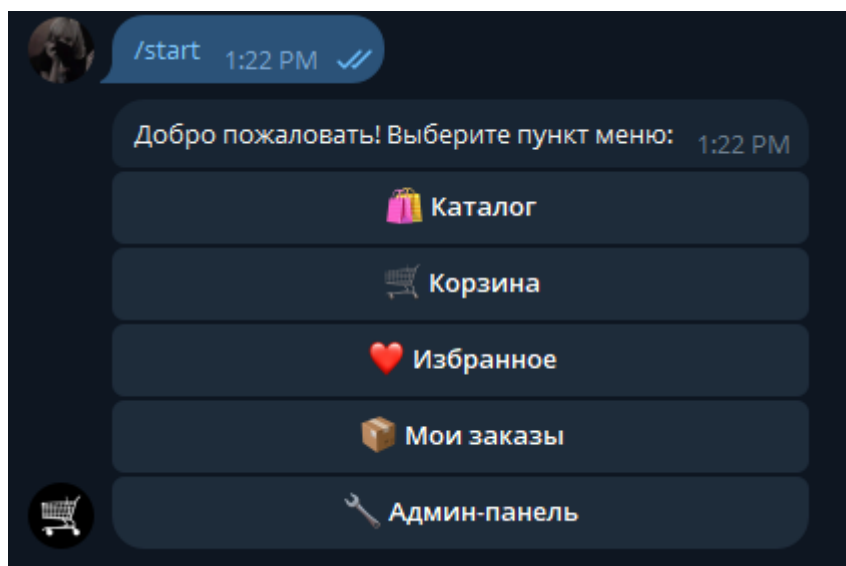


Рисунок 12—Запуск команды /start

Проверка состояния сущности users в базе данных после добавления адреса представлена на рисунке 13.

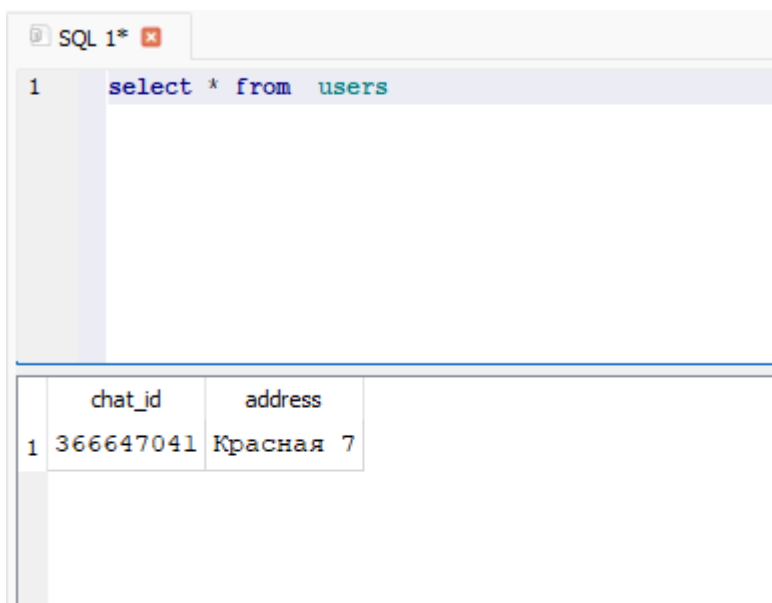


Рисунок 13—База данных после добавления адреса

5.2 Добавление товара в корзину

После просмотра каталога пользователь может выбрать товар, указать размер и добавить его в корзину. Если такой товар с выбранным размером уже есть, бот уведомит, что он уже добавлен, либо увеличит количество.

На рисунках 14 и 15 представлено отображение карточки товара и его размерной сетки.



Рисунок 14– Отображение карточки товара

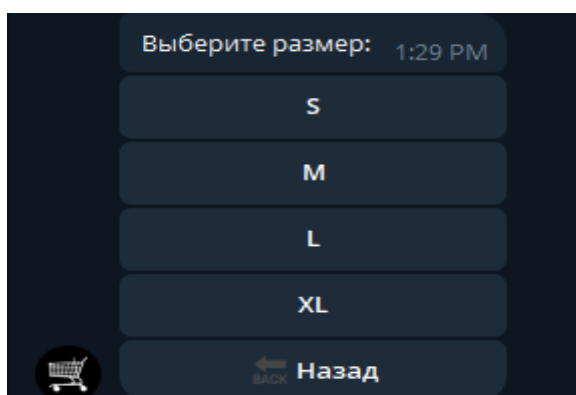


Рисунок 15–Отображение размерной сетки

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

36

Далее добавим товар в корзину, выбрав размер. На рисунке 16 и 17 представлено уведомление о добавлении товара в корзину и изменение таблицы cart.

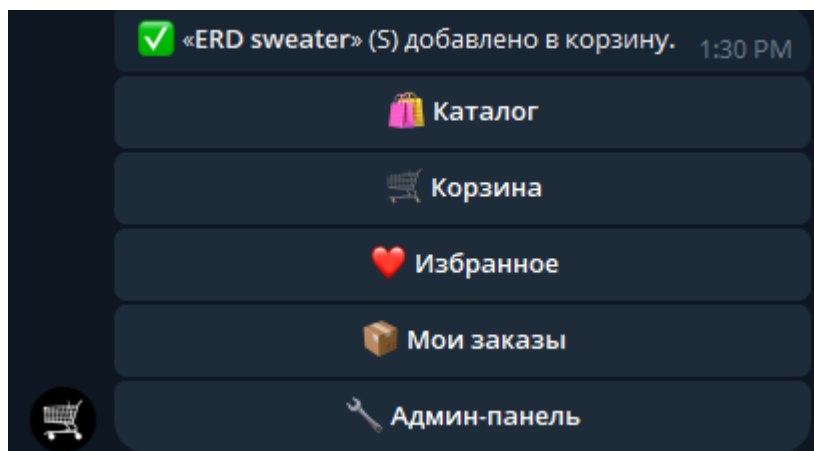


Рисунок 16–Уведомление о добавлении товара в корзину

SQL 1*				
1	select * from cart			
	id	chat_id	product	size
1	2	366647041	ERD sweater	S

Рисунок 17– Состояние таблицы cart после добавления товара в корзину

В ходе тестирования все сценарии прошли успешно. При неправильных данных пользователю была выведена ошибка. С верно введенными данными произошел переход на главную страницу где отображено имя пользователя, который выполнил вход.

5.3 Оформление заказа

Пользователь может перейти в корзину и оформить заказ. Перед оформлением бот отображает состав заказа, цену, стоимость доставки и предлагает ввести промокод. После подтверждения заказ сохраняется, корзина очищается.

На рисунке 18 отображено добавление товара в корзину.

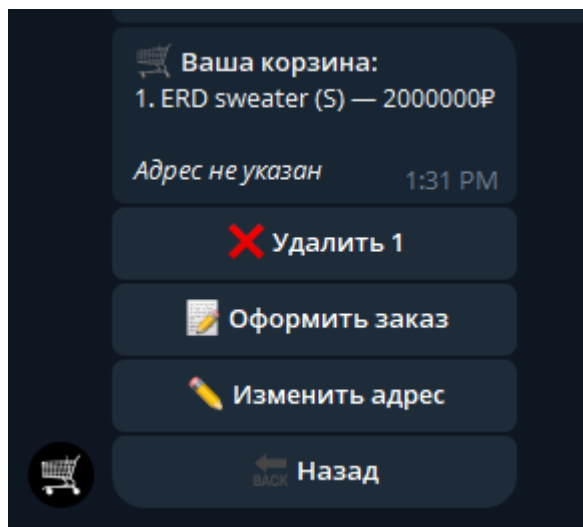


Рисунок 18–Выбранный товар в корзине

Если пользователь еще не оформлял заказ, то при нажатии на кнопку Оформить заказ пользователю будет предложено ввести адрес доставки. После ввода адреса пользователю будет предложено ввести промокод.

На рисунках 19 и 20 представлен ввод адреса и промокода.

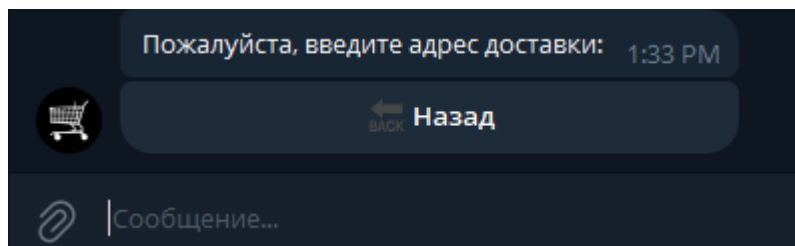


Рисунок 19–Ввод адреса доставки

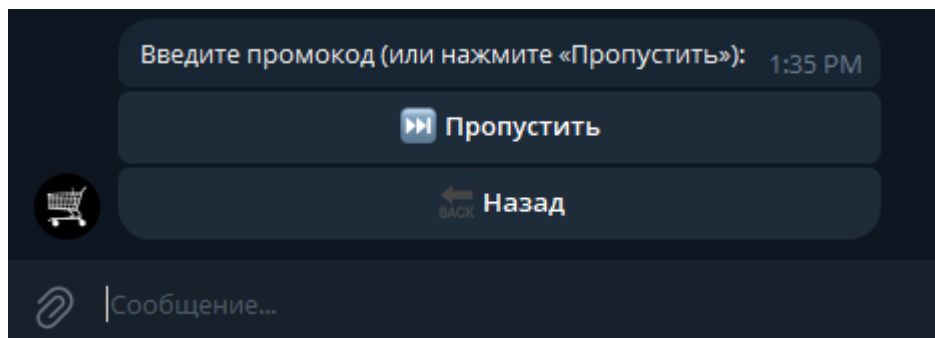


Рисунок 20–Ввод промокода (при наличии)

После ввода промокода и адреса пользователь должен подтвердить заказ. На рисунках 21 и 22 представлено подтверждение заказа.

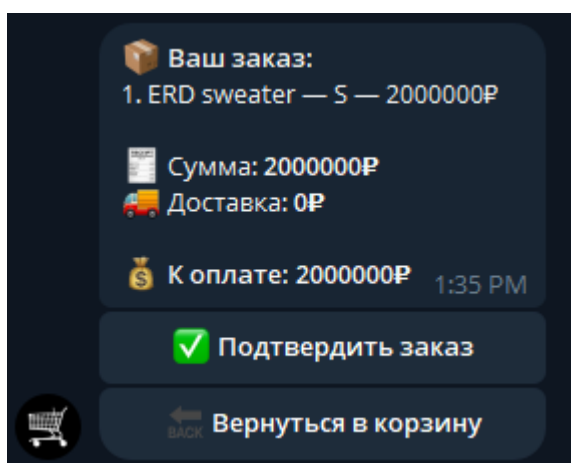


Рисунок 21–Подтверждение заказа

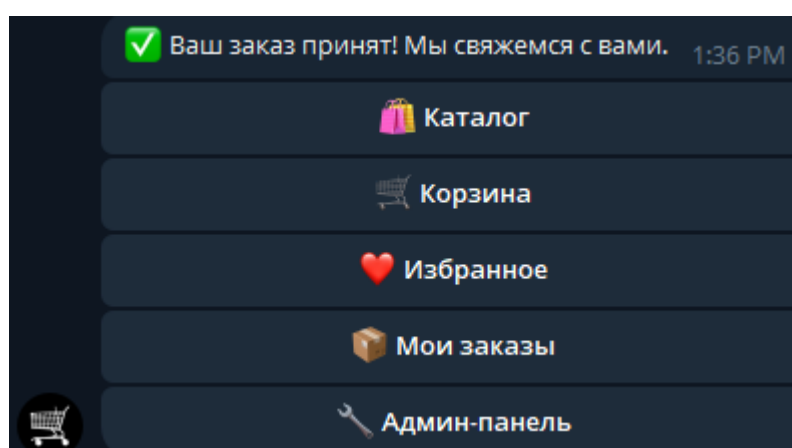
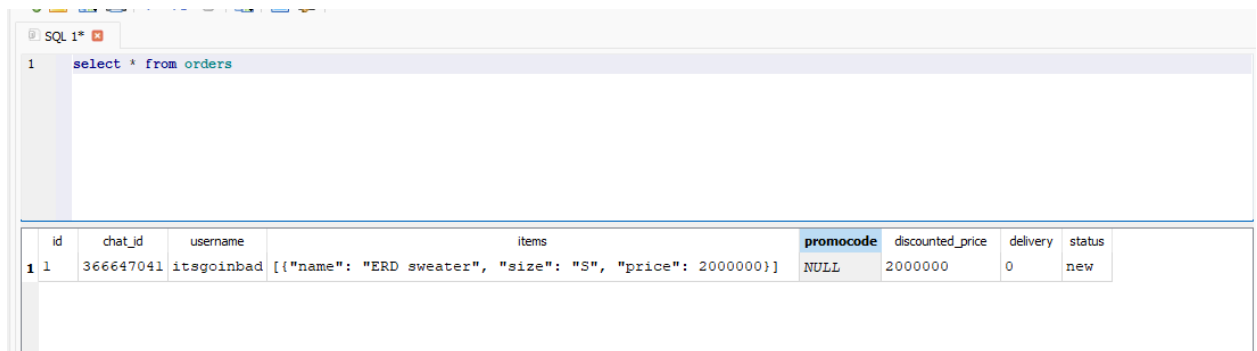


Рисунок 22–Уведомление о сохранении заказа

На рисунке 23 представлена таблица orders в базе данных после оформления заказа.



	id	chat_id	username	items	promocode	discounted_price	delivery	status
1	1	366647041	itsgoinbad	[{"name": "ERD sweater", "size": "S", "price": 2000000}]	NULL	2000000	0	new

Рисунок 23—Состояние таблицы orders после создания заказа

После оформления заказа корзина автоматически очищается.

На рисунке 24 представлена корзина после оформления заказа.

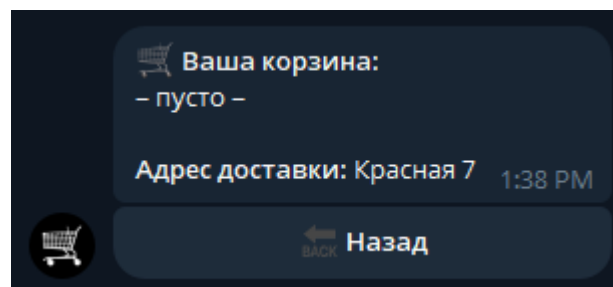


Рисунок 24—Очищенная корзина после оформления

5.4 Подтверждение и управление заказами администратором

Администратор имеет доступ к панели управления. Он может:

- просматривать список новых заказов;
- подтверждать или отклонять заказы;
- редактировать/удалять товары;
- управлять категориями.

На рисунке 25 и 26 представлено подтверждение заказа администратором.

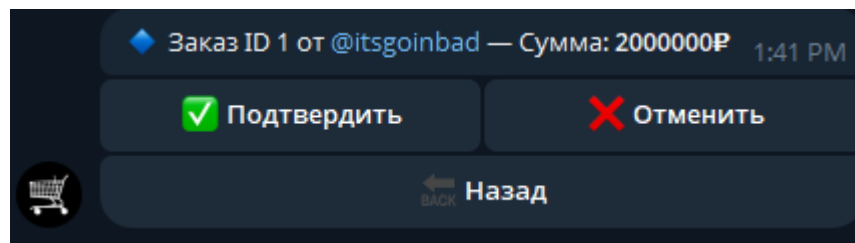


Рисунок 25– Список новых заказов в админ-панели

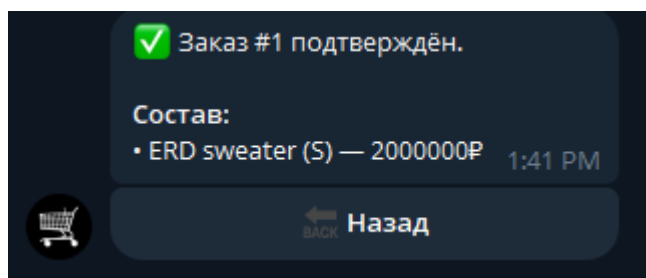


Рисунок 26–Подтверждение заказа админом

На рисунке 27 изображена таблица orders после подтверждения заказа.

SQL 1*

1 select * from orders

	id	chat_id	username	items	promocode	discounted_price	delivery	status
1	1	366647041	itsgoinbad	[{"name": "ERD sweater", "size": ...	NULL	2000000	0	confirmed

Рисунок 27–Изменение статуса в таблице orders

На рисунке 28 продемонстрировано изменение статуса заказа в боте.

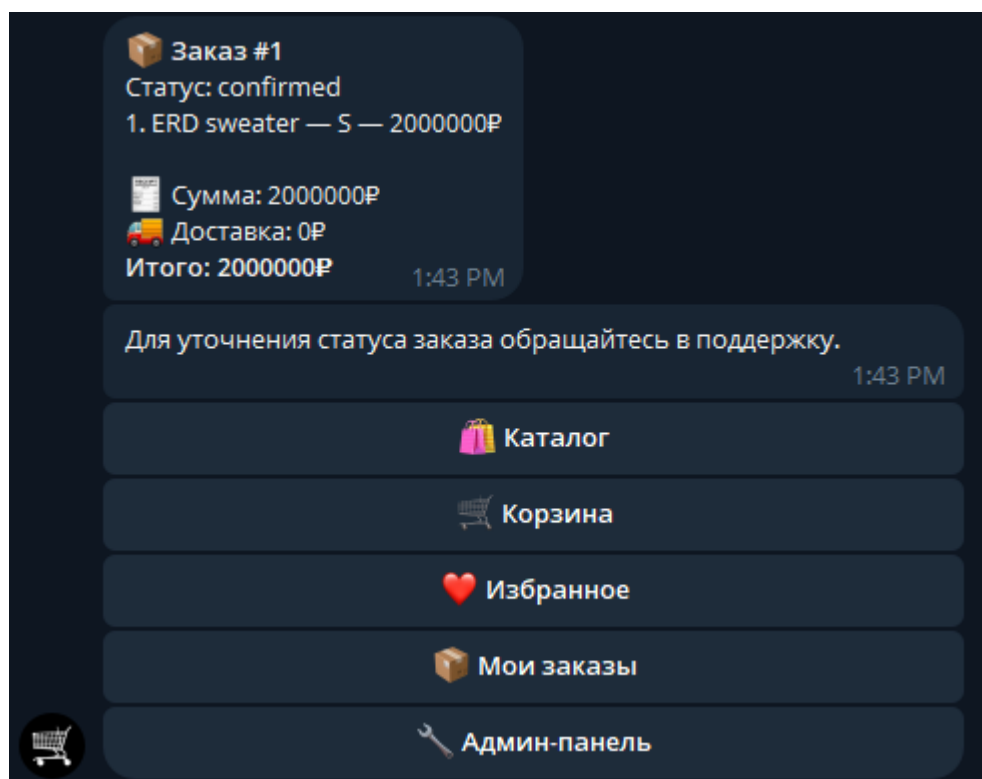


Рисунок 28—Уведомление пользователя о изменении статуса заказа

5.5 Редактирование товаров администратором

Процесс редактирования реализован в виде пошагового диалога, где бот поочерёдно запрашивает новые значения. Администратор может оставить поле без изменений, отправив команду пропуска.

На рисунке 29 изображен полный функционал администратора.

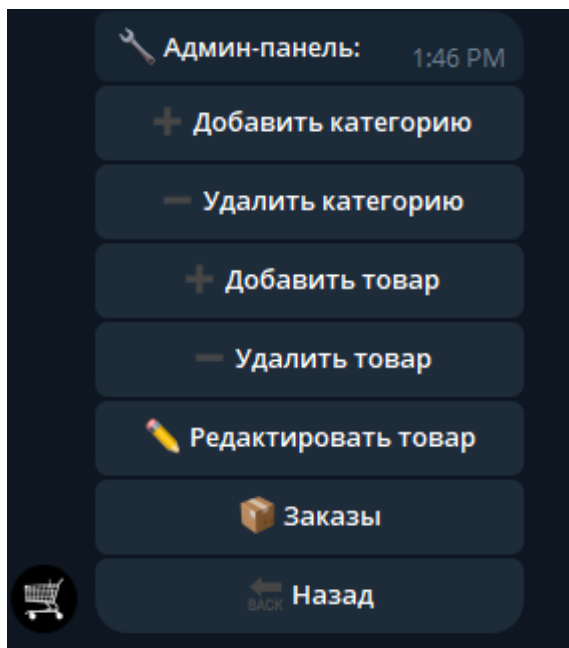


Рисунок 29–Панель администратора

При нажатии на кнопку Редактировать товар, бот запросит новую цену или новое изображение товара. На рисунках 30 и 31 показано изменение цены товара.

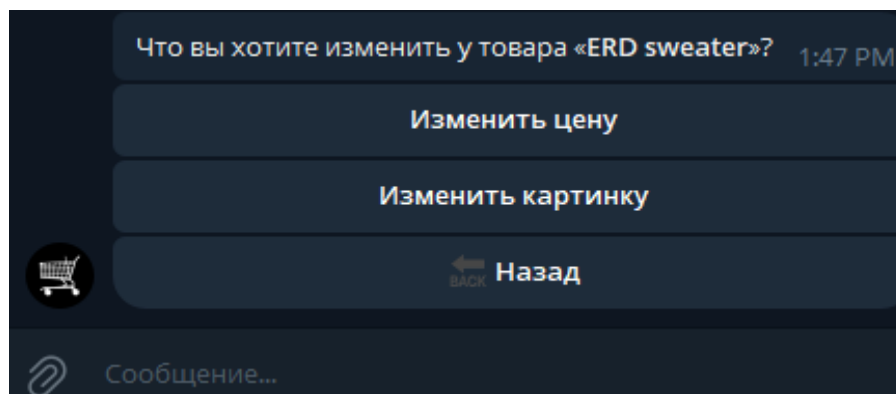


Рисунок 30–Редактирование товара администратором

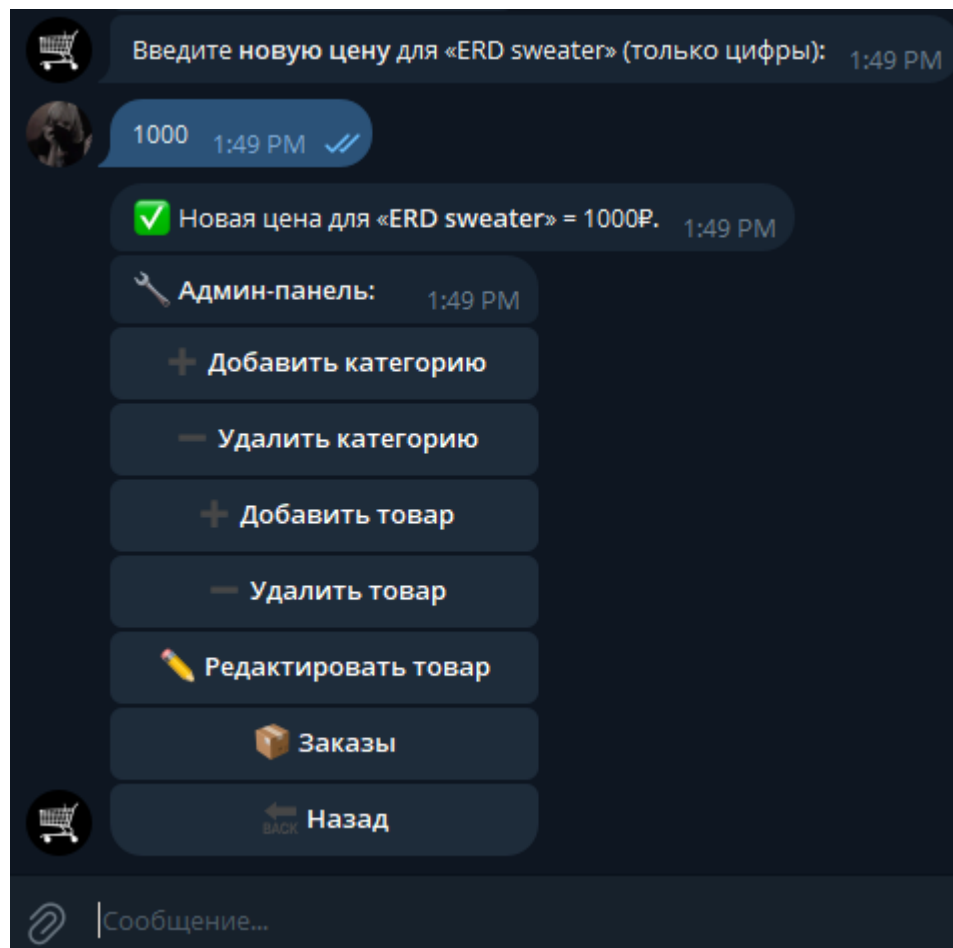


Рисунок 31–Уведомление об успешном изменении цены товара

На рисунке 32 изображена таблицы products после изменения цены.

SQL 1*

```
1 select * from products
```

	id	name	gender	category	price	image
1	1	ERD sweater	Мужской	Sweater	1000	AgACAgIAAxkBAAILgmhVN6THwP4dH6Fhp1F...

Рисунок 32–Состояние таблицы products после изменения цены товара

6 Эргономические требования

При разработке и использовании Telegram-бота для электронной коммерции особое внимание уделялось эргономике взаимодействия с пользователем. Telegram является универсальной платформой, поэтому важно было обеспечить интуитивно понятный интерфейс, исключающий перегрузку информацией и снижающий вероятность ошибок со стороны пользователя.

Бот формирует сообщения с чёткой визуальной и смысловой структурой: товары отображаются с фото, ценой и кратким описанием; кнопки подписаны однозначно (например, «Добавить в корзину», «Выбрать размер», «Подтвердить заказ»); навигация по меню не вызывает затруднений даже у пользователей без технического опыта. Использование инлайн-кнопок помогает упростить управление заказом, свести число действий к минимуму и сделать процесс максимально комфортным.

Все сообщения структурированы так, чтобы избежать перегрузки экрана: используются короткие абзацы, визуальные разделители, эмодзи-иконки - всё это снижает зрительную нагрузку и делает процесс покупок в мессенджере удобным.

6.1 Организация рабочего места разработчика

Разработка Telegram-бота велась на персональном компьютере, что требует соблюдения санитарно-гигиенических норм при длительной работе. В соответствии с рекомендациями по охране труда были соблюдены следующие условия:

- монитор установлен на уровне глаз на расстоянии 60–70 см;
- рабочее место освещается равномерно, без резких теней и бликов;
- используется эргономичное кресло с регулировкой высоты и поддержкой спины;

– делаются регулярные перерывы - 5–10 минут каждый час с упражнениями для глаз и рук.

Также обеспечивается контроль микроклимата помещения: температурный режим в пределах 21–23 °С, нормальная влажность и регулярное проветривание. Всё это позволяет снизить утомляемость разработчика, предотвратить нарушение осанки и повысить производительность труда.

6.2 Правовое соответствие

В процессе разработки Telegram-бота учитывались положения Федерального закона №152-ФЗ «О персональных данных». Бот собирает ограниченный набор персональных данных: Telegram ID, имя пользователя, сведения о заказах и адрес доставки. Эти данные используются строго в целях обработки заказов и не передаются третьим лицам.

На текущем этапе бот функционирует в режиме локального тестирования, однако архитектура системы изначально предусматривает реализацию защиты данных в продакшн-среде. В числе предусмотренных мер:

- регистрация пользователей происходит автоматически по Telegram ID, что исключает ввод лишней информации и снижает риски;
- все действия пользователей и администраторов логируются;
- реализована ролевая система доступа (пользователь, администратор);
- структура кода предусматривает возможность встраивания политики конфиденциальности и пользовательского соглашения с их подтверждением через бот.

6.3 Требования к тестированию и качеству

Для обеспечения безопасной и стабильной эксплуатации Telegram-бота были выполнены мероприятия по тестированию и контролю качества.

					<i>ПГУ 1.090301.36.001 ПЗ</i>	Лист
						46
Изм.	Лист	№ докум.	Подпись	Дата		

Надёжность программного продукта является ключевым условием для исключения сбоев и недовольства пользователей.

Принятые меры включают:

- проверку работы бота в разных клиентах Telegram (мобильная, веб, десктопная версия);
- обработку некорректного ввода и нештатных ситуаций (отсутствие размеров, неверные команды);
- тестирование сценариев оформления заказа, работы промокодов, добавления товаров в корзину;
- логирование ошибок и откатов транзакций в случае сбоев;
- проверку устойчивости при резком завершении сессии, обрыве соединения и повторных действиях.

Благодаря этим мерам, Telegram-бот демонстрирует корректное поведение даже при повышенной нагрузке или ошибках со стороны пользователя. Это позволяет минимизировать потенциальные риски и обеспечить положительный пользовательский опыт.

Заключение

В рамках выполнения данной выпускной квалификационной работы была разработана и протестирована система в виде Telegram-бота, предназначенного для реализации сценариев электронной коммерции. В ходе проекта были последовательно решены следующие задачи: проведён анализ актуальности темы и архитектурных решений, выполнен выбор языка программирования, технологий и инструментов, спроектирована структура базы данных, реализован основной функционал и проведено тестирование.

Разработанное клиент-серверное приложение, реализованное в виде Telegram-бота, позволяет пользователям удобно взаимодействовать с системой без установки дополнительных программ или регистрации: оформлять заказы, просматривать каталог товаров, добавлять товары в корзину и получать уведомления. Администратор, в свою очередь, получает удобный инструментарий для управления товарами и заказами, включая подтверждение, удаление и редактирование данных.

В процессе разработки были получены практические навыки работы с библиотекой `python-telegram-bot`, организации клиент-серверного взаимодействия, проектирования структур SQLite-базы данных и реализации пошаговых интерфейсов для конечного пользователя.

В будущем систему можно дополнить интеграцией с платёжными системами, расширенной статистикой заказов, системой уведомлений по расписанию, а также улучшить визуальную часть - например, добавив поддержку кастомных изображений товаров и интерактивных фильтров. Полученный опыт разработки Telegram-бота с полной бизнес-логикой станет ценным в профессиональной деятельности и поможет в дальнейшем карьерном развитии.

Список использованных источников

1. Официальная документация Telegram Bot API. – Core.telegram.org. – Режим доступа: <https://core.telegram.org/bots/api> (дата обращения: 09.05.2025).
2. Документация python-telegram-bot. – Docs.python-telegram-bot.org. – Режим доступа: <https://docs.python-telegram-bot.org> (дата обращения: 09.05.2025).
3. SQLite: Встроенная реляционная СУБД. – SQLite.org. – Режим доступа: <https://www.sqlite.org> (дата обращения: 10.05.2025).
4. Хабр: «Создание архитектуры Telegram-бота» / Павел Рудь. – Habr.com. – Режим доступа: <https://habr.com/ru/articles/541728/> (дата обращения: 13.05.2025).
5. Medium: «Создание интернет-магазина в Telegram с нуля» / Виктор Колесников. – Medium.com. – Режим доступа: <https://medium.com/@kolesnikov.telegrambotshop> (дата обращения: 13.05.2025).
6. Real Python: «Working with SQLite in Python». – Realpython.com. – Режим доступа: <https://realpython.com/python-sqlite-sqlalchemy/> (дата обращения: 12.05.2025).
7. GitHub: Примеры Telegram-ботов на Python. – Github.com. – Режим доступа: <https://github.com/python-telegram-bot/python-telegram-bot> (дата обращения: 14.05.2025).

Приложение А

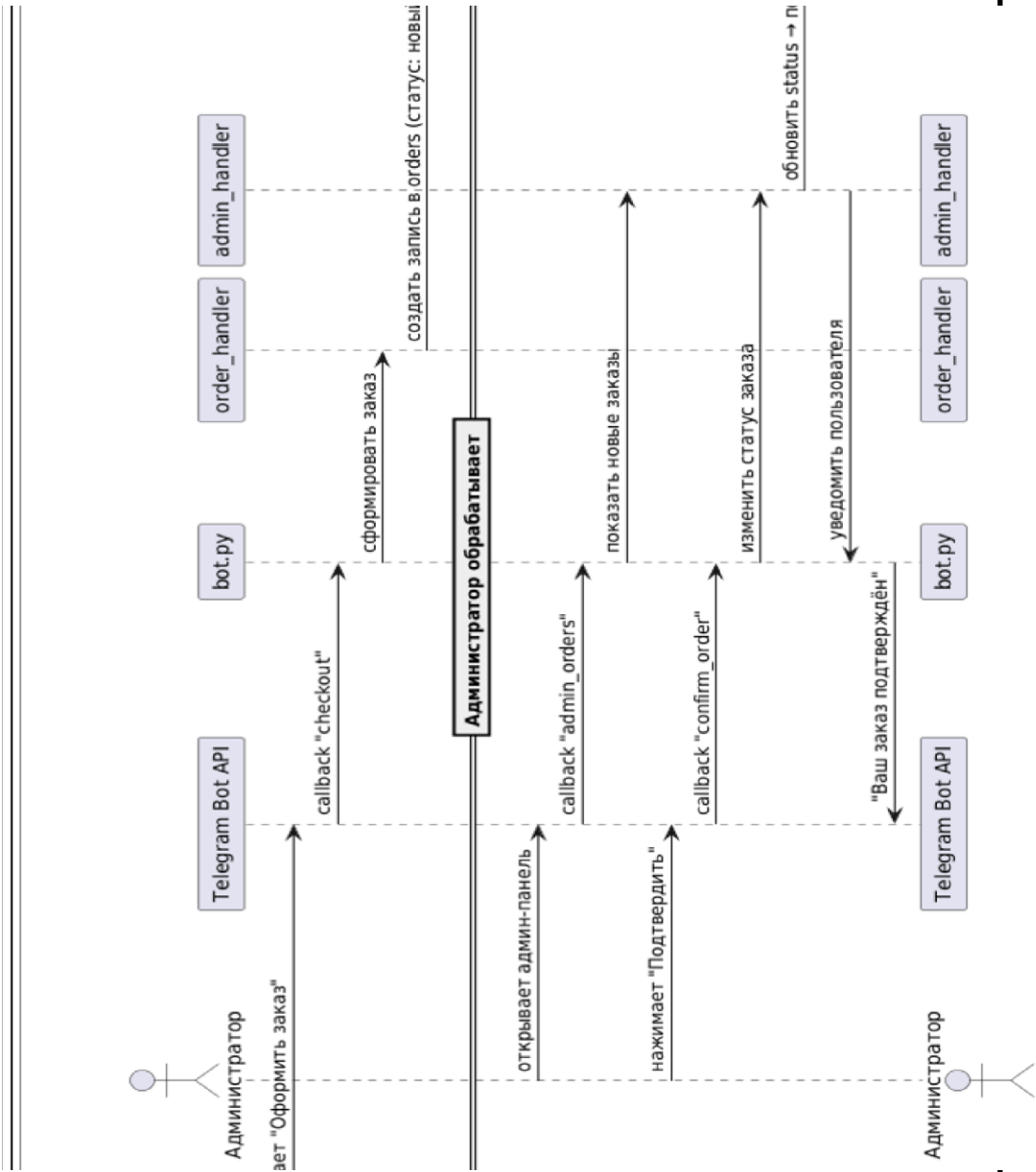
(обязательное)

Чат-бот для электронной коммерции

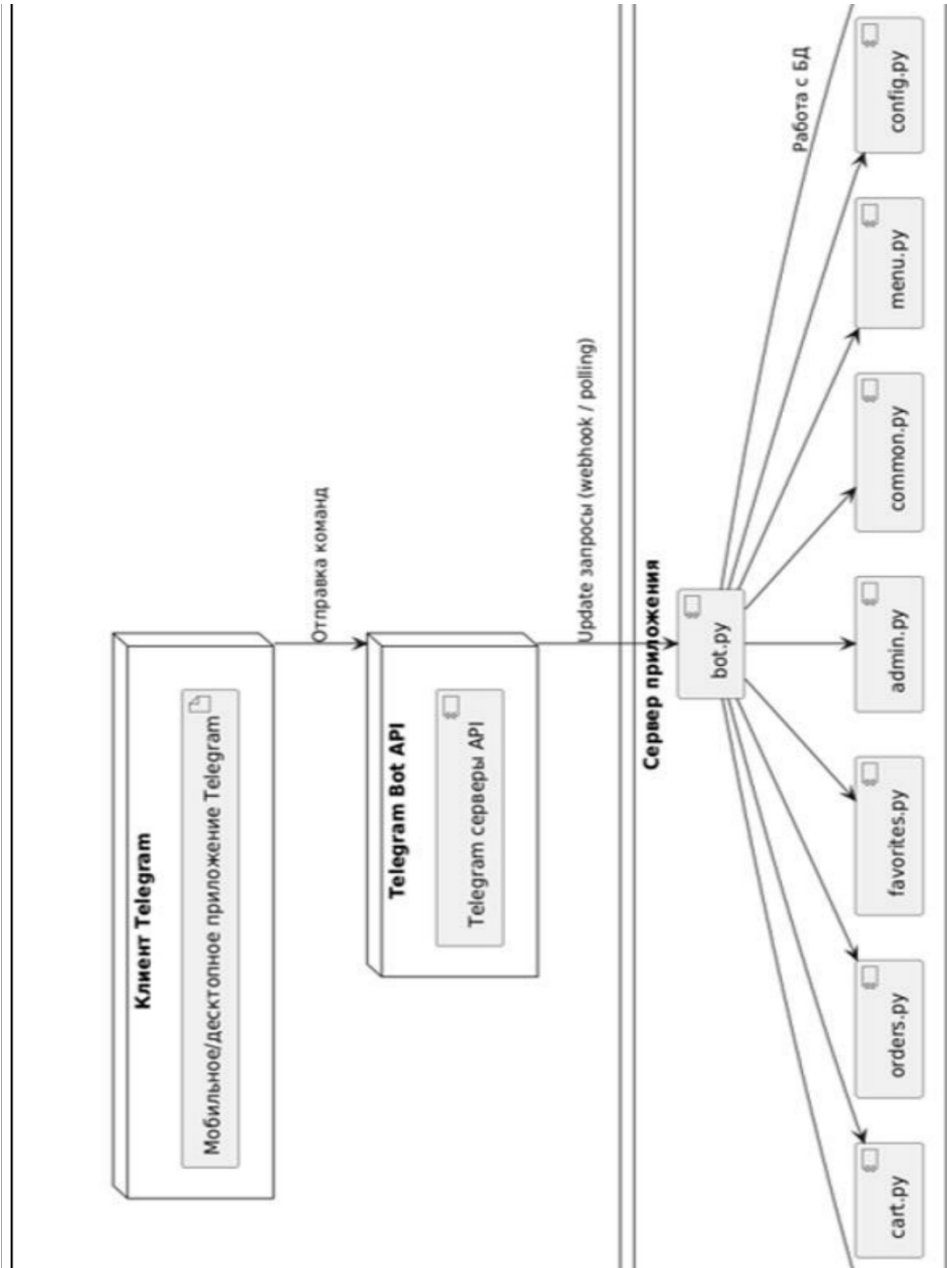
UML-диаграммы

					ПГУ 1.090301.36.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 - Диаграмма последовательностей оформления и подтверждения заказа

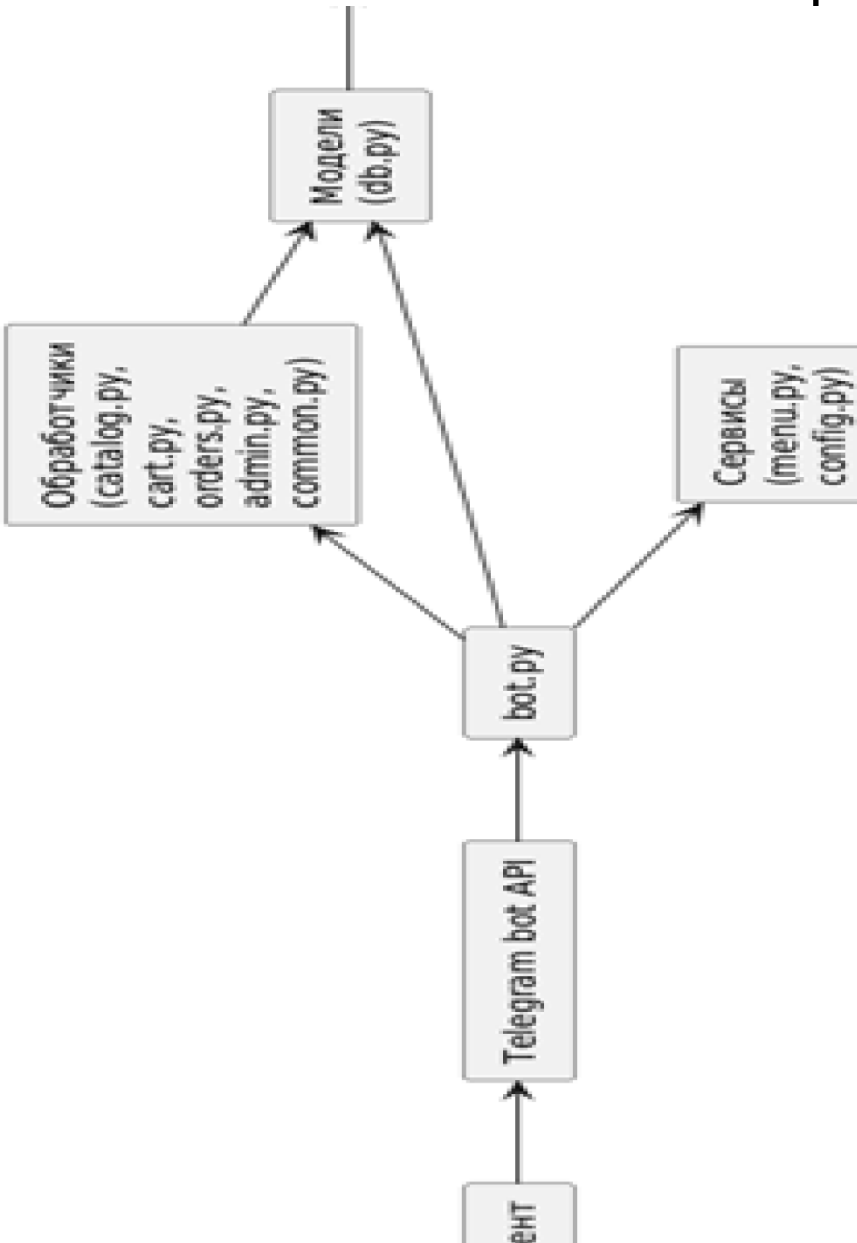


Приложение А.2 - Диаграмма развертывания

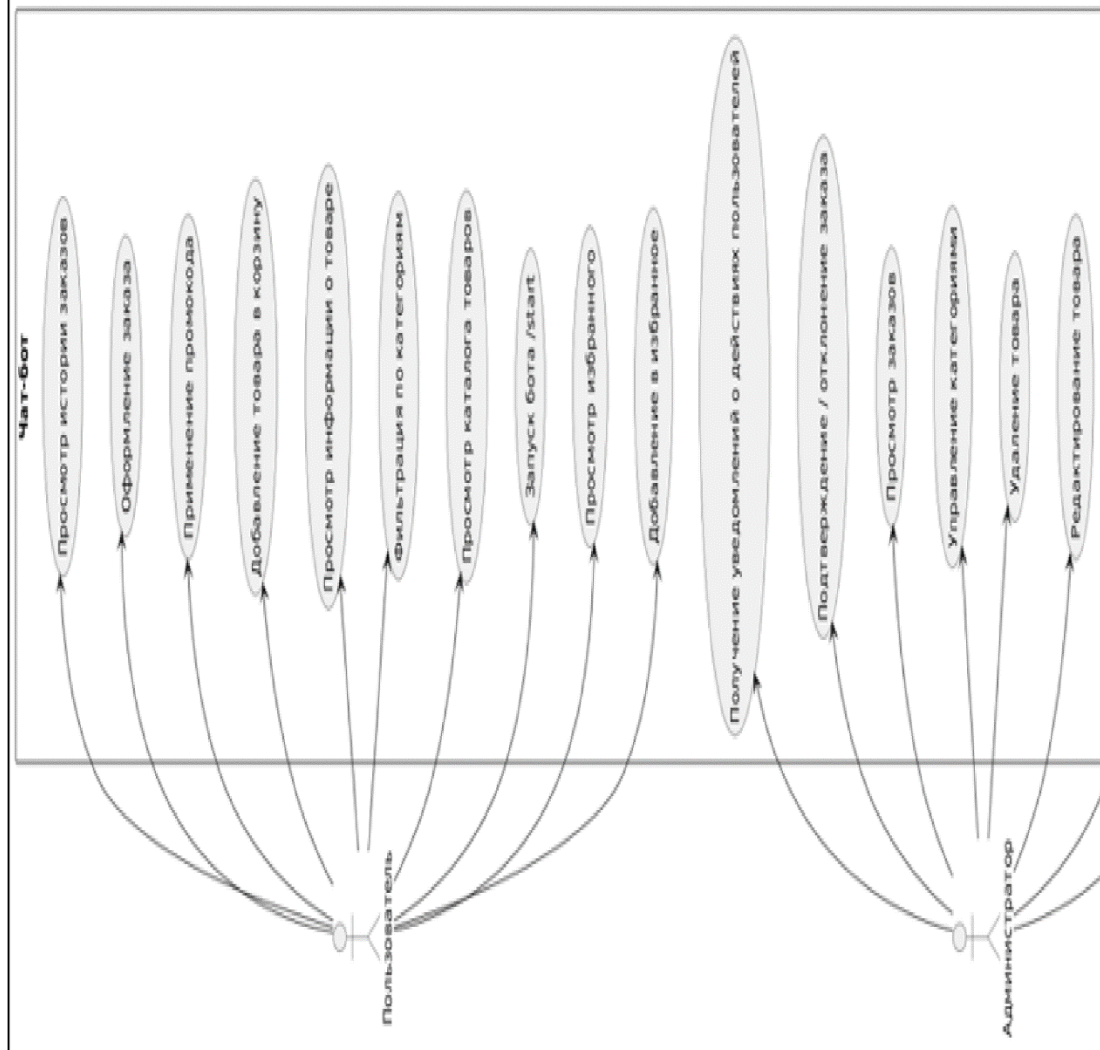


Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.3 - Диаграмма модулей



Приложение А.4 - Диаграмма вариантов использования



Приложение Б

(обязательное)

Чат-бот для электронной коммерции

Листинг программы

					ПГУ 1.090301.36.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Файл bot.py

```
import logging
from telegram.constants import ParseMode
from telegram.ext import (
    ApplicationBuilder,
    CommandHandler,
    CallbackQueryHandler,
    MessageHandler,
    filters,
)
import config
config.init_db()

from handlers.common import setup_common_handlers
from handlers.admin import (
    admin_panel,
    admin_add_category,
    admin_add_cat_gender,
    handle_new_category,
    setup_admin_handlers,
)
from handlers.catalog import setup_catalog_handlers
from handlers.cart import setup_cart_handlers
from handlers.favorites import setup_fav_handlers
from handlers.orders import setup_order_handlers
from utils.menu import main_menu

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s -
%(message)s",
    level=logging.INFO,
)
logger = logging.getLogger(__name__)

async def start_cmd(update, context):
    cid = update.effective_chat.id
    # Сбрасываем предыдущие состояния
    config.user_states[cid] = None
    await context.bot.send_message(
        chat_id=cid,
        text="Добро пожаловать! Выберите пункт меню:",
        reply_markup=main_menu(cid),
        parse_mode=ParseMode.HTML,
    )

if __name__ == '__main__':
    config.init_db()

    app = ApplicationBuilder().token(config.TOKEN).build()
    app.add_handler(CommandHandler('start', start_cmd))
    setup_common_handlers(app)

    setup_catalog_handlers(app)
    setup_cart_handlers(app)
```

```

        setup_fav_handlers(app)
        setup_order_handlers(app)

        app.add_handler(CallbackQueryHandler(admin_panel,
        pattern="^admin_panel$"))

    app.add_handler(CallbackQueryHandler(admin_add_category,
    pattern="^admin_add_category$"))

    app.add_handler(CallbackQueryHandler(admin_add_cat_gender,
    pattern="^admin_add_cat_gender:"))

        setup_admin_handlers(app)

        print("Bot started...")
        app.run_polling()

```

Файл db.py

```

import sqlite3
import os
import json

DB_PATH = os.path.join(os.path.dirname(__file__),
"shop.db")

def get_connection():
    """
    Открывает соединение с базой (sqlite3.Row), но не
    создаёт таблицы.
    """
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    """
    Создаёт (если нужно) и мигрирует таблицы:
        - categories
        - products
        - cart
        - favorites
        - orders (с автоматическим добавлением колонок items,
        promocode, discounted_price, delivery, username)
    """
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("""
    CREATE TABLE IF NOT EXISTS categories (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        gender TEXT NOT NULL,

```

```

        UNIQUE(name, gender)
    ) """)

cur.execute("""
CREATE TABLE IF NOT EXISTS products (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    gender TEXT NOT NULL,
    category TEXT NOT NULL,
    price INTEGER NOT NULL,
    image TEXT,
    UNIQUE(name, gender, category)
) """)

cur.execute("""
CREATE TABLE IF NOT EXISTS cart (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    chat_id INTEGER NOT NULL,
    product TEXT NOT NULL,
    size TEXT NOT NULL
) """)

cur.execute("""
CREATE TABLE IF NOT EXISTS favorites (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    chat_id INTEGER NOT NULL,
    product TEXT NOT NULL,
    size TEXT NOT NULL
) """)

cur.execute("""
CREATE TABLE IF NOT EXISTS orders (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    chat_id INTEGER NOT NULL,
    status TEXT DEFAULT 'new'
) """)

cur.execute("PRAGMA table_info(orders)")
existing_cols = {row["name"] for row in cur.fetchall()}

if "items" not in existing_cols:
    cur.execute("ALTER TABLE orders ADD COLUMN items
TEXT NOT NULL DEFAULT '[]'")
if "promocode" not in existing_cols:
    cur.execute("ALTER TABLE orders ADD COLUMN
promocode TEXT")
if "discounted_price" not in existing_cols:
    cur.execute("ALTER TABLE orders ADD COLUMN
discounted_price INTEGER")
if "delivery" not in existing_cols:
    cur.execute("ALTER TABLE orders ADD COLUMN delivery
INTEGER")
if "username" not in existing_cols:
    cur.execute("ALTER TABLE orders ADD COLUMN username
TEXT")

conn.commit()

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

58

```

conn.close()

def _ensure_username_column():

    conn = get_connection()
    cur = conn.cursor()
    cur.execute("PRAGMA table_info(orders)")
    existing_cols = {row["name"] for row in cur.fetchall()}

    if "username" not in existing_cols:
        cur.execute("ALTER TABLE orders ADD COLUMN username
TEXT")
        conn.commit()

    conn.close()

def add_category(name: str, gender: str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "INSERT INTO categories(name, gender) VALUES (?,
?),",
        (name, gender)
    )
    conn.commit()
    conn.close()

def remove_category(name: str, gender: str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "DELETE FROM categories WHERE name = ? AND gender =
?",
        (name, gender)
    )
    conn.commit()
    conn.close()

def get_categories_by_gender(gender: str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT name FROM categories WHERE gender = ? ORDER
BY name",
        (gender,)
    )
    rows = cur.fetchall()
    conn.close()
    return [row["name"] for row in rows]

def add_product(name: str, gender: str, category: str,
price: int, image: str = None):

```

```

        conn = get_connection()
        cur = conn.cursor()
        cur.execute(
            "INSERT INTO products(name, gender, category,
price, image) VALUES (?, ?, ?, ?, ?)",
            (name, gender, category, price, image),
        )
        conn.commit()
        conn.close()

def remove_product(name: str, gender: str, category: str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "DELETE FROM products WHERE name = ? AND gender = ?
AND category = ?",
        (name, gender, category),
    )
    conn.commit()
    conn.close()

def get_products_by_gender_and_category(gender: str,
category: str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT name FROM products WHERE gender = ? AND
category = ? ORDER BY name",
        (gender, category),
    )
    rows = cur.fetchall()
    conn.close()
    return [row["name"] for row in rows]

def get_price(product_name: str) -> int:
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT price FROM products WHERE name = ?",
        (product_name,),
    )
    row = cur.fetchone()
    conn.close()
    return row["price"] if row else 0

def get_product_image(product_name: str) -> str | None:
    """
    Возвращает file_id картинки товара (или None).
    """
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT image FROM products WHERE name = ?",

```

```

        (product_name,),
    )
    row = cur.fetchone()
    conn.close()
    if row and row["image"]:
        return row["image"]
    return None

def add_to_cart(chat_id: int, product: str, size: str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "INSERT INTO cart(chat_id, product, size) VALUES
(?, ?, ?)",
        (chat_id, product, size),
    )
    conn.commit()
    conn.close()

def remove_from_cart(chat_id: int, index: int):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT id FROM cart WHERE chat_id = ?
ORDER BY id", (chat_id,))
    rows = cur.fetchall()
    if 0 <= index < len(rows):
        item_id = rows[index]["id"]
        cur.execute("DELETE FROM cart WHERE id = ?",
(item_id,))
    conn.commit()
    conn.close()

def get_cart(chat_id: int):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT product, size FROM cart WHERE chat_id = ?
ORDER BY id",
        (chat_id,),
    )
    rows = cur.fetchall()
    conn.close()
    return [{"name": r["product"], "size": r["size"]} for r
in rows]

def clear_cart(chat_id: int):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("DELETE FROM cart WHERE chat_id = ?",
(chat_id,))
    conn.commit()
    conn.close()

```

Изм.	Лист	№ докум.	Подпись	Дата

```

def add_to_favorites(chat_id: int, product: str, size:
str):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "INSERT INTO favorites(chat_id, product, size)
VALUES (?, ?, ?)",
        (chat_id, product, size),
    )
    conn.commit()
    conn.close()

def remove_from_favorites(chat_id: int, index: int):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT id FROM favorites WHERE chat_id = ?
ORDER BY id", (chat_id,))
    rows = cur.fetchall()
    if 0 <= index < len(rows):
        item_id = rows[index]["id"]
        cur.execute("DELETE FROM favorites WHERE id = ?",
(item_id,))
    conn.commit()
    conn.close()

def get_favorites(chat_id: int):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT product, size FROM favorites WHERE chat_id
= ? ORDER BY id",
        (chat_id,),
    )
    rows = cur.fetchall()
    conn.close()
    return [{"name": r["product"], "size": r["size"]} for r
in rows]

def add_order(user_id: int, username: str | None, items:
list[dict], promocode: str | None = None):

    conn = get_connection()
    cur = conn.cursor()
    items_json = json.dumps(items, ensure_ascii=False)
    total = sum(it["price"] for it in items)
    cur.execute(
        "INSERT INTO orders(chat_id, username, items,
promocode, discounted_price, delivery) "
        "VALUES (?, ?, ?, ?, ?, ?)",
        (user_id, username, items_json, promocode, total,
None)
    )

```

```

        conn.commit()
        conn.close()

    def get_orders_by_status(status: str, limit: int = 10) ->
list[sqlite3.Row]:

        _ensure_username_column()

        conn = get_connection()
        rows = conn.execute(
            "SELECT id, chat_id, username, items, promocode,
discounted_price, delivery, status "
            "FROM orders WHERE status = ? ORDER BY id DESC
LIMIT ?",
            (status, limit)
        ).fetchall()
        conn.close()
        return rows

    def update_order_status(order_id: int, new_status: str):
        conn = get_connection()
        cur = conn.cursor()
        cur.execute(
            "UPDATE orders SET status = ? WHERE id = ?",
            (new_status, order_id)
        )
        conn.commit()
        conn.close()

    def get_orders_by_user_id(chat_id: int) ->
list[sqlite3.Row]:

        _ensure_username_column()

        conn = get_connection()
        rows = conn.execute(
            "SELECT id, chat_id, username, items, promocode,
discounted_price, delivery, status "
            "FROM orders WHERE chat_id = ? ORDER BY id DESC",
            (chat_id,)
        ).fetchall()
        conn.close()
        return rows

```

Файл config.py

```

import config
from telegram import InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.ext import CallbackQueryHandler
from utils.menu import main_menu

```

```

async def back_to_main(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    await context.bot.send_message(
        chat_id=cid,
        text="Главное меню:",
        reply_markup=main_menu(cid)
    )

def setup_common_handlers(app):
    app.add_handler(CallbackQueryHandler(back_to_main,
pattern="^back_to_main$"))

```

Файл utils/menu.py

```

import config
from telegram import InlineKeyboardButton,
InlineKeyboardMarkup

def root_gender_keyboard():
    kb = [[InlineKeyboardButton(g,
callback_data=f"choose_gender:{g}")] for g in config.GENDERS]
    kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="back_to_main")])
    return InlineKeyboardMarkup(kb)

def category_keyboard(gender: str):
    cats = config.get_categories_by_gender(gender)
    kb = [[InlineKeyboardButton(cat,
callback_data=f"select_category:{gender}:{cat}")] for cat in
cats]
    kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="menu_catalog")])
    return InlineKeyboardMarkup(kb)

def product_keyboard(gender: str, category: str):
    prods =
config.get_products_by_gender_and_category(gender, category)
    kb = [[InlineKeyboardButton(prod,
callback_data=f"select_product:{gender}:{category}:{prod}")] for
prod in prods]
    kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data=f"choose_gender:{gender}")])
    return InlineKeyboardMarkup(kb)

def size_keyboard(gender, category, product, destination):

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        size_type = config.category_types.get((gender,
category))
        print(f"[DEBUG] gender={gender}, category={category},
size_type={size_type}, dest={destination}")
        if size_type == "другое":
            return None # не строим кнопки размеров
        sizes = config.SIZE_OPTIONS.get(gender,
{ }).get(size_type, [])
        print(f"[DEBUG] sizes={sizes}")
        kb = [
            [
                InlineKeyboardButton(
                    sz,

callback_data=f"size:{destination}:{gender}:{category}:{product}
:{sz}"

                )
            ]
            for sz in sizes
        ]
        kb.append([
            InlineKeyboardButton(
                "⬅️ Назад",

callback_data=f"select_product:{gender}:{category}:{product}"

            )
        ])
        return InlineKeyboardMarkup(kb)

    async def send_single_menu(bot, cid, *args, **kwargs):
        from config import last_bot_message_id
        if cid in last_bot_message_id:
            try:
                await bot.delete_message(chat_id=cid,
message_id=last_bot_message_id[cid])
            except Exception as e:
                pass
            msg = await bot.send_message(chat_id=cid, *args,
**kwargs)
            last_bot_message_id[cid] = msg.message_id
            return msg

    async def send_single_photo(bot, cid, *args, **kwargs):
        from config import last_bot_message_id
        if cid in last_bot_message_id:
            try:
                await bot.delete_message(chat_id=cid,
message_id=last_bot_message_id[cid])
            except Exception:
                pass
            msg = await bot.send_photo(chat_id=cid, *args,
**kwargs)
            last_bot_message_id[cid] = msg.message_id
            return msg

    def main_menu(chat_id: int):

```

```

        kb = [
            [InlineKeyboardButton("📖 Каталог",
callback_data="menu_catalog")],
            [InlineKeyboardButton("🛒 Корзина",
callback_data="menu_cart")],
            [InlineKeyboardButton("❤️ Избранное",
callback_data="menu_fav")],
            [InlineKeyboardButton("📦 Мои заказы",
callback_data="menu_orders")],
        ]
        if chat_id in config.ADMIN_IDS:
            kb.append([InlineKeyboardButton("🔑 Админ-панель",
callback_data="admin_panel")])
        return InlineKeyboardMarkup(kb)

def get_items_text(items: list):
    if not items:
        return "- пусто -"
    lines = []
    for idx, it in enumerate(items, start=1):
        lines.append(f"{idx}. {it['name']} ({it['size']}) -
{it['price']}₽")
    return "\n".join(lines)

```

Файл handlers/admin

```

# handlers/admin.py
import logging
import json
from telegram import InlineKeyboardButton,
InlineKeyboardMarkup, Update
from telegram.constants import ParseMode
from telegram.ext import CallbackQueryHandler,
MessageHandler, filters, ContextTypes

import config
from utils.menu import main_menu, send_single_menu

from config import (
    add_category, remove_category,
get_categories_by_gender,
    add_product, remove_product,
get_products_by_gender_and_category,
    get_orders_by_status, update_order_status,
get_orders_by_user_id,
    update_product_price, update_product_image,
get_db_connection
)

logger = logging.getLogger(__name__)
class StateFilter(filters.MessageFilter):
    def __init__(self, state_name: str):
        super().__init__()
        self.state_name = state_name

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

66

```

def filter(self, message) -> bool:
    chat = getattr(message, "chat", None)
    if not chat:
        return False
    cid = chat.id
    st = config.user_states.get(cid)
    if isinstance(st, (tuple, list)):
        return bool(st) and st[0] == self.state_name
    return st == self.state_name

STATE_ADD_CATEGORY = StateFilter("adding_category")
STATE_REM_CATEGORY = StateFilter("removing_category")
STATE_ADD_PROD_NAME = StateFilter("adding_product_name")
STATE_ADD_PROD_PRICE = StateFilter("adding_product_price")
STATE_ADD_PROD_IMAGE = StateFilter("adding_product_image")
STATE_EDIT_FIELDS = StateFilter("editing_fields")
STATE_EDIT_VALUE = StateFilter("editing_value")
STATE_CANCEL_ORDER = StateFilter("awaiting_cancel_order")
STATE_SEARCH_USER = StateFilter("searching_user_orders")

def html_escape(text: str) -> str:
    return (
        text.replace("&", "&amp;")
        .replace("<", "&lt;")
        .replace(">", "&gt;")
        .replace("'", "&quot;")
        .replace('"', "&#39;")
    )

async def send_admin_panel(bot, cid):
    kb = [
        [InlineKeyboardButton("➕ Добавить категорию",
            callback_data="admin_add_category")],
        [InlineKeyboardButton("➖ Удалить категорию",
            callback_data="admin_remove_category")],
        [InlineKeyboardButton("➕ Добавить товар",
            callback_data="admin_add_product")],
        [InlineKeyboardButton("➖ Удалить товар",
            callback_data="admin_remove_product")],
        [InlineKeyboardButton("✎ Редактировать товар",
            callback_data="admin_edit_product")],
        [InlineKeyboardButton("📦 Заказы",
            callback_data="admin_orders")],
        [InlineKeyboardButton("⬅️ Назад",
            callback_data="back_to_main")],
    ]
    await send_single_menu(
        bot,
        cid,
        text="🔗 <b>Админ-панель</b>:",
    )

```

```

        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(kb),
    )

    async def admin_panel(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        """
        CallbackQueryHandler → меню «Админ-панель»
        """
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        try:
            # Удаляем «прошлое» сообщение с кнопками, чтобы не
захламлять чат
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass
        await send_admin_panel(context.bot, cid)

    async def admin_add_category(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        # Выводим кнопки с полами (из config.GENDERS)
        kb = [[InlineKeyboardButton(g,
callback_data=f"admin_add_cat_gender:{g}")] for g in
config.GENDERS]
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
        await send_single_menu(
            context.bot,
            cid,
            text="Выберите <b>пол</b> для новой категории:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_add_cat_gender(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        """
        После того как админ нажал «+ Добавить категорию» и
выбрал пол → просим ввести название.
        """
        query = update.callback_query
        await query.answer()

```

```

        cid = query.message.chat.id
        _, gender = query.data.split(":", 1)

        config.user_states[cid] = ("adding_category", gender)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        await send_single_menu(
            context.bot,
            cid,
            text=(
                f"Введите <b>название</b> новой категории для
<i>{html_escape(gender)}</i>.\n"
                "Пример: <code>Спортивная одежда</code>"
            ),
            parse_mode=ParseMode.HTML,
        )

    async def handle_new_category(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        msg = getattr(update.message, "text", "")
        state = config.user_states.get(cid)

        print(f"[handle_new_category] TRIGGERED")
        print(f"[handle_new_category] cid={cid}, state={state},
text={msg}")

        if not state or state[0] != "adding_category":
            print(f"[handle_new_category] ❌ Invalid state,
exiting")
            return

        _, gender = state
        new_cat = msg.strip()
        if not new_cat:
            await update.message.reply_text("❌ Название не
может быть пустым.")
            return

        try:
            add_category(new_cat, gender)
            print(f"[handle_new_category] ✅ Category added:
{new_cat} / {gender}")
        except Exception as e:
            print(f"[handle_new_category] ❌ Error while adding
category: {e}")
            await update.message.reply_text("❌ Такая категория
уже существует.")
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

```

```

        # Показываем кнопки для выбора типа размерной сетки
        kb = [
            [InlineKeyboardButton("Верх",
                callback_data=f"admin_set_cat_type:верх:{gender}:{new_cat}")],
            [InlineKeyboardButton("Низ",
                callback_data=f"admin_set_cat_type:низ:{gender}:{new_cat}")],
            [InlineKeyboardButton("Обувь",
                callback_data=f"admin_set_cat_type:обувь:{gender}:{new_cat}")],
            [InlineKeyboardButton("Другое (без размеров)",
                callback_data=f"admin_set_cat_type:другое:{gender}:{new_cat}")],
        ]
        await update.message.reply_text(
            "Выберите <b>тип размерной сетки</b> для новой
категории:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb)
        )

    async def admin_set_cat_type(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()

        try:
            await
context.bot.delete_message(chat_id=query.message.chat.id,
message_id=query.message.message_id)
        except Exception:
            pass

        _, size_type, gender, cat = query.data.split(":")
        config.category_types[(gender, cat)] = size_type
        await context.bot.send_message(
            chat_id=query.message.chat.id,
            text=f"Тип размерной сетки для категории
<b>{cat}</b> установлен: <b>{size_type}</b>",
            parse_mode=ParseMode.HTML,
        )
        config.user_states[query.message.chat.id] = None
        await send_admin_panel(context.bot,
query.message.chat.id)

    async def admin_remove_category(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        kb = [[InlineKeyboardButton(g,
callback_data=f"admin_rem_cat_gender:{g}")]] for g in
config.GENDERS]

```

```

        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel"))
        await context.bot.send_message(
            chat_id=cid,
            text="Выберите <b>пол</b> категории для удаления:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_rem_cat_gender(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        _, gender = query.data.split(":", 1)

        config.user_states[cid] = ("removing_category", gender)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        cats = get_categories_by_gender(gender)
        if not cats:
            await context.bot.send_message(
                chat_id=cid,
                text=f"❗ У вас нет категорий для
<i>{html_escape(gender)}</i>.",
                parse_mode=ParseMode.HTML,
            )
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

        kb = [[InlineKeyboardButton(cat,
callback_data=f"admin_do_rem_category:{cat}") for cat in cats]
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel"))
        await send_single_menu(
            context.bot,
            cid,
            text=f"Категории для
<b>{html_escape(gender)}</b>:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_do_rem_category(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        state = config.user_states.get(cid)

```

```

        if not state or state[0] != "removing_category":
            await send_admin_panel(context.bot, cid)
            return

_, gender = state
cat = query.data.split(":", 1)[1]
try:
    remove_category(cat, gender)
except Exception:
    config.user_states[cid] = None
    await send_admin_panel(context.bot, cid)
    return

try:
    await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
except:
    pass

await send_single_menu(
    context.bot,
    cid,
    text=f"✅ Категория «<b>{html_escape(cat)}</b>»
удалена из <i>{html_escape(gender)}</i>.",
    parse_mode=ParseMode.HTML,
)
config.user_states[cid] = None
await send_admin_panel(context.bot, cid)

async def admin_add_product(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id
    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    kb = [[InlineKeyboardButton(g,
callback_data=f"admin_add_prod_gender:{g}")] for g in
config.GENDERS]
    kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
    await context.bot.send_message(
        chat_id=cid,
        text="Выберите <b>пол</b> нового товара:",
        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(kb),
    )

async def admin_add_prod_gender(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query

```

```

        await query.answer()
        cid = query.message.chat.id
        _, gender = query.data.split(":", 1)

        config.user_states[cid] = ("adding_product_cat",
gender)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        cats = get_categories_by_gender(gender)
        if not cats:
            await context.bot.send_message(
                chat_id=cid,
                text=f"❗ У вас нет категорий для
<i>{html_escape(gender)}</i>.",
                parse_mode=ParseMode.HTML,
            )
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

        kb = [[InlineKeyboardButton(cat,
callback_data=f"admin_add_prod_cat:{cat}")] for cat in cats]
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
        await send_single_menu(
            context.bot,
            cid,
            text=f"Выберите категорию для
<b>{html_escape(gender)}</b>:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_add_prod_cat(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        prev = config.user_states.get(cid)
        if not prev or prev[0] != "adding_product_cat":
            await send_admin_panel(context.bot, cid)
            return

        _, gender = prev
        chosen_cat = query.data.split(":", 1)[1]

        config.user_states[cid] = ("adding_product_name",
gender, chosen_cat)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)

```

```

except:
    pass

await send_single_menu(
    context.bot,
    cid,
    text=(
        f"Введите <b>название</b> нового товара в
категории «<i>{html_escape(chosen_cat)}</i>»: "
    ),
    parse_mode=ParseMode.HTML,
)

async def add_product_name(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    cid = update.effective_chat.id
    state = config.user_states.get(cid)
    logger.info(f"add_product_name STATE: {state}")
    import builtins
    builtins.open("admin_test_log.txt",
"a").write(f"TRIGGER {update.effective_chat.id}
{getattr(update.message, 'text', '')}\n")
    if not state or state[0] != "adding_product_name":
        return
    _, gender, chosen_cat = state
    name = update.message.text.strip()
    if not name:
        await update.message.reply_text("✗ Название товара
не может быть пустым.")
        return
    existing = get_products_by_gender_and_category(gender,
chosen_cat)
    if name in existing:
        await update.message.reply_text("✗ Такой товар уже
есть в этой категории.")
        return
    config.user_states[cid] = ("adding_product_price",
gender, chosen_cat, name)
    await update.message.reply_text("Введите <b>цену</b>
нового товара (только цифры):", parse_mode=ParseMode.HTML)

async def add_product_price(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    cid = update.effective_chat.id
    state = config.user_states.get(cid)
    logger.info(f"add_product_price STATE: {state}")
    if not state or state[0] != "adding_product_price":
        return
    _, gender, chosen_cat, name = state
    text = update.message.text.strip()
    if not text.isdigit():
        await update.message.reply_text("✗ Цена должна
быть числом (только цифры).")
        return
    price = int(text)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        config.user_states[cid] = ("adding_product_image",
gender, chosen_cat, name, price)
        await update.message.reply_text("Пришлите
<b>картинку</b> нового товара (или отправьте <code>/skip</code>,
чтобы пропустить):", parse_mode=ParseMode.HTML)

    async def add_product_image(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        state = config.user_states.get(cid)
        logger.info(f"add_product_image STATE: {state}")
        if not state or state[0] != "adding_product_image":
            return
        _, gender, chosen_cat, name, price = state
        image = update.message.photo[-1].file_id if
update.message.photo else None
        try:
            add_product(name, gender, chosen_cat, price, image)
        except Exception as e:
            logger.error(f"Error while adding product: {e}")
            await update.message.reply_text("✗ Не удалось
добавить товар.")
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return
        await update.message.reply_text(
            f"✅ Товар «<b>{html_escape(name)}</b>» ({price}₽)
успешно добавлен в категорию
«<i>{html_escape(chosen_cat)}</i>».",
            parse_mode=ParseMode.HTML,
        )
        config.user_states[cid] = None
        await send_admin_panel(context.bot, cid)

    async def skip_product_image(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        state = config.user_states.get(cid)
        logger.info(f"skip_product_image STATE: {state}")
        if not state or state[0] != "adding_product_image":
            return
        _, gender, chosen_cat, name, price = state
        try:
            add_product(name, gender, chosen_cat, price, None)
        except Exception as e:
            logger.error(f"Error while skipping image for
product: {e}")
            await update.message.reply_text("✗ Не удалось
добавить товар.")
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return
        await update.message.reply_text(
            f"✅ Товар «<b>{html_escape(name)}</b>» ({price}₽)
успешно добавлен в категорию «<i>{html_escape(chosen_cat)}</i>»

```

```

без картинки.",
        parse_mode=ParseMode.HTML,
    )
    config.user_states[cid] = None
    await send_admin_panel(context.bot, cid)

    async def admin_remove_product(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        kb = [[InlineKeyboardButton(g,
callback_data=f"admin_rem_prod_gender:{g}")] for g in
config.GENDERS]
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
        await context.bot.send_message(
            chat_id=cid,
            text="Выберите <b>пол</b> товара для удаления:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_rem_prod_gender(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        _, gender = query.data.split(":", 1)

        config.user_states[cid] = ("removing_product_cat",
gender)

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        cats = get_categories_by_gender(gender)
        if not cats:
            await send_single_menu(
                context.bot,
                cid,
                text=f"❗ У вас нет категорий для
<i>{html_escape(gender)}</i>.",
                parse_mode=ParseMode.HTML,
            )

```

```

        config.user_states[cid] = None
        await send_admin_panel(context.bot, cid)
        return

    kb = [[InlineKeyboardButton(cat,
callback_data=f"admin_rem_prod_cat:{cat}")] for cat in cats]
    kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
    await send_single_menu(
        context.bot,
        cid,
        text=f"Категории для
<b>{html_escape(gender)}</b>:",
        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(kb),
    )

    async def admin_rem_prod_cat(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        prev = config.user_states.get(cid)
        if not prev or prev[0] != "removing_product_cat":
            await send_admin_panel(context.bot, cid)
            return

        _, gender = prev
        chosen_cat = query.data.split(":", 1)[1]

        config.user_states[cid] = ("removing_product", gender,
chosen_cat)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        prods = get_products_by_gender_and_category(gender,
chosen_cat)
        if not prods:
            await send_single_menu(
                context.bot,
                cid,
                text=f"❗ В категории
«<i>{html_escape(chosen_cat)}</i>» нет товаров.",
                parse_mode=ParseMode.HTML,
            )
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

        kb = [[InlineKeyboardButton(prod,
callback_data=f"admin_do_rem_prod:{prod}")] for prod in prods]
        kb.append([InlineKeyboardButton("⬅️ Назад",

```

```

callback_data="admin_panel"]])
    await context.bot.send_message(
        chat_id=cid,
        text=f"Товары в категории
<b>{html_escape(chosen_cat)}</b>:",
        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(kb),
    )

    async def admin_do_rem_prod(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        prev = config.user_states.get(cid)
        if not prev or prev[0] != "removing_product":
            await send_admin_panel(context.bot, cid)
            return

        _, gender, chosen_cat = prev
        prod_name = query.data.split(":", 1)[1]
        try:
            remove_product(prod_name, gender, chosen_cat)
        except Exception as e:
            logger.error(f"Error removing product: {e}")
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        await send_single_menu(
            context.bot,
            cid,
            text=f"☑ Товар «<b>{html_escape(prod_name)}</b>»
удалён из категории «<i>{html_escape(chosen_cat)}</i>».",
            parse_mode=ParseMode.HTML,
        )
        config.user_states[cid] = None
        await send_admin_panel(context.bot, cid)

    async def admin_edit_product(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:

```

```

        pass

        kb = [[InlineKeyboardButton(g,
callback_data=f"admin_edit_prod_gender:{g}")] for g in
config.GENDERS]
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
        await send_single_menu(
            context.bot,
            cid,
            text="Выберите <b>пол</b> товара для
редактирования:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_edit_prod_gender(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        _, gender = query.data.split(":", 1)

        config.user_states[cid] = ("editing_product_cat",
gender)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        cats = get_categories_by_gender(gender)
        if not cats:
            await send_single_menu(
                context.bot,
                cid,
                text=f"❗ У вас нет категорий для
<i>{html_escape(gender)}</i>.",
                parse_mode=ParseMode.HTML,
            )
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

        kb = [[InlineKeyboardButton(cat,
callback_data=f"admin_edit_prod_cat:{cat}")] for cat in cats]
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")])
        await send_single_menu(
            context.bot,
            cid,
            text=f"Категории для
<b>{html_escape(gender)}</b>:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),

```

```

    )

    async def admin_edit_prod_cat(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id
        prev = config.user_states.get(cid)
        if not prev or prev[0] != "editing_product_cat":
            await send_admin_panel(context.bot, cid)
            return

        _, gender = prev
        chosen_cat = query.data.split(":", 1)[1]

        config.user_states[cid] = ("editing_product", gender,
chosen_cat)
        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        prods = get_products_by_gender_and_category(gender,
chosen_cat)
        if not prods:
            await send_single_menu(
                context.bot,
                cid,
                text=f"❏ В категории
«<i>{html_escape(chosen_cat)}</i>» нет товаров.",
                parse_mode=ParseMode.HTML,
            )
            config.user_states[cid] = None
            await send_admin_panel(context.bot, cid)
            return

        kb = [[InlineKeyboardButton(prod,
callback_data=f"admin_edit_prod_select:{prod}")] for prod in
prods]

        kb.append([InlineKeyboardButton("❏ Назад",
callback_data="admin_panel")])
        await send_single_menu(
            context.bot,
            cid,
            text=f"Товары в категории
<b>{html_escape(chosen_cat)}</b>:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_edit_prod_select(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query

```

```

        await query.answer()
        cid = query.message.chat.id
        prod_name = query.data.split(":", 1)[1]
        prev = config.user_states.get(cid)
        if not prev or prev[0] != "editing_product":
            await send_admin_panel(context.bot, cid)
            return

        _, gender, chosen_cat = prev
        config.user_states[cid] = ("editing_fields", gender,
        chosen_cat, prod_name)
        try:
            await context.bot.delete_message(chat_id=cid,
            message_id=query.message.message_id)
        except:
            pass

        kb = [
            [InlineKeyboardButton("Изменить цену",
            callback_data=f"admin_edit_field:price:{prod_name}")],
            [InlineKeyboardButton("Изменить картинку",
            callback_data=f"admin_edit_field:image:{prod_name}")],
            [InlineKeyboardButton("⬅️ Назад",
            callback_data="admin_panel")],
        ]
        await send_single_menu(
            context.bot,
            cid,
            text=f"Что вы хотите изменить у товара
            «<b>{html_escape(prod_name)}</b>»? ",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_edit_field(update: Update, context:
    ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        prev = config.user_states.get(cid)
        if not prev or prev[0] != "editing_fields":
            await send_admin_panel(context.bot, cid)
            return

        _, gender, chosen_cat, prod_name = prev
        _, field, _ = query.data.split(":", 2)

        config.user_states[cid] = ("editing_value", gender,
        chosen_cat, prod_name, field)
        try:
            await context.bot.delete_message(chat_id=cid,
            message_id=query.message.message_id)
        except:
            pass

```

```

        if field == "price":
            await context.bot.send_message(
                chat_id=cid,
                text=f"Введите <b>новую цену</b> для
«{html_escape(prod_name)}» (только цифры):",
                parse_mode=ParseMode.HTML,
            )
        else:
            await send_single_menu(
                context.bot,
                cid,
                text="Пришлите <b>новую картинку</b> для
товара:",
                parse_mode=ParseMode.HTML,
            )

    async def admin_edit_value(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        state = config.user_states.get(cid)
        logger.info(f"admin_edit_value STATE: {state}")
        if not state or state[0] != "editing_value":
            return
        _, gender, chosen_cat, prod_name, field = state
        if field == "price":
            text = update.message.text.strip()
            if not text.isdigit():
                await update.message.reply_text("✗ Цена должна
быть числом. Попробуйте ещё раз.")
                return
            new_price = int(text)
            try:
                update_product_price(prod_name, gender,
chosen_cat, new_price)
            except Exception as e:
                logger.error(f"Error updating price: {e}")
                await update.message.reply_text("✗ Не удалось
изменить цену.")
            config.user_states[cid] = None
            return
            await update.message.reply_text(
                f"✓ Новая цена для
«<b>{html_escape(prod_name)}</b>» = {new_price}Р.",
                parse_mode=ParseMode.HTML,
            )
        else:
            if not update.message.photo:
                await update.message.reply_text("✗ Это не
фото. Пожалуйста, отправьте изображение.")
                return
            file_id = update.message.photo[-1].file_id
            try:
                update_product_image(prod_name, gender,
chosen_cat, file_id)

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

82

```

except Exception as e:
    logger.error(f"Error updating image: {e}")
    await update.message.reply_text("❌ Не удалось
сохранить новую картинку.")
    config.user_states[cid] = None
    return
    await update.message.reply_text(
        f"✅ Новая картинка для
«<b>{html_escape(prod_name)}</b>» сохранена.",
        parse_mode=ParseMode.HTML,
    )
    config.user_states[cid] = None
    await send_admin_panel(context.bot, cid)
    async def admin_orders_menu(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid_admin = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid_admin,
message_id=query.message.message_id)
        except:
            pass

        kb = [
            [InlineKeyboardButton("🆕 Новые заказы",
callback_data="admin_new_orders")],
            [InlineKeyboardButton("✅ Подтверждённые",
callback_data="admin_confirmed_orders")],
            [InlineKeyboardButton("❌ Отменённые",
callback_data="admin_canceled_orders")],
            [InlineKeyboardButton("🔍 Поиск по пользователю",
callback_data="admin_search_by_user")],
            [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_panel")],
        ]
        await context.bot.send_message(
            chat_id=cid_admin,
            text="📁 <b>Раздел «Заказы»</b>:",
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup(kb),
        )

    async def admin_new_orders(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid_admin = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid_admin,
message_id=query.message.message_id)
        except:

```

```

pass

rows = get_orders_by_status("new", limit=10)
if not rows:
    await context.bot.send_message(
        chat_id=cid_admin,
        text="📭 Новых заказов нет.",
        reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")]
        ])
    )
    return

text_lines = []
keyboard = []
for row in rows:
    order_id      = row["id"]
    chat_id_user  = row["chat_id"]
    username      = row["username"]
    total         = row["discounted_price"] or 0

    if username:
        who = "@" + username
    else:
        try:
            user_obj = await
context.bot.get_chat(chat_id_user)
            who = "@" + user_obj.username if
user_obj.username else f"<code>{chat_id_user}</code>"
        except:
            who = f"<code>{chat_id_user}</code>"

    text_lines.append(f"💎 Заказ ID {order_id} от {who}
- Сумма: <b>{total}₽</b>")
    keyboard.append([
        InlineKeyboardButton("✅ Подтвердить",
callback_data=f"admin_confirm_order:{order_id}"),
        InlineKeyboardButton("❌ Отменить",
callback_data=f"admin_cancel_order:{order_id}")
    ])

    keyboard.append([InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")])

    await context.bot.send_message(
        chat_id=cid_admin,
        text="\n".join(text_lines),
        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(keyboard)
    )

import json
from config import get_db_connection

```

```

    async def admin_confirm_order(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        cid = query.message.chat.id
        await query.answer()
        order_id = int(query.data.split(":")[1])

        update_order_status(order_id, "confirmed")

        # Получаем состав заказа
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute("SELECT items FROM orders WHERE id = ?",
(order_id,))
        row = cur.fetchone()
        conn.close()

        order_items = json.loads(row["items"]) if row else []
        order_text = "\n".join(
            f"• {item['name']} ({item['size']}) -
{item['price']}₽"
            for item in order_items
        )

        await context.bot.edit_message_text(
            chat_id=cid,
            message_id=query.message.message_id,
            text=(
                f"✅ Заказ #{order_id} подтверждён.\n\n"
                f"<b>Состав:</b>\n{order_text}"
            ),
            reply_markup=InlineKeyboardMarkup([
                [InlineKeyboardButton("⬅️ BACK Назад",
callback_data="admin_orders")]
            ]),
            parse_mode=ParseMode.HTML
        )

```

```

    async def admin_cancel_order(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        cid = query.message.chat.id
        await query.answer()
        order_id = int(query.data.split(":")[1])

        update_order_status(order_id, "canceled")

        # Получаем состав заказа
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute("SELECT items FROM orders WHERE id = ?",
(order_id,))
        row = cur.fetchone()
        conn.close()

```

```

        order_items = json.loads(row["items"]) if row else []
        order_text = "\n".join(
            f"• {item['name']} ({item['size']}) -
{item['price']}₽"
            for item in order_items
        )

        await context.bot.edit_message_text(
            chat_id=cid,
            message_id=query.message.message_id,
            text=(
                f"✕ Заказ #{order_id} отменён.\n\n"
                f"<b>Состав:</b>\n{order_text}"
            ),
            reply_markup=InlineKeyboardMarkup([
                [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")]
            ]),
            parse_mode=ParseMode.HTML
        )

    async def admin_confirmed_orders(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid_admin = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid_admin,
message_id=query.message.message_id)
        except:
            pass

        rows = get_orders_by_status("confirmed", limit=10)
        if not rows:
            await context.bot.send_message(
                chat_id=cid_admin,
                text="🚫 Подтверждённых заказов нет.",
                reply_markup=InlineKeyboardMarkup([
                    [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")]
                ])
            )
            return

        text_lines = []
        for row in rows:
            order_id      = row["id"]
            chat_id_user  = row["chat_id"]
            username       = row["username"]
            total          = row["discounted_price"] or 0

            if username:
                who = "@" + username
            else:

```

```

        try:
            user_obj = await
context.bot.get_chat(chat_id_user)
            who = "@" + user_obj.username if
user_obj.username else f"<code>{chat_id_user}</code>"
        except:
            who = f"<code>{chat_id_user}</code>"

        text_lines.append(f"💎 ID {order_id} от {who} -
Сумма: <b>{total}₽</b>")

        await context.bot.send_message(
            chat_id=cid_admin,
            text="\n".join(text_lines),
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup([
                [InlineKeyboardButton("⬅️ BACK Назад",
callback_data="admin_orders")]
            ])
        )

    async def admin_canceled_orders(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid_admin = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid_admin,
message_id=query.message.message_id)
        except:
            pass

        rows = get_orders_by_status("canceled", limit=10)
        if not rows:
            await context.bot.send_message(
                chat_id=cid_admin,
                text="🚫 Отменённых заказов нет.",
                reply_markup=InlineKeyboardMarkup([
                    [InlineKeyboardButton("⬅️ BACK Назад",
callback_data="admin_orders")]
                ])
            )
            return

        text_lines = []
        for row in rows:
            order_id = row["id"]
            chat_id_user = row["chat_id"]
            username = row["username"]
            total = row["discounted_price"] or 0

            if username:
                who = "@" + username
            else:

```

```

        try:
            user_obj = await
context.bot.get_chat(chat_id_user)
            who = "@" + user_obj.username if
user_obj.username else f"<code>{chat_id_user}</code>"
        except:
            who = f"<code>{chat_id_user}</code>"

        text_lines.append(f"💎 ID {order_id} от {who} -
Сумма: <b>{total}₽</b>")

        await context.bot.send_message(
            chat_id=cid_admin,
            text="\n".join(text_lines),
            parse_mode=ParseMode.HTML,
            reply_markup=InlineKeyboardMarkup([
                [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")]
            ])
        )

    async def admin_search_by_user(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        config.user_states[cid] = "searching_user_orders"
        await send_single_menu(
            context.bot,
            cid,
            text=(
                "Введите <b>chat_id</b> пользователя или
<b>@username</b> для поиска его заказов.\n"
                "Пример: <code>123456789</code> или
<code>@username</code>"
            ),
            parse_mode=ParseMode.HTML,
        )

    async def handle_search_by_user(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        state = config.user_states.get(cid)
        logger.info(f"handle_search_by_user STATE: {state}")
        TEXT: {update.message.text})
        if state != "searching_user_orders":
            return

```

```

text = update.message.text.strip()
conn = get_db_connection()
cur = conn.cursor()

if text.startswith("@"):
    username = text[1:].strip()
    cur.execute(
        "SELECT id, chat_id, username,
discounted_price, status "
        "FROM orders WHERE username = ? ORDER BY id
DESC LIMIT 10",
        (username,)
    )
    rows = cur.fetchall()
    if not rows:
        await update.message.reply_text(
            f"❌ Нет заказов для пользователя
@{username}.",
            reply_markup=InlineKeyboardMarkup([
                [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")]
            ])
        )
    else:
        msg_lines = [f"🔍 Заказы для @{username}:"]
        for row in rows:
            msg_lines.append(
                f"\n<b>Заказ #{row['id']}</b>\n"
                f"chat_id:
<code>{row['chat_id']}</code>\n"
                f"Статус: <i>{row['status']}</i>\n"
                f"Сумма: {row['discounted_price']}₽\n"
            )
        await
update.message.reply_text("\n".join(msg_lines),
parse_mode=ParseMode.HTML)
    else:
        if not text.isdigit():
            await update.message.reply_text("❌ Неверный
формат. Введите @username или chat_id (только цифры).")
            conn.close()
            return

        user_id = int(text)
        cur.execute(
            "SELECT id, chat_id, username,
discounted_price, status "
            "FROM orders WHERE chat_id = ? ORDER BY id DESC
LIMIT 10",
            (user_id,)
        )
        rows = cur.fetchall()
        if not rows:
            await update.message.reply_text(
                f"❌ Нет заказов для chat_id

```

```

{user_id}.",
        reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("⬅️ Назад",
callback_data="admin_orders")]
        ])
    )
    else:
        msg_lines = [f"🔍 Заказы для chat_id
<code>{user_id}</code>:"]
        for row in rows:
            uname = row["username"]
            uname_display = "None" if not uname else
"@ " + uname

            msg_lines.append(
                f"\n<b>Заказ #{row['id']}</b>\n"
                f"username: {uname_display}\n"
                f"Статус: <i>{row['status']}</i>\n"
                f"Сумма: {row['discounted_price']}₽\n"
            )
        await
update.message.reply_text("\n".join(msg_lines),
parse_mode=ParseMode.HTML)

    conn.close()
    config.user_states[cid] = None
    await admin_orders_menu(update, context)
def setup_admin_handlers(app):

    app.add_handler(CallbackQueryHandler(admin_panel,
pattern="^admin_panel$"))

app.add_handler(CallbackQueryHandler(admin_add_category,
pattern="^admin_add_category$"))

app.add_handler(CallbackQueryHandler(admin_add_cat_gender,
pattern="^admin_add_cat_gender:"))
    app.add_handler(MessageHandler(STATE_ADD_CATEGORY,
handle_new_category))

app.add_handler(CallbackQueryHandler(admin_set_cat_type,
pattern="^admin_set_cat_type:"))

app.add_handler(CallbackQueryHandler(admin_remove_category,
pattern="^admin_remove_category$"))

app.add_handler(CallbackQueryHandler(admin_rem_cat_gender,
pattern="^admin_rem_cat_gender:"))

app.add_handler(CallbackQueryHandler(admin_do_rem_category,
pattern="^admin_do_rem_category:"))

    app.add_handler(CallbackQueryHandler(admin_add_product,
pattern="^admin_add_product$"))

app.add_handler(CallbackQueryHandler(admin_add_prod_gender,

```

Изм.	Лист	№ докум.	Подпись	Дата

```

pattern="^admin_add_prod_gender:"))

app.add_handler(CallbackQueryHandler(admin_add_prod_cat,
pattern="^admin_add_prod_cat:"))
    app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & STATE_ADD_PROD_NAME, add_product_name))
    app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & STATE_ADD_PROD_PRICE, add_product_price))
    app.add_handler(MessageHandler(filters.PHOTO &
STATE_ADD_PROD_IMAGE, add_product_image))

app.add_handler(MessageHandler(filters.Regex(r'^/skip$') &
STATE_ADD_PROD_IMAGE, skip_product_image))
    app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & STATE_EDIT_VALUE, admin_edit_value))
    app.add_handler(MessageHandler(filters.PHOTO &
STATE_EDIT_VALUE, admin_edit_value))
    app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & STATE_SEARCH_USER, handle_search_by_user))

app.add_handler(CallbackQueryHandler(admin_remove_product,
pattern="^admin_remove_product$"))

app.add_handler(CallbackQueryHandler(admin_rem_prod_gender,
pattern="^admin_rem_prod_gender:"))

app.add_handler(CallbackQueryHandler(admin_rem_prod_cat,
pattern="^admin_rem_prod_cat:"))
    app.add_handler(CallbackQueryHandler(admin_do_rem_prod,
pattern="^admin_do_rem_prod:"))

app.add_handler(CallbackQueryHandler(admin_edit_product,
pattern="^admin_edit_product$"))

app.add_handler(CallbackQueryHandler(admin_edit_prod_gender,
pattern="^admin_edit_prod_gender:"))

app.add_handler(CallbackQueryHandler(admin_edit_prod_cat,
pattern="^admin_edit_prod_cat:"))

app.add_handler(CallbackQueryHandler(admin_edit_prod_select,
pattern="^admin_edit_prod_select:"))
    app.add_handler(CallbackQueryHandler(admin_edit_field,
pattern="^admin_edit_field:"))
    app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & STATE_EDIT_VALUE, admin_edit_value))
    app.add_handler(MessageHandler(filters.PHOTO &
STATE_EDIT_VALUE, admin_edit_value))

    app.add_handler(CallbackQueryHandler(admin_orders_menu,
pattern="^admin_orders$"))
    app.add_handler(CallbackQueryHandler(admin_new_orders,
pattern="^admin_new_orders$"))

app.add_handler(CallbackQueryHandler(admin_confirm_order,

```

Изм.	Лист	№ докум.	Подпись	Дата

ИГУ 1.090301.36.001 ПЗ

Лист

91

```

pattern="^admin_confirm_order:"))

app.add_handler(CallbackQueryHandler(admin_cancel_order,
pattern="^admin_cancel_order:"))

app.add_handler(CallbackQueryHandler(admin_confirmed_orders,
pattern="^admin_confirmed_orders$"))

app.add_handler(CallbackQueryHandler(admin_canceled_orders,
pattern="^admin_canceled_orders$"))

app.add_handler(CallbackQueryHandler(admin_search_by_user,
pattern="^admin_search_by_user$"))
    app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & STATE_SEARCH_USER, handle_search_by_user))

```

Файл handlers/cart.py

```

from telegram import InlineKeyboardButton,
InlineKeyboardMarkup, Update
from telegram.constants import ParseMode
from telegram.ext import CallbackQueryHandler,
ContextTypes, MessageHandler, filters
import config
from utils.menu import get_items_text, main_menu,
send_single_menu
from config import (
    PROMO_RATES, get_cart, get_price, calculate_delivery,
add_order, get_user_address,
    remove_from_cart, set_user_address
)

async def handle_promocode(update, context):
    cid = update.effective_chat.id
    state = config.user_states.get(cid)
    if state != "entering_promocode":
        return

    promocode = update.message.text.strip().upper()
    config.user_states[cid] = ("review_order", promocode)
    await update.message.delete()
    return await show_order_summary(update, context,
promocode)

async def skip_promocode(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id
    config.user_states[cid] = ("review_order", None)
    try:
        await context.bot.delete_message(cid,
query.message.message_id)
    except:

```

```

pass

return await show_order_summary(update, context, None)

async def show_order_summary(update, context, promocode):
    cid = update.effective_chat.id if update.message else
update.callback_query.message.chat.id
    cart_items = config.get_cart(cid)

    if not cart_items:
        await context.bot.send_message(cid, "✕ Корзина
пуста.")
        return

    total = sum(item["price"] for item in cart_items)
    promo_discount =
config.PROMO_RATES.get(promocode.upper(), 0) if promocode else 0
    discounted = int(total * (1 - promo_discount))

    delivery = 0
    if discounted < 10000:
        delivery = 1000
        if discounted >= 2000:
            delivery = int(delivery * 0.85)

    final = discounted + delivery

    lines = [f"📦 <b>Ваш заказ:</b>"]
    for i, item in enumerate(cart_items, 1):
        lines.append(f"{i}. {item['name']} - {item['size']}
- {item['price']}₽")

    lines.append(f"\n📑 Сумма: <b>{total}₽</b>")
    if promo_discount:
        lines.append(f"🏷️ Скидка по промокоду: <b>-
{int(promo_discount * 100)}%</b>")
        lines.append(f"Итого со скидкой:
<b>{discounted}₽</b>")
    lines.append(f"🚚 Доставка: <b>{delivery}₽</b>")
    lines.append(f"\n💰 <b>К оплате: {final}₽</b>")

    await context.bot.send_message(
        cid,
        "\n".join(lines),
        parse_mode="HTML",
        reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("✅ Подтвердить заказ",
callback_data="confirm_order")],
            [InlineKeyboardButton("⬅️ BACK Вернуться в корзину",
callback_data="menu_cart")]
        ])
    )

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

93

```

    async def confirm_order(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        try:
            await context.bot.delete_message(cid,
query.message.message_id)
        except:
            pass

        state = config.user_states.get(cid)

        if not state or state[0] != "review_order":
            return

        promocode = state[1]
        cart_items = config.get_cart(cid)
        total = sum(item["price"] for item in cart_items)
        discount = config.PROMO_RATES.get(promocode.upper(), 0)
        if promocode else 0
        discounted = int(total * (1 - discount))

        delivery = 0
        if discounted < 10000:
            delivery = 1000
            if discounted >= 2000:
                delivery = int(delivery * 0.85)

        username = update.effective_user.username
        config.add_order(cid, cart_items, promocode,
discounted_price=discounted, delivery=delivery,
username=username)
        config.clear_cart(cid)
        config.user_states[cid] = None

        await context.bot.send_message(
            chat_id=cid,
            text="✅ Ваш заказ принят! Мы свяжемся с вами.",
            reply_markup=main_menu(cid)
        )

    async def cart_checkout(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        address = config.get_user_address(cid)
        if address:
            config.user_states[cid] = "entering_promocode"
            await send_single_menu(
                context.bot,
                cid,
                text="Введите промокод (или нажмите

```

```

«Пропустить»):",
        parse_mode="HTML",
        reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("»» Пропустить",
callback_data="skip_promocode")],
            [InlineKeyboardButton("⬅️ Назад",
callback_data="menu_cart")]
        ])
    )
else:
    config.user_states[cid] = "entering_address"
    await send_single_menu(
        context.bot,
        cid,
        text="Пожалуйста, введите адрес доставки:",
        parse_mode="HTML",
        reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("⬅️ Назад",
callback_data="menu_cart")]
        ])
    )

async def handle_enter_address(update, context):
    cid = update.effective_chat.id
    state = config.user_states.get(cid)
    if state != "entering_address":
        return

    address = update.message.text.strip()
    if not address:
        await update.message.reply_text("❌ Адрес не может
быть пустым.")
        return

    config.set_user_address(cid, address)

    config.user_states[cid] = "entering_promocode"
    await update.message.reply_text(
        "Введите промокод (или нажмите «Пропустить»):",
        reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("»» Пропустить",
callback_data="skip_promocode")],
            [InlineKeyboardButton("⬅️ Назад",
callback_data="menu_cart")]
        ])
    )

async def menu_cart(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    cart_items = config.get_cart(cid)
    text = get_items_text(cart_items)

```

```

# >>> ДОБАВЛЯЕМ ПОЛУЧЕНИЕ АДРЕСА
address = config.get_user_address(cid)
if address:
    text += f"\n\n<b>Адрес доставки:</b> {address}"
else:
    text += "\n\n<i>Адрес не указан</i>"

kb = []
for idx, _ in enumerate(cart_items):
    kb.append([InlineKeyboardButton(f"✕ Удалить {idx+1}", callback_data=f"remove_cart:{idx}")])
    if cart_items:
        kb.append([InlineKeyboardButton("📝 Оформить заказ", callback_data='cart_checkout')])
        kb.append([InlineKeyboardButton("✎ Изменить адрес", callback_data='change_address')])
        kb.append([InlineKeyboardButton("⬅️ Назад", callback_data='back_to_main')])

    try:
        await context.bot.delete_message(chat_id=cid, message_id=query.message.message_id)
    except:
        pass

    await send_single_menu(
        context.bot,
        cid,
        text=f"🛒 <b>Ваша корзина:</b>\n{text}",
        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(kb)
    )

async def change_address(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id
    config.user_states[cid] = "entering_address"
    await send_single_menu(
        context.bot,
        cid,
        text="Введите новый адрес доставки:",
        parse_mode="HTML"
    )

async def remove_cart(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    idx = int(query.data.split(":", 1)[1])
    config.remove_from_cart(cid, idx)

    return await menu_cart(update, context)

def setup_cart_handlers(app):

```

```

        app.add_handler(CallbackQueryHandler(menu_cart,
pattern="^menu_cart$"))
        app.add_handler(CallbackQueryHandler(cart_checkout,
pattern="^cart_checkout$"))
        app.add_handler(CallbackQueryHandler(remove_cart,
pattern="^remove_cart:"))
        app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND, handle_enter_address))
        app.add_handler(CallbackQueryHandler(change_address,
pattern="^change_address$"))
        app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND, handle_enter_address))
        app.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND, handle_promocode))
        app.add_handler(CallbackQueryHandler(skip_promocode,
pattern="^skip_promocode$"))
        app.add_handler(CallbackQueryHandler(confirm_order,
pattern="^confirm_order$"))

```

Файл handlers/catalog.py

```

from telegram import InlineKeyboardButton,
InlineKeyboardMarkup, Update
from telegram.ext import CallbackQueryHandler,
MessageHandler, filters, ContextTypes
import config
from utils.menu import (
    root_gender_keyboard,
    category_keyboard,
    product_keyboard,
    size_keyboard,
    main_menu,
    send_single_menu,
    send_single_photo
)

async def open_catalog(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    await send_single_menu(
        context.bot,
        cid,
        text="Выберите <b>пол</b>:",
        parse_mode="HTML",
        reply_markup=root_gender_keyboard(),
    )

```

```

    async def choose_gender(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    _, gender = query.data.split(":", 1)
    config.user_states[cid] = ("browsing_gender", gender)

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    await send_single_menu(
        context.bot,
        cid,
        text=f"Категории для <b>{gender}</b>:",
        parse_mode="HTML",
        reply_markup=category_keyboard(gender),
    )

    async def select_category(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    _, gender, category = query.data.split(":", 2)
    config.user_states[cid] = ("browsing_category", gender,
category)

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    await send_single_menu(
        context.bot,
        cid,
        text=f"Товары в категории <b>{category}</b>
({gender}):",
        parse_mode="HTML",
        reply_markup=product_keyboard(gender, category),
    )

    async def select_product(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    _, gender, category, prod = query.data.split(":", 3)
    config.user_states[cid] = ("browsing_product", gender,

```

Изм.	Лист	№ докум.	Подпись	Дата

```

category, prod)

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    # Показываем картинку товара (если есть) и кнопку
выбора размеров
    # Попробуем получить file_id (telegram) или отправить
заглушку
    row = None
    conn = config.get_db_connection()
    cur = conn.cursor()
    cur.execute(
        "SELECT image, price FROM products WHERE name = ?
AND gender = ? AND category = ?",
        (prod, gender, category)
    )
    row = cur.fetchone()
    conn.close()

    text = f"«<b>{prod}</b>» - {row['price']}₽"
    kb = [
        [InlineKeyboardButton("Выбрать размер",
callback_data=f"choose_size:{gender}:{category}:{prod}")],
        [InlineKeyboardButton("☆ В избранное",
callback_data=f"fav_choose_size:{gender}:{category}:{prod}")],
        [InlineKeyboardButton("⬅️ Назад",
callback_data=f"select_category:{gender}:{category}")]]

    if row["image"]:
        await send_single_photo(
            context.bot,
            cid,
            photo=row["image"],
            caption=text,
            parse_mode="HTML",
            reply_markup=InlineKeyboardMarkup(kb)
        )
    else:
        await send_single_menu(
            context.bot,
            cid,
            text=text,
            parse_mode="HTML",
            reply_markup=InlineKeyboardMarkup(kb)
        )

    async def choose_size(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

```

```

_, gender, category, prod = query.data.split(":", 3)
config.user_states[cid] = ("choosing_size", gender,
category, prod)

try:
    await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
except:
    pass

kb = size_keyboard(gender, category, prod,
destination="cart")
if kb is None:
    # "Другое" - добавить товар без выбора размера
    from config import add_to_cart
    add_to_cart(cid, prod, size="-")
    await send_single_menu(
        context.bot,
        cid,
        text=f"Товар <b>{prod}</b> добавлен в корзину
(без размера)!",
        parse_mode="HTML",
        reply_markup=main_menu(cid)
    )
    return

    await send_single_menu(
        context.bot,
        cid,
        text="Выберите размер:",
        parse_mode="HTML",
        reply_markup=kb,
    )

async def fav_choose_size(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    _, gender, category, prod = query.data.split(":", 3)
    cid = query.message.chat.id

    kb = size_keyboard(gender, category, prod,
destination="fav")
    if kb is None:
        from config import add_to_favorites
        add_to_favorites(cid, prod, size="-")
        await context.bot.send_message(
            chat_id=cid,
            text=f"Товар <b>{prod}</b> добавлен в избранное
(без размера)!",
            parse_mode="HTML",
            reply_markup=main_menu(cid)
        )
    return

```

```

        await send_single_menu(
            context.bot,
            cid,
            text="Выберите размер для избранного:",
            parse_mode="HTML",
            reply_markup=kb,
        )

    async def size_cart(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        parts = query.data.split(":", 5)
        _, destination, gender, category, prod, sz = parts
        config.user_states[cid] = None
        config.add_to_cart(cid, prod, sz)

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        await send_single_menu(
            context.bot,
            cid,
            text=f"✅ «<b>{prod}</b>» ({sz}) добавлено в
корзину.",
            parse_mode="HTML",
            reply_markup=main_menu(cid),
        )

    async def size_fav(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        # size:fav:gender:category:prod:size
        _, destination, gender, category, prod, sz =
query.data.split(":", 5)
        cid = query.message.chat.id
        from config import add_to_favorites
        add_to_favorites(cid, prod, sz)
        await send_single_menu(
            context.bot,
            cid,
            text=f"Товар <b>{prod}</b> ({sz}) добавлен в
избранное!",
            parse_mode="HTML",
            reply_markup=main_menu(cid)
        )

def setup_catalog_handlers(app):
    app.add_handler(CallbackQueryHandler(open_catalog,

```

```

pattern="^menu_catalog$"))
    app.add_handler(CallbackQueryHandler(choose_gender,
pattern="^choose_gender:"))
    app.add_handler(CallbackQueryHandler(select_category,
pattern="^select_category:"))
    app.add_handler(CallbackQueryHandler(select_product,
pattern="^select_product:"))
    app.add_handler(CallbackQueryHandler(choose_size,
pattern="^choose_size:"))
    app.add_handler(CallbackQueryHandler(fav_choose_size,
pattern="^fav_choose_size:"))
    app.add_handler(CallbackQueryHandler(size_fav,
pattern="^size:fav:"))
    app.add_handler(CallbackQueryHandler(size_cart,
pattern="^size:"))

```

Файл handlers/common.py

```

import config
from telegram import InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.ext import CallbackQueryHandler
from utils.menu import main_menu

# Хендлер « Назад» (возвращает в главное меню)
async def back_to_main(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    await context.bot.send_message(
        chat_id=cid,
        text="Главное меню:",
        reply_markup=main_menu(cid)
    )

def setup_common_handlers(app):
    app.add_handler(CallbackQueryHandler(back_to_main,
pattern="^back_to_main$"))

```

Файл handlers/favorites.py

```

from telegram import InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.constants import ParseMode
from telegram.ext import CallbackQueryHandler
import config
from utils.menu import get_items_text, main_menu,

```

```

send_single_menu

async def menu_fav(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    fav_items = config.get_favorites(cid)
    text = get_items_text(fav_items)

    kb = []
    for idx, item in enumerate(fav_items):
        kb.append([
            InlineKeyboardButton(f"🛒 В корзину",
callback_data=f"fav_to_cart:{idx}"),
            InlineKeyboardButton(f"✕ Удалить",
callback_data=f"remove_fav:{idx}")
        ])
        kb.append([InlineKeyboardButton("⬅️ Назад",
callback_data='back_to_main')])

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    await send_single_menu(
        context.bot,
        cid,
        text=f"❤️ <b>Ваше избранное:</b>\n{text}" if
fav_items else "У вас пока нет избранных товаров.",
        parse_mode=ParseMode.HTML,
        reply_markup=InlineKeyboardMarkup(kb)
    )

async def remove_fav(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    idx = int(query.data.split(":", 1)[1])
    config.remove_from_favorites(cid, idx)

    return await menu_fav(update, context)

async def fav_to_cart(update, context):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id
    idx = int(query.data.split(":", 1)[1])
    favs = config.get_favorites(cid)
    if not (0 <= idx < len(favs)):
        await context.bot.send_message(chat_id=cid,
text="✕ Товар не найден.")

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        return
    item = favs[idx]
    config.add_to_cart(cid, item['name'], item['size'])
    config.remove_from_favorites(cid, idx)
    await send_single_menu(
        context.bot,
        cid,
        text=f"Товар <b>{item['name']}</b> добавлен в
корзину!",
        parse_mode=ParseMode.HTML
    )
    await menu_fav(update, context)

def setup_fav_handlers(app):
    app.add_handler(CallbackQueryHandler(menu_fav,
pattern="^menu_fav$"))
    app.add_handler(CallbackQueryHandler(remove_fav,
pattern="^remove_fav:"))
    app.add_handler(CallbackQueryHandler(fav_to_cart,
pattern="^fav_to_cart:"))

```

Файл handlers/orders.py

```

import json
import logging
import config
from telegram import InlineKeyboardButton,
InlineKeyboardMarkup, Update
from telegram.constants import ParseMode
from telegram.ext import CallbackQueryHandler,
MessageHandler, filters, ContextTypes
from config import get_orders # убедись, что функция
возвращает заказы пользователя

from db import (
    add_order,
    get_orders_by_status,
    update_order_status,
    get_orders_by_user_id,
)
from utils.menu import main_menu

logger = logging.getLogger(__name__)

async def menu_orders(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid = query.message.chat.id

    try:
        await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
    except:
        pass

    orders = get_orders(cid)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

if not orders:
    await context.bot.send_message(
        chat_id=cid,
        text="У вас пока нет заказов.",
        reply_markup=main_menu(cid)
    )
    return

for row in orders:
    order_id = row["id"]
    status = row["status"]
    total = row["discounted_price"] or 0
    delivery = row["delivery"] or 0
    items = json.loads(row["items"])

    lines = [f"<b>📦 Заказ #{order_id}</b>", f"Статус:
{status}"]
    for i, item in enumerate(items, 1):
        lines.append(f"{i}. {item['name']} -
{item['size']} - {item['price']}₽")
    lines.append(f"\n📑 Сумма: {total}₽")
    lines.append(f"🚚 Доставка: {delivery}₽")
    lines.append(f"<b>Итого: {total + delivery}₽</b>")

    await context.bot.send_message(
        chat_id=cid,
        text="\n".join(lines),
        parse_mode=ParseMode.HTML
    )

    await context.bot.send_message(
        chat_id=cid,
        text="Для уточнения статуса заказа обращайтесь в
поддержку.",
        reply_markup=main_menu(cid),
        parse_mode=ParseMode.HTML
    )

async def handle_confirm_from_admin(update: Update,
context: ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()
    cid_admin = query.message.chat.id

    try:
        order_id = int(query.data.split(":", 1)[1])
    except ValueError:
        await context.bot.send_message(chat_id=cid_admin,
text="❌ Неверный формат ID заказа.")
        return

    update_order_status(order_id, "confirmed")
    await context.bot.send_message(
        chat_id=cid_admin,
        text=f"✅ Заказ ID {order_id} подтверждён.",
        reply_markup=main_menu(cid_admin)

```

```

    )

#
    async def handle_cancel_from_admin(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid_admin = query.message.chat.id

        try:
            order_id = int(query.data.split(":", 1)[1])
        except ValueError:
            await context.bot.send_message(chat_id=cid_admin,
text="✗ Неверный формат ID заказа.")
            return

        update_order_status(order_id, "canceled")
        await context.bot.send_message(
            chat_id=cid_admin,
            text=f"✗ Заказ ID {order_id} отменён.",
            reply_markup=main_menu(cid_admin)
        )

    async def admin_manual_cancel(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid,
message_id=query.message.message_id)
        except:
            pass

        config.user_states[cid] = "awaiting_cancel_order"
        await context.bot.send_message(
            chat_id=cid,
            text=(
                "Введите <b>chat_id</b> пользователя и <b>номер</b>
заказа</b> через пробел,\n"
                "например: <code>123456789 2</code> (в нашей
базе сортировка DESC).",
            ),
            parse_mode=ParseMode.HTML,
        )

    async def handle_cancel_order_input(update: Update,
context: ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        if config.user_states.get(cid) !=
"awaiting_cancel_order":
            return

        parts = update.message.text.strip().split()
        if len(parts) != 2 or not all(p.isdigit() for p in

```

```

parts):
    await update.message.reply_text(
        "✗ Неверный формат. Должно быть:
<code>123456789 2</code>.",
        parse_mode=ParseMode.HTML,
    )
    return

    user_id, idx = map(int, parts)
    idx -= 1 # пользователь вводит «1-based», а у нас «0-
based»

    rows = get_orders_by_user_id(user_id)
    if not rows or idx < 0 or idx >= len(rows):
        await update.message.reply_text("✗ Заказ не
найден.", parse_mode=ParseMode.HTML)
        return

    order_db_id = rows[idx]["id"]
    try:
        update_order_status(order_db_id, "canceled")
    except Exception as e:
        logger.error(f"Error cancelling order: {e}")
        await update.message.reply_text("✗ Не удалось
отменить заказ.", parse_mode=ParseMode.HTML)
        config.user_states[cid] = None
        # После этого можно вернуть меню заказов
        return

    await update.message.reply_text(
        f"✅ Заказ #{idx+1} пользователя
<code>{user_id}</code> отменён.",
        parse_mode=ParseMode.HTML
    )
    try:
        await context.bot.send_message(
            chat_id=user_id,
            text="⚠️ <b>Ваш заказ отменён
администратором.</b>",
            parse_mode=ParseMode.HTML,
            reply_markup=main_menu(user_id),
        )
    except:
        pass

    config.user_states[cid] = None

    async def admin_search_by_user(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        query = update.callback_query
        await query.answer()
        cid = query.message.chat.id

        try:
            await context.bot.delete_message(chat_id=cid,

```

```

message_id=query.message.message_id)
    except:
        pass

    # Устанавливаем state и просим админа ввести chat_id
или @username
    config.user_states[cid] = "searching_user_orders"
    await context.bot.send_message(
        chat_id=cid,
        text=(
            "Введите <b>chat_id</b> пользователя или
<b>@username</b> для поиска его заказов.\n"
            "Пример: <code>123456789</code> или
<code>@username</code>"
        ),
        parse_mode=ParseMode.HTML,
    )

    async def handle_search_by_user(update: Update, context:
ContextTypes.DEFAULT_TYPE):
        cid = update.effective_chat.id
        state = config.user_states.get(cid)
        if state != "searching_user_orders":
            return

        text = update.message.text.strip()
        # Немного логируем входящую команду, чтобы в логах было
ВИДНО
        logger.info(f"AdminSearchByUser: got '{text}' from
admin {cid}")

        conn = config.get_db_connection()
        cur = conn.cursor()

        if text.startswith("@"):
            username = text[1:].strip()
            cur.execute(
                "SELECT id, chat_id, username,
discounted_price, status "
                "FROM orders WHERE username = ? ORDER BY id
DESC LIMIT 10",
                (username,)
            )
            rows = cur.fetchall()
            if not rows:
                await update.message.reply_text(f"❏ Нет
заказов для пользователя @{username}.")
            else:
                msg_lines = [f"🔍 Заказы для @{username}:"]
                for row in rows:
                    msg_lines.append(
                        f"\n<b>Заказ #{row['id']}</b>\n"
                        f"chat_id:
<code>{row['chat_id']}</code>\n"
                        f"Статус: <i>{row['status']}</i>\n"
                        f"Сумма: {row['discounted_price']}₽\n"
                    )

```

```

        await
update.message.reply_text("\n".join(msg_lines),
parse_mode=ParseMode.HTML)

    else:
        if not text.isdigit():
            await update.message.reply_text("✗ Неверный
формат. Введите @username или chat_id (только цифры).")
            conn.close()
            return
        user_id = int(text)
        cur.execute(
            "SELECT id, chat_id, username,
discounted_price, status "
            "FROM orders WHERE chat_id = ? ORDER BY id DESC
LIMIT 10",
            (user_id,)
        )
        rows = cur.fetchall()
        if not rows:
            await update.message.reply_text(f"❏ Нет
заказов для chat_id {user_id}.")
        else:
            msg_lines = [f"🔍 Заказы для chat_id
<code>{user_id}</code>:" ]
            for row in rows:
                uname = row["username"]
                uname_display = "None" if not uname else
"@ " + uname

                msg_lines.append(
                    f"\n<b>Заказ #{row['id']}</b>\n"
                    f"username: {uname_display}\n"
                    f"Статус: <i>{row['status']}</i>\n"
                    f"Сумма: {row['discounted_price']}₽\n"
                )
            await
update.message.reply_text("\n".join(msg_lines),
parse_mode=ParseMode.HTML)

        conn.close()

        config.user_states[cid] = None
        # После вывода результата вернёмся в подменю «Заказы»
        await menu_orders(update, context)
    def setup_order_handlers(app):
        app.add_handler(CallbackQueryHandler(menu_orders,
pattern="^admin_orders$"))
        app.add_handler(CallbackQueryHandler(menu_orders,
pattern="^menu_orders$"))

    app.add_handler(CallbackQueryHandler(handle_confirm_from_admin,
pattern="^admin_confirm_order:"))

    app.add_handler(CallbackQueryHandler(handle_cancel_from_admin,
pattern="^admin_cancel_order:"))

```

```

app.add_handler(CallbackQueryHandler(admin_manual_cancel,
pattern="^admin_manual_cancel$"))
    app.add_handler(MessageHandler(
        filters.TEXT
        & ~filters.COMMAND
        & filters.MessageFilter(lambda m:
config.user_states.get(m.chat.id) == "awaiting_cancel_order"),
        handle_cancel_order_input
    ))

app.add_handler(CallbackQueryHandler(admin_search_by_user,
pattern="^admin_search_by_user$"))
    app.add_handler(MessageHandler(
        filters.TEXT
        & ~filters.COMMAND
        & filters.MessageFilter(lambda m:
config.user_states.get(m.chat.id) == "searching_user_orders"),
        handle_search_by_user
    ))

```

Приложение В

(обязательное)

Чат-бот для электронной коммерции

Презентация

					<i>ПГУ 1.090301.36.001 ПЗ</i>	<i>Лист</i>
						<i>111</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Тема работы: Чат-бот для электронной коммерции

Автор работы:

студент гр.21ВВП2 Миликов А.Р.

Научный руководитель:

к.э.н., доцент, каф. ВТ ПГУ
Слепцов Н.В.

Пенза 2025

1

Актуальность разработки

В современном мире развитие электронной коммерции идёт стремительными темпами. Всё больше пользователей отдают предпочтение онлайн-покупкам, особенно в удобном и простом формате - без необходимости устанавливать отдельные приложения или искать сайты в поисковиках. Telegram, как один из самых популярных мессенджеров в мире и особенно в странах СНГ, становится новой площадкой для взаимодействия между продавцами и покупателями.



2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

112

Цель работы

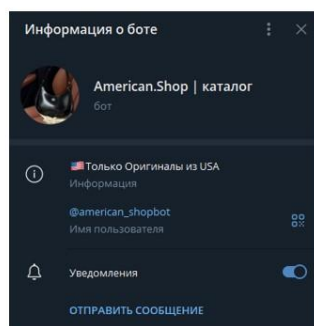
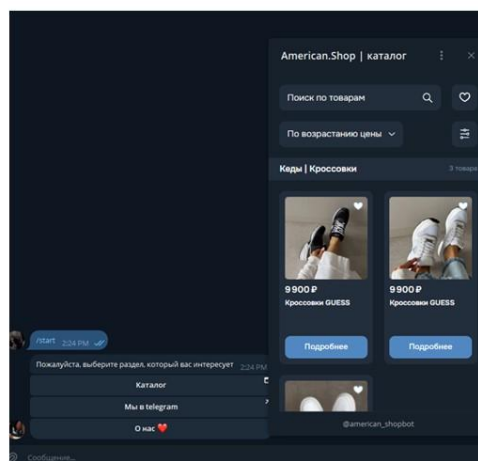
Цель работы - разработка полнофункционального Telegram-бота для электронной коммерции, предоставляющего пользователям возможность просматривать каталог товаров, добавлять товары в корзину и избранное, оформлять заказы с применением промокодов, а также администраторам — управлять категориями и товарами, подтверждать заказы и получать уведомления о действиях пользователей

Объектами разработки являются:

- Telegram-бот с обработчиками команд и кнопок, реализованными на Python (python-telegram-bot);
- база данных SQLite, содержащая информацию о пользователях, товарах, заказах, категориях, избранном и корзине;
- каталог товаров с фильтрацией по категориям и отображением подробной информации (цена, размер, фото);
- корзина с выбором размера, оформление заказа с расчётом стоимости доставки и учётом скидок;
- панель администратора с возможностью добавления, редактирования и удаления товаров и категорий;

3

Аналоги



4

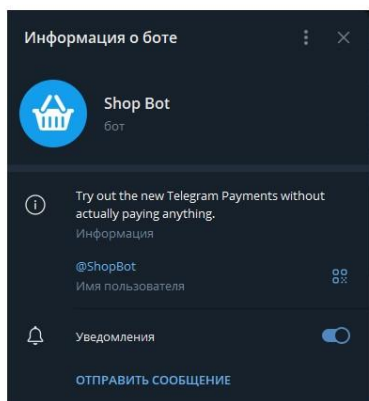
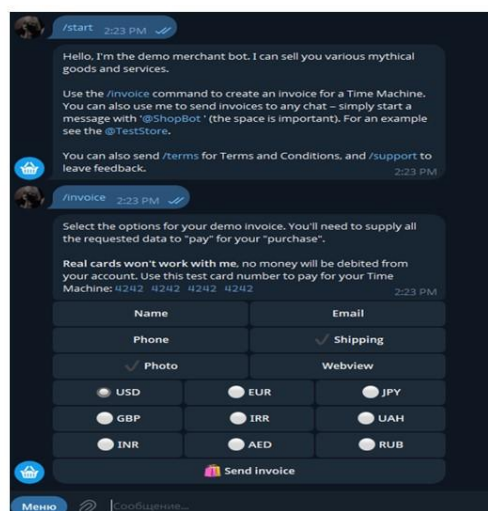
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

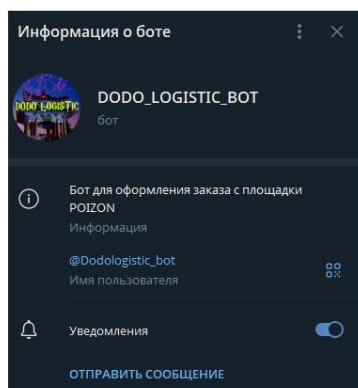
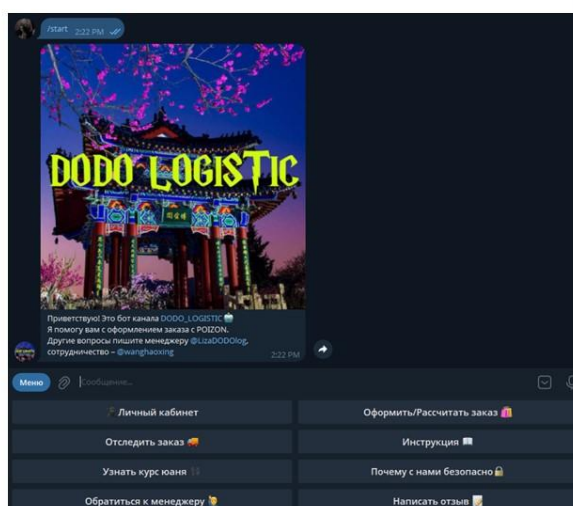
113

Аналоги



5

Аналоги



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

114

Стек технологий

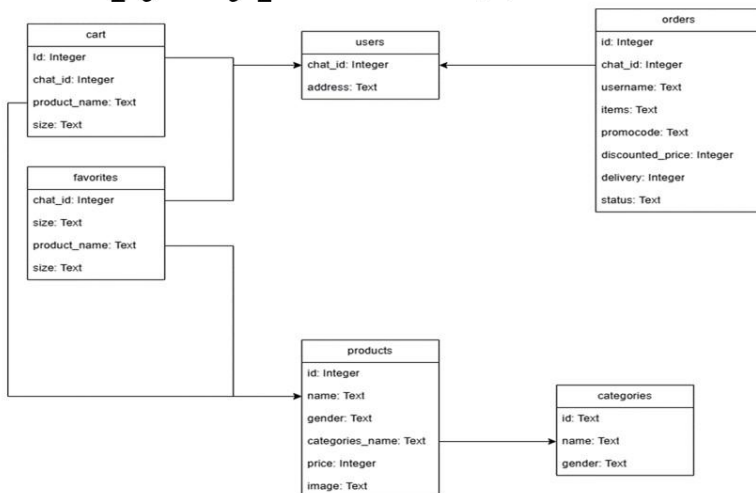
omnilib/**aiosqlite**

asyncio bridge to the standard sqlite3 module



7

Схема структуры базы данных



8

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

115

Диаграмма вариантов использования



9

Диаграмма вариантов использования



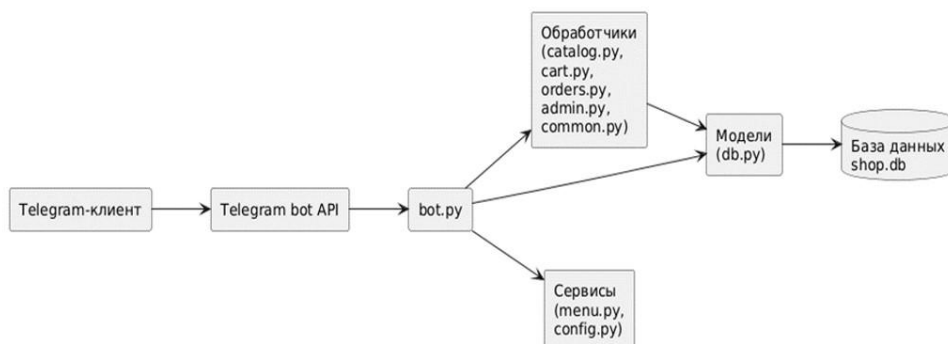
10

Изм.	Лист	№ докум.	Подпись	Дата

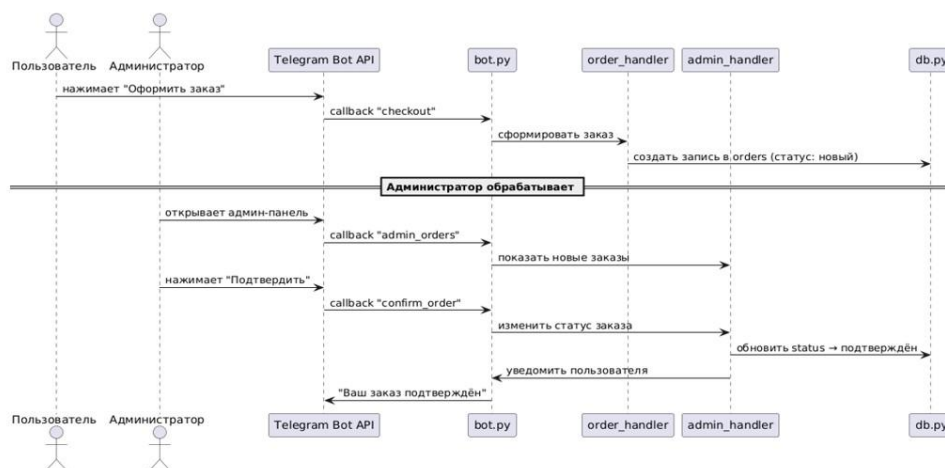
ПГУ 1.090301.36.001 ПЗ

Лист

116



11



2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

117

Заключение

В рамках выполнения данной выпускной квалификационной работы была разработана и протестирована система в виде Telegram-бота, предназначенного для реализации сценариев электронной коммерции

Разработанное клиент-серверное приложение, реализованное в виде Telegram-бота, позволяет:

- удобно взаимодействовать с системой без установки дополнительных программ или регистрации
- просматривать каталог товаров;
- оформлять заказы через корзину;
- получать уведомление о заказах;
- администраторам управлять пользователями и модерировать заказы.

Возможно усовершенствовать систему, добавив:

- интеграцию с платежными системами;
- расширенную статистику заказов;
- улучшенную визуальную часть;
- автоматическую систему уведомлений и рассылок;
- возможность пользователям размещать свои вещи на продажу.

13

Спасибо за внимание!

14

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

118