

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

**БИЗНЕС-ПРОЕКТ «РАЗРАБОТКА ИНТЕРНЕТ-
МАГАЗИНА НА ОСНОВЕ TELEGRAM-БОТА»**

Студент Евд 28.05.2026 Евдокимов Р.Е.
(подпись, дата) (ФИО полностью)
Руководитель Юр 29.05.2026 Юрова О.В.
(подпись, дата) (фамилия, инициалы)
Нормоконтролёр Быч 29.05.2026 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 22.05.2026 №14)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой

отметка

(протокол заседания ГЭК от 11.06.2026 № 27)

Секретарь ГЭК

Е.И. Гурин

Е.И. Гурин

Пенза 2026

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»



«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«07» 04 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Евдокимов Руслан Евгеньевич гр.22ВВП1 факультета вычислительной техники.

2. Тема работы: Бизнес-проект «Разработка интернет-магазина на основе Telegram-бота»

Тема утверждена приказом ПГУ № 347/о от 07.04.2026 г.

3. Руководитель бакалаврской работы к.т.н. доцент Юрова Ольга Викторовна

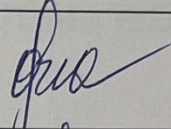
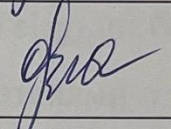
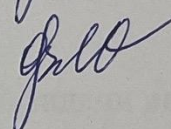
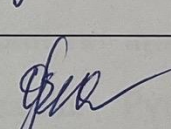
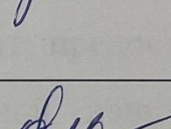
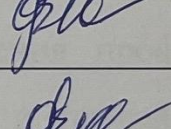
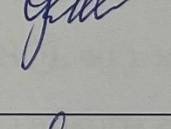
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программный комплекс интернет-магазина на базе Telegram-бота на выбранном языке программирования. Приложение должно обладать модульной асинхронной архитектурой, интуитивно понятным интерфейсом и в достаточной для практической эксплуатации степени обеспечивать комплексную автоматизацию торговых процессов, включая ведение каталога, обработку заказов и проведение платежей

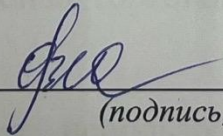
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: проанализировать актуальность разработки, изучить существующие аналоги интернет-магазинов в мессенджерах, выделить их достоинства и недостатки. Выбрать язык программирования, асинхронные библиотеки и систему управления базами данных для реализации проекта. Выявить необходимые сценарии взаимодействия пользователей и администраторов с приложением и реализовать их на базе конечных автоматов. Описать архитектуру проекта, настроить инструменты и среду разработки, спроектировать схему БД, разработать программу, описать реализованные функции и алгоритмы, проанализировать результаты функционального тестирования, составить рекомендации к дальнейшей разработке.

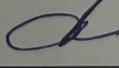
6.Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	40	07.04.2026 – 11.04.2026	
2	Анализ требований	32	12.04.2026 – 15.04.2026	
3	Формализация технических требований и проектирование структуры базы данных	64	16.04.2026 – 23.04.2026	
4	Проектирование и программная реализация пользовательского интерфейса	88	24.04.2026 – 04.05.2026	
5	Разработка административной панели и подсистемы управления заказами	96	05.05.2026 – 16.05.2026	
6	Функциональное тестирование программного продукта и устранение дефектов	16	17.05.2026 – 18.05.2026	
7	Оформление пояснительной записки	24	19.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026г.

Руководитель бакалаврской работы  Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент  Евдокимов Р.Е.
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 144 лист, 59 рисунков, 3 таблицы, 7 источников, 3 приложения.

TELEGRAM-БОТ, ИНТЕРНЕТ-МАГАЗИН, АСИНХРОННАЯ
АРХИТЕКТУРА, PYTHON, AIOGRAM, SQLITE, АВТОМАТИЗАЦИЯ
ТОРГОВЛИ, ОНЛАЙН-ОПЛАТА

Цель работы – разработать интернет-магазин на основе Telegram-бота для автоматизации процесса продаж и взаимодействия с пользователями в рамках мессенджера.

Наиболее эффективным путем достижения поставленной задачи является создание асинхронного приложения с модульной архитектурой по принципу MVC. Программа разработана на языке программирования Python с использованием фреймворка aiogram и СУБД SQLite. Бот предоставляет возможности просмотра каталога товаров с постраничной навигацией, динамического управления корзиной, пошагового заполнения профиля и проведения онлайн-оплаты через встроенный платежный интерфейс. Все данные о пользователях, товарах и заказах защищены от SQL-инъекций и обрабатываются через асинхронную библиотеку aioredis. Для гибкого ведения бизнеса и изменения статусов заказов в систему интегрирована административная панель. В итоге было разработано приложение, которое полностью реализует функционал торговой платформы внутри мессенджера Telegram.

В ходе проектирования использовались технологии асинхронного программирования, машины состояний и промежуточного ПО. Реализованный функционал успешно протестирован и полностью готов к эксплуатации.

					ПГУ1.090301.09.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Бизнес-проект «Разработка интернет-магазина на основе Telegram-бота» Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Евдокимов Р.Е.		28.05.2026					
Провер.		Юрова О.В.		28.05.2026			4	144	
Н. Контр.		Бычков А. С.		28.05.2026		ФВТ, каф. ВТ гр. 22ВВП1			
Утверд.		Митрохин М.А.		28.05.2026					

Содержание

Перечень сокращений и обозначений	6
Введение	7
1 Анализ предметной области	9
1.1 Анализ решения поставленной задачи	10
1.2 Обзор аналогов	11
1.3 Постановка задачи проекта	15
1.4 Минимальные системные требования	16
1.4.1 Требования для пользователя	16
1.4.2 Требования для сервера	17
1.5 Выбор инструментов разработки	17
1.5.1 Выбор языка программирования	17
1.5.2 Выбор базы данных	18
2 Реализация	20
2.1 Установка и настройка инструментов разработки	20
2.2 Описание архитектуры программы	25
2.3 Описание разработки модулей	28
2.3.1 Диспетчер и точка входа	28
2.3.2 Хранилище данных	29
2.3.3 Вспомогательный инструментарий	32
2.3.4 Модуль построения клавиатур	34
2.3.5 Пользовательские модули	36
2.3.6 Административные модули	43
2.4 Тестирование приложения	50
2.4.1 Методика и поэтапный процесс отладки	50
2.4.2 Тестирование пользовательской части приложения	51
2.4.3 Тестирование административной части приложения	54
2.4.4 Результаты проведенных испытаний	57
3 Руководство пользователя	58

3.1	Основные операции для клиента.....	58
3.1.1	Просмотр каталога товаров.....	58
3.1.2	Управление профилем	61
3.1.3	Просмотр истории заказов	63
3.2	Основные операции для администратора.....	66
3.3	Управление товарными позициями.....	66
3.3.1	Управление заказами	70
3.3.2	Назначение новых администраторов	73
4	Анализ результатов.....	74
4.1	Результаты разработки интернет-магазина на базе Telegram-бота	74
4.2	Рекомендации к дальнейшей разработке.....	76
5	Безопасность жизнедеятельности.....	77
5.1	Эргономика и организация рабочего места	77
5.2	Микроклимат и освещение	77
5.3	Электробезопасность	78
5.4	Пожарная безопасность	78
6	Стартап проект	79
6.1	Сегменты целевой аудитории.....	79
6.2	Ключевые процессы.....	80
6.3	Каналы взаимодействия	82
6.4	Источники дохода	82
6.5	Ключевые ресурсы.....	83
6.6	Риски проекта	84
	Заключение	85
	Список использованных источников	87
	Приложение А – UML-диаграммы.....	88
	Приложение Б – Листинг программы	107
	Приложение В – Презентация.....	151

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

ИТ – информационные технологии

ПЭВМ – персональная электронно-вычислительная машина (компьютер разработчика)

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс для взаимодействия компонентов и внешних сервисов

B2B (Business-to-Business) – коммерческое взаимодействие между компаниями, бизнес для бизнеса

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

FSM (Finite State Machine) – конечный автомат для управления пошаговыми сценариями диалога

MVC (Model-View-Controller) – паттерн, разделяющий данные, интерфейс и логику

UML (Unified Modeling Language) – язык графического моделирования структур и процессов

VPN (Virtual Private Network) – виртуальная частная сеть для защищенного соединения

					ПГУ1.090301.09.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В рамках данной работы рассматривается разработка интернет-магазина на основе Telegram-бота. Приложение предназначено для взаимодействия пользователей с каталогом товаров, оформления заказов и управления основными функциями магазина через мессенджер Telegram.

Telegram-боты являются удобным инструментом для автоматизации различных процессов и позволяют создавать сервисы с быстрым и понятным интерфейсом. Благодаря этому пользователь может выполнять необходимые действия непосредственно внутри мессенджера без использования дополнительных приложений или сайтов.

Разрабатываемая система гармонично объединяет в себе понятный пользовательский интерфейс и надежную серверную часть, отвечающую за безопасное хранение и обработку данных о доступной продукции, зарегистрированных пользователях и истории их покупок. В качестве основного технологического стека проекта были обоснованно выбраны язык программирования Python, асинхронный фреймворк aiogram для эффективного взаимодействия с Telegram Bot API, а также легковесная база данных SQLite.

В процессе реализации особое внимание уделялось детальной оптимизации навигационных сценариев, корректной обработке ошибок и созданию отказоустойчивой архитектуры. Такой подход обеспечивает стабильную эксплуатацию системы и упрощает её дальнейшее масштабирование без радикальной переработки исходного кода.

Цели выпускной квалификационной работы:

- проведение сравнительного анализа существующих технологических решений в сфере электронной коммерции и обоснование преимуществ выбранного формата платформы;
- формализация комплекса функциональных и технических требований к программному интерфейсу, внутренней бизнес-логике и транзакционной подсистеме;

- проектирование архитектуры программного комплекса и разработка реляционной структуры базы данных с обеспечением ограничений целостности;
- программная разработка модульного серверного ядра системы и реализация компонентов управления данными;
- проектирование и внедрение диалоговых сценариев пошагового взаимодействия для распределенных ролей пользователей и администраторов;
- интеграция платежного шлюза и реализация алгоритмов верификации данных для защиты от транзакционных ошибок;
- разработка вспомогательных сервисных компонентов для автоматической оптимизации интерфейса и очистки истории взаимодействия;
- проведение многоэтапного функционального тестирования программного продукта на различных типах устройств для оценки его отказоустойчивости;
- оценка практической эффективности разработанного программного решения и определение направлений его дальнейшей модернизации.

1 Анализ предметной области

Сейчас люди на постоянной основе пользуются мессенджерами. Они отправляют файлы, ищут информацию в них, переписываются, делают покупки. Это стало новым, простым способом общаться друг с другом и решать дела.

Раньше, чтобы купить вещь, нужно было посетить офлайн-магазин. Теперь практически каждый может покупать нужное, не выходя из дома, имея только доступ к интернету. Есть разные способы: сайты, крупные платформы вроде Ozon, Wildberries, Яндекс Маркета. Но у этих способов есть свои недостатки, особенно для тех, у кого небольшой магазин.

Маркетплейсы позволяют привлекать большое количество клиентов и упрощают процесс доставки товаров. Однако они взимают значительные комиссии, предусматривают штрафные санкции, а также характеризуются высокой конкуренцией. В результате небольшим магазинам часто приходится снижать цены, что существенно уменьшает их прибыль.

Создание собственного интернет-магазина является альтернативным решением, однако оно связано с финансовыми затратами и значительными трудозатратами. Необходимо разработать дизайн, обеспечить адаптивность для различных устройств, а также реализовать функциональную часть сайта и его поддержку.

В данных условиях Telegram-бот может выступать эффективным решением. Мессенджер Telegram является популярной платформой с большой аудиторией пользователей. Он позволяет не только общаться, но и реализовывать различные сервисы, включая покупку товаров. Пользователь может просматривать каталог, выбирать товары, добавлять их в корзину и оформлять заказ непосредственно в чате, без перехода на сторонние ресурсы. Это сокращает путь от выбора товара до его покупки.

Дополнительно следует отметить, что разработка Telegram-бота требует меньших затрат по сравнению с созданием полноценного интернет-магазина. Бот

					ПГУ1.090301.09.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

проще в разработке и поддержке, а также позволяет гибко добавлять новые функции. Таким образом, интернет-магазин в формате Telegram-бота является актуальным решением как для пользователей, так и для представителей малого бизнеса.

1.1 Анализ решения поставленной задачи

Интернет-магазин на базе Telegram-бота – это современное программное решение, предназначенное для комплексной автоматизации торговых процессов и обеспечения прямого взаимодействия с целевой аудиторией через интерфейс мессенджера. Подобные асинхронные сервисы позволяют эффективно заменять традиционные коммерческие веб-ресурсы, предоставляя пользователям возможность комфортно совершать покупки в привычной цифровой среде. Торговые чат-боты позволяют наглядно и безопасно организовать полный цикл продаж: просматривать актуальный каталог продукции, динамически управлять содержимым корзины и оформлять заказы непосредственно внутри диалогового окна, без перехода на сторонние ресурсы. Это существенно сокращает путь потенциального клиента от момента выбора товара до его непосредственной покупки. Данный формат отлично подходит как для потребителей, ценящих скорость и удобство интерфейса, так и для представителей малого бизнеса, которым необходимо оперативно развернуть площадку без значительных финансовых вложений.

Использование готовой инфраструктуры популярной платформы полностью исключает необходимость разработки дорогостоящих сайтов. В результате обеспечивается высокая гибкость сопровождения проекта, простота масштабирования его модульной архитектуры и стабильное функционирование всей системы при минимальных финансовых затратах на аренду оборудования.

					ПГУ1.090301.09.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

1.2 Обзор аналогов

В рамках анализа существующих решений были рассмотрены Telegram-боты, реализующие различные сценарии взаимодействия с пользователями. Особое внимание уделено архитектуре интерфейса, функциональным возможностям и используемым технологиям, включая классические чат-боты и Telegram Mini Apps.

В качестве первого аналога рассмотрим @avoVPN_bot, представленный на рисунке 1.1, который представляет собой сервис предоставления VPN-услуг на основе подписочной модели. Пользователь взаимодействует с системой через последовательный диалог, в рамках которого осуществляется выбор тарифа и оформление доступа к сервису. Функциональность бота ограничена одной услугой, что упрощает его структуру, но одновременно исключает необходимость реализации таких элементов, как каталог товаров, корзина или система многопозиционных заказов. Таким образом, архитектура решения ориентирована на узкоспециализированную задачу и не предусматривает расширяемости в сторону полноценной торговой платформы.

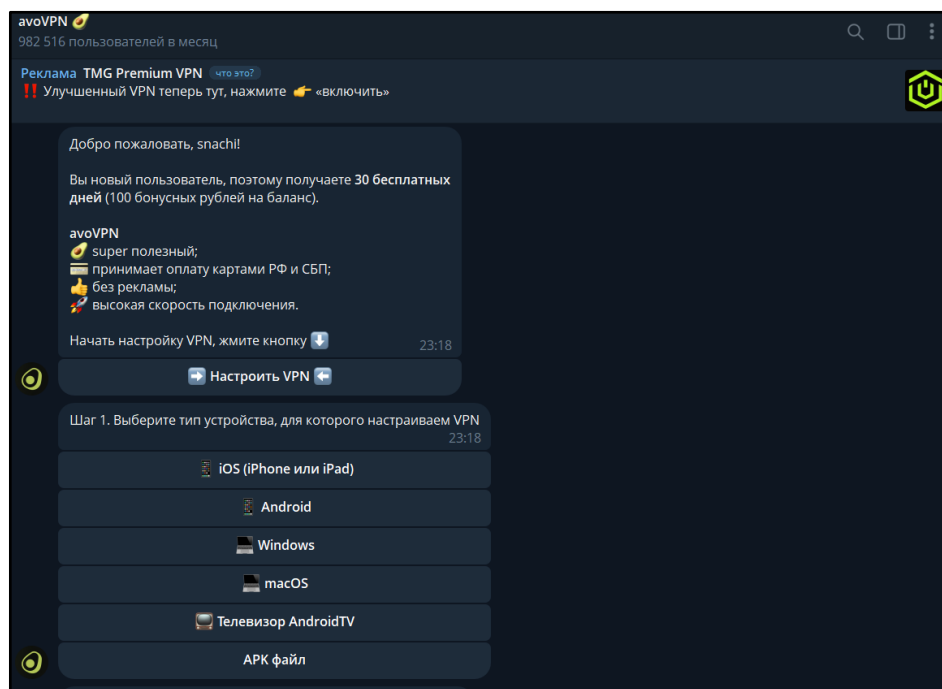


Рисунок 1.1 – Обзор аналога «@avoVPN_bot»

Следующим рассмотрим @IMP_egezebot, представленный на рисунке 1.2. Данный бот используется в образовательных целях и предоставляет пользователям задания и тесты для подготовки к экзаменам. Он демонстрирует реализацию интерактивных сценариев и обработку пользовательских ответов. Несмотря на наличие сложной логики взаимодействия, функциональность ограничена образовательной областью и не включает возможности работы с товарами и заказами.

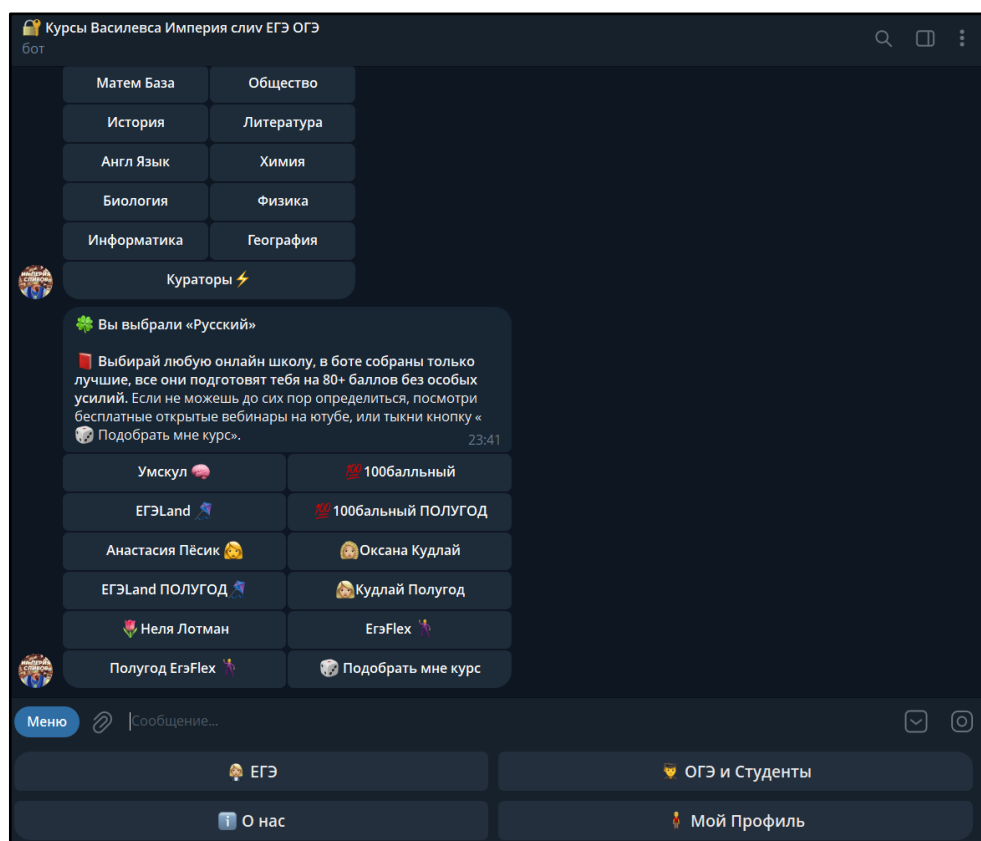


Рисунок 1.2 – Обзор аналога «@IMP_egezebot»

Бот @SpaceCosmeticsShop_Bot, представленный на рисунке 1.3, является примером интернет-магазина, реализованного с использованием технологии Telegram Mini Apps. При взаимодействии с ботом пользователь открывает встроенное веб-приложение, которое функционирует как полноценный сайт внутри Telegram. В приложении реализованы каталог товаров, карточки продукции и оформление заказа. Такой подход обеспечивает более удобный и

визуально насыщенный интерфейс. Однако подобные решения часто имеют упрощённую бизнес-логику: ограниченные возможности управления заказами, слабую персонализацию и недостаточную проработку аналитики.

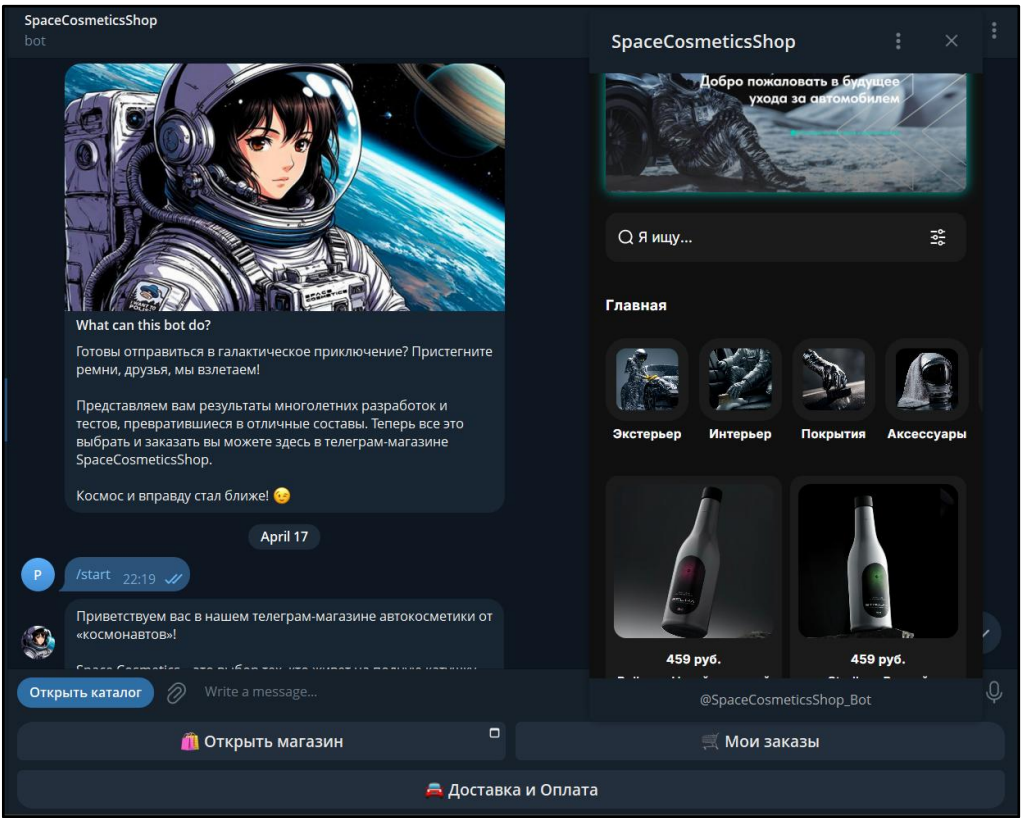


Рисунок 1.3 – Обзор аналога «@SpaceCosmeticsShop_Bot»

В следующем примере рассмотрим @gamesplaymarket_bot, представленный на рисунке 1.4, который также использует формат Mini Apps и ориентирован на продажу цифровых товаров, таких как игровые валюты и подписки. Пользователь взаимодействует с веб-интерфейсом внутри Telegram, где доступен каталог и процесс покупки. Основным преимуществом является высокая скорость обработки заказов. В то же время система остаётся узкоспециализированной и не предусматривает гибкой настройки логики интернет-магазина или расширенного управления заказами. Отсутствие возможности кастомизации исходного кода делает невозможным адаптацию подобной платформы под специфические бизнес-процессы или уникальные маркетинговые задачи конкретного продавца.

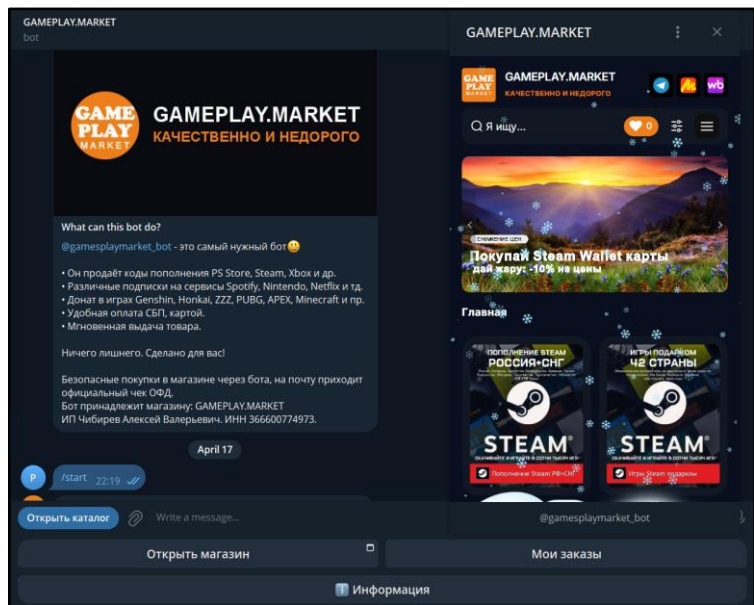


Рисунок 1.4 – Обзор аналога «@gamesplaymarket_bot»

Последним рассмотренным аналогом будет @Cveti_kursk_bot, представленный на рисунке 1.5, который представляет собой интернет-магазин цветов с доставкой. Пользователю доступен каталог товаров с возможностью выбора и оформления заказа. Реализован базовый сценарий покупки, однако функциональность ограничена: отсутствует полноценная корзина, гибкая работа с заказами и персонализация взаимодействия с пользователем.

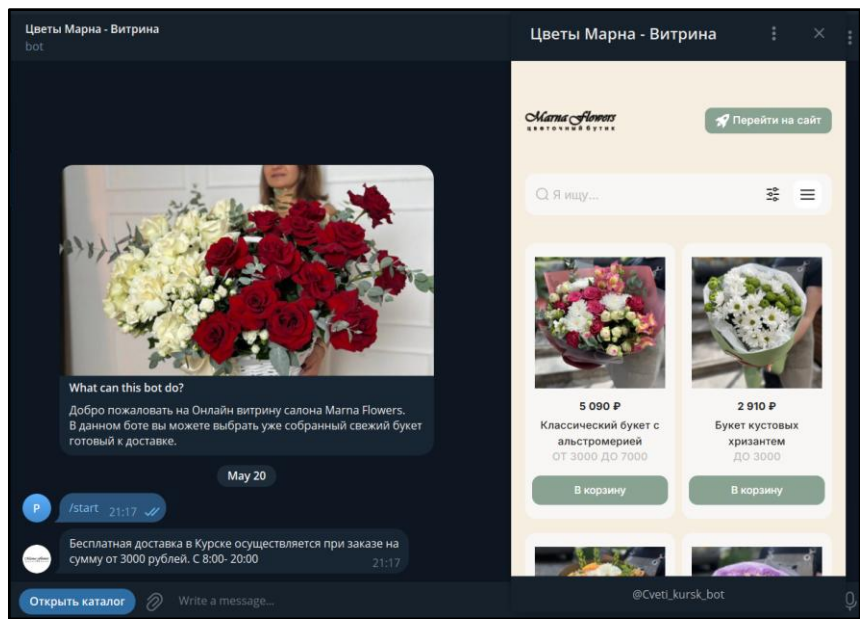


Рисунок 1.5 – Обзор аналога «@Cveti_kursk_bot»

Таким образом, проведенный анализ существующих решений показал, что классические чат-боты либо жестко ограничены одной узкоспециализированной услугой, либо полностью лишены коммерческого функционала, такого как корзина и система многопозиционных заказов. В свою очередь, современные решения на базе технологии Telegram Mini Apps обеспечивают визуально насыщенный интерфейс, но часто страдают от упрощенной бизнес-логики, слабой персонализации и недостаточных возможностей управления заказами и аналитики. Существующие же коммерческие боты в классическом формате диалога не предоставляют пользователю полноценного функционала корзины и гибких сценариев взаимодействия.

Это подтверждает актуальность проектирования и разработки интернет-магазина, который совместит в себе интуитивно понятную навигацию классического интерфейса с мощной асинхронной серверной частью, способной обеспечить полноценный цикл обработки заказов, сквозную валидацию данных и развитый административный функционал управления торговой платформой.

1.3 Постановка задачи проекта

Требуется разработать интернет-магазин на основе Telegram-бота, обеспечивающий возможность просмотра товаров, оформления заказов и взаимодействия с пользователем в рамках мессенджера. В рамках выполнения проекта должны быть выполнены следующие аспекты:

- пользовательский интерфейс бота должен обеспечивать удобную навигацию между основными разделами: главное меню, каталог товаров с возможностью навигации по страницам и просмотра товаров, профиль пользователя, корзина, а также функции оформления заказа и отслеживания их статусов;
- система должна обеспечивать хранение и обработку данных о товарах, пользователях, корзинах и заказах;

- бот должен обеспечивать возможность добавления товаров в корзину, изменения её содержимого и оформления заказа;
- система должна обеспечивать обработку пользовательских запросов и корректную работу всех сценариев взаимодействия;
- приложение должно предусматривать интеграцию платежной системы для осуществления онлайн-оплаты заказов;
- система должна обеспечивать корректную обработку ошибок, включая ввод некорректных данных и нестабильное соединение;
- приложение должно быть развернуто на сервере и обеспечивать стабильную и непрерывную работу.

1.4 Минимальные системные требования

1.4.1 Требования для пользователя

Поскольку взаимодействие с интернет-магазином осуществляется через мессенджер Telegram, пользователю необходимо устройство с установленным приложением Telegram и доступом к сети Интернет.

Приложение может использоваться на смартфонах, планшетах и персональных компьютерах. Для стабильной работы достаточно устройства со следующими характеристиками:

- операционная система Android 7.0 и выше;
- iOS 13 и выше;
- Windows 10 и выше;
- установленное приложение Telegram;
- подключение к сети интернет;
- не менее 2 ГБ оперативной памяти.

Дополнительная установка специализированного программного обеспечения не требуется, так как все взаимодействие с системой выполняется непосредственно через Telegram.

					ПГУ1.090301.09.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

1.4.2 Требования для сервера

Для размещения и стабильной работы Telegram-бота требуется сервер или облачная платформа с поддержкой Python-приложений и постоянным подключением к сети Интернет.

Для корректной работы системы достаточно следующих характеристик:

- операционная система Ubuntu 20.04, Windows 10;
- процессор с частотой не менее 1 ГГц;
- не менее 1 ГБ оперативной памяти;
- не менее 2 ГБ свободного места на диске;
- установленный Python версии 3.11 и выше;
- стабильное подключение к сети Интернет;
- установленные библиотеки aiogram и aioredis.

В качестве системы хранения данных используется SQLite, поэтому установка отдельного сервера базы данных не требуется. Разработанное приложение может быть размещено как на локальном компьютере, так и на облачных платформах для размещения Telegram-ботов.

1.5 Выбор инструментов разработки

При разработке интернет-магазина на основе Telegram-бота особое внимание было уделено выбору инструментов и технологий, так как от этого напрямую зависит стабильность работы системы, удобство разработки и возможность дальнейшего расширения функционала.

1.5.1 Выбор языка программирования

В качестве первого возможного решения был рассмотрен Node.js, работающий на базе программной платформы JavaScript. Данный инструмент ценится за высокую скорость обработки сетевых запросов и событийно-ориентированную асинхронную архитектуру, которая позволяет эффективно справляться с большим потоком одновременных подключений в реальном времени.

					ПГУ1.090301.09.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

Однако Node.js накладывает определенные ограничения при проектировании торговых систем. Разработка классической бизнес-логики, управления сложными пошаговыми состояниями пользователей и валидации транзакций на JavaScript требует написания избыточного объема кода, а существующие библиотеки для интеграции с Telegram Bot API уступают аналогам по уровню стандартизации и документированности.

В качестве альтернативного варианта был проанализирован язык программирования Python. Его ключевыми преимуществами являются лаконичный синтаксис, высокая читаемость кода и развитая экосистема готовых модулей, что существенно сокращает сроки разработки и упрощает процесс последующего сопровождения серверной части приложения [1].

Для Python создана мощная асинхронная библиотека aiogram, которая предоставляет готовые архитектурные абстракции для маршрутизации событий и встроенную машину состояний. Это позволяет организовать стабильную параллельную обработку сотен клиентских запросов без риска блокировки основного потока выполнения программы.

На основе проведенного сравнения для реализации проекта был выбран язык программирования Python. Он обеспечивает оптимальное сочетание простоты написания модулей и производительности асинхронной серверной части, превосходя Node.js по уровню готовности специализированного инструментария для разработки Telegram-ботов.

1.5.2 Выбор базы данных

Для организации слоя хранения данных рассматривалась мощная реляционная СУБД PostgreSQL, которая широко применяется в высоконагруженных промышленных проектах. Она предлагает развитые механизмы обеспечения целостности информации, поддержку сложных аналитических запросов и параллельное выполнение операций на уровне таблиц. Несмотря на очевидные достоинства, PostgreSQL требует обязательного развертывания, тонкой настройки и непрерывного администрирования отдельного

полноценного сервера баз данных. Для интернет-магазина в формате мессенджера подобные возможности являются избыточными, а усложнение архитектуры влечет за собой дополнительные финансовые затраты на хостинг и увеличивает трудоемкость деплоя приложения.

В качестве второго аналога была рассмотрена СУБД SQLite, представляющая собой встраиваемую реляционную базу данных. Она сохраняет всю необходимую информацию в одном локальном файле на диске сервера и функционирует без запуска выделенного серверного процесса, что минимизирует потребление оперативной памяти и процессора. В связке с асинхронной библиотекой `aiosqlite` данное решение обеспечивает безопасность финансовых расчетов за счет поддержки атомарных транзакций и исключает сетевые задержки при обмене данными. Нормализованная структура таблиц позволяет легко оперировать информацией о пользователях, товарах и корзинах, гарантируя высокую скорость CRUD-операций [2].

В итоге для торговой платформы оптимальным решением стал выбор СУБД SQLite. Она полностью покрывает технические потребности малого бизнеса, гарантирует реляционную целостность данных и исключает необходимость сложного администрирования инфраструктуры, выгодно отличаясь от PostgreSQL экономичностью и простотой развертывания.

2 Реализация

2.1 Установка и настройка инструментов разработки

Перед началом разработки интернет–магазина на основе Telegram-бота на Python необходимо установить программу позволяющую работать с данным языком. Нужно перейти на официальный сайт «<https://www.python.org/downloads/>» и скачать бесплатную и полнофункциональную среду Python 3.12.10 (32-bit). После скачивания дистрибутива, запускаем его и проходим процедуру установки программы как показано на рисунке 2.1.

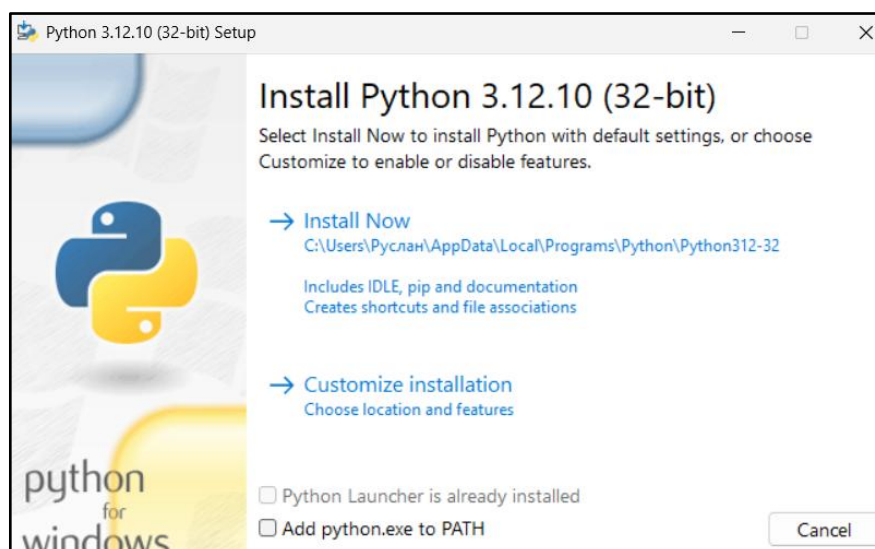


Рисунок 2.1 – Установка Python

Для подтверждения корректности завершения инсталляции вызывается командная строка операционной системы, где выполняется проверка версии интерпретатора, результат работы которой представлен на рисунке 2.2.

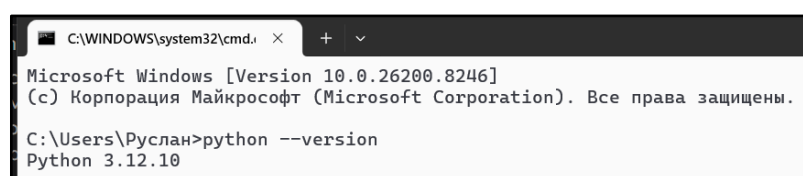


Рисунок 2.2 – Проверка установленной версии Python

					ПГУ1.090301.09.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

Следующим этапом настройки рабочей среды является развертывание специализированных программных зависимостей. Взаимодействие с Telegram Bot API осуществляется посредством асинхронного фреймворка aiogram, для загрузки которого применяется стандартный менеджер пакетов pip. Команда инсталляции фиксированной версии библиотеки вводится в консоль терминала, что представлено на рисунке 2.3.

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.26200.8246]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Руслан>pip install aiogram==3.22.0
Requirement already satisfied: aiogram==3.22.0 in c:\users\руслан\appdata\local\programs\python\python312\lib\site-packages (3.22.0)
Requirement already satisfied: aiofiles<24.2,>=23.2.1 in c:\users\руслан\appdata\local\programs\python\python312\lib\site-packages (from aiogram==3.22.0) (24.1.0)
Requirement already satisfied: aiohttp<3.13,>=3.9.0 in c:\users\руслан\appdata\local\programs\python\python312\lib\site-packages (from aiogram==3.22.0) (3.12.15)
Requirement already satisfied: certifi>=2023.7.22 in c:\users\руслан\appdata\local\programs\python\python312\lib\site-packages (from aiogram==3.22.0) (2025.10.5)
Requirement already satisfied: magic-filter<1.1,>=1.0.12 in c:\users\руслан\appdata\local\programs\python\python312\lib\site-packages (from aiogram==3.22.0) (1.0.12)
  
```

Рисунок 2.3 – Установка aiogram

По завершении скачивания связанных модулей выполняется контрольный запрос метаданных пакета, позволяющий удостовериться в успешной интеграции фреймворка, как показано на рисунке 2.4.

```

C:\Users\Руслан>pip show aiogram
Name: aiogram
Version: 3.22.0
Summary: Modern and fully asynchronous framework for Telegram Bot API
Home-page: https://aiogram.dev/
Author:
Author-email: Alex Root Junior <jroot.junior@gmail.com>
License-Expression: MIT
Location: C:\Users\Руслан\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires: aiofiles, aiohttp, certifi, magic-filter, pydantic, typing-extensions
Required-by:
  
```

Рисунок 2.4 – Проверка установленной версии aiogram

Для установки основного инструмента проектирования переходим на сайт «<https://code.visualstudio.com/download>» и скачиваем нужный дистрибутив. Процесс развертывания Visual Studio Code начинается с запуска официального мастера установки, где на начальном этапе пользователь подтверждает принятие условий лицензионного соглашения корпорации Microsoft, что показано на рисунке 2.5.

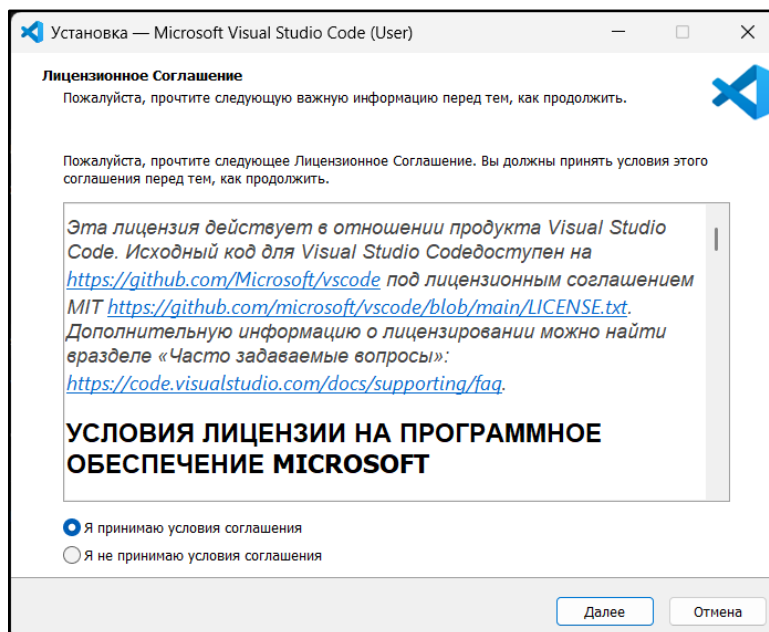


Рисунок 2.5 – Установка Visual Studio Code

В последующем диалоговом окне конфигурации инсталлятора производится настройка дополнительных задач, включая обязательную регистрацию редактора в системе под поддерживаемые типы файлов и добавление путей в PATH для обеспечения сквозного вызова среды из консоли, как отображено на рисунке 2.6.

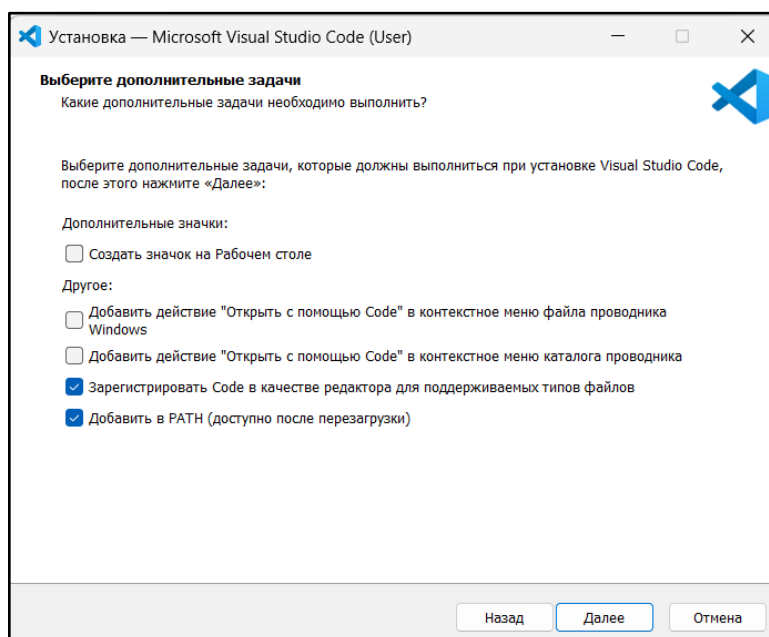


Рисунок 2.6 – Установка Visual Studio Code

На финальном подготовительном шаге мастер обобщает выбранные параметры и подтверждает полную готовность программного комплекса к непосредственному копированию бинарных файлов на жесткий диск, что представлено на рисунке 2.7.

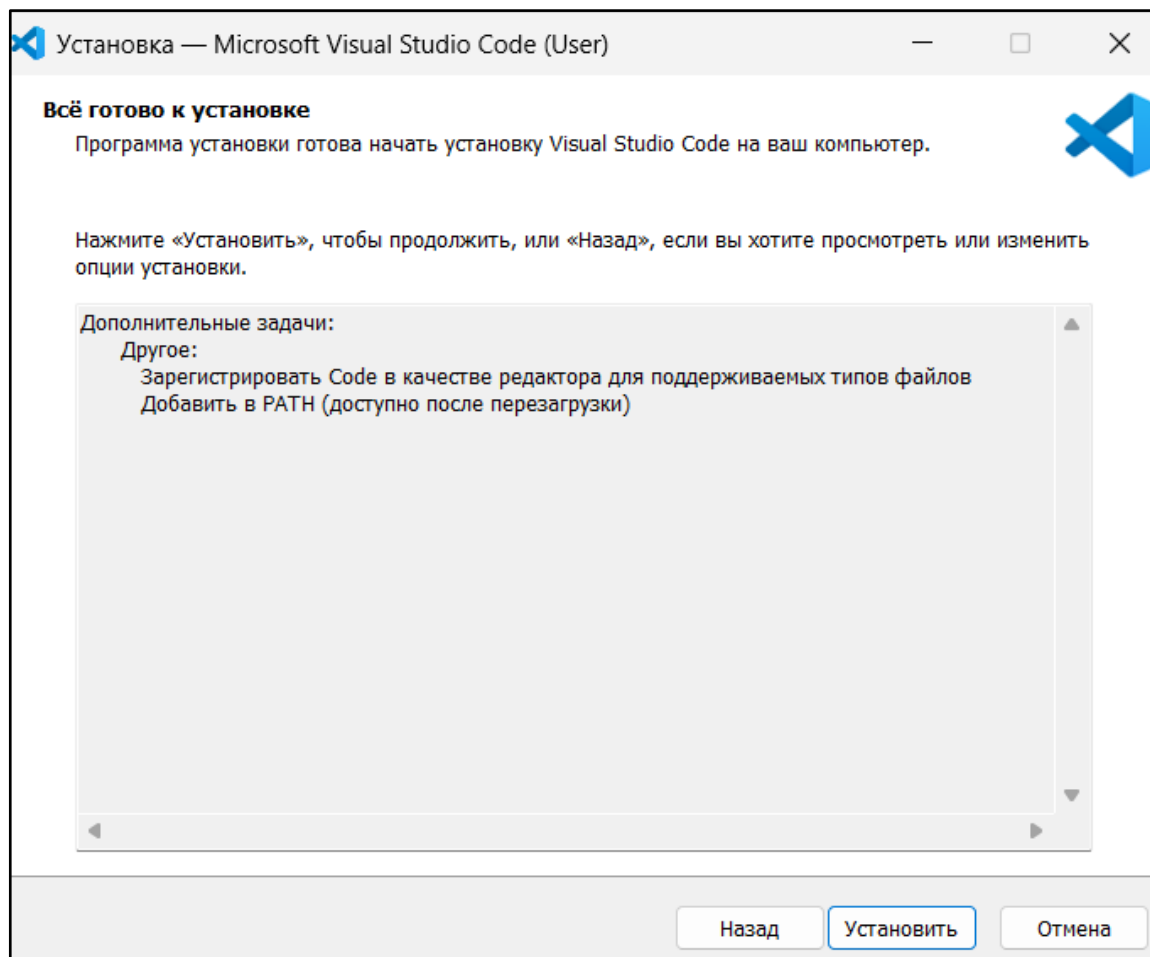


Рисунок 2.7 – Установка Visual Studio Code

Для обеспечения асинхронного взаимодействия Telegram-бота с локальной базой данных требуется развертывание библиотеки `aiosqlite`. Установка данного модуля осуществляется через командную строку операционной системы с использованием менеджера пакетов `pip`, как показано на рисунке 2.8

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.26200.8457]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Руслан>pip install aiosqlite
Requirement already satisfied: aiosqlite in c:\users\руслан\appdata\local\programs\python\python312\lib\site-packages (0.22.1)

[notice] A new release of pip is available: 25.0.1 -> 26.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\Руслан>pip show aiosqlite
Name: aiosqlite
Version: 0.22.1
Summary: asyncio bridge to the standard sqlite3 module
Home-page:
Author:
Author-email: Amethyst Reese <amethyst@n7.gg>
License:
Location: C:\Users\Руслан\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires:
Required-by:

```

Рисунок 2.8 – Установка Visual Studio Code

Следующим этапом является подготовка инструмента для проектирования базы данных. Для этих целей будет использован программный комплекс DB Browser for SQLite. После прохождения базовых шагов конфигурации в мастере установки выводится финальное диалоговое окно, подтверждающее успешное завершение инсталляции графического клиента, что представлено на рисунке 2.9.

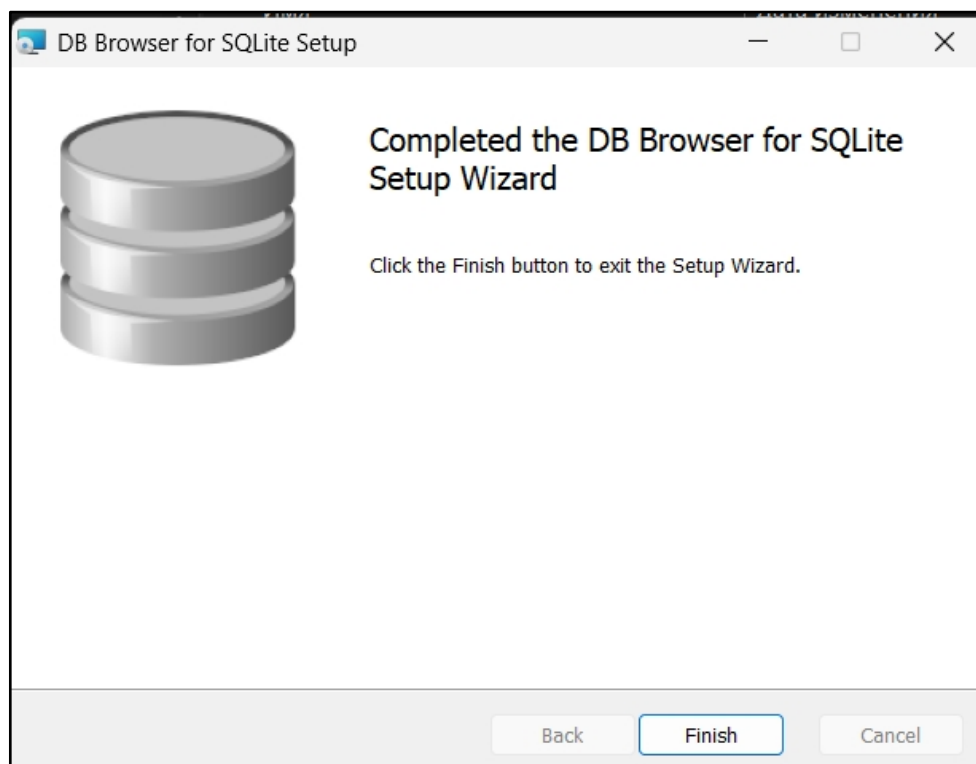


Рисунок 2.9 – Установка Visual Studio Code

2.2 Описание архитектуры программы

Индивидуальный проект построен на модульной архитектуре, которая делает кодовую базу гибкой, упрощает ее поддержку и открывает возможности для расширения функциональности без существенного переписывания уже работающих модулей. Каждый компонент решает строго определенный круг задач. Принцип разделения ответственности реализован через такую декомпозицию, а фундаментом всей системы стал асинхронный фреймворк aiogram версии 3.x, который предоставляет удобные абстракции для маршрутизации запросов, управления состояниями и обработки событий.

Архитектура проекта опирается на три условных слоя, организованных по принципу MVC, где первый слой объединяет представление и маршрутизацию, а входящие в него роутеры закреплены за каталогом товаров, корзиной, пользовательским профилем и управлением заказами и регистрируются централизованно в главном диспетчере dp внутри модуля main.py, который служит точкой входа для всех маршрутов. Чтобы расширить функциональность, достаточно создать новый файл-роутер и зарегистрировать его – существующий код при этом остается нетронутым [3].

Слой бизнес-логики отличается наибольшей программной насыщенностью среди всех компонентов архитектуры, а его ядро формируют хэндлеры, распределенные по отдельным модулям, где реализованы пользовательские сценарии взаимодействия с ботом. Библиотека aiogram предоставляет машину состояний, которую хэндлеры используют для организации пошаговых диалогов. Когда администратор добавляет товар или пользователь заполняет профиль, механизм работы FSM становится особенно очевиден. Данные собираются последовательно через цепочки состояний ProfileStates и AddProductStates, а колбэки жестко контролируют переходы между ними, предотвращая некорректный ввод вне установленного порядка. Всю навигацию и взаимодействие обеспечивают inline-клавиатуры. Логика их генерации сосредоточена в классах AdminKeyboards и UserKeyboards модуля keyboards.py, и

такая централизация устранила дублирование кнопок, обеспечив единообразие интерфейса во всех разделах бота.

В нижней, третьей позиции архитектуры располагается слой доступа к данным, реализованный в модуле database.py через класс Database, который воплощает паттерн «Репозиторий» и скрывает сырые SQL-запросы за понятными методами, позволяющими взаимодействовать с базой без написания низкоуровневого кода. Асинхронность всех операций обеспечивает библиотека aioredis. Основной поток обработки сообщений благодаря этому никогда не блокируется, а для бота, обслуживающего множество пользователей параллельно, подобное архитектурное решение оказывается принципиальным, поскольку любая задержка в одном запросе не затрагивает обработку остальных [4].

Метод create_tables формирует тщательно нормализованную схему базы данных, как показано на рисунке 2.10, причем цены хранятся исключительно в копейках через целочисленные поля price и price_at_purchase, что устраняет погрешности округления, свойственные числам с плавающей точкой, и гарантирует точность финансовых расчетов при формировании счетов и отчетов.

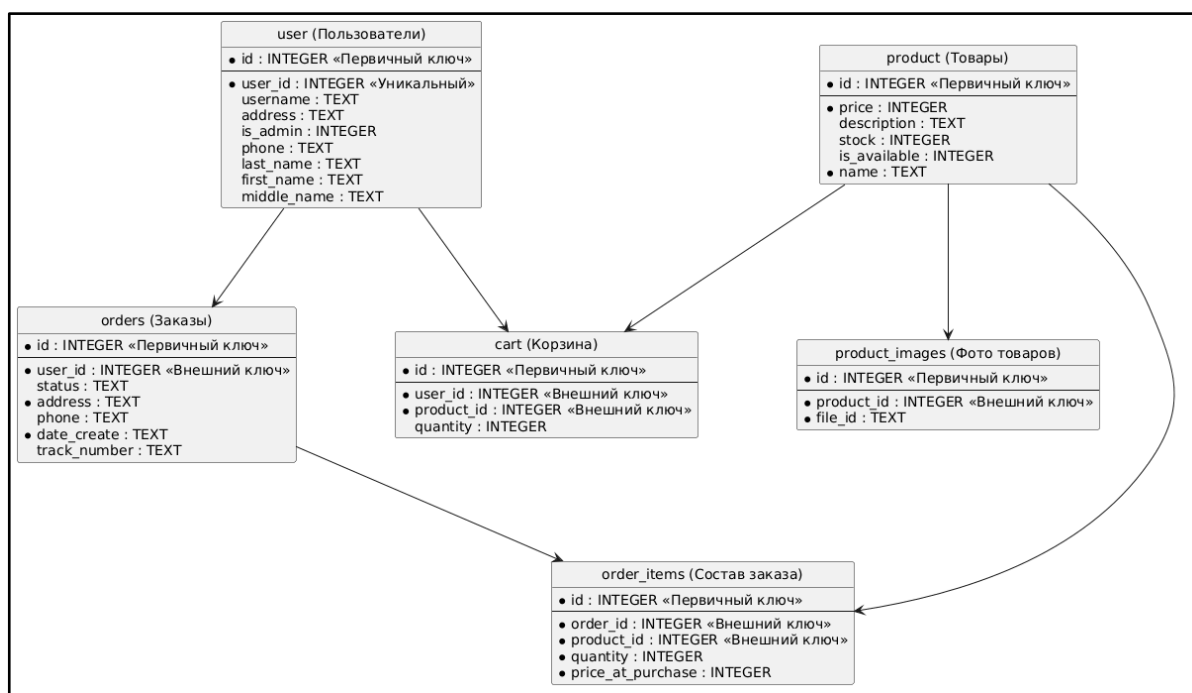


Рисунок 2.10 – UML-диаграмма базы данных

Оплата выстроена как самостоятельный архитектурный модуль, опирающийся на принцип предварительного отбора товаров: пользователь помечает галочками нужные позиции в корзине с помощью кнопок `select_pay_` и `unselect_pay_`, а идентификаторы выбранных товаров попадают в поле `selected_items` внутри FSM-состояния. Хэндлер `process_payment` проверяет наличие отмеченных позиций на складе и корректность профиля покупателя. Убедившись, что данные верны, он формирует счет и отправляет его пользователю. Для отладки применяется тестовый платежный токен Telegram, который позволяет пройти полный цикл оплаты без реальных денежных транзакций; когда понадобится промышленная эксплуатация, достаточно будет заменить его на рабочий токен от платежного провайдера.

В файле `main.py` зарегистрирован middleware `save_user_messages_middleware`, который перехватывает каждое входящее событие от пользователя и через функцию `save_msg` записывает идентификаторы сообщений в оперативную память, обеспечивая критически важную для интерфейса функциональность. Утилита `clear_chat` работает в связке с этим механизмом и удаляет устаревшие сообщения-меню непосредственно перед выводом новых, поэтому при переходе из главного меню в каталог все предшествующие кнопки исчезают, а на экране остается только актуальный контент. Визуальный мусор устраняется полностью, навигация по разделам магазина становится интуитивно понятной. Ручное тестирование всех сценариев подтвердило корректность работы этого решения.

Трехслойная архитектура приложения четко разграничивает представление, бизнес-логику и доступ к данным, что обеспечивает модульность системы и упрощает тестирование. При добавлении новых возможностей существенная переработка уже написанного кода не потребуется, поскольку архитектура изначально предусматривает расширение функционала.

2.3 Описание разработки модулей

Архитектура проекта опиралась на принцип независимых функциональных модулей, при котором каждый файл (за исключением конфигурационных) решает строго определенную задачу и существует автономно. Такой подход позволил вести итеративную разработку, продвигаясь в нескольких частях системы сразу, и упростил отладку. На рисунке 2.11 можно увидеть все разработанные модули. Ниже каждый модуль рассмотрен отдельно.

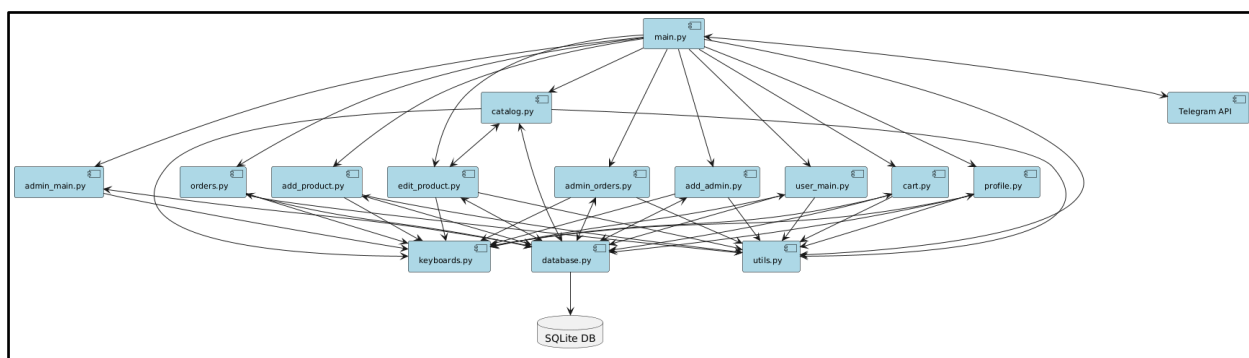


Рисунок 2.11 – UML-диаграмма компонентов

2.3.1 Диспетчер и точка входа

Модуль служит ядром приложения и выполняет исключительно конфигурационные и диспетчерские функции, не содержа при этом никакой бизнес-логики; при запуске создается объект бота с токеном от BotFather, а также главный диспетчер, связанный с хранилищем состояний MemoryStorage. Оперативную память для FSM-состояний выбрали осознанно, поскольку диалоги кратковременны и сохранять их между перезапусками нет необходимости.

Модуль регистрирует outer-middleware save_user_messages_middleware, и этот компонент берет на себя центральную задачу. Middleware срабатывает на каждое входящее событие Update после того, как хэндлер завершил обработку, сохраняя message_id пользовательского сообщения в историю. Функция main() централизованно подключает все роутеры проекта через вызовы dp.include_router(), причем порядок подключения здесь существенен: роутеры с

более специфичными фильтрами должны располагаться раньше, иначе диспетчер не сможет корректно маршрутизировать запросы. После подключения всех компонентов запускается бесконечный цикл long-polling через `dp.start_polling()`, и бот непрерывно опрашивает серверы Telegram, ожидая новых событий. Полная схема представлена на рисунке 2.12.

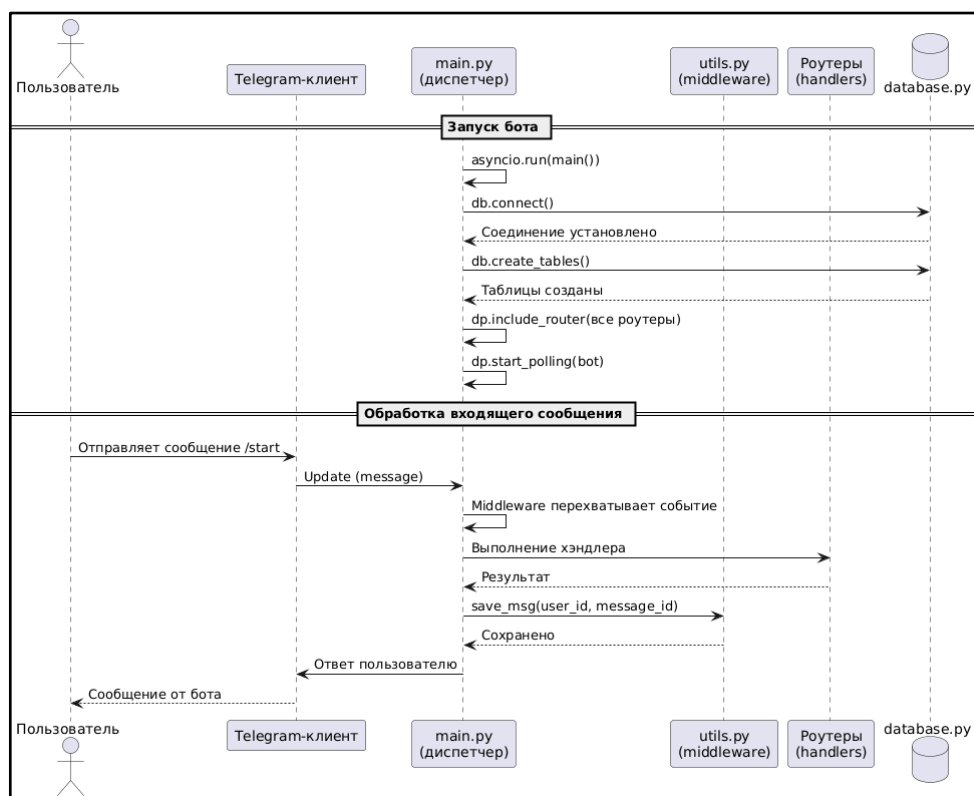


Рисунок 2.12 – UML-диаграмма модуля main.py

2.3.2 Хранилище данных

Модуль организован по классическому паттерну проектирования «Репозиторий» и функционирует как централизованная единственная точка доступа к базе данных SQLite. При этом программный класс Database полностью инкапсулирует в себе сетевое соединение через специальный вызов асинхронного подключения и содержит исчерпывающий набор методов для выполнения базовых CRUD-операций над всеми ключевыми сущностями разработанного приложения. Асинхронная архитектура на данном программном уровне является принципиальной, поскольку традиционные синхронные запросы к файлу базы

данных неизбежно заблокировали бы единый главный событийный поток программы и полностью остановили бы обработку входящих сообщений от остальных активных пользователей мессенджера на все время выполнения конкретной дисковой операции.

Метод `create_tables` отвечает за автоматическое формирование полной схемы реляционной базы данных. Он превентивно создает необходимые внутренние таблицы с наложением жестких структурных ограничений целостности, включая правила первичного ключа, уникальности, запрета пустых значений и параметров по умолчанию, а также корректно устанавливает логические реляционные связи через внешние ключи с обязательным каскадным удалением зависимых записей. За каждой такой операцией стоит строго параметризованный SQL-запрос со специальными плейсхолдерами знаков вопроса, полностью защищающими подсистему хранения от потенциальных вредоносных SQL-инъекций со стороны внешних источников.

Метод `get_product_by_id` спроектирован иначе и выполняет два последовательных независимых запроса вместо одного ресурсоемкого и сложного объединения таблиц. Первый запрос извлекает основные текстовые и числовые характеристики выбранного товара, а второй запрос собирает все связанные уникальные идентификаторы из таблицы медиафайлов. Схема вызова асинхронных методов репозитория и пошаговая последовательность выполнения соответствующих SQL-запросов наглядно продемонстрированы на рисунке 2.13. Такой подход позволяет минимизировать объем передаваемых данных между драйвером SQLite и приложением, поскольку при использовании JOIN для каждого товара происходило бы дублирование основных полей для каждой фотографии. Кроме того, разделение запросов упрощает отладку и тестирование каждого этапа извлечения данных независимо. В случае отсутствия фотографий второй запрос просто возвращает пустой список, что не требует дополнительной логики обработки NULL-значений на стороне бота. Данное решение также лучше масштабируется при добавлении новых типов связанных сущностей.

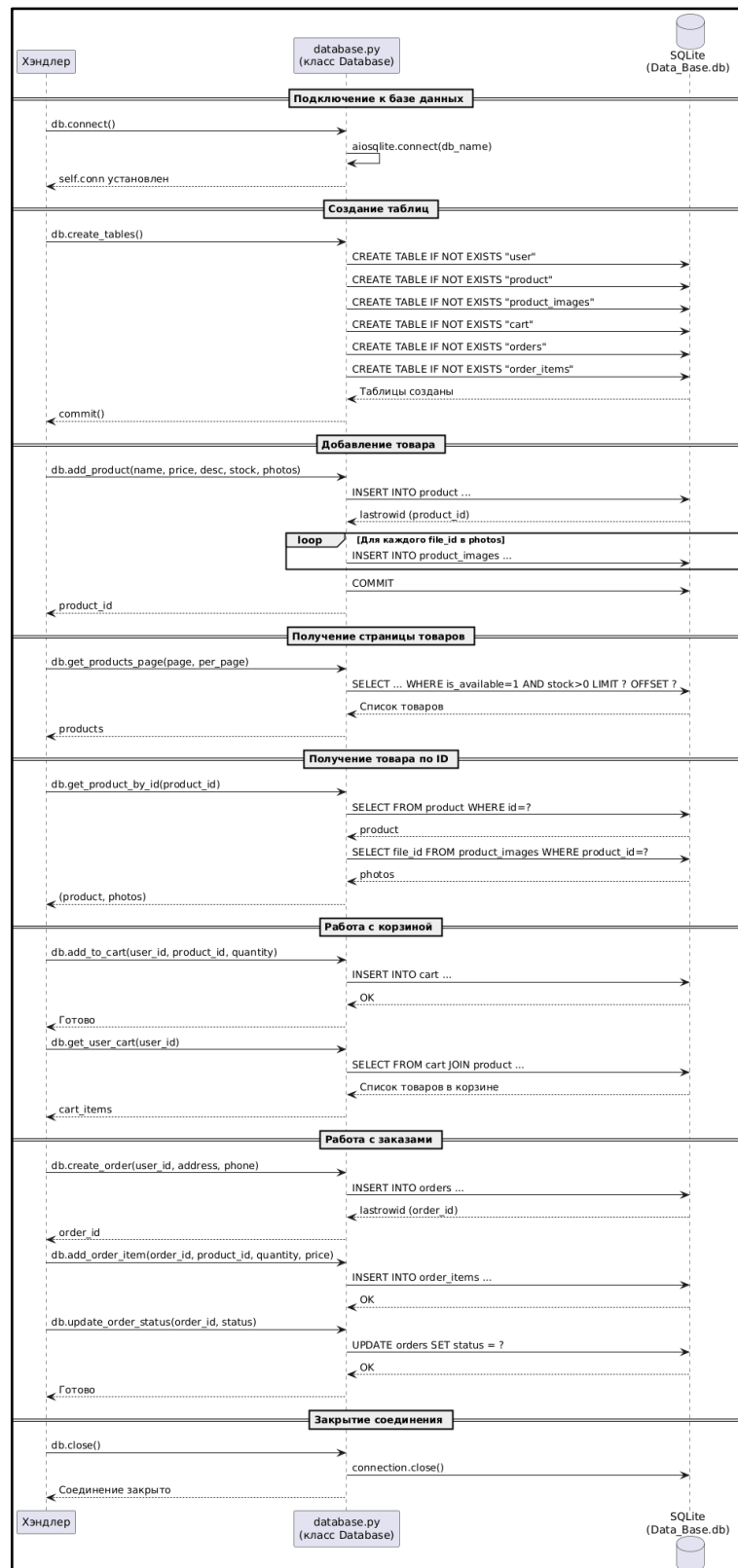


Рисунок 2.13 – UML-диаграмма модуля database.py

2.3.3 Вспомогательный инструментарий

Практически каждый хэндлер проекта обращается к общим утилитам этого модуля, и ключевой среди них остается система «чистого чата». Она построена на глобальном словаре `user_messages_history`, где хранятся идентификаторы сообщений для каждого пользователя в оперативной памяти, и на двух вспомогательных функциях. Функция `save_msg(user_id, msg_id)` записывает новый идентификатор в список конкретного пользователя. Функция `clear_chat(bot, user_id)` извлекает накопленные идентификаторы и передает их методу `bot.delete_messages()` для массового удаления; поскольку Telegram API позволяет удалять за один вызов не более ста сообщений, список предварительно разбивается на чанки подходящего размера. Когда удалить сообщение не получается, программа перехватывает исключение, фиксирует его в логе и продолжает выполнение без остановки. Такой подход позволяет поддерживать интерфейс в актуальном состоянии и полностью исключает зависание бота при попытке взаимодействия с устаревшими данными.

Модуль содержит отдельный блок функций для работы с ценами. Когда администратор вводит строку вроде «23.03» или «23,03», она поступает в `parse_price`, где удаляются пробелы, запятая заменяется точкой, а итоговое значение конвертируется в целое число копеек (2303). Обратное преобразование выполняет `format_price`, превращая это число в читаемое представление для пользователя. В базе данных хранятся и обрабатываются исключительно целочисленные значения, благодаря чему погрешности округления полностью исключены. Реализованный комплекс утилит обеспечивает централизованную валидацию и унификацию данных, существенно упрощая код основных обработчиков проекта. Алгоритмы сохранения ID сообщений, пакетного удаления чанков истории, а также подсистемы парсинга и форматирования ценовых значений подробно продемонстрированы на рисунке 2.14.

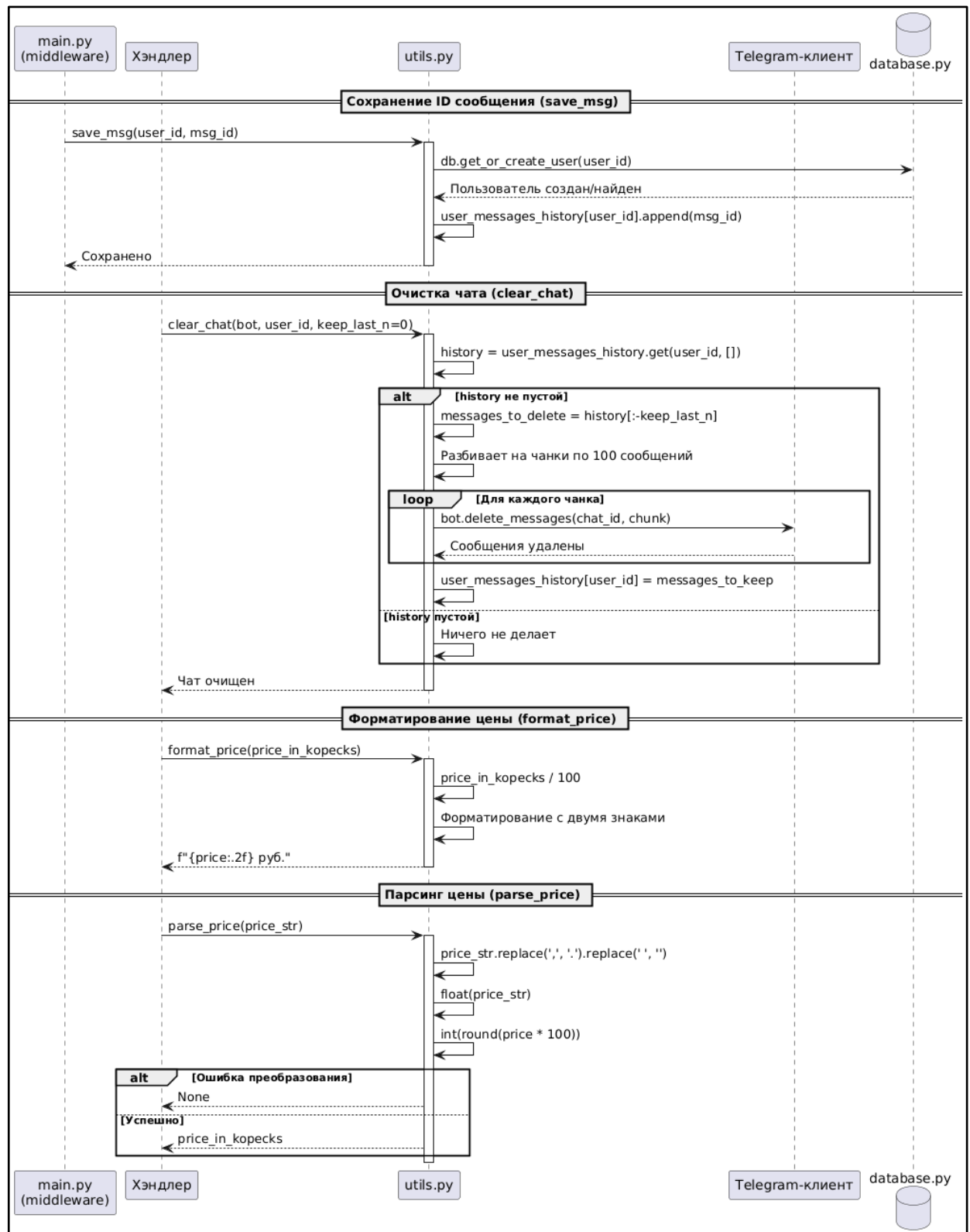


Рисунок 2.14 – UML-диаграмма модуля utils.py

2.3.4 Модуль построения клавиатур

Модуль целиком отвечает за формирование inline-клавиатур и определение визуального оформления кнопок. Классы `UserKeyboards` и `AdminKeyboards` содержат исключительно статические методы, причем каждый из них создает экземпляр `InlineKeyboardBuilder`, наполняет его необходимыми кнопками с привязанными `callback_data` и возвращает сконструированную клавиатуру. Вынос разметки в изолированные классы позволяет полностью отделить слой графического представления от контроллеров обработки событий, строго следуя архитектурному принципу разделения ответственности.

Такой подход к организации кода дает несколько практических преимуществ, и главное из них связано с простотой сопровождения. Когда логика отображения сосредоточена в одном месте, изменить текст кнопки или ее расположение можно редактированием единственного фрагмента. Хэндлеры при этом остаются компактными, потому что сборка клавиатур вынесена в отдельные методы – программист просто вызывает `UserKeyboards.cart_item_kb(p_id, quantity, is_selected)` и получает готовый результат. Параметры методов позволяют динамически менять состав элементов интерфейса в зависимости от ситуации. Метод `product_preview_kb` показывает кнопку «Редактировать» только администраторам (при `is_admin=True`), а кнопка «В корзину» превращается в «Удалить из корзины», когда флаг `in_cart` сигнализирует о том, что товар уже добавлен. Один метод работает в разных контекстах, и дублировать код не приходится. Централизованная генерация интерфейсных элементов также гарантирует строгое единообразие структуры callback-данных во всех модулях приложения. Взаимодействие хэндлеров с фабрикой интерфейсов при формировании различных типов inline-клавиатур и динамическом изменении состава кнопок подробно отражено на рисунке 2.15.

2.3.5 Пользовательские модули

Эта группа модулей реализует клиентский путь, то есть все сценарии взаимодействия обычного покупателя с магазином.

Файл `user_main.py` открывает пользователю вход в приложение. Команда `/start` запускает процесс регистрации или авторизации, после чего модуль обращается к базе данных через `db.get_or_create_user`, очищает историю чата и выводит главное меню. Функция `db.is_admin(user_id)` определяет, какой интерфейс получит пользователь (стандартный вариант содержит каталог и корзину, а расширенный дополнительно включает кнопку «Админ панель»), и в том же модуле размещен хэндлер `back_to_menu`, который возвращает пользователя из любого раздела приложения и сбрасывает текущее FSM-состояние. Взаимодействие компонентов подробно продемонстрировано на рисунке 2.16.

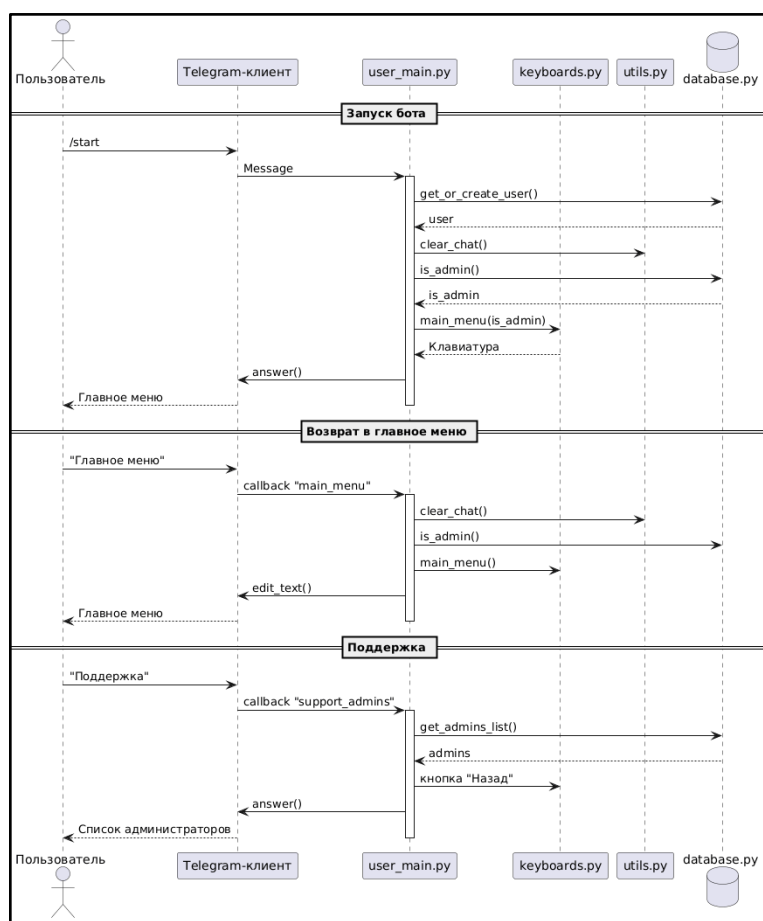


Рисунок 2.16 – UML-диаграмма модуля `user_main.py`

Модуль catalog.py управляет постраничной навигацией по списку товаров, а константа PRODUCTS_PER_PAGE задает количество карточек на одной странице. Хэндлер show_catalog обрабатывает первичное открытие каталога и переключение между страницами через колбэки catalog_page_. Номер текущей страницы сохраняется в поле catalog_page внутри FSM-состояния пользователя, и после выхода из карточки товара человек возвращается на ту страницу, с которой начал просмотр. Динамический расчет общего количества доступных страниц происходит в реальном времени на основе актуального пула товаров в репозитории, что исключает некорректное отображение пустых окон интерфейса. Сохранение текущей позиции в оперативной памяти конечного автомата существенно повышает скорость отклика приложения и снижает нагрузку на базу данных при циклических переходах.

Функция send_product_card работает в нескольких контекстах и формирует детальный просмотр товара, обеспечивает отображение позиции при администраторском редактировании, показывает товары, отсутствующие на складе. Если у товара загружено несколько фотографий, они группируются и отправляются через MediaGroupBuilder, а кнопки действий уходят отдельным сообщением. Использование отдельной генерации медиафайлов и клавиатуры обусловлено техническими ограничениями Bot API, не позволяющими прикреплять интерактивную inline-панель напрямую к альбому. Такой инженерный подход позволяет полностью сохранить управляемость торговой позицией без ухудшения визуальной информативности презентуемой карточки. Пошаговый алгоритм навигации по страницам, динамического формирования карточек продукции и обработки запросов пользователя в интерфейсе каталога подробно продемонстрирован на рисунке 2.17.

					ПГУ1.090301.09.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

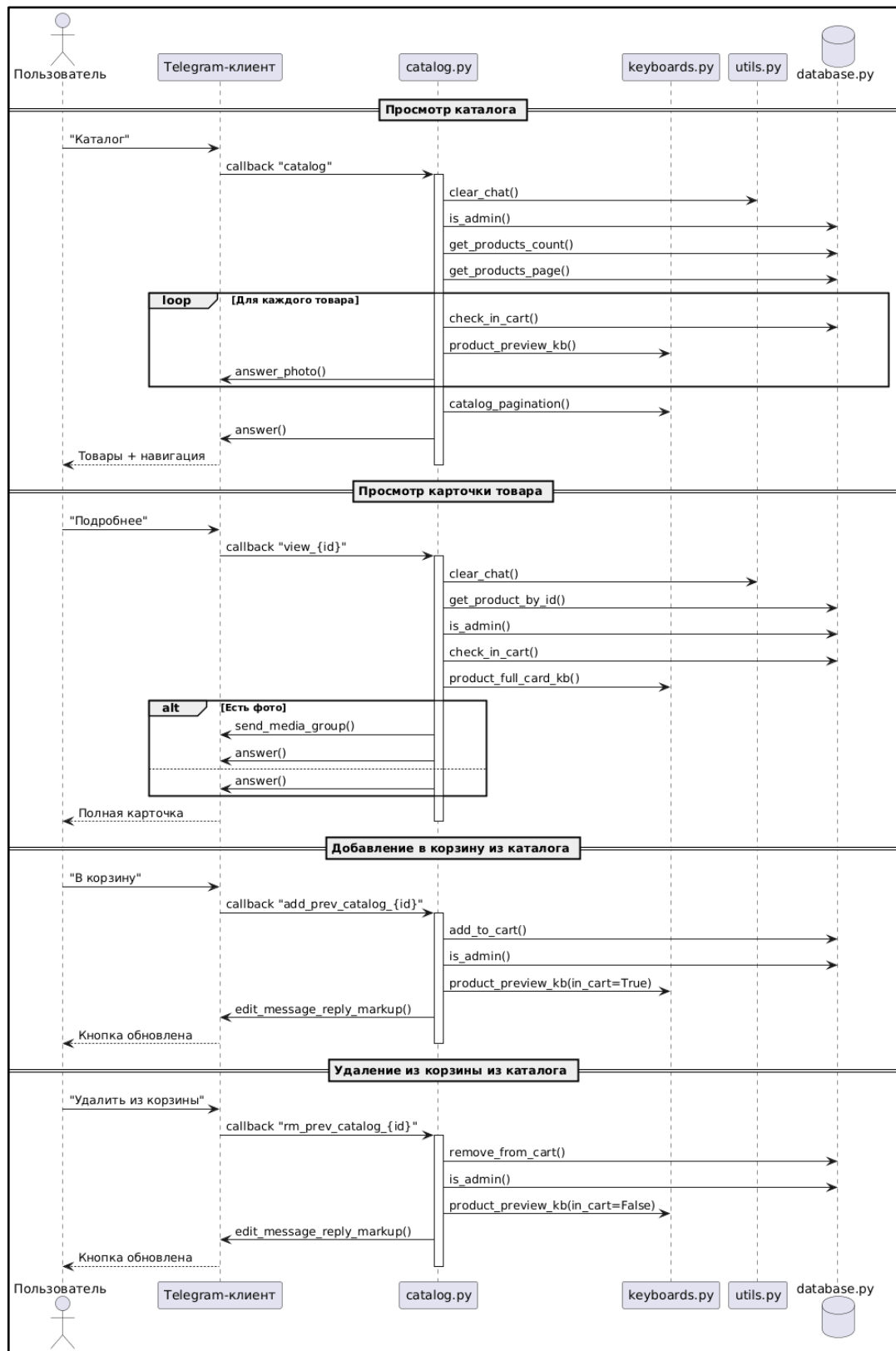


Рисунок 2.17 – UML-диаграмма модуля catalog.py

Модуль `cart.py` среди пользовательских компонентов обладает наибольшей функциональной плотностью. Корзина здесь управляется через конечный автомат FSM, и в состоянии пользователя хранится перечень товаров, отмеченных для оплаты (`selected_items`). Хэндлеры `cart_plus_` и `cart_minus_` изменяют количество единиц товара, причем при увеличении `cart_plus_` сверяет запрашиваемое значение со складскими остатками и предотвращает превышение доступного запаса. Хэндлер `toggle_pay_selection` отвечает за включение товара в список выбранных или его исключение, срабатывая по колбэкам `select_pay_` и `unselect_pay_`, а подпись к сообщению при этом обновляется динамически через `edit_caption`. Применение сквозной перезаписи текстовых и графических блоков через методы редактирования минимизирует количество системного мусора в диалоговом окне мессенджера, обеспечивая высокую эргономику интерфейса.

Оформление заказа проходит через цепочку из трех последовательных хэндлеров, каждый из которых выполняет строго определенную функцию в платежном цикле. Первым срабатывает `process_payment`, проверяющий заполненность профиля пользователя и наличие товаров в корзине, после чего он формирует объекты `LabeledPrice` и отправляет счет через `send_invoice`. Непосредственно перед списанием средств вступает `process_pre_checkout_query`, где корзина повторно сверяется со складскими остатками, и при обнаружении расхождения платеж отклоняется. Заключительный хэндлер `on_success_payment` фиксирует заказ в базе данных, регистрирует позиции, списывает товары со склада и очищает корзину. Весь платежный цикл использует тестовый токен, что позволяет проводить отладку без реальных транзакций. Двухэтапная верификация доступности продукции на складе исключает возникновение состояния гонки (`race condition`) при одновременном оформлении заказа на дефицитную позицию несколькими покупателями. Полная последовательность взаимодействия элементов интерфейса, валидации остатков и прохождения всех этапов платежного цикла наглядно продемонстрирована на рисунке 2.18.

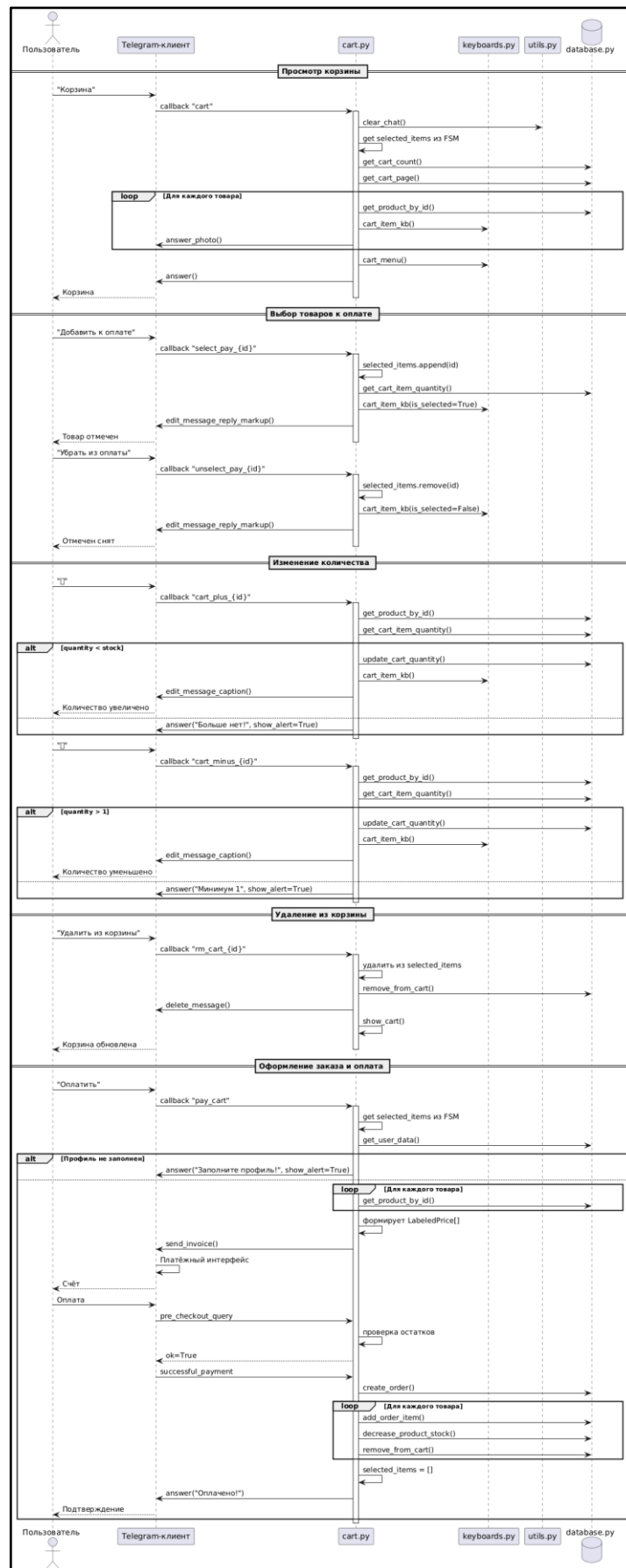


Рисунок 2.18 – UML-диаграмма модуля cart.py

Модуль profile.py управляет личными данными пользователя и задействует конечный автомат для пошагового заполнения профиля, а класс ProfileStates описывает цепочку состояний, которая открывается ожиданием фамилии и завершается ожиданием адреса. Сам ввод адреса разбит на несколько этапов и начинается с состояния waiting_for_delivery_service, где пользователь выбирает службу доставки, таких как СДЭК или Почта России. После этого система показывает кнопку-ссылку на Яндекс.Карты, пользователь переходит по ней, находит подходящий пункт выдачи и копирует адрес в чат, переходя тем самым в состояние waiting_for_address. Телефонный номер проверяется регулярным выражением, что исключает попадание в базу явно ошибочных данных. Полная последовательность пошаговых переходов машины состояний, сценарии валидации ввода и порядок взаимодействия компонентов при настройке личных данных пользователя наглядно отражены на рисунке 2.19. Модуль orders.py управляет просмотром истории заказов и предлагает пользователю два режима отображения: активные заказы, которые еще находятся в обработке или в пути, и завершенные со статусом «доставлен», причем переключение между этими представлениями реализовано через методы get_user_orders_page, фильтрующие записи в базе данных в зависимости от выбранного режима. Пагинация разбивает длинную историю на страницы. Хэндлер show_order_info формирует развернутую карточку конкретного заказа, куда входят состав корзины, цены товаров, адрес доставки и трек-номер для отслеживания отправленных посылок. Интерактивные кнопки под карточкой позволяют пользователю оперативно переключаться между детализацией конкретной покупки и общим списком истории без засорения чата. Алгоритм взаимодействия компонентов системы представлен на рисунке 2.20.

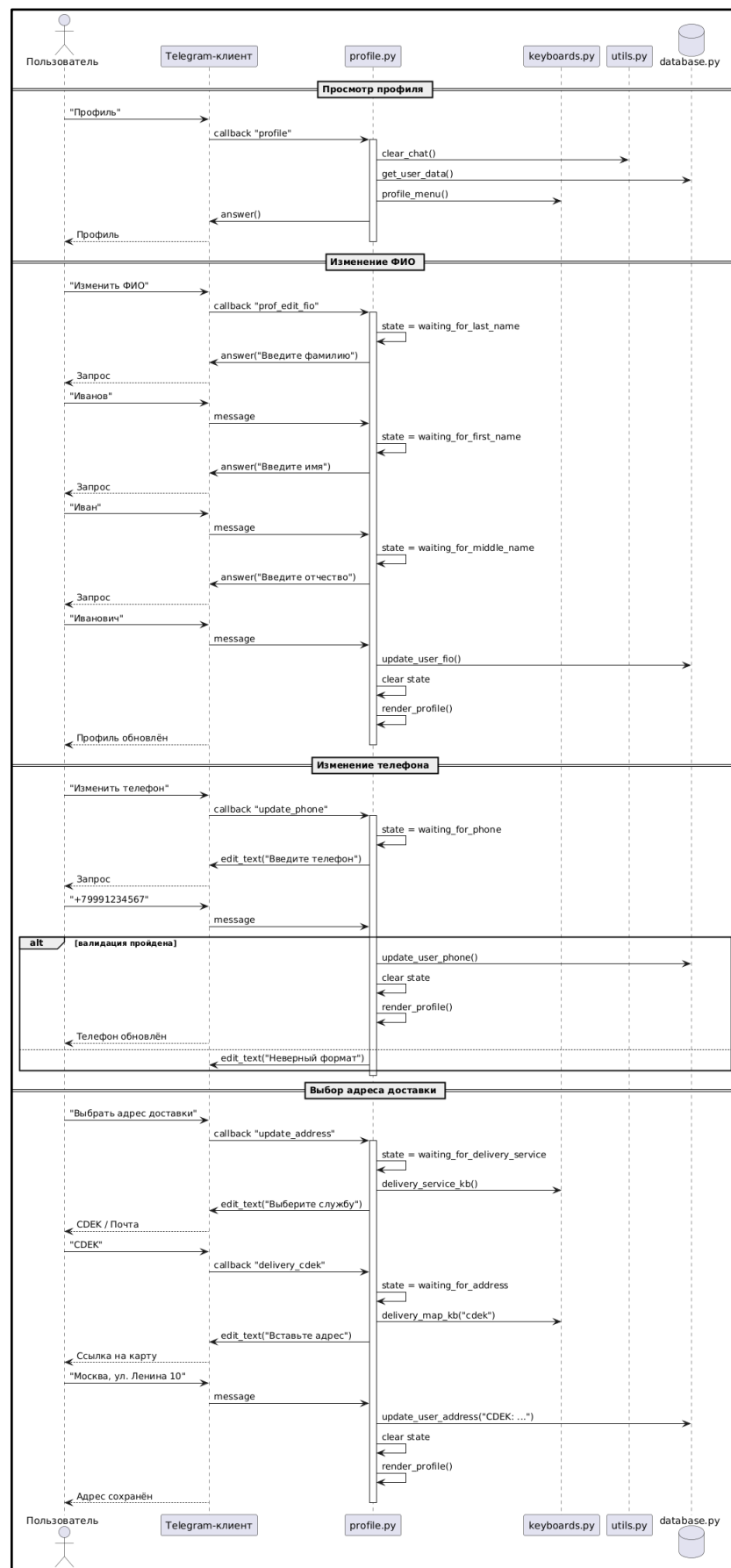


Рисунок 2.19 – UML-диаграмма модуля profile.py

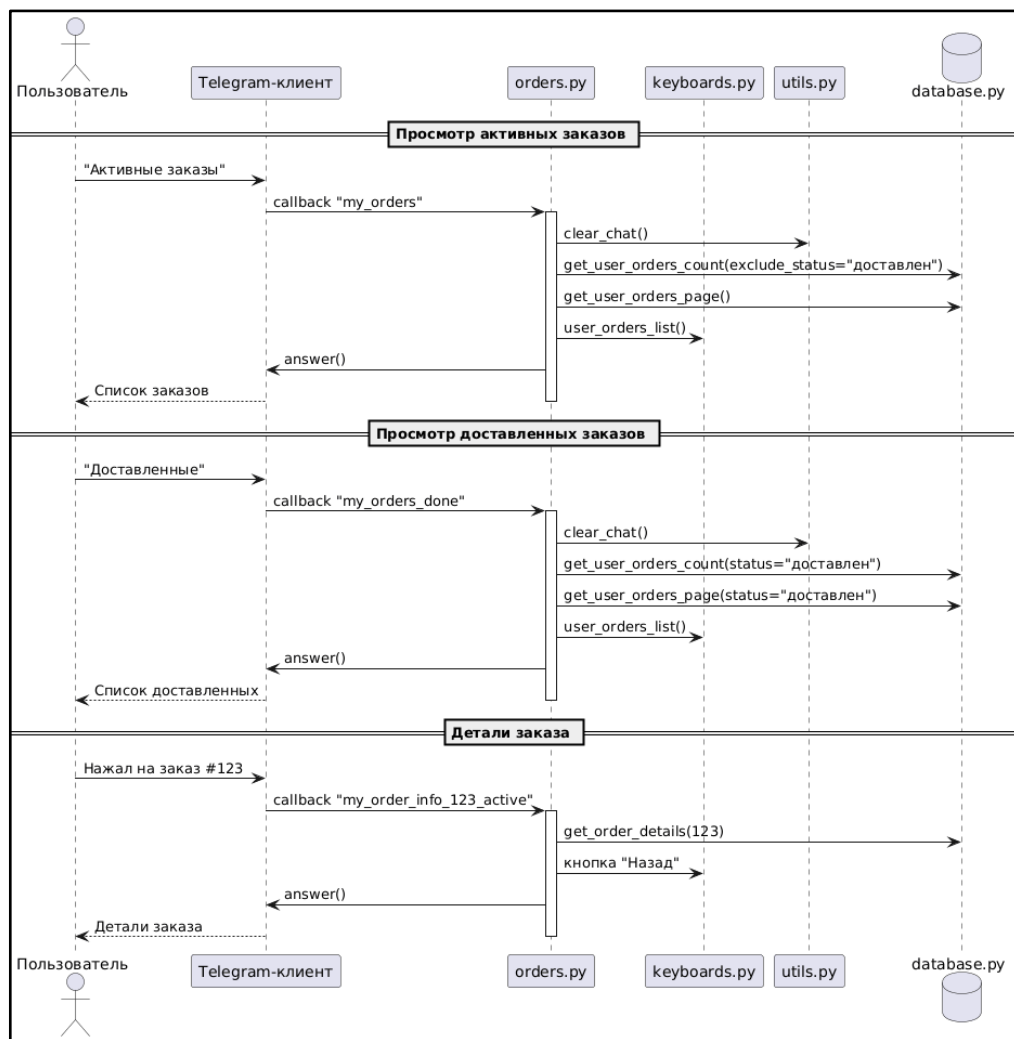


Рисунок 2.20 – UML-диаграмма модуля orders.py

2.3.6 Административные модули

Эта группа модулей формирует панель управления магазином, доступную только пользователям с флагом `is_admin = 1`.

Модуль `admin_main.py` решает две задачи, определяющие его роль в системе. Он сбрасывает незавершенные FSM-состояния администратора и выводит главное меню панели управления, тем самым становясь отправной точкой для навигации по административному интерфейсу. Взаимодействие компонентов при аутентификации администратора, очистке чата и генерации стартовой inline-клавиатуры панели управления отражено на рисунке 2.21.

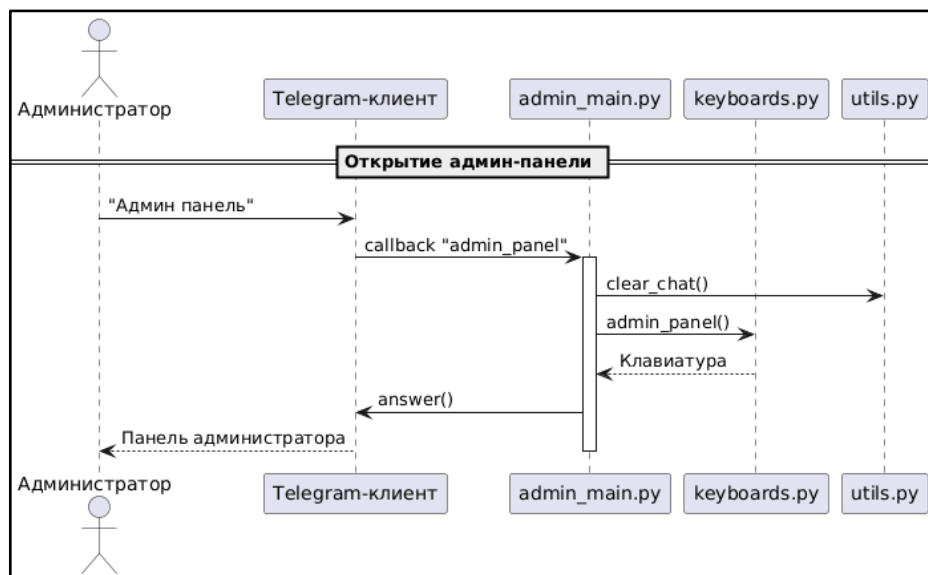


Рисунок 2.21 – UML-диаграмма модуля admin_main.py

Модуль add_product.py реализует нелинейный сценарий взаимодействия на основе конечного автомата, который позволяет администратору возвращаться к предыдущим этапам в любой момент. Класс AddProductStates определяет пять состояний, соответствующих последовательным фазам добавления товара.

Загрузка фотографий реализована сложнее остальных этапов. Хэндлер handle_photos принимает изображения от администратора и записывает их file_id во временный список, привязанный к текущему состоянию FSM. При загрузке каждого снимка система формирует пару идентификаторов, благодаря которой можно при необходимости удалить из чата и пользовательское сообщение, и ответ бота. Хэндлер delete_photo обрабатывает колбэк photo_delete_ – он извлекает индекс конкретного изображения, очищает соответствующие записи из списков и отправляет запросы к Telegram API для удаления связанных сообщений. Навигационные хэндлеры обеспечивают гибкость процесса, позволяя администратору переключиться на любой шаг без потери ранее введенных данных. Полный алгоритм отражен на рисунке 2.22.

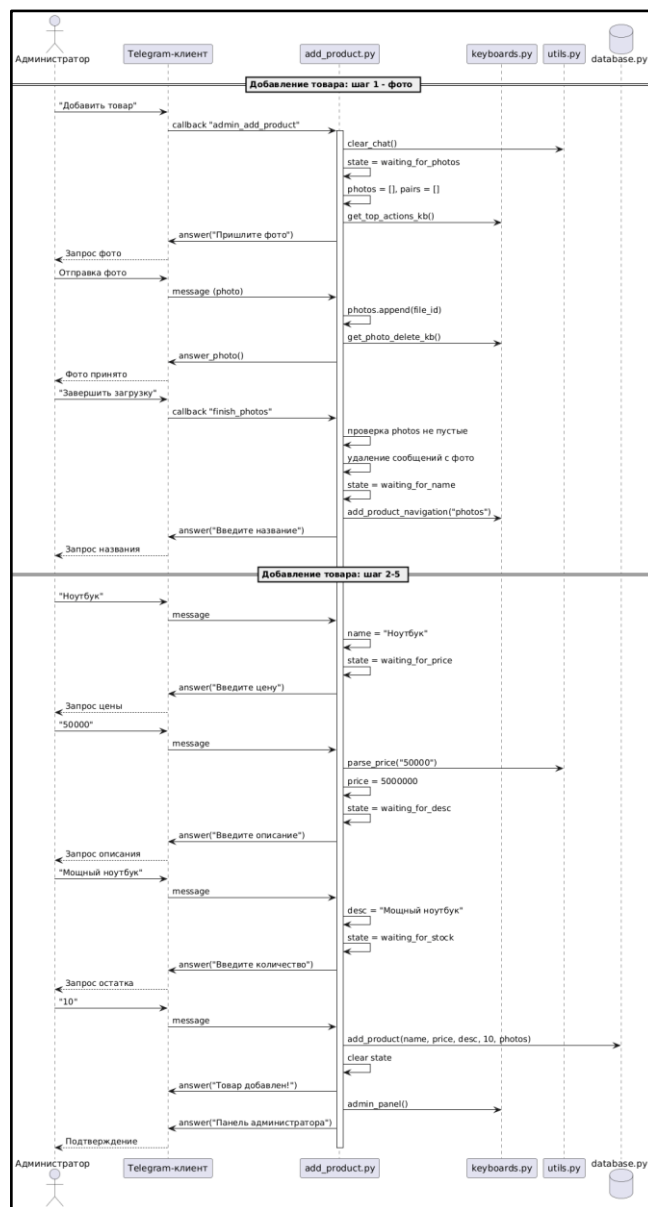


Рисунок 2.22 – UML-диаграмма модуля add_product.py

Весь функционал редактирования товара сосредоточен в модуле edit_product.py, который задействует два независимых FSM-сценария. EditStates.waiting_for_text корректирует текстовые атрибуты такие как: название, описание, цену и остаток, а EditStates.waiting_for_new_photos отвечает за работу с изображениями. Когда администратор переходит в режим редактирования фотографий, система извлекает текущие снимки из базы данных, отображает их вместе с кнопками удаления и позволяет убрать ненужные изображения, загрузить новые, а затем сохранить обновленный набор. Своевременный сброс состояний

конечного автомата после фиксации изменений автоматически возвращает пользователя в корневое меню управления текущей позицией. Перед удалением товара система запрашивает подтверждение действия через `callback confirm_delete_`, и только после согласия администратора изменяет статус записи. Сама запись при этом не стирается физически – вместо удаления выставляется флаг `is_available = 0`. Такой подход мягкого удаления защищает от случайной утраты информации и оставляет возможность восстановить товар в дальнейшем. Пошаговые сценарии изменения текстовых полей, перезаписи пула медиафайлов и алгоритм мягкого удаления позиции наглядно представлены на рисунке 2.23.

Модуль `admin_orders.py` управляет заказами в административной панели, и весь бизнес-процесс обработки строится на последовательной смене четырех статусов: «оформлен», «собран», «отправлен», «доставлен». Метод `admin_order_details` в модуле `AdminKeyboards` контролирует логику переходов, отображая только те кнопки, которые допустимы для текущего состояния заказа. Если заказ находится в статусе «собран», интерфейс предлагает единственную кнопку «Отправить»; при статусе «отправлен» доступны варианты «Завершить» и «Изменить трек-номер». Хэндлер `confirm_status_change` требует обязательного подтверждения при любом изменении статуса, защищая систему от случайных нажатий, а для ввода трек-номера предусмотрено отдельное состояние FSM `AdminOrderStates.waiting_for_track`, обеспечивающее удобный формат работы администратора. Каждая успешная смена этапа мгновенно синхронизируется с базой данных через асинхронные запросы репозитория. Такой строгий контроль переходов полностью исключает риск нарушения логической цепочки движения заказа по этапам обработки. Пошаговые сценарии переключения категорий, верификации изменения статусов и алгоритм фиксации накладных в базе данных подробно продемонстрированы на рисунке 2.24.

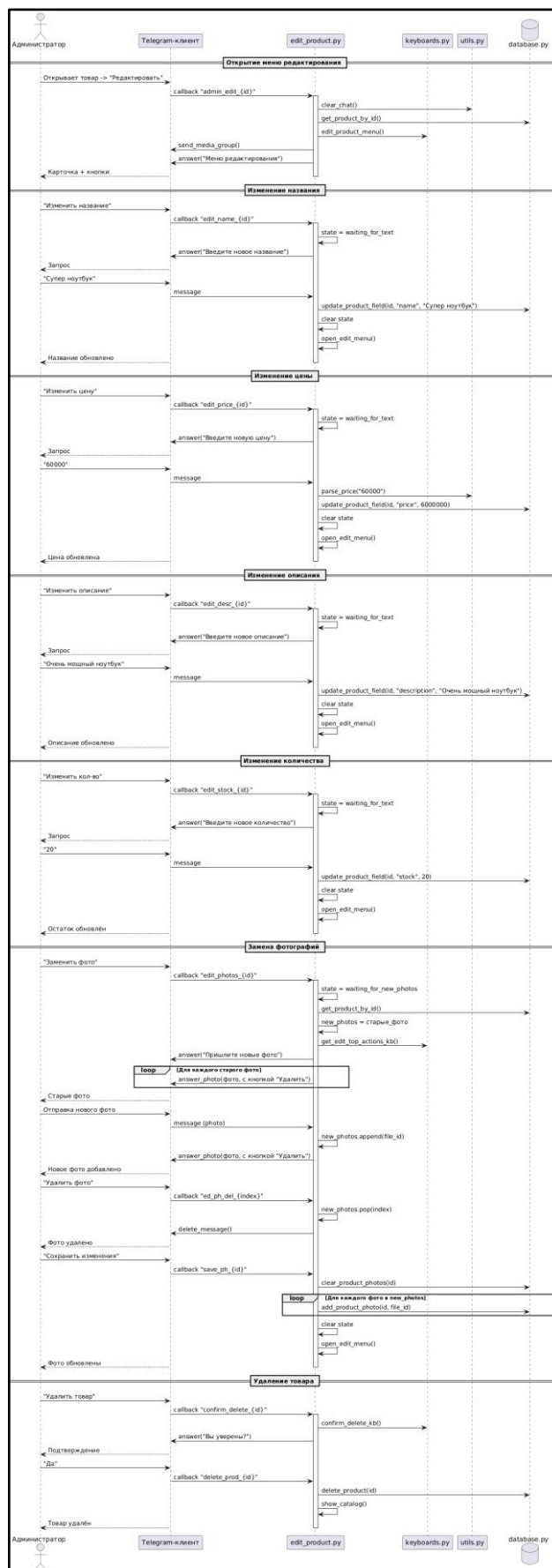


Рисунок 2.23 – UML-диаграмма модуля edit_product.py

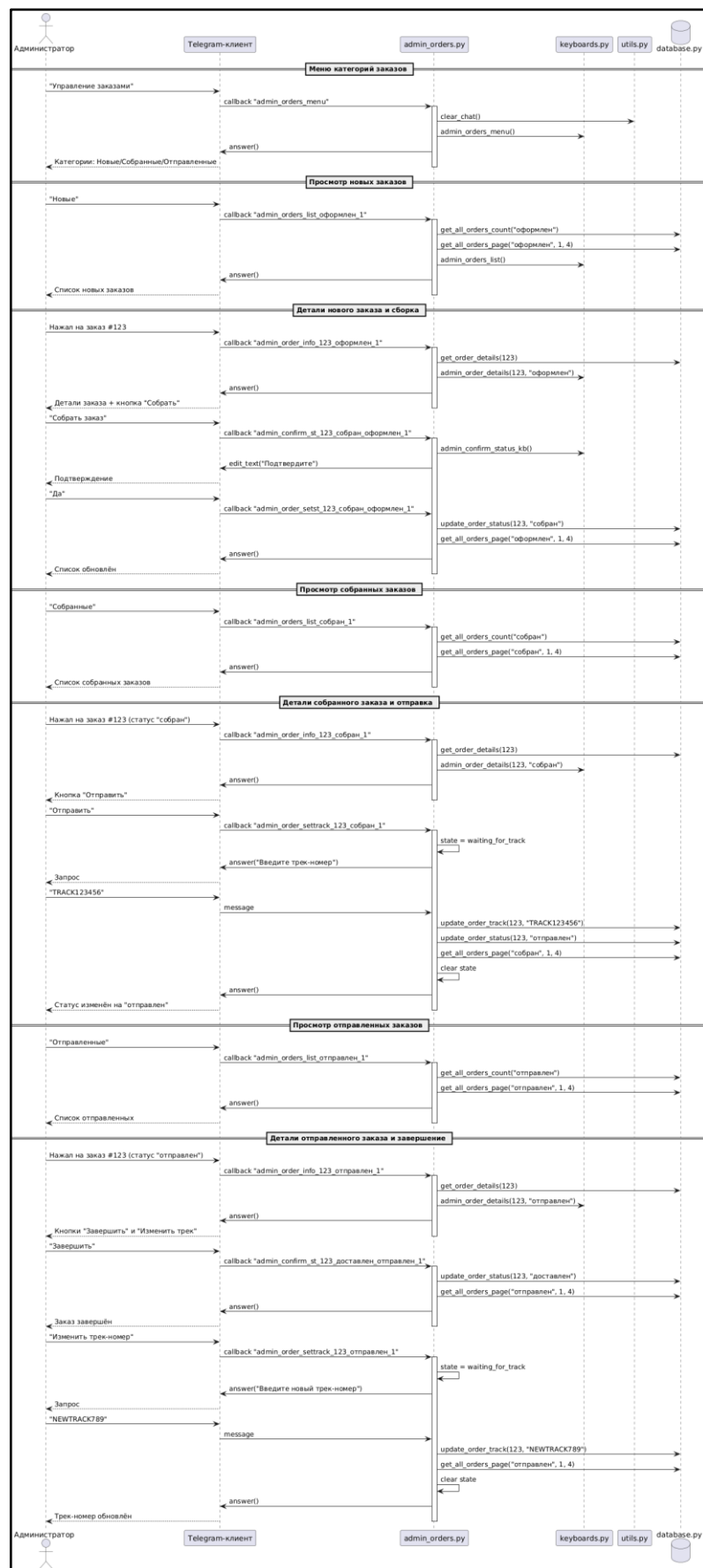


Рисунок 2.24 – UML-диаграмма модуля admin_orders.py

Модуль `add_admin.py` реализует назначение новых администраторов через двухшаговый FSM–процесс, в котором система сначала запрашивает `@username`, затем метод `db.get_user_by_username` ищет пользователя в базе данных, и при успешном обнаружении записи следует запрос на подтверждение операции, исключающий ошибочное назначение прав несуществующему пользователю. Детальная проработка обоих модулей обеспечила корректное функционирование административных сценариев интернет-магазина и сформировала надежную основу для дальнейшего расширения функциональности. Пошаговый алгоритм проверки пользователя по базе, верификации прав и межкомпонентного взаимодействия при добавлении нового администратора наглядно продемонстрирован на рисунке 2.25.

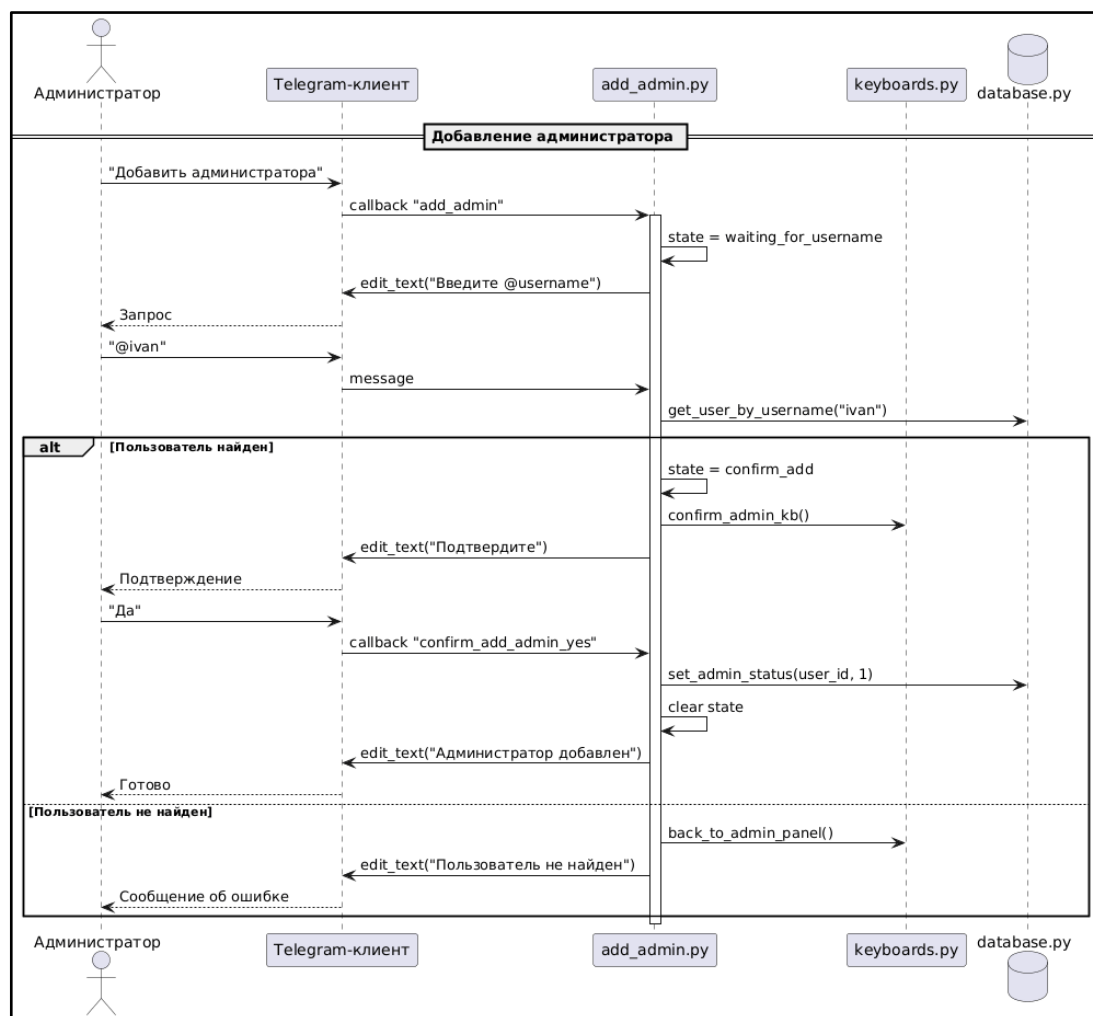


Рисунок 2.25 – UML-диаграмма модуля `add_admin.py`

2.4 Тестирование приложения

После завершения разработки интернет-магазина на основе Telegram-бота было проведено комплексное тестирование всех основных компонентов и функциональных модулей системы. Главной целью испытаний являлась проверка корректности работы пользовательского интерфейса, точности обработки поступающих запросов, надежности взаимодействия с базой данных SQLite и общей устойчивости программного комплекса к ошибочным или нестандартным действиям пользователей.

Проверка разработанного приложения выполнялась вручную (ручное функциональное тестирование) в различных клиентских средах, включая кроссплатформенное приложение Telegram Desktop и мобильное приложение Telegram Android, с привлечением специально подготовленных наборов тестовых данных.

2.4.1 Методика и поэтапный процесс отладки

В процессе разработки отладка программы выполнялась поэтапно. На первом этапе производилась изолированная отладка каждого программного модуля по отдельности, после чего проверялось их межкомпонентное взаимодействие в рамках общей архитектуры проекта. Для поиска и устранения дефектов применялись встроенные инструменты интегрированной среды разработки Visual Studio Code, включая расстановку точек останова (breakpoints), детальное пошаговое выполнение кода и инспектирование текущих значений переменных в оперативной памяти.

В ходе ручной отладки были выявлены и устранены следующие критические проблемы:

- проблемы с асинхронной обработкой параллельных дисковых запросов. Использование стандартных синхронных вызовов к файлу базы данных приводило к кратковременной блокировке главного событийного потока приложения, из-за чего задерживались ответы другим пользователям. Данная

уязвимость была ликвидирована за счет полного перехода на асинхронную библиотеку `aiosqlite` и паттерн «Репозиторий»;

– сбои при обновлении динамических `inline`-клавиатур. При быстром переключении меню возникали ситуации наложения интерфейсов или дублирования элементов. Проблема решена централизацией логики генерации кнопок внутри изолированных фабричных классов и внедрением специализированного промежуточного ПО (`middleware`) для своевременной очистки истории чата;

– ошибки валидации данных в машине состояний (FSM). При прерывании пошагового заполнения профиля или добавлении товара конечный автомат мог зависать в промежуточном состоянии. В систему были добавлены глобальные обработчики команд сброса состояний (например, при нажатии кнопки «Назад» или «Отмена»), возвращающие пользователя в корневое меню;

– погрешности округления при расчете дробных стоимостей. Для исключения математических ошибок при операциях с рублями и копейками все ценовые значения были переведены на хранение исключительно в виде целых чисел копеек, с автоматическим парсингом строк и обратным форматированием при выводе на экран.

2.4.2 Тестирование пользовательской части приложения

Для проверки корректности клиентского интерфейса и валидации пользовательского опыта разработан перечень тест-кейсов, покрывающих сценарии взаимодействия покупателя с интернет-магазином. Проверки охватывают навигацию по каталогу, управление корзиной и оформление заказов, что позволяет выявить логические ошибки и сбои отображения. Полный список тестовых сценариев для пользовательской части представлен в таблице 1.

Таблица 1 – Тестовые сценарии пользовательской части

Сценарий тестирования	Выполняемые шаги	Ожидаемый результат	Фактический результат
Запуск бота и первичная регистрация	1. Отправить команду /start в чате с ботом.	Новый пользователь автоматически заносится в БД, выводится приветственный текст и главное меню. При повторном вызове дубликат не создается.	Соответствует ОР
Навигация по главному меню	1. Нажать на любую активную кнопку в главном меню бота.	Осуществляется переход в выбранный раздел, интерфейс перестраивается, старое меню-сообщение удаляется из чата.	Соответствует ОР
Открытие каталога товаров	1. В главном меню нажать кнопку «Каталог».	Отображаются графические карточки доступных товаров с фото, наименованием и ценой.	Соответствует ОР
Пагинация каталога продукции	1. Нажать кнопку «След.». 2. Нажать кнопку «Пред.».	Происходит корректное циклическое переключение страниц каталога без задержек.	Соответствует ОР
Просмотр полной карточки товара	1. Нажать инлайн-кнопку	Бот выводит расширенное текстовое описание товара,	Соответствует ОР

Сценарий тестирования	Выполняемые шаги	Ожидаемый результат	Фактический результат
	«Подробнее» под товаром.	характеристики и точный складской остаток.	
Добавление товара в корзину	1. Нажать кнопку «В корзину» в карточке.	Выбранный товар успешно добавляется, кнопка динамически меняется на «Удалить из корзины».	Соответствует ОР
Просмотр содержимого корзины	1. Нажать кнопку «Корзина» в меню.	Отображается структурированный список всех добавленных позиций и итоговая сумма.	Соответствует ОР
Изменение количества товара	1. Использовать инлайн-кнопки + и - в корзине.	Количество единиц товара и общая стоимость заказа мгновенно пересчитываются.	Соответствует ОР
Удаление товара из корзины	1. Нажать инлайн-кнопку «Удалить».	Товар полностью стирается из текущей корзины, список позиций обновляется.	Соответствует ОР
Валидация номера телефона	1. Ввести заведомо некорректный формат номера.	Срабатывают регулярные выражения, выводится сообщение об	Соответствует ОР

Сценарий тестирования	Выполняемые шаги	Ожидаемый результат	Фактический результат
		ошибке, система просит повторить ввод.	
Оформление и оплата заказа	1. Заполнить профиль (адрес, контакты). 2. Нажать инлайн-кнопку «Оплатить».	При заполнении профиля формируется чек, при успешной тестовой оплате заказ создается в БД, корзина очищается. В случае нехватки товара на складе операция блокируется.	Соответствует ОР
Просмотр истории заказов	1. Открыть раздел пользовательских заказов.	Корректно отображаются все текущие и завершенные заказы пользователя с их текущими логистическими статусами.	Соответствует ОР

2.4.3 Тестирование административной части приложения

Административный модуль бота проверялся на предмет разграничения прав доступа и корректности управления контентом интернет-магазина. Результаты ручного тестирования административных функций сведены в таблицу 2.

Таблица 2 – Тестовые сценарии административной части

Сценарий тестирования	Выполняемые шаги	Ожидаемый результат	Фактический результат
Вход в панель администратора	1. Нажать кнопку «Админ панель» (доступна только при наличии прав).	Успешно открывается защищенное меню администратора с инструментами управления.	Соответствует ОР
Добавление новой товарной позиции	1. Ввести пошагово текстовые данные и загрузить пул медиафайлов.	Новая номенклатурная единица успешно сохраняется в БД и мгновенно публикуется в каталоге.	Соответствует ОР
Отмена процесса добавления товара	1. На любом шаге FSM нажать кнопку «Отмена».	Пошаговый процесс прерывается, временные данные сбрасываются, бот возвращает в корень панели.	Соответствует ОР
Редактирование наименования товара	1. Выбрать товар и изменить текстовое поле названия.	Строковое значение успешно перезаписывается в таблице базы данных.	Соответствует ОР
Редактирование отпускной цены	1. Ввести новое числовое значение цены.	Цена успешно валидируется, переводится в копейки и обновляется.	Соответствует ОР

Сценарий тестирования	Выполняемые шаги	Ожидаемый результат	Фактический результат
Редактирование складского остатка	1. Изменить количество доступного инвентаря.	Новое количество фиксируется в СУБД, влияя на доступность для покупки.	Соответствует ОР
Удаление товара (мягкое удаление)	1. Подтвердить удаление выбранной позиции.	Товар физически не удаляется из БД, но ему выставляется флаг доступности в ноль, и он скрывается из каталога клиентов.	Соответствует ОР
Просмотр поступающих заказов	1. Открыть служебный раздел администрирования заказов.	Отображается структурированный список всех новых и необработанных оплаченных заказов.	Соответствует ОР
Смена логистического статуса заказа	1. Изменить статус заказа и при необходимости ввести трек-номер.	Статус успешно обновляется в базе данных («собран», «отправлен», «доставлен»).	Соответствует ОР
Назначение нового администратора	1. Ввести @username целевого пользователя.	Происходит проверка юзера по базе данных, после подтверждения пользователь наделяется полными правами администратора.	Соответствует ОР

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

56

2.4.4 Результаты проведенных испытаний

По результатам проведенного комплексного функционального тестирования установлено, что все ключевые модули разработанного интернет-магазина на основе Telegram-бота функционируют корректно и стабильно. Все зафиксированные пользовательские и административные сценарии выполняются в строгом соответствии с ожидаемыми результатами. Ошибки в логике навигации, сбои при обработке некорректного пользовательского ввода или критические транзакционные дефекты полностью отсутствуют.

Полученные в ходе испытаний показатели полностью подтверждают высокую эксплуатационную готовность разработанного программного комплекса, его отказоустойчивость под нагрузкой и пригодность к внедрению в реальные коммерческие процессы малого бизнеса, а также оставляют широкие возможности для последующего масштабирования и наращивания функционала системы без изменения её архитектурного ядра.

					ПГУ1.090301.09.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

3 Руководство пользователя

Разработанное приложение представляет собой Telegram-бота, предоставляющего функционал интернет-магазина. Взаимодействие с пользователем осуществляется через систему кнопок, что делает интерфейс интуитивно понятным и не требующим ввода специальных

3.1 Основные операции для клиента

После запуска бота командой /start пользователь попадает в главное меню. Как это показано на рисунке 3.1.

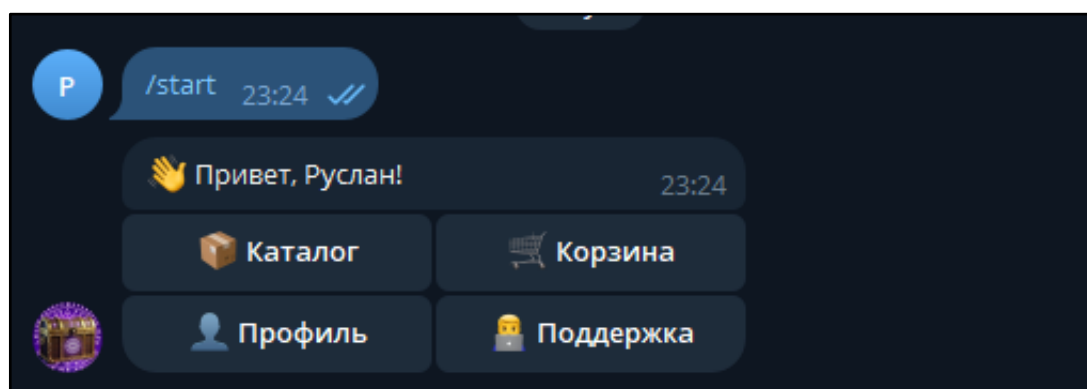


Рисунок 3.1 – Главное меню пользователя

Как видно на рисунке, из главного меню доступны четыре основных раздела: «Каталог», «Корзина», «Профиль» и «Поддержка».

3.1.1 Просмотр каталога товаров

Для просмотра доступных товаров необходимо нажать кнопку «Каталог». На экране отобразится страница с карточками товаров. Если товаров больше четырёх, в нижней части экрана появятся навигационные кнопки «Пред.» и «След.» для перехода между страницами, как это показано на рисунке 3.2. Дополнительно над кнопками переключения отображается текущий номер страницы каталога, что упрощает навигацию по всему доступному ассортименту магазина.

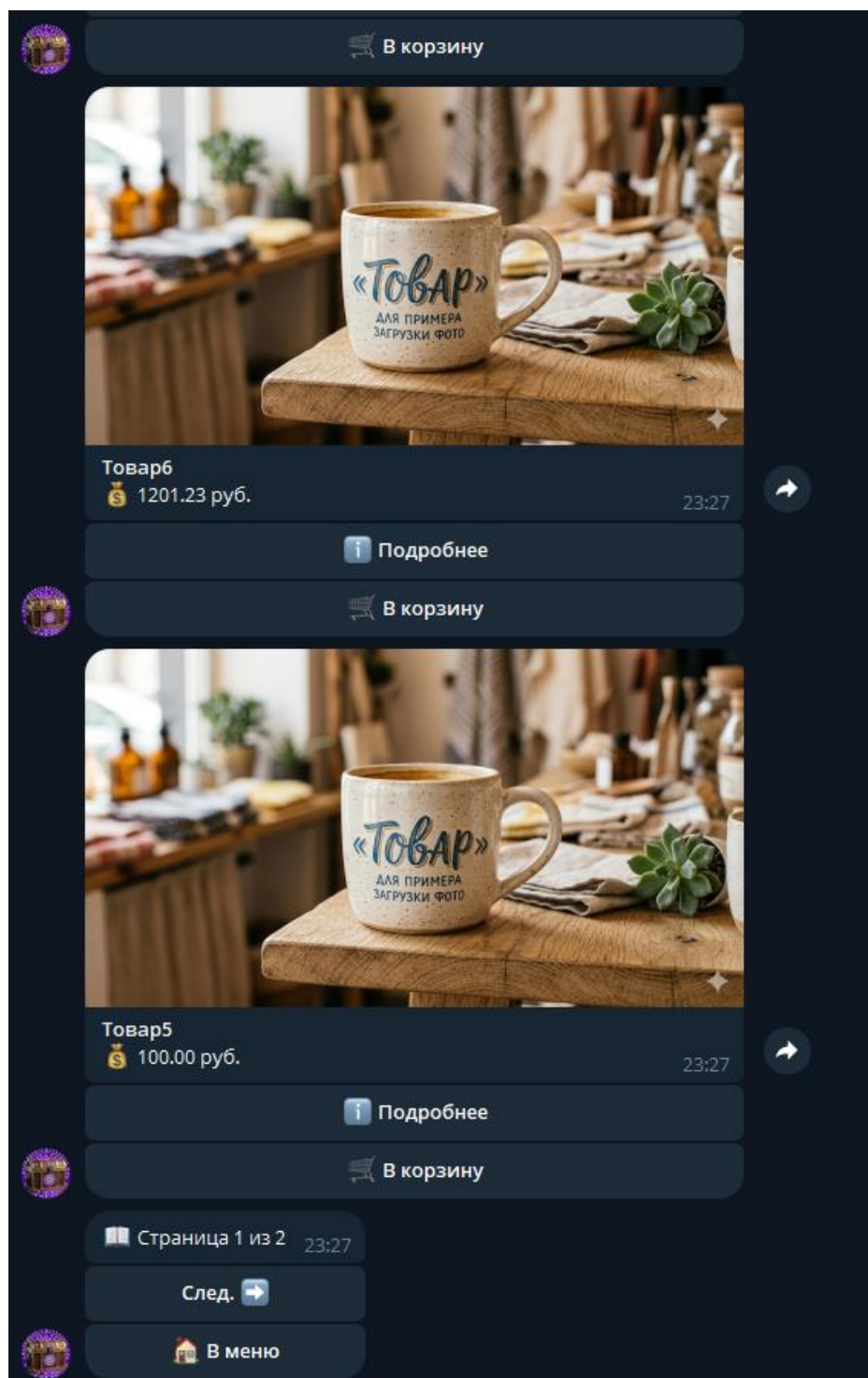


Рисунок 3.2 – Страница каталога с карточками товаров

Управлять товарами можно непосредственно из каталога. Для добавления товара в корзину без перехода в его подробное описание предназначена кнопка «В корзину». Если товар уже был добавлен в корзину ранее, текст кнопки изменится

на «Удалить из корзины», как показано на рисунке 3.3, позволяя убрать позицию из заказа не покидая страницу просмотра товаров.

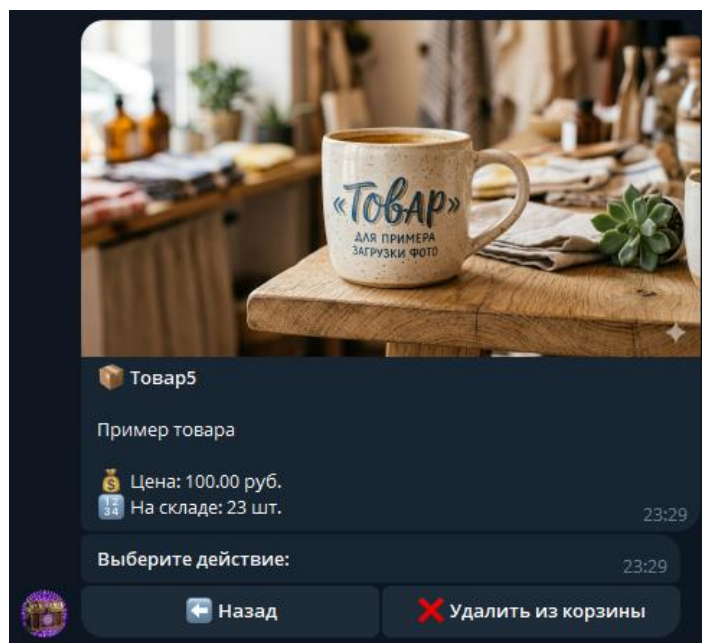


Рисунок 3.3 – Добавление товара в корзину

Для получения подробной информации нужно нажать кнопку «Подробнее». Откроется полная карточка товара, как показано на рисунке 3.4, где помимо названия и цены будет доступно полное описание товара и его остаток на складе.



Рисунок 3.4 – Полная карточка товара

Находясь в полной карточке товара, пользователь может:

- нажать «Назад» для возврата к списку товаров;
- использовать кнопку «В корзину» (или «Удалить из корзины»), которая выполняет ту же функцию, что и в общем каталоге, обеспечивая единообразное поведение интерфейса. После добавления или удаления товара кнопка также динамически меняет свой текст.

3.1.2 Управление профилем

Для указания личной информации и деталей доставки необходимо перейти в раздел «Профиль», как показано на рисунке 3.5.

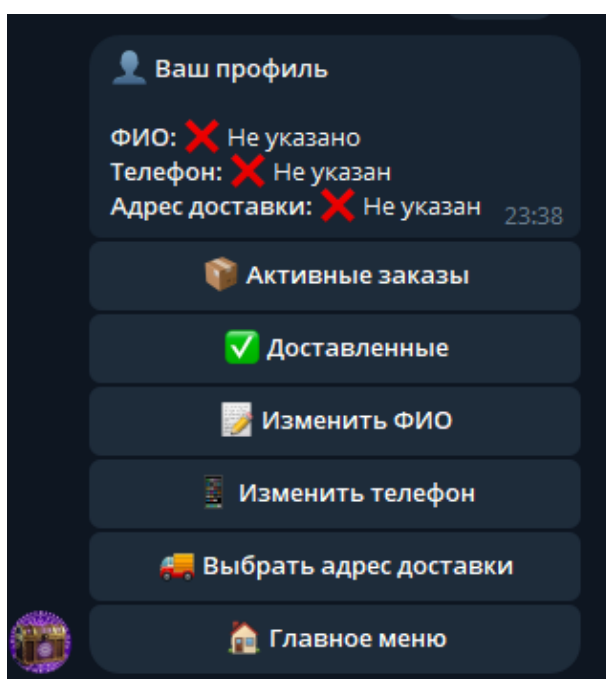


Рисунок 3.5 – Главный экран профиля

Здесь отображаются текущие данные пользователя: ФИО, телефон и адрес доставки. Если поле не заполнено, отображается пометка «Не указан».

Для редактирования данных используются следующие кнопки:

- «Изменить ФИО»: запускает пошаговый процесс ввода фамилии, имени и отчества. Бот последовательно запрашивает каждое поле. Для пропуска отчества предлагается ввести прочерк, как показано на рисунке 3.6;

– «Изменить телефон»: после нажатия нужно ввести номер телефона в ответном сообщении. Бот проводит простую валидацию формата, как показано на рисунке 3.7;

– «Выбрать адрес доставки»: открывает выбор службы доставки (СДЭК или Почта России), как показано на рисунке 3.8. После выбора службы необходимо нажать на кнопку со ссылкой на карту, скопировать адрес нужного пункта выдачи и вставить его в ответном сообщении, как показано на рисунке 3.9. Адрес будет сохранен с префиксом выбранной службы доставки.

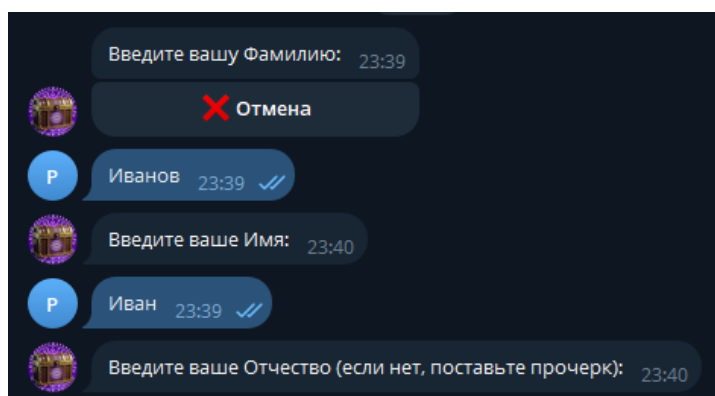


Рисунок 3.6 – Изменение ФИО

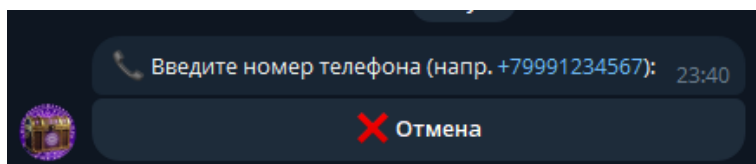


Рисунок 3.7 – Изменение телефона

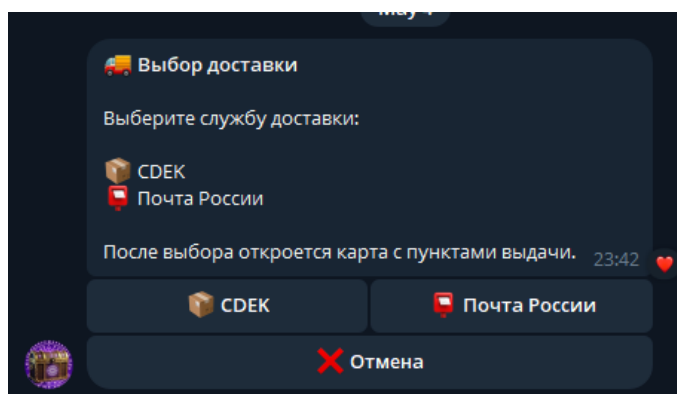


Рисунок 3.8 – Выбор службы доставки

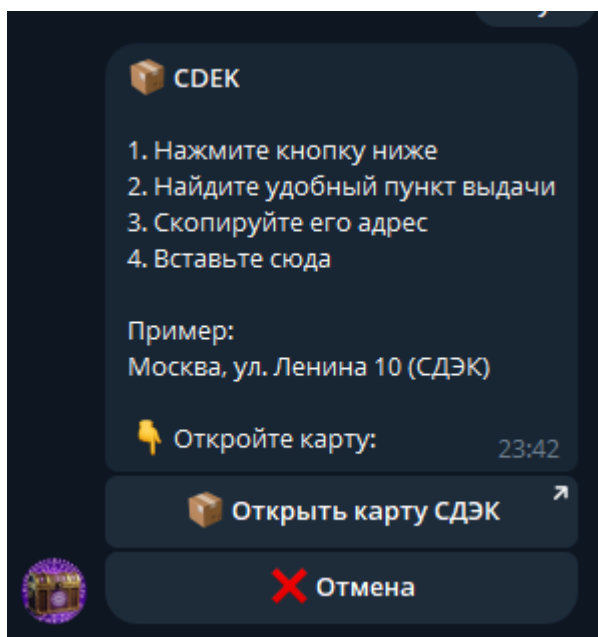


Рисунок 3.9 – Указание адреса доставки

3.1.3 Просмотр истории заказов

Для перехода к содержимому корзины используется кнопка «Корзина» в главном меню. Внутри корзины пользователь может выполнять следующие действия для каждой позиции:

- изменение количества: Кнопки «-» и «+» позволяют уменьшить или увеличить количество единиц конкретного товара. Если количество достигает остатка на складе, дальнейшее увеличение блокируется;
- выбор для оплаты: Кнопка «Добавить к оплате» помечает товар как выбранный. Это позволяет оплатить не всю корзину, а только часть позиций. Для отмены выбора служит кнопка «Убрать из оплаты»;
- удаление: Кнопка «Удалить из корзины» немедленно убирает позицию из корзины;

После того как необходимые товары выбраны, необходимо нажать кнопку «Оплатить», как показано на рисунке 3.10, расположенную в нижнем меню корзины. Если пользователь выбрал товары, но не заполнил полностью свой профиль, бот выдаст предупреждение и остановит операцию.

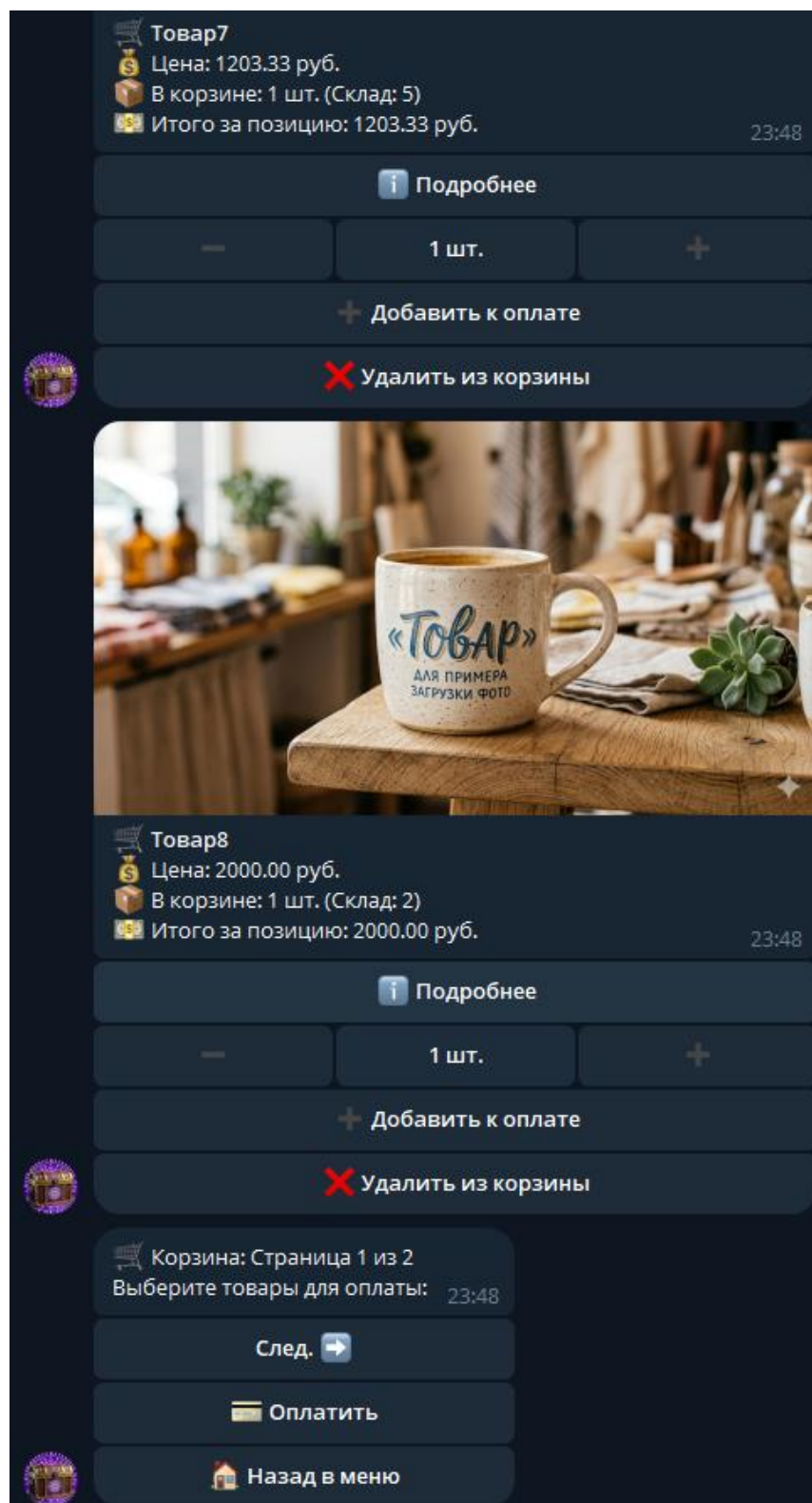


Рисунок 3.10 – Указание адреса доставки

После будет отображена общая сумма оплаты товаров, где надо будет подтвердить оплату нажав на кнопку «Заплатит». При успешной проверке система

Telegram сформирует счет на оплату, где будет указана общая стоимость выбранных товаров, как показано на рисунке 3.11.

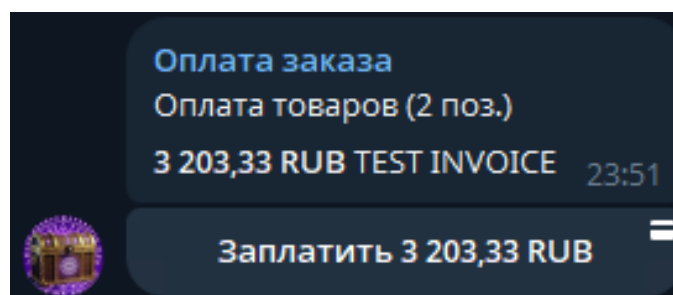


Рисунок 3.11 – Счет на оплату

При успешной проверке система Telegram сформирует счет на оплату, как показано на рисунке 3.12.

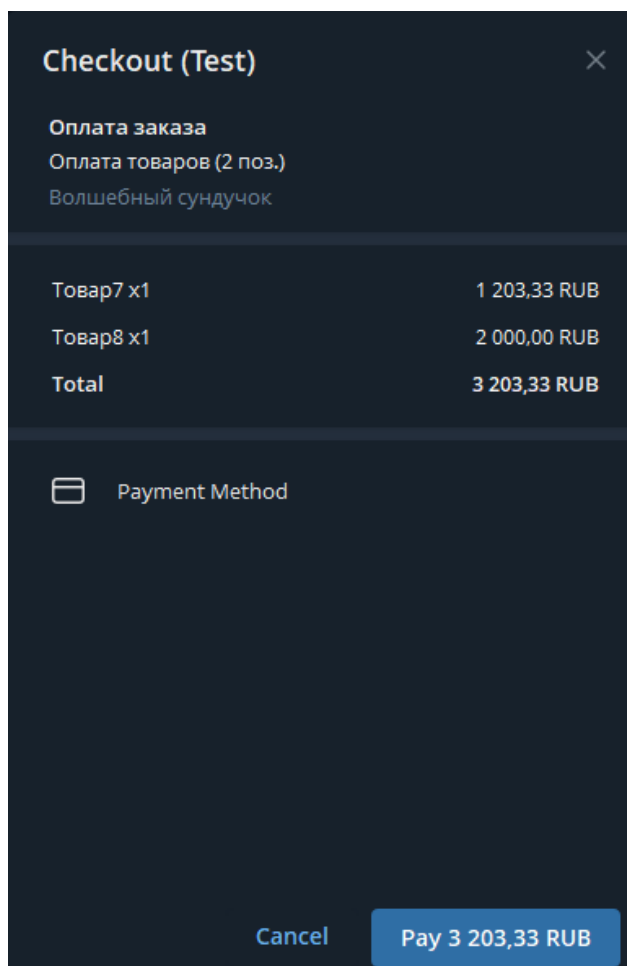


Рисунок 3.12 – Оплата заказа

После подтверждения оплаты бот создает заказ, уменьшает остатки товаров на складе и очищает корзину. Пользователю выводится сообщение с итоговой суммой, как показано на рисунке 3.13.

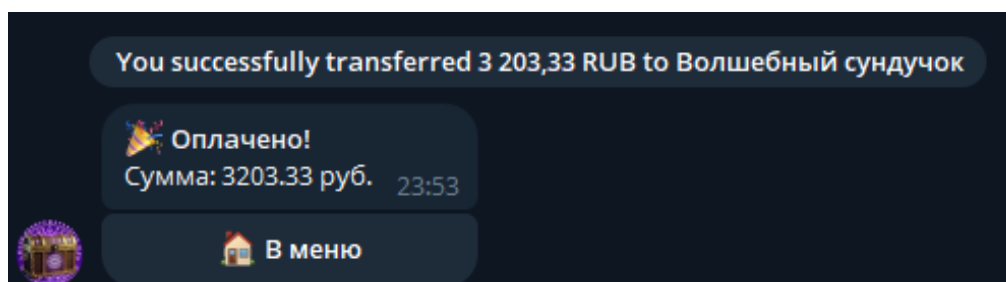


Рисунок 3.13 – Товар оплачен

3.2 Основные операции для администратора

Если пользователь обладает правами администратора, после команды /start его главное меню будет содержать кнопки «Каталог» и «Админ панель», как продемонстрировано на рисунке 3.14.

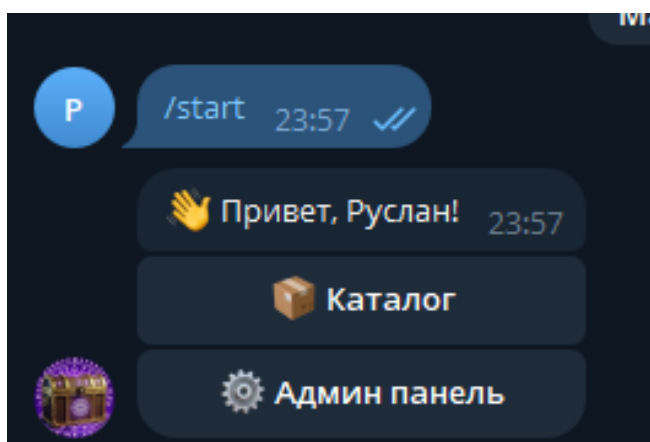


Рисунок 3.14 – Главное меню администратора

3.3 Управление товарными позициями

В админ-панели администратору доступны функции для изменения ассортимента магазина, как продемонстрировано на рисунке 3.15.

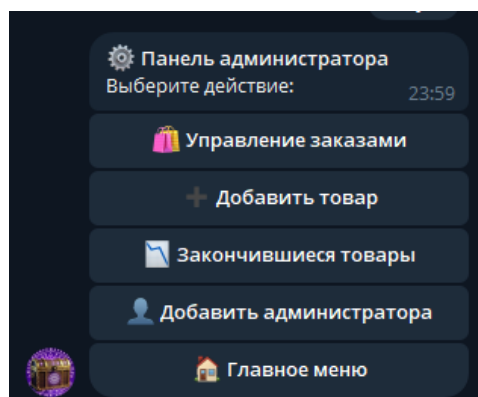


Рисунок 3.15 – Панель администратора

Добавление нового товара: после нажатия «Добавить товар» запускается многошаговое добавление нового товара. В данном меню можно загрузить несколько фото, а также удалить ошибочно загруженные до завершения шага нажатием кнопки «Удалить» под каждой фотографией. Для перехода к следующему шагу необходимо нажать «Завершить загрузку», как показано на рисунке 3.16;



Рисунок 3.16 – Загрузка изображения

На следующем этапе система запрашивает наименование позиции, которое вводится текстовым сообщением и фиксируется в качестве заголовка будущей карточки, как показано на рисунке 3.17;

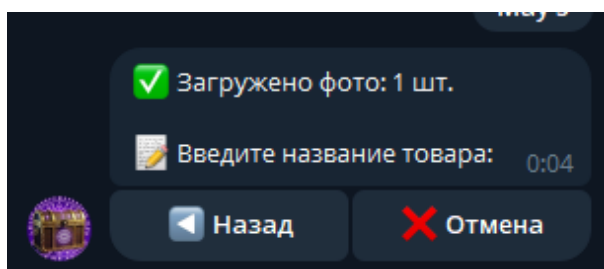


Рисунок 3.17 – Загрузка изображения

Далее надо указать стоимость товарной позиции вводится в рублях, как показано на рисунке 3.18;

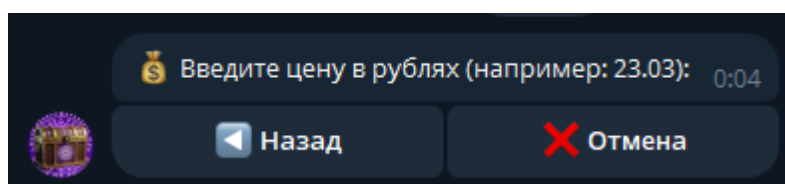


Рисунок 3.18 – Ввод цены товара

После этого необходимо добавить описные товара, раскрывающие особенности и параметры продукции для покупателя, которое вводятся обычным текстовым сообщением, как показано на рисунке 3.19;

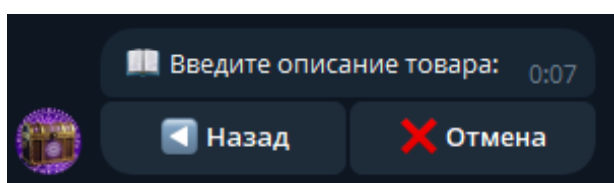


Рисунок 3.19 – Ввод описания товара

Финальным параметром которое необходимо указать является остаток на складе, который вводится целым числом, как показано на рисунке 3.20.

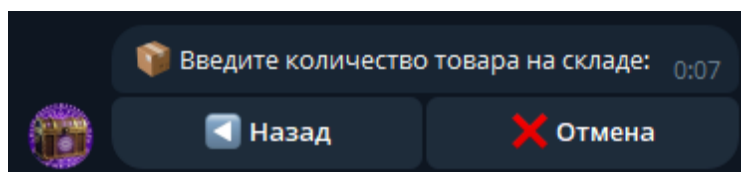


Рисунок 3.20 – Ввод остатка на складе товара

На любом этапе после ввода фотографий можно вернуться на предыдущий шаг с помощью кнопки «Назад» или полностью отменить процесс, нажав «Отмена». После ввода остатка товар окончательно сохраняется в базу данных, как отображено на рисунке 3.21.

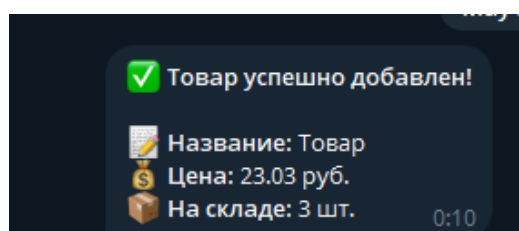


Рисунок 3.21 – Товар добавлен

Редактирование товара: при просмотре каталога (кнопка «Каталог») у администратора под каждой карточкой есть кнопка «Редактировать». Также раздел «закончившиеся товары» показывает позиции с нулевым остатком, где тоже доступно редактирование. При нажатии открывается меню управления товаром, как показано на рисунке 3.22:

- «Заменить фото»: запускает процесс замены всего набора фотографий. Старые фото удаляются, на их место загружаются новые;
- «Изменить название» / «Изменить описание» / «Изменить цену» / «Изменить кол-во»: бот запрашивает новое значение и после проверки сохраняет его в базе;
- «Удалить товар»: выполняет мягкое удаление (архивацию). Товар пропадает из каталога клиентов, но физически остается в базе данных;

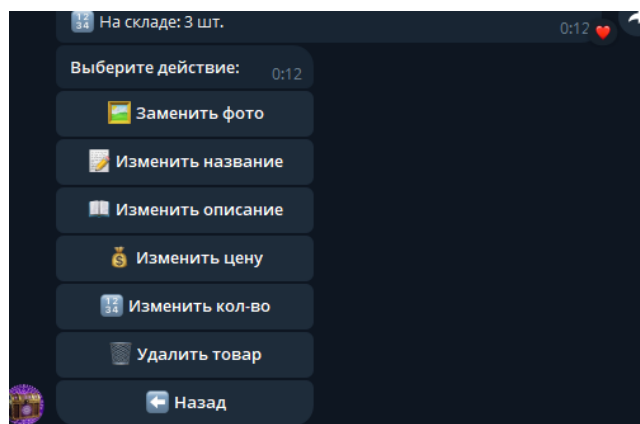


Рисунок 3.22 – Меню редактирования товара

3.3.1 Управление заказами

Для работы с заказами в админ-панели необходимо нажать «Управление заказами». Откроется меню, где заказы сгруппированы по статусам: «Новые», «Собранные», «Отправленные», как показано на рисунке 3.23.

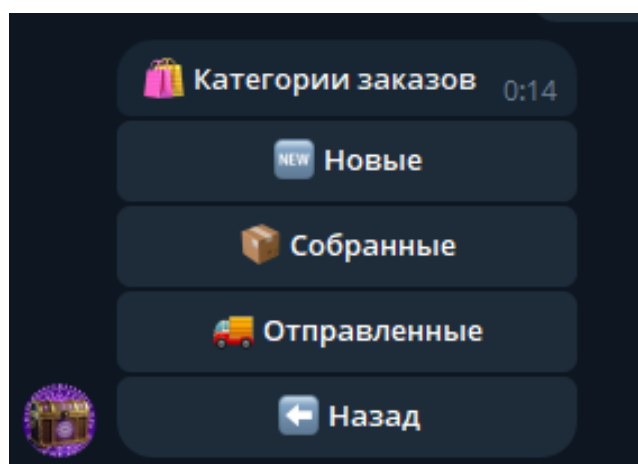


Рисунок 3.23 – Меню категорий заказов

При переходе в любую категорию администратор видит список заказов с возможностью пагинации. Нажав на нужный заказ, администратор попадает в его карточку, где видит ФИО покупателя, телефон, адрес и состав. В зависимости от текущего статуса заказа ему доступны следующие действия:

– для новых заказов (статус «оформлен»): кнопка «Собрать заказ».

Переводит заказ в статус «Собран», как показано на рисунке 3.24;

– для собранных заказов: кнопка «Отправить (ввести трек)», как показано на рисунке 3.25. После нажатия бот ожидает ввода трек-номера, который присваивается заказу с одновременной сменой статуса на «Отправлен», как показано на рисунке 3.26;

– для отправленных заказов: доступна кнопка «Завершить (Доставлен)» для закрытия заказа, и кнопка «Изменить трек-номер» для исправления в уже введенном номере отслеживания, как показано на рисунке 3.27.

Все операции по смене статуса требуют дополнительного подтверждения во избежание случайных нажатий.

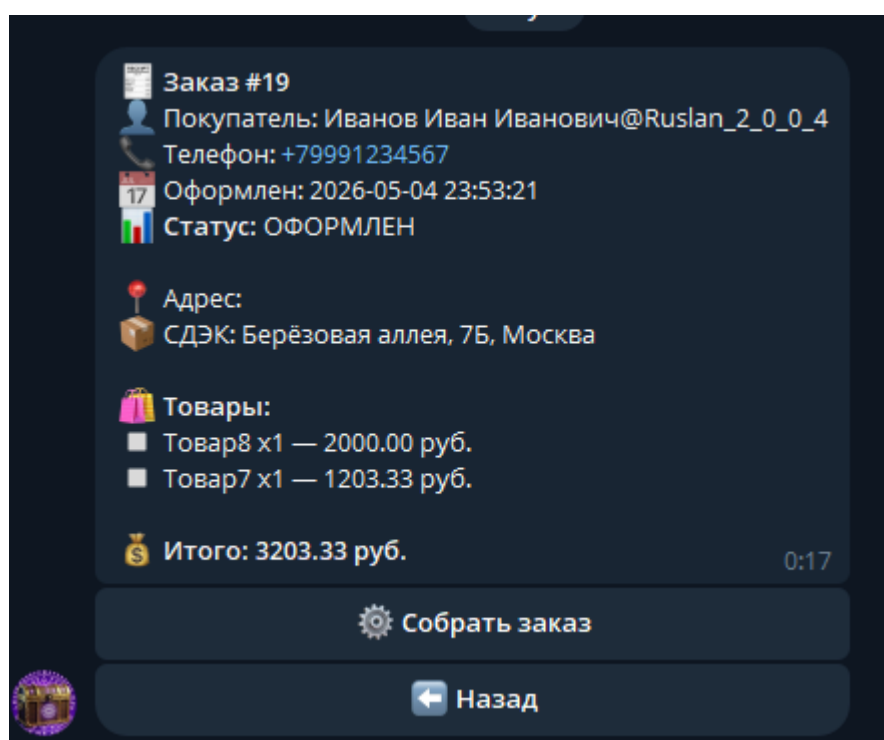


Рисунок 3.24 – Новый заказ

Заказ #19
 Покупатель: Иванов Иван Иванович@Ruslan_2_0_0_4
 Телефон: +79991234567
 Оформлен: 2026-05-04 23:53:21
 Статус: СОБРАН

Адрес:
 СДЭК: Берёзовая аллея, 7Б, Москва

Товары:
 ■ Товар8 x1 — 2000.00 руб.
 ■ Товар7 x1 — 1203.33 руб.

Итого: 3203.33 руб. 0:18

Отправить (ввести трек)

Назад

Рисунок 3.25 – Собранный заказ

Введите трек-номер для заказа #19: 0:18

Рисунок 3.26 – Ввод токена для собранных заказов

Заказ #19
 Покупатель: Иванов Иван Иванович@Ruslan_2_0_0_4
 Телефон: +79991234567
 Оформлен: 2026-05-04 23:53:21
 Статус: ОТПРАВЛЕН
 Трек-номер: 123

Адрес:
 СДЭК: Берёзовая аллея, 7Б, Москва

Товары:
 ■ Товар8 x1 — 2000.00 руб.
 ■ Товар7 x1 — 1203.33 руб.

Итого: 3203.33 руб. 0:20

Завершить (Доставлен)

Изменить трек-номер

Назад

Рисунок 3.27 – Ввод токена для собранных заказов

3.3.2 Назначение новых администраторов

Для того чтобы выдать права администратора другому пользователю, текущий администратор должен нажать кнопку «Добавить администратора», ввести @username интересующего его человека в ответном сообщении и подтвердить действие в появившемся диалоге.

					ПГУ1.090301.09.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

4 Анализ результатов

4.1 Результаты разработки интернет-магазина на базе Telegram-бота

В результате выполнения данного технического задания было реализовано приложение магазина на базе Telegram-бота с модульной асинхронной архитектурой. Программный комплекс написан на языке программирования Python с использованием фреймворка aiogram и встраиваемой СУБД SQLite.

В созданном приложении пользователи и администраторы получили полный набор инструментов для автоматизации коммерческих процессов и взаимодействия внутри мессенджера. Покупателям доступны постраничная навигация по каталогу товаров, динамическое управление содержимым корзины, пошаговое заполнение профиля с валидацией номера телефона и интеграция с внешними картами для выбора адреса доставки. Также реализованы безопасное проведение онлайн-оплаты через встроенный платежный интерфейс и просмотр истории заказов с разделением на активные и завершенные позиции. Полный спектр доступных клиенту сценариев и прецедентов взаимодействия наглядно отражает диаграмма вариантов использования для пользователя, представленная на рисунке 4.1. Для управления платформой в бот интегрирована расширенная административная панель, позволяющая добавлять товары с загрузкой пула медиафайлов, редактировать параметры, применять механизм мягкого удаления позиций, пошагово изменять статусы заказов с вводом трек-номеров и гибко назначать права новым модераторам. Все доступные административные функции и границы полномочий управляющего персонала зафиксированы на диаграмме вариантов использования для администратора на рисунке 4.2.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

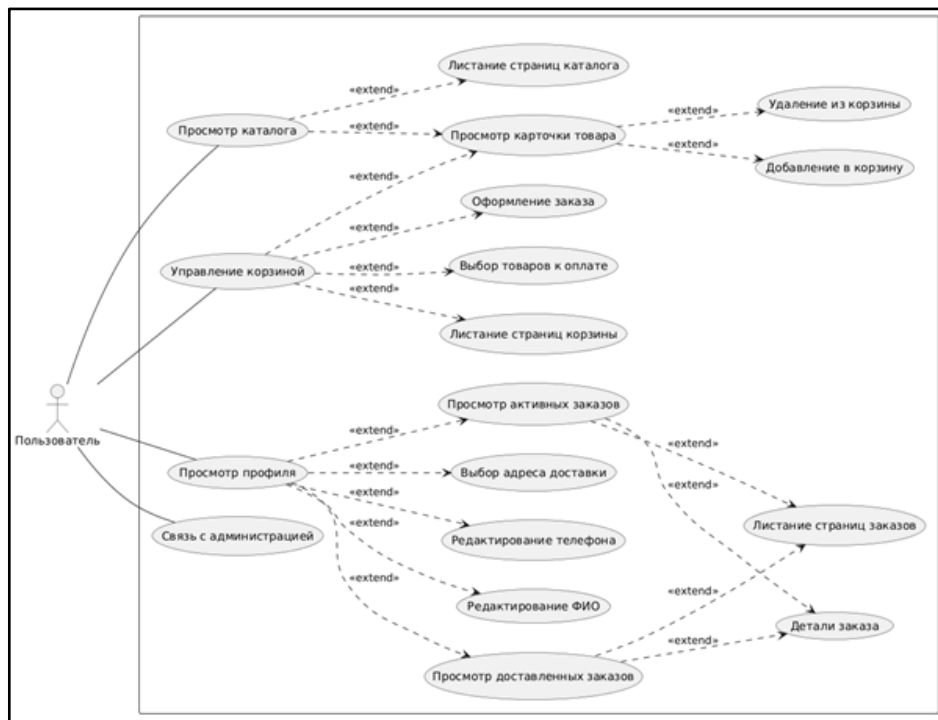


Рисунок 4.1 – UML-диаграмма прецедентов пользователя

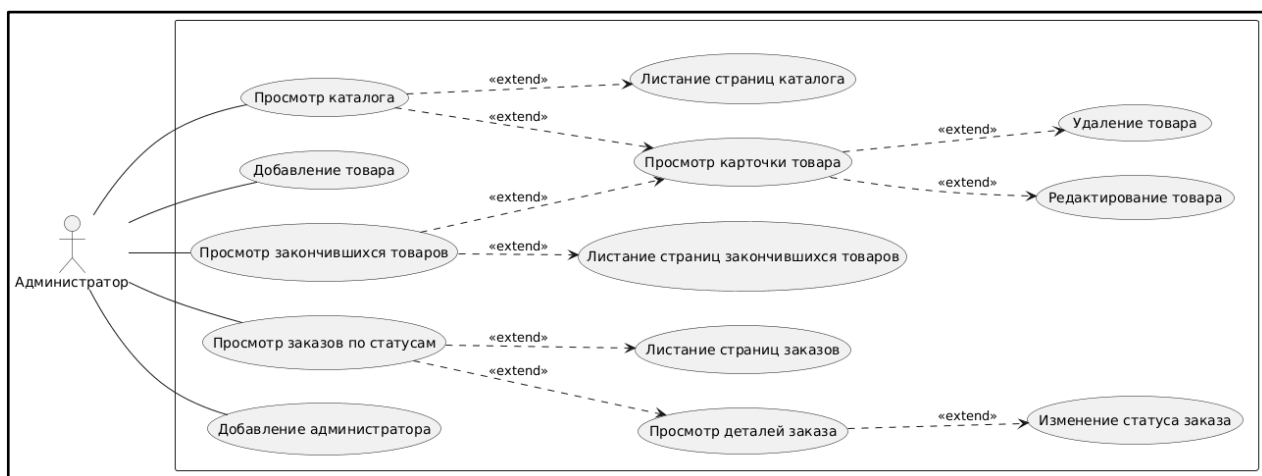


Рисунок 4.2 – UML-диаграмма прецедентов администратора

Реализованного функционала полностью достаточно для развертывания готовой торговой площадки малого бизнеса, что позволяет предпринимателям эффективно заменить традиционные веб-ресурсы и избежать жестких условий крупных маркетплейсов. Выбранные асинхронные компоненты, машина состояний и параметризованные запросы библиотеки `aiosqlite` обеспечивают высокую отказоустойчивость системы под нагрузкой, безопасность данных от

SQL-инъекций и бесперебойную параллельную обработку сотен клиентских сессий. Спроектированная структура позволяет в дальнейшем легко масштабировать проект и наращивать коммерческий функционал без переработки ядра системы.

4.2 Рекомендации к дальнейшей разработке

На текущем этапе разработанное программное обеспечение полностью соответствует всем требованиям технического задания, демонстрирует стабильную отказоустойчивость и в полной мере обеспечивает автоматизацию базовых коммерческих процессов интернет-магазина. Тем не менее, для последующего масштабирования платформы и ее адаптации под условия высокой рыночной конкуренции проект имеет существенный потенциал для дальнейшей модернизации.

В качестве приоритетных направлений развития системы рекомендуется рассмотреть переход со встроенной СУБД SQLite на централизованную реляционную базу данных PostgreSQL. Это позволит обеспечить поддержку высокой горизонтальной масштабируемости, повысить безопасность хранения информации при многопоточном доступе и ускорить обработку сложных аналитических запросов.

Также целесообразно интегрировать полноценный графический веб-интерфейс администратора на базе технологии Telegram Mini Apps, что позволит владельцу бизнеса оперативно отслеживать сквозную аналитику продаж и управлять номенклатурой товаров вне диалогового окна мессенджера. Дополнительно логичным шагом станет автоматизация логистического модуля посредством прямого подключения к API курьерских служб, что полностью исключит необходимость ручного ввода трек-номеров администратором и автоматизирует расчет стоимости доставки в реальном времени.

Реализация предложенных рекомендаций точно улучшат приложение, повысив его конкурентоспособность и общую коммерческую эффективность.

					ПГУ1.090301.09.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

5 Безопасность жизнедеятельности

Разработка и отладка программного комплекса интернет-магазина на базе Telegram-бота осуществлялись в условиях офисного или домашнего рабочего места разработчика, оборудованного персональным компьютером. Основными вредными и опасными факторами на рабочем месте программиста являются повышенное зрительное напряжение, длительная гиподинамия, электромагнитные излучения и опасность поражения электрическим током.

5.1 Эргономика и организация рабочего места

Организация рабочего пространства выполнена в соответствии с требованиями ГОСТ 12.2.032-78 «Рабочее место при выполнении работ сидя» [5]. Высота рабочей поверхности стола и параметры кресла отрегулированы под антропометрические данные разработчика для снижения статической нагрузки на опорно-двигательный аппарат. Монитор расположен на расстоянии 600–700 мм от глаз, а угол направления взгляда составляет 15-20° ниже горизонтали, что соответствует эргономическим требованиям ГОСТ Р ИСО 9241-303-2012 к электронным видеодисплеям и минимизирует утомляемость зрительного анализатора.

5.2 Микроклимат и освещение

Параметры микроклимата и уровня освещенности на рабочем месте поддерживались в пределах оптимальных значений, регламентированных СанПиН 1.2.3685-21 «Гигиенические нормативы и требования к обеспечению безопасности и/или безвредности для человека факторов среды обитания». Температура воздуха в помещении в холодный период года составляла 22–24 °С при относительной влажности 40–60%. Естественное и искусственное освещение спроектировано таким образом, чтобы освещенность на поверхности стола в зоне размещения документов составляла не менее 300–500 лк. Для исключения прямой

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

и отраженной блескости монитора применены антибликовые покрытия и солнцезащитные жалюзи на окнах.

5.3 Электробезопасность

Поскольку используемая вычислительная техника питается от сети переменного тока напряжением 220 В, существует риск поражения электрическим током при повреждении изоляции токоведущих частей. В соответствии с ГОСТ 12.1.009-2017 «Электробезопасность. Термины и определения», на рабочем месте реализованы следующие защитные меры:

- все корпуса ПЭВМ и периферийного оборудования имеют защитное заземление;
- подключение оборудования выполнено через сертифицированные сетевые фильтры с автоматической защитой от перегрузок;
- целостность изоляции питающих кабелей проверяется визуально перед каждым включением техники в соответствии с требованиями ГОСТ 12.1.038-82.

5.4 Пожарная безопасность

Помещение с вычислительной техникой относится к категории В из-за наличия твердых горючих материалов: пластик, изоляция проводов, бумага. Пожарная безопасность организована в строгом соответствии с ГОСТ 12.1.004-91 «Пожарная безопасность. Общие требования». Помещение укомплектовано первичными средствами пожаротушения – углекислотным огнетушителем класса ОУ-2, предназначенным для тушения электроустановок под напряжением до 1000 В. Пути эвакуации содержатся свободными, а схема эвакуации размещена на видном месте. Путем соблюдения регламентированных перерывов обеспечивается поддержание высокой работоспособности и сохранение здоровья разработчика на протяжении всего жизненного цикла создания программного продукта [6].

					ПГУ1.090301.09.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

6 Стартап проект

Основатель инициативы намерен создать интернет-магазин на базе Telegram-бота. Проект разрабатывается в Пензе. Цель деятельности предприятия – извлечение прибыли от продаж для улучшения материального благосостояния создателя. Также задача состоит в автоматизации рутины и обеспечении простого, безопасного взаимодействия между магазином и клиентом. Нужно предоставить покупателям современное решение, которое позволит им удобно просматривать каталог товаров, управлять корзиной, оформлять заказы и отслеживать их статус. Первоначально продукт ориентирован на обычных розничных покупателей, с перспективой масштабирования бизнеса и расширения функционала площадки по мере развития проекта.

6.1 Сегменты целевой аудитории

Разрабатываемый продукт ориентирован на розничных покупателей, а в перспективе – на оптовых заказчиков и крупный B2B-сектор. Каждый сегмент аудитории имеет свои специфические потребности: корпоративные клиенты в первую очередь заинтересованы в автоматизации процесса закупок, получении индивидуальных ценовых предложений и минимизации ошибок при оформлении партий. Мы предлагаем гибкие условия, которые варьируются в зависимости от объемов закупок клиента.

Успешная реализация проекта требует формирования надежной партнерской сети. Ключевыми партнерами станут поставщики товаров и логистические службы. Взаимная выгода заключается в обеспечении стабильного канала сбыта для партнеров и бесперебойного предоставления качественных товаров для клиентов магазина.

Розничные пользователи и оптовые покупатели станут основными потребителями системы. Сотрудничество с ними позволит предоставить удобное и эффективное решение для совершения покупок через платформу Telegram.

					ПГУ1.090301.09.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

Взаимная выгода заключается в предоставлении продукта, который поможет клиентам упростить процесс оформления заказа, оптимизировать поиск товаров и сэкономить время и ресурсы на взаимодействии со стандартными веб-сайтами.

6.2 Ключевые процессы

Ниже перечислены основные ключевые процессы при разработке стартап-проекта:

- создание ИП;
- поиск финансирования;
- разработка программного обеспечения;
- поиск каналов сбыта и взаимодействие с корпоративными клиентами с целью продажи ПО;
- техническая поддержка и развитие данного продукта.

Разработка программного обеспечения будет осуществляться под кодами ОКВЭД 62.02 «Консультирование по компьютерным вопросам и связанная деятельность» и 62.09 «Деятельность, связанная с использованием вычислительной техники». В качестве налогового режима выбрана УСН из-за простоты расчета и уплаты налогов, а также благодаря низкой ставке – в Пензенской области она составляет 6% от дохода [7]. Налоги будут оплачиваться ежеквартально, а декларация подаваться раз в год. Документы для регистрации решено подать через МФЦ, чтобы избежать уплаты госпошлины. Выписка из ЕГРИП придет на электронную почту в течение трех дней.

Для запуска проекта формируется команда: разработчики, бухгалтер, операторы технической поддержки и маркетолог. Такой подход позволит оптимизировать расходы на старте и быстро приступить к работе.

Этапы разработки ПО описаны в разделе 1.

Поиск каналов сбыта осуществляется с помощью коммерческих предложений. Предлагаемое решение в виде интернет-магазина на базе

					ПГУ1.090301.09.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

мессенджера имеет несколько значимых достоинств, которые помогут привлечь заказчиков и выделить проект среди конкурентов.

Во-первых, оно автоматизирует процесс обработки заказов, снижая нагрузку на менеджеров и минимизируя влияние человеческого фактора. Во-вторых, обеспечивает прямой, быстрый и безопасный канал коммуникации с покупателями через привычный интерфейс Telegram, что напрямую повышает конверсию и лояльность аудитории.

Можно отметить, что использование передовых кроссплатформенных технологий и интеграция в экосистему популярного мессенджера повышает имидж компании-заказчика и демонстрирует ее инновационный подход к бизнесу. Клиенты воспринимают такую компанию как современную, внимательную к новым тенденциям и готовую предложить удобные цифровые сервисы.

Кроме того, предлагаемое решение отличается гибкой архитектурой и широким функционалом, охватывающим весь цикл: от навигации по каталогу до оформления корзины. Это позволяет занять уверенную позицию и быть востребованным проектом в сегменте электронной коммерции. Ниже в таблице 3 приведены услуги и цены, предлагаемые в рамках коммерческого предложения.

Таблица 3 – Краткое содержание аналоговых решений

Наименование	Цена реализации
Проектирование архитектуры и базы данных	15 000
Разработка клиентской части бота	40 000
Разработка панели администратора	25 000
Интеграция платежной системы	10 000
Развертывание ПО на сервере	10 000
Техническая поддержка и сопровождение	5 000

6.3 Каналы взаимодействия

Основным каналом продаж станут прямые презентации для B2B-сектора – обзвоны и встречи с владельцами розничного и оптового бизнеса. На реальных примерах покажем, как магазин в мессенджере упрощает путь покупателя, повышает конверсию и сокращает издержки на ручную обработку заказов.

После заключения договора мы обеспечим полный цикл внедрения: от развертывания бота на сервере и первоначальной настройки базы данных каталога до обучения персонала клиента работе с админ-панелью. Процесс интеграции выстроен так, чтобы бизнес начал получать заказы максимально быстро.

Особое внимание уделим техническому сопровождению. Служба поддержки будет оперативно решать возникающие вопросы, а регулярные обновления позволят масштабировать функционал. Для поддержания долгосрочных отношений планируется регулярный сбор обратной связи от заказчиков, что позволит постоянно совершенствовать продукт.

6.4 Источники дохода

Модель монетизации включает несколько ключевых направлений. Основной объем выручки формируется за счет продажи программного обеспечения – разработки и детальной настройки индивидуальных Telegram-магазинов для корпоративных заказчиков. Ориентировочная стоимость базового проекта для одного клиента составляет 60 000 рублей.

Дополнительный финансовый поток обеспечивает услуга внедрения и развертывания стоимостью 10 000 рублей. Она включает интеграцию продукта в существующую ИТ-инфраструктуру заказчика, установку ПО на сервер и первичную адаптацию под конкретные бизнес-процессы. Кроме того, предусмотрено комплексное абонентское техническое сопровождение за 3 000 рублей в месяц, охватывающее оперативные консультации пользователей и устранение возможных неполадок, что повышает общую лояльность клиентской базы.

Архитектура системы изначально предполагает масштабируемость. Расширение функционала за счет внедрения дополнительных программных модулей обойдется заказчику от 10 000 рублей за каждый новый блок. Это обеспечивает стабильный приток средств по мере роста коммерческих потребностей бизнеса клиента.

Исходя из анализа регионального рынка, количество потенциальных средних и крупных торговых предприятий в Пензе, нуждающихся в автоматизации продаж, достигает 10. При прогнозируемой конверсии в 20% стартовую базу составят 2 организации. Ожидаемый финансовый результат рассчитывается как произведение количества привлеченных заказчиков на суммарную стоимость реализации проекта. На основе этих данных базовый доход от первоначальной разработки и внедрения составит 140 000 рублей, а подключение первичных модулей масштабирования принесет еще 20 000 рублей. В итоге общий ожидаемый разовый доход на старте оценивается в 160 000 рублей, без учета стабильных ежемесячных поступлений от услуг поддержки.

6.5 Ключевые ресурсы

Для успешного запуска интернет-магазина на базе Telegram-бота необходимо комплексное обеспечение ключевыми ИТ-ресурсами. Основой системы является надежный виртуальный выделенный сервер для бесперебойного хостинга приложения. Серверная инфраструктура должна быть дополнена системой управления базами данных для хранения каталога товаров, профилей пользователей и истории заказов. Обязательным элементом является настройка автоматического резервного копирования для защиты коммерческих данных.

На программном уровне требуется интегрированная среда разработки, интерпретатор Python, асинхронный фреймворк aiogram и специализированное программное обеспечение для администрирования баз данных.

Успешное внедрение продукта требует не только технического оснащения, но и грамотной организации рабочего процесса. Ключевым элементом становятся

программисты, которые будут заниматься написанием кода и адаптацией функционала под конкретные бизнес-задачи заказчика. Для продуктивной работы команды необходимо профильное рабочее пространство, оборудованное мощными рабочими станциями для написания и тестирования кода, средами для отладки, а также переговорными зонами для согласования технических заданий с клиентами.

Команда должна включать бэкенд-разработчиков с уверенным знанием Python и опытом работы с Telegram Bot API. Также необходимы специалисты по проектированию баз данных и дизайнеры интерфейсов для создания логичной и удобной навигации внутри мессенджера. Для координации работы потребуются менеджеры проектов, которые будут контролировать этапы тестирования, сроки релизов и бюджет разработки.

6.6 Риски проекта

При запуске стартапа важно учитывать возможные риски и форс-мажорные обстоятельства. Среди ключевых угроз является низкий спрос со стороны корпоративных клиентов, жесткая конкуренция с готовыми конструкторами сайтов и маркетплейсами, а также изменения в законодательстве в сфере электронной коммерции и регулирования мессенджеров. Отдельно стоит выделить технические риски: сбои в работе Telegram API, недоступность серверов хостинга и уязвимости баз данных. Кроме того, работу проекта могут нарушить макроэкономическая нестабильность и внешние ограничения на информационную инфраструктуру. Для снижения потенциальных потерь необходимо заранее разработать стратегии минимизации угроз, включая настройку автоматического резервного копирования, использование отказоустойчивых серверов и гибкую адаптацию продукта под меняющиеся условия рынка.

					ПГУ1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

Заключение

В ходе выполнения индивидуального проекта был разработан интернет-магазин на основе Telegram-бота, полностью соответствующий поставленной цели и задачам. В результате проведённой работы было создано приложение, обеспечивающее удобный просмотр каталога товаров, управление корзиной, оформление заказов с интеграцией тестовой платёжной системы, а также гибкое взаимодействие пользователей с системой непосредственно в мессенджере Telegram.

Все заявленные функциональные требования были реализованы. Пользовательский интерфейс бота построен на системе inline-клавиатур и поддерживает интуитивно понятную навигацию между основными разделами: главное меню, каталог с пагинацией, профиль, корзина и история заказов. Система хранения данных на базе SQLite в связке с асинхронной библиотекой aioredis обеспечивает надёжное хранение информации о товарах, пользователях, корзинах и заказах, что соответствует современным подходам к организации данных в небольших проектах.

Особое внимание было уделено обработке пользовательских запросов и сценариев взаимодействия. Машина состояний, предоставляемая библиотекой aiogram, позволила организовать пошаговые диалоги для заполнения профиля, добавления и редактирования товаров без потери данных при случайном прерывании действий. Интеграция тестового платёжного токена Telegram дала возможность полноценно отладить процесс оплаты, а в дальнейшем, как отмечено в документации платёжного сервиса, замена на рабочий токен потребует минимальных изменений в коде.

Обработка ошибок реализована на всех уровнях приложения: от валидации ввода пользователя до перехвата исключений при сетевых сбоях и удалении сообщений, что подтверждено в ходе функционального тестирования. Приложение успешно развёрнуто на сервере и обеспечивает стабильную

					<i>ПГУ1.090301.09.001 ПЗ</i>	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

непрерывную работу благодаря асинхронной архитектуре и корректной маршрутизации запросов.

Архитектура проекта, построенная по принципу разделения на модули, позволяет легко расширять функционал: добавлять новые категории товаров, интегрировать другие платёжные шлюзы или подключать системы аналитики без существенной переработки уже написанного кода.

Таким образом, разработанный Telegram-бот представляет собой полноценное, готовое к эксплуатации решение для малого бизнеса, которое сочетает низкий порог входа для пользователей и удобные инструменты управления для администратора. Результаты тестирования подтвердили корректность работы всех сценариев, устойчивость к ошибочным действиям и высокую степень удобства интерфейса.

					ПГУ1.090301.09.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. cyberleninka.ru.: [Электронный ресурс]. URL <https://cyberleninka.ru/article/n/sfery-primeneniya-yazyka-python> (дата обращения 19.05.2026)
2. cyberleninka.ru.: [Электронный ресурс]. URL <https://cyberleninka.ru/article/n/ispolzovanie-subd-v-proektah-mashinnogo-obucheniya-i-analiza-dannyh> (дата обращения 19.05.2026)
3. cyberleninka.ru.: [Электронный ресурс]. URL <https://cyberleninka.ru/article/n/sfery-primeneniya-yazyka-python> (дата обращения 19.05.2026)
4. cyberleninka.ru.: [Электронный ресурс]. URL <https://cyberleninka.ru/article/n/issledovanie-razrabotki-asinhronnoy-monitoringovoy-sluzhby-na-baze-modulya-asyncio-yazyka-programmirovaniya-python-s-primeneniem> (дата обращения 19.05.2026)
5. docs.cntd.ru.: [Электронный ресурс]. URL <https://docs.cntd.ru/document/1200003913> (дата обращения 19.05.2026)
6. cyberleninka.ru.: [Электронный ресурс]. URL <https://cyberleninka.ru/article/n/fizkultura-v-zhizni-programmista> (дата обращения 19.05.2026)
7. nalog.gov.ru.: [Электронный ресурс]. URL <https://www.nalog.gov.ru/rn77/taxation/taxes/usn/> (дата обращения 20.05.2026)

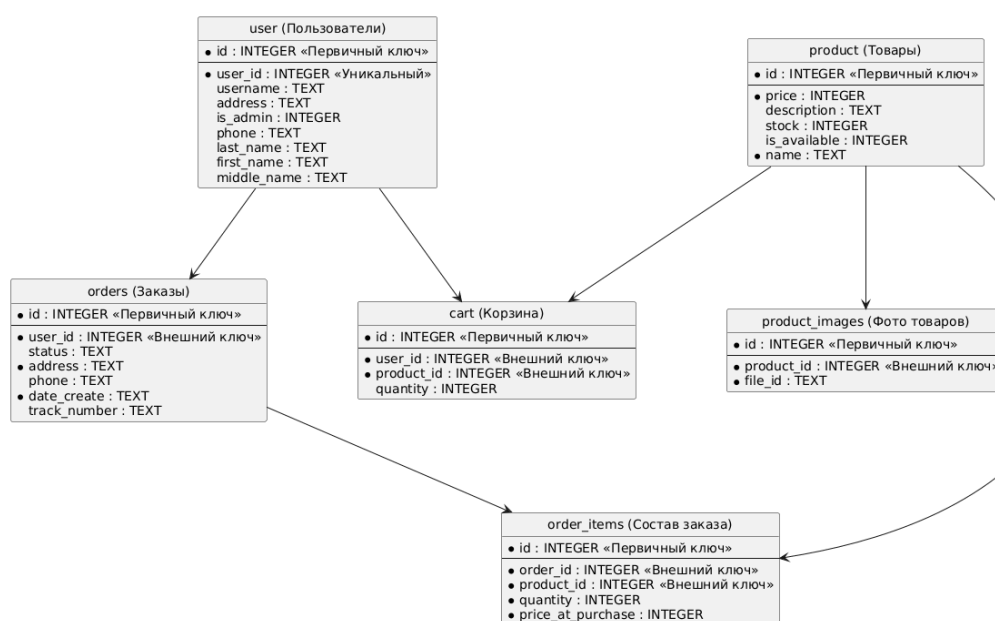
Приложение А
(обязательное)

Бизнес-проект «Разработка интернет-магазина на основе Telegram-бота»

UML-диаграммы

					<i>ПГУ1.090301.09.001 ПЗ</i>	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML-диаграмма базы данных



					ПГУ1.090301.09.002 Д6		
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмма логической базы данных		
Разработ.	Евдокимов Р.Е.						
Пров.	Юрова О.В.						
Утверд.							
Исполн.	Бычков А.С.						
Утв.	Митрошин М.А.				Лист 1 из 1		
					Гр. 22ВВП1		

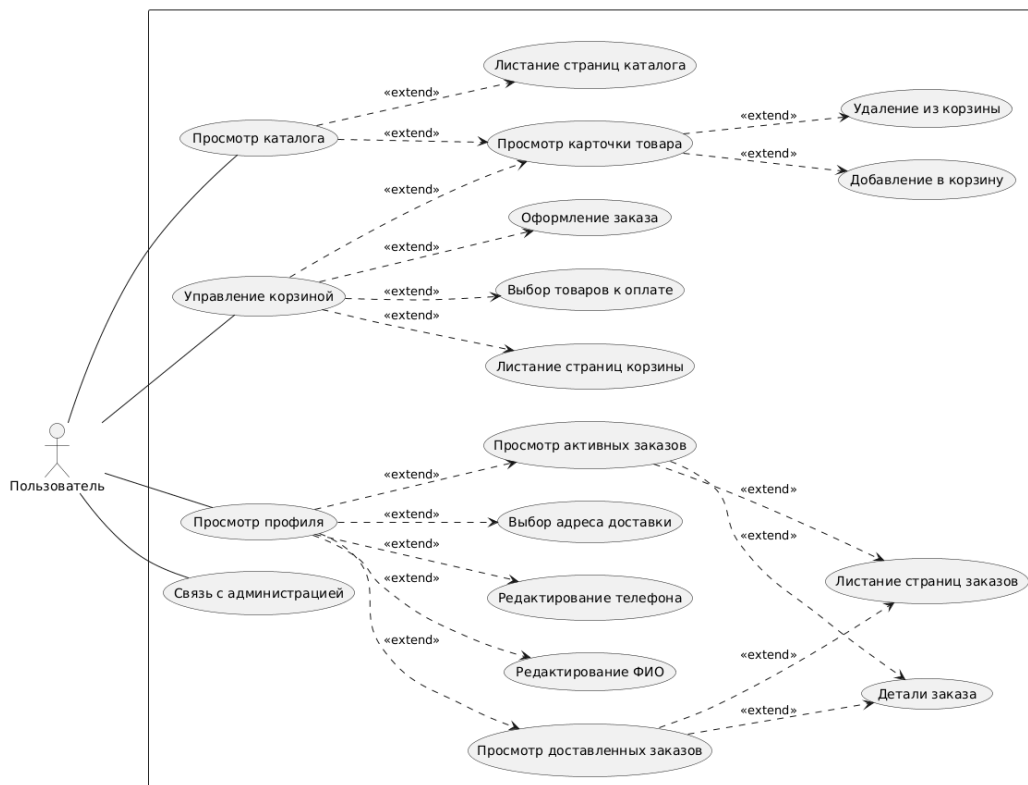
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

89

Приложение А.2 – UML-диаграмма прецедентов пользователя



					ПГУ1.090301.09.003 Д13		
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм прецедентов пользователя		
Разработ.	Евдокимов Р.Е.						
Пров.	Юрова О.В.				Лист	1	Листов
Текст.					1		1
Реценз.					Гр. 22ВВП1		
Начектр.	Бычков А.С.						
Утв.	Митрохин М.А.						

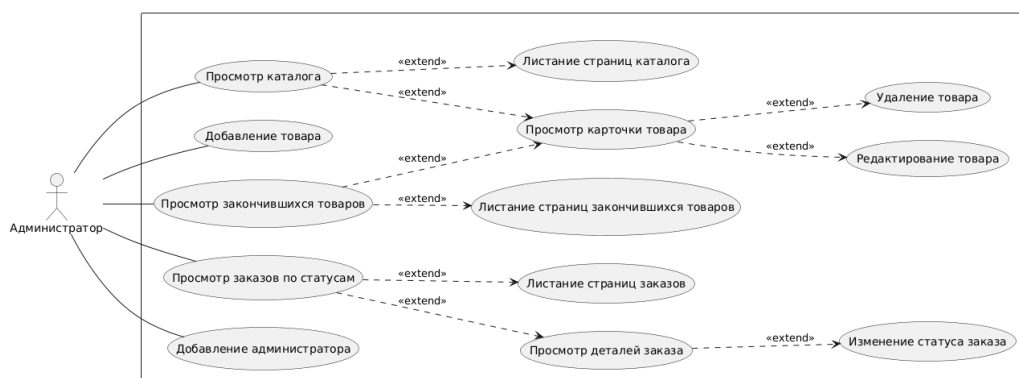
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

90

Приложение А.3 – UML-диаграмма прецедентов администратора



					ПГУ1.090301.09.004 Д13			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмма прецедентов администратора			
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Утверд.								
Реценз.	Бычков А.С.							
Исполн.	Матроскин М.А.				Лист 1 Листов 1			
Утв.					Гр. 22ВВП1			

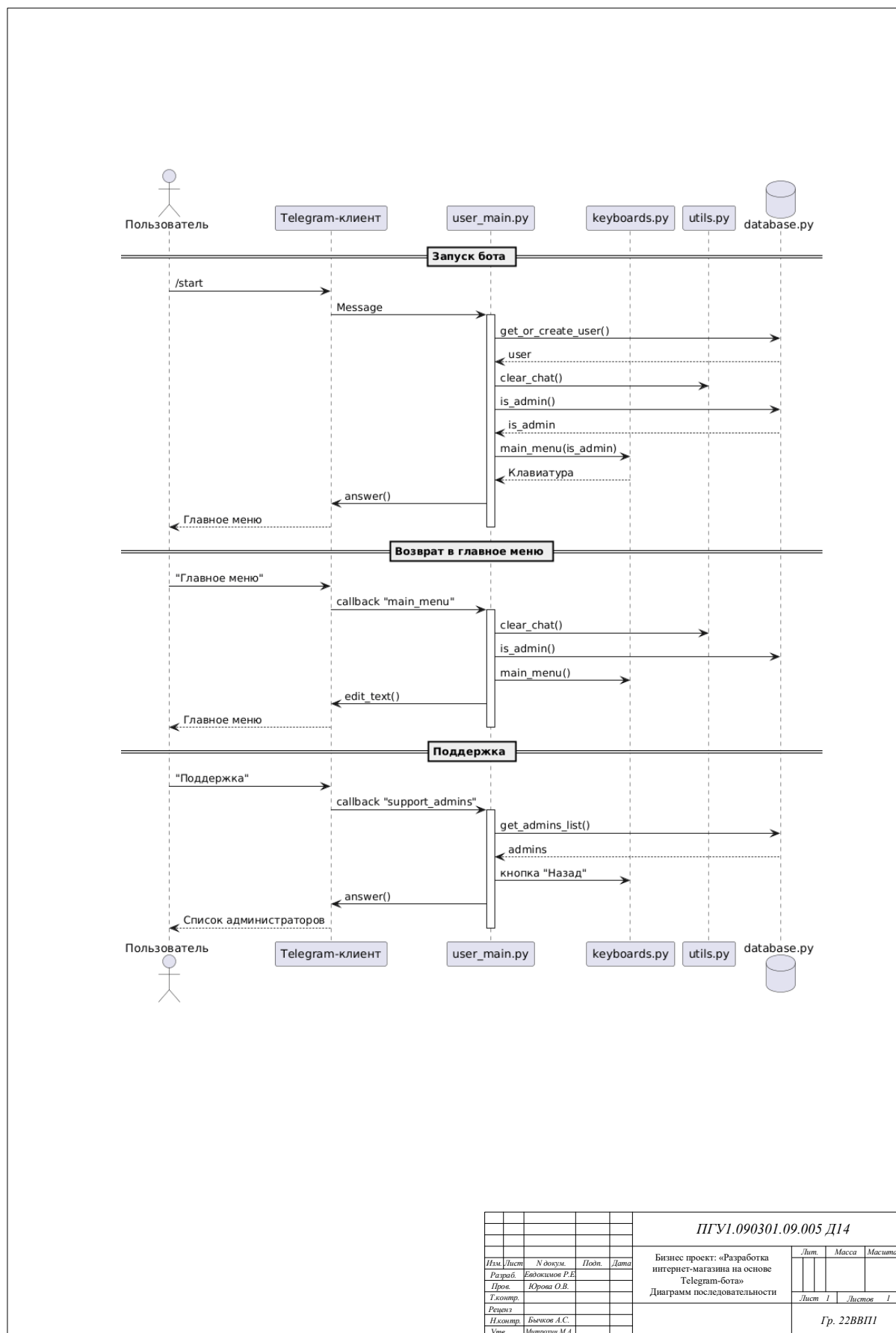
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

91

Приложение А.4 – UML-диаграмма модуля user_main.py



					ПГУ1.090301.09.005 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности			
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.								
Реценз.	Бычков А.С.							
И.контр.					Гр. 22ВВП1			
Утв.	Митрошин М.А.							

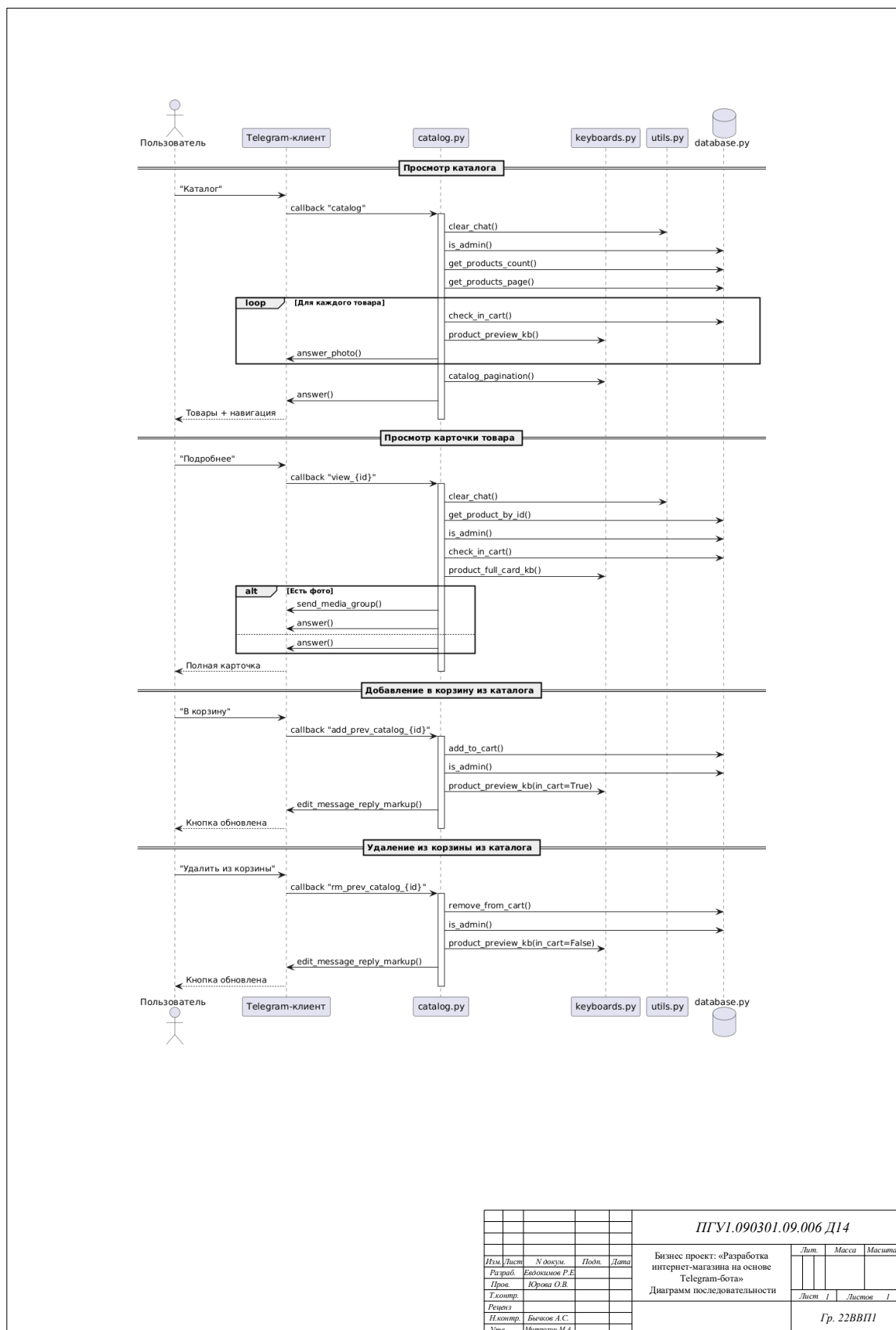
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

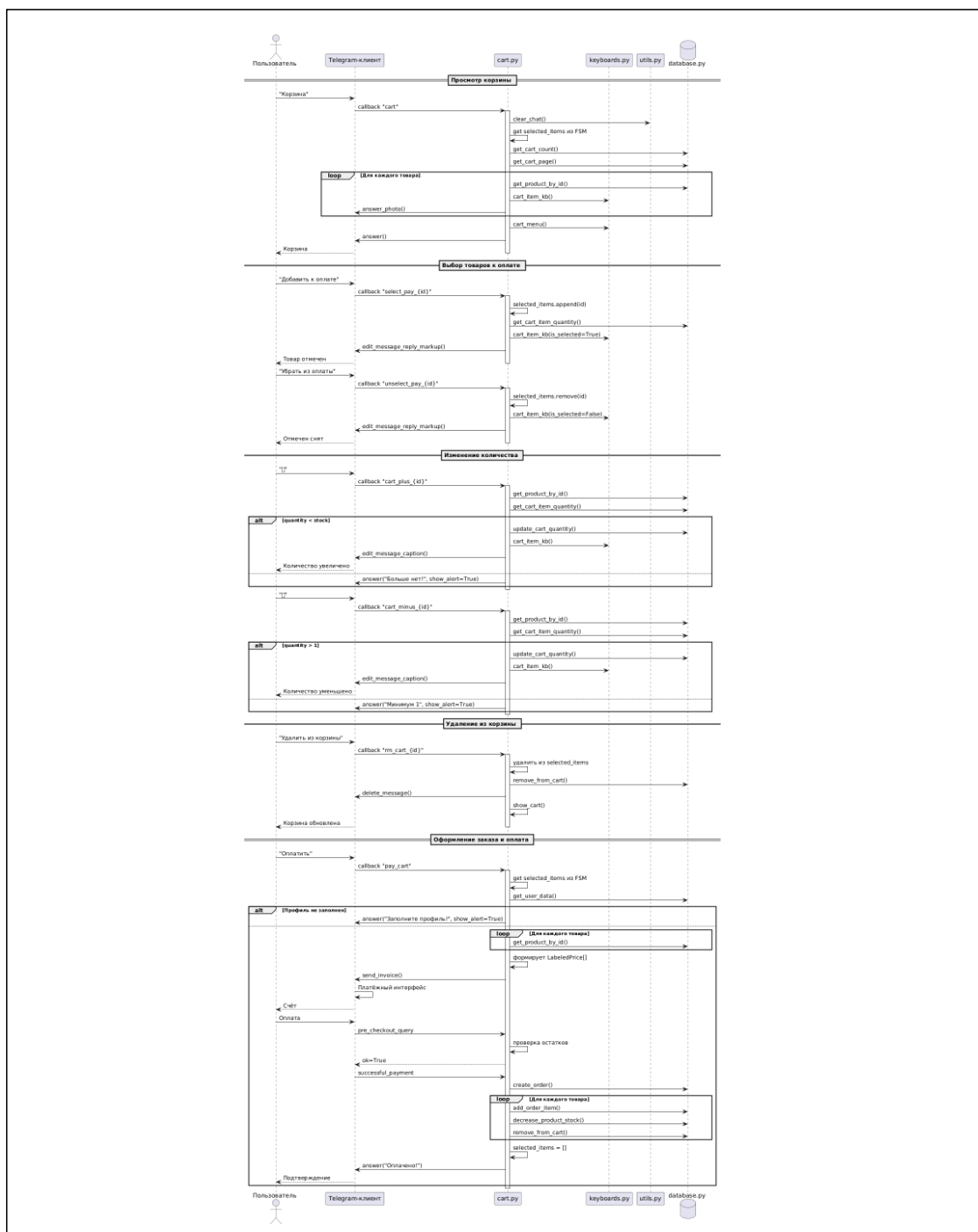
Лист

92

Приложение А.5 – UML-диаграмма модуля catalog.py

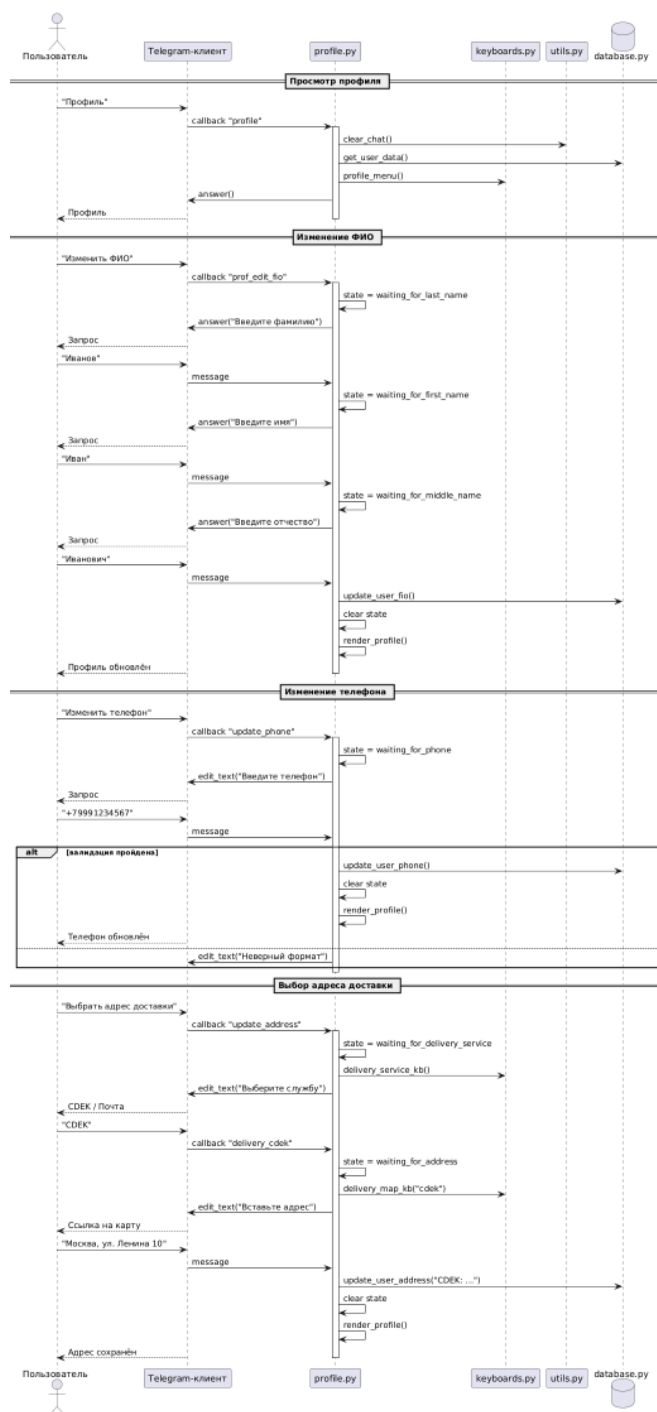


Приложение А.6 – UML диаграмма модуля cart.py



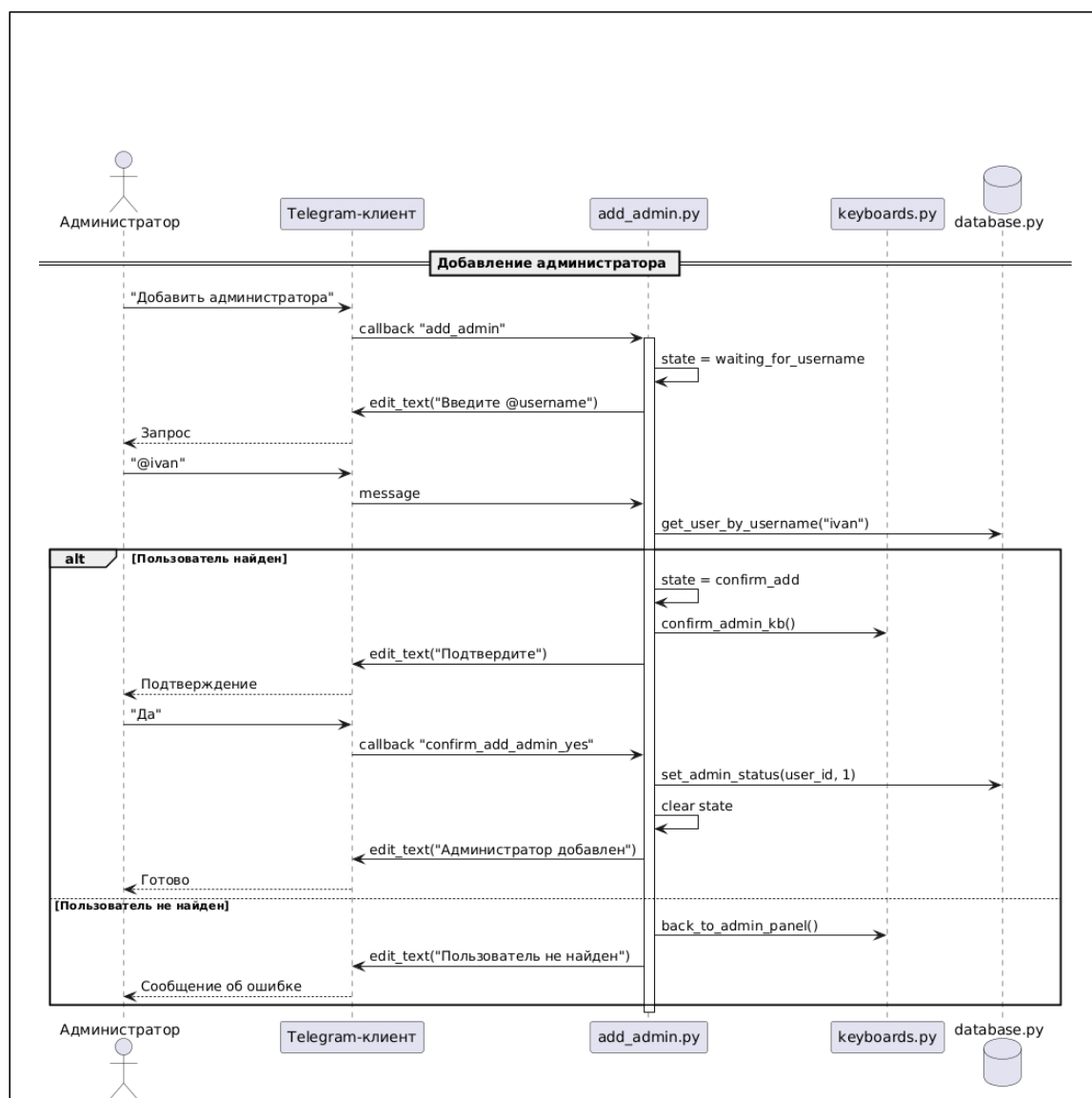
					ПГУ1.090301.09.007 Д14		
Изм.	Лист	N докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности		
Разраб.	Евдокимов Р.Е						
Пров.	Юрова О.В.						
Т.контр.							
Реценз							
Н.контр.	Бычков А.С.				Гр. 22ВВП1		
Утв.	Митрохин М.А.						

Приложение А.7 – UML диаграмма модуля profile.py



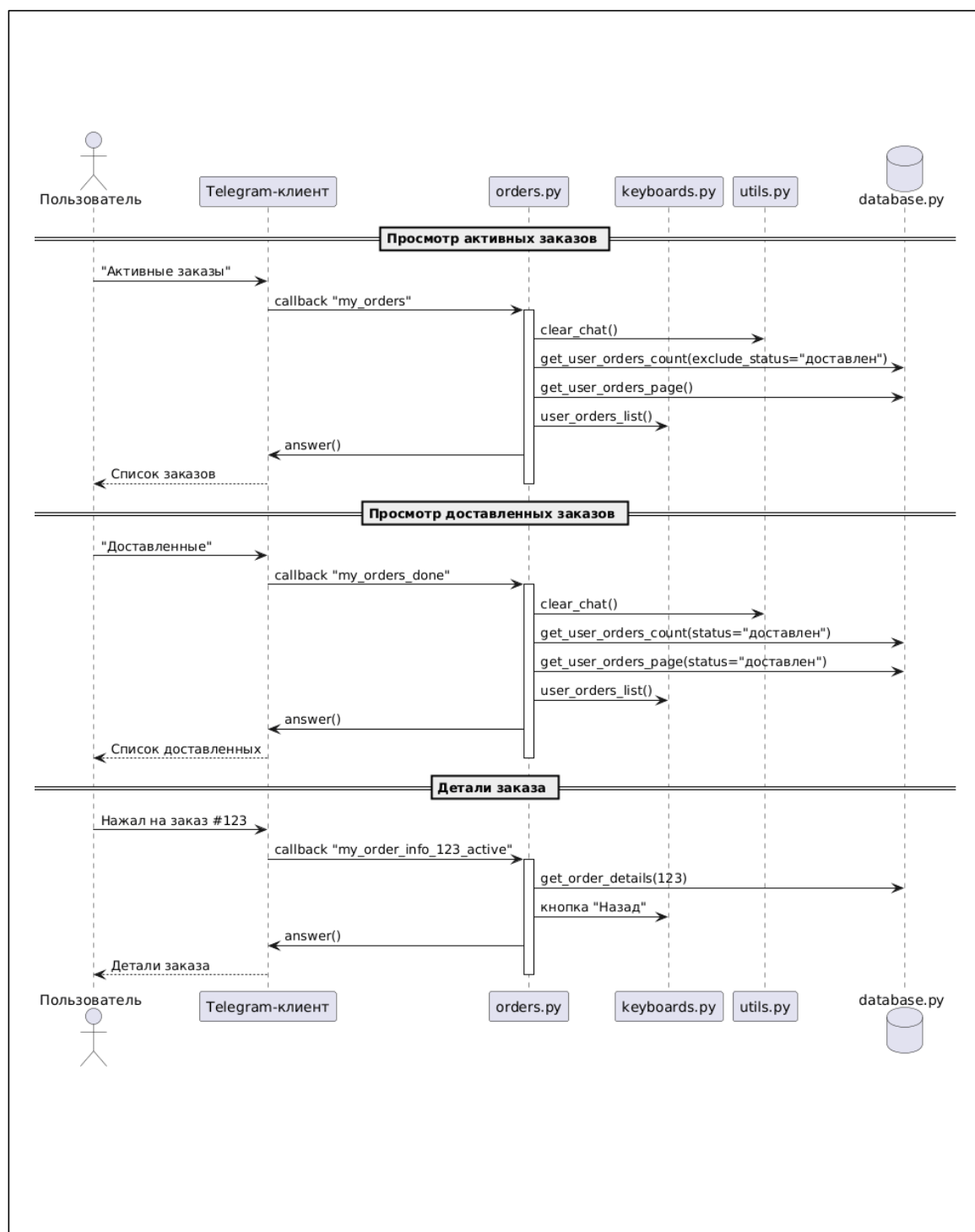
					ПГУ1.090301.09.008 Д14			
Изм.	Лист	N докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лист	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т. контр.						Лист	1	Листов
Реценз								1
Н.контр.	Бычков А.С.				Гр. 22ВВП1			
Утв.	Митрохин М.А.							

Приложение А.8 – UML диаграмма модуля add_admin.py



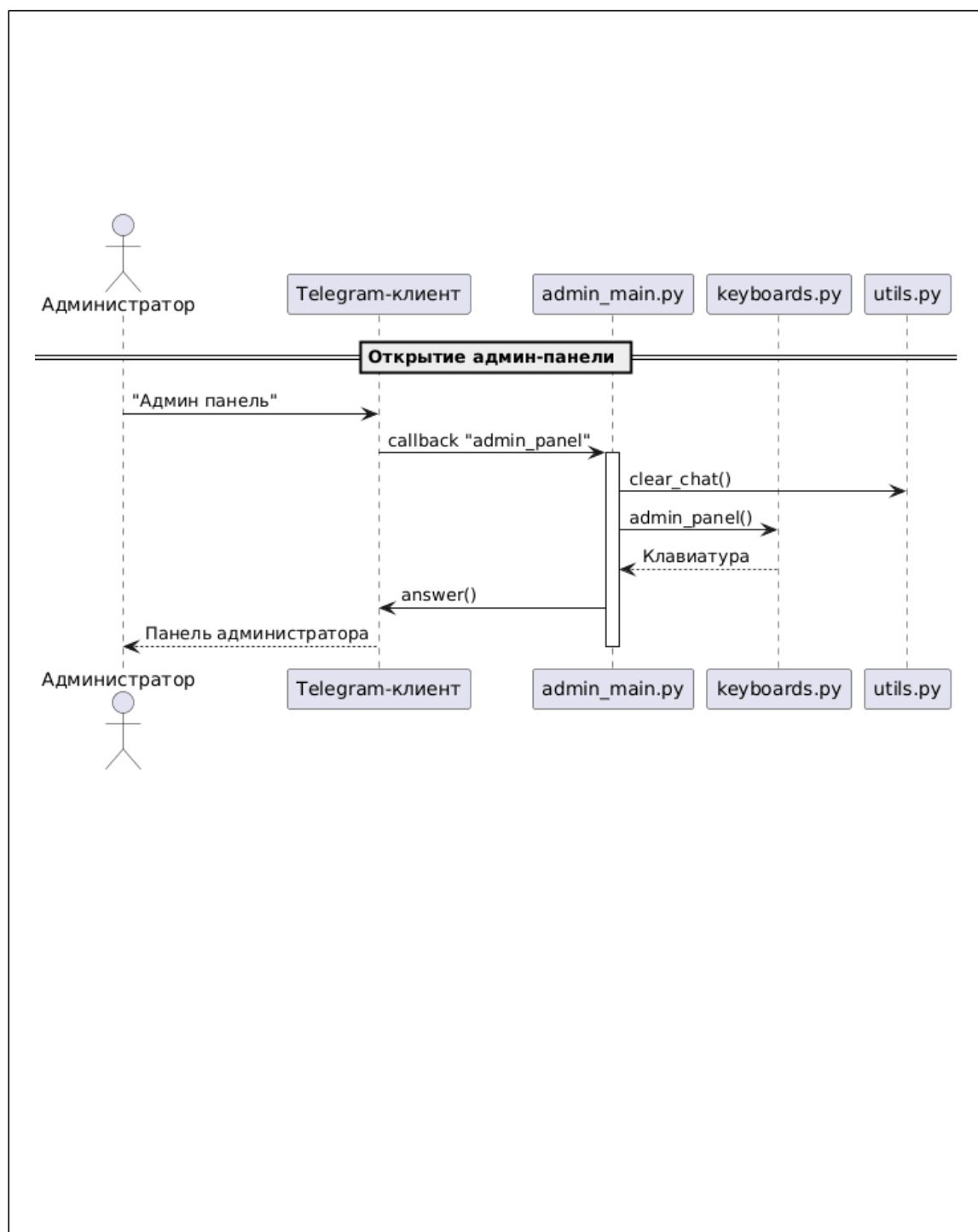
					ПГУ1.090301.09.009 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности			
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.								
Реценз								
Н.контр.	Бычков А.С.				Гр. 22ВВП1			
Утв.	Митрохин М.А.							
					Лит.	Масса	Масштаб	
					Лист 1	Листов 1		

Приложение А.9 – UML диаграмма модуля orders.py



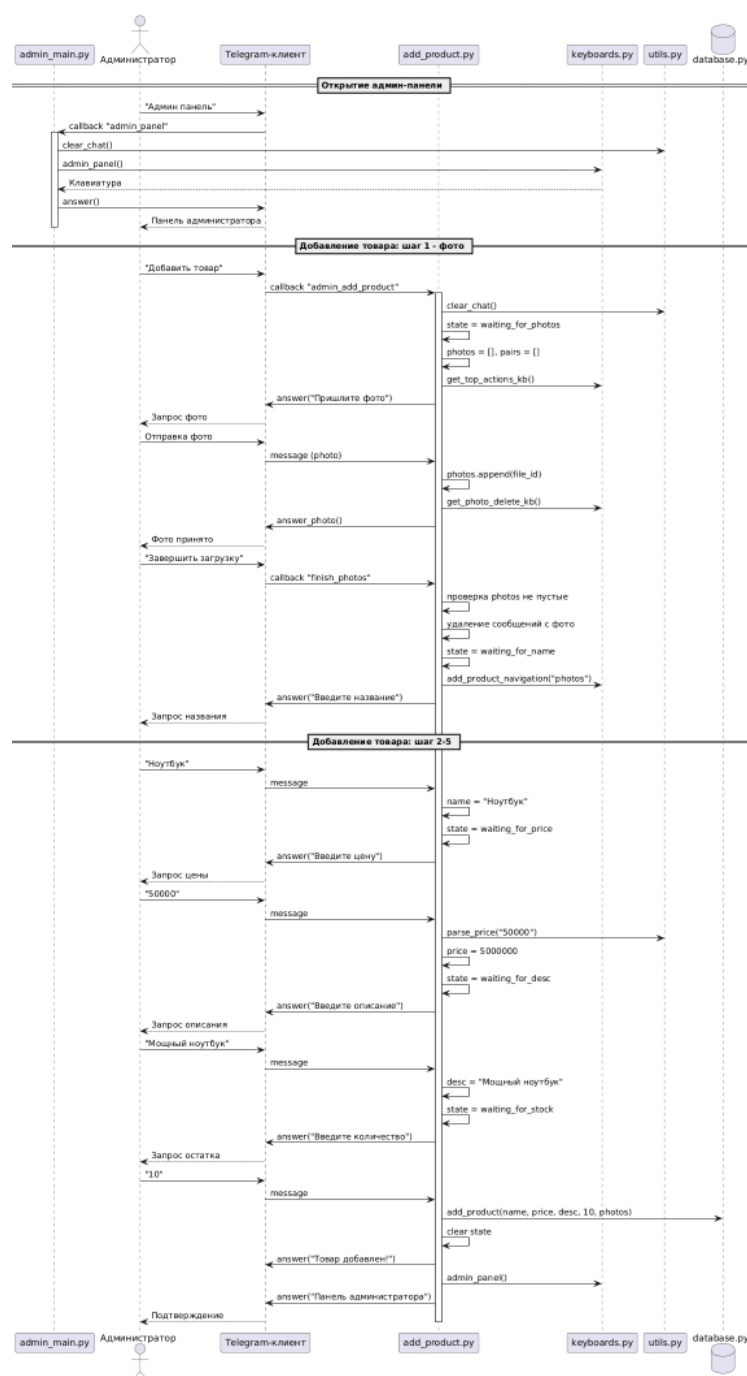
					ПГУ1.090301.09. 010 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВП1		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

Приложение А.10 – UML диаграмма модуля admin_main.py



					ПГУ1.090301.09.011 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.								
Реценз								
Н.контр.	Бычков А.С.				Гр. 22ВВП1	Лист 1	Листов 1	
Утв.	Митрохин М.А.							

Приложение А.11 – UML диаграмма модуля add_product.py



ПГУ1.090301.09.012 Д14

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Евдокимов Р.Е.		
Пров.		Юрова О.В.		
Т.контр.				
Реценз				
Н.контр.		Бычков А.С.		
Утв.		Митрохин М.А.		

Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота»
Диаграмм последовательности

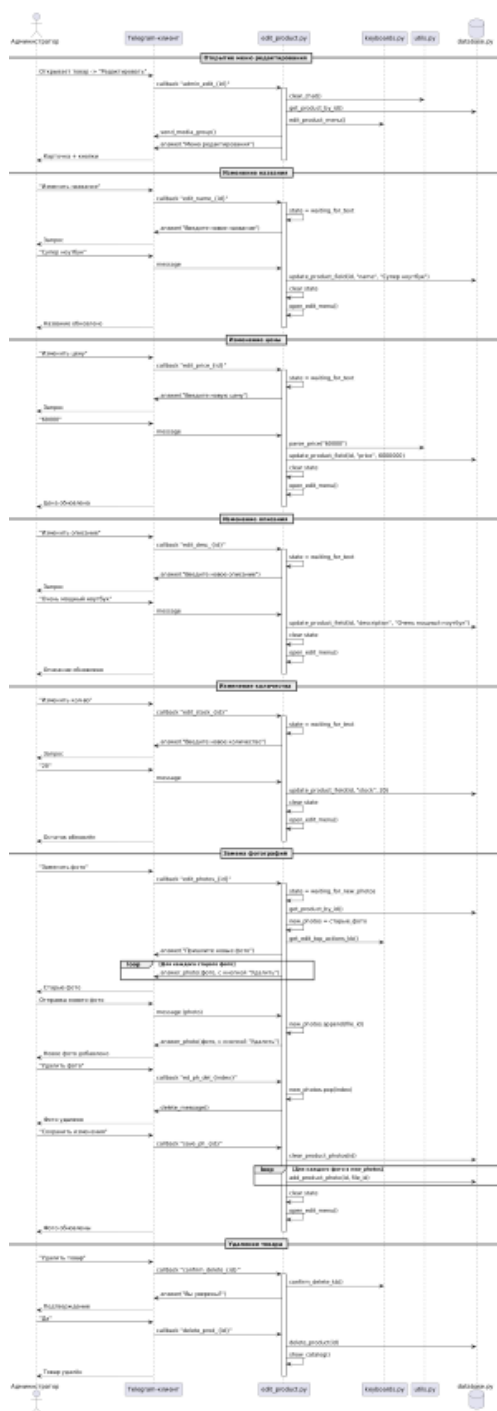
Лит.	Масса	Масштаб
Лист 1	Листов 1	
Гр. 22ВВІІІ		

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

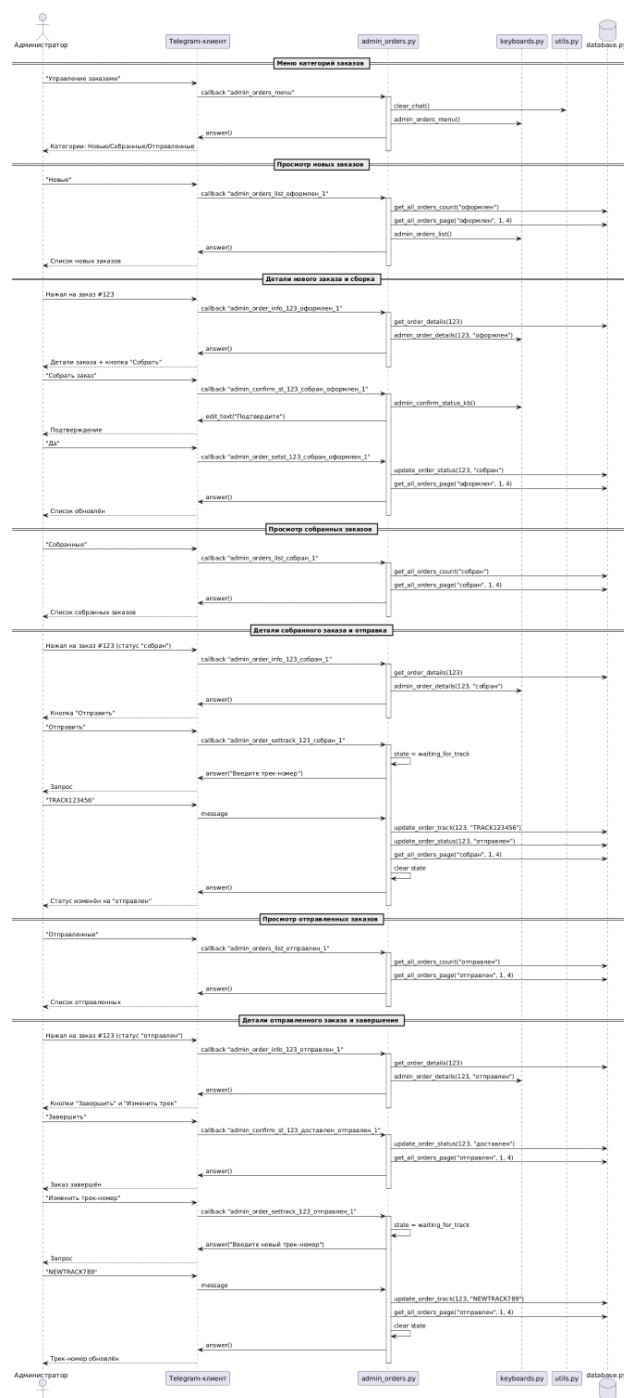
Лист
99

Приложение А.12 – UML диаграмма модуля edit_product.py



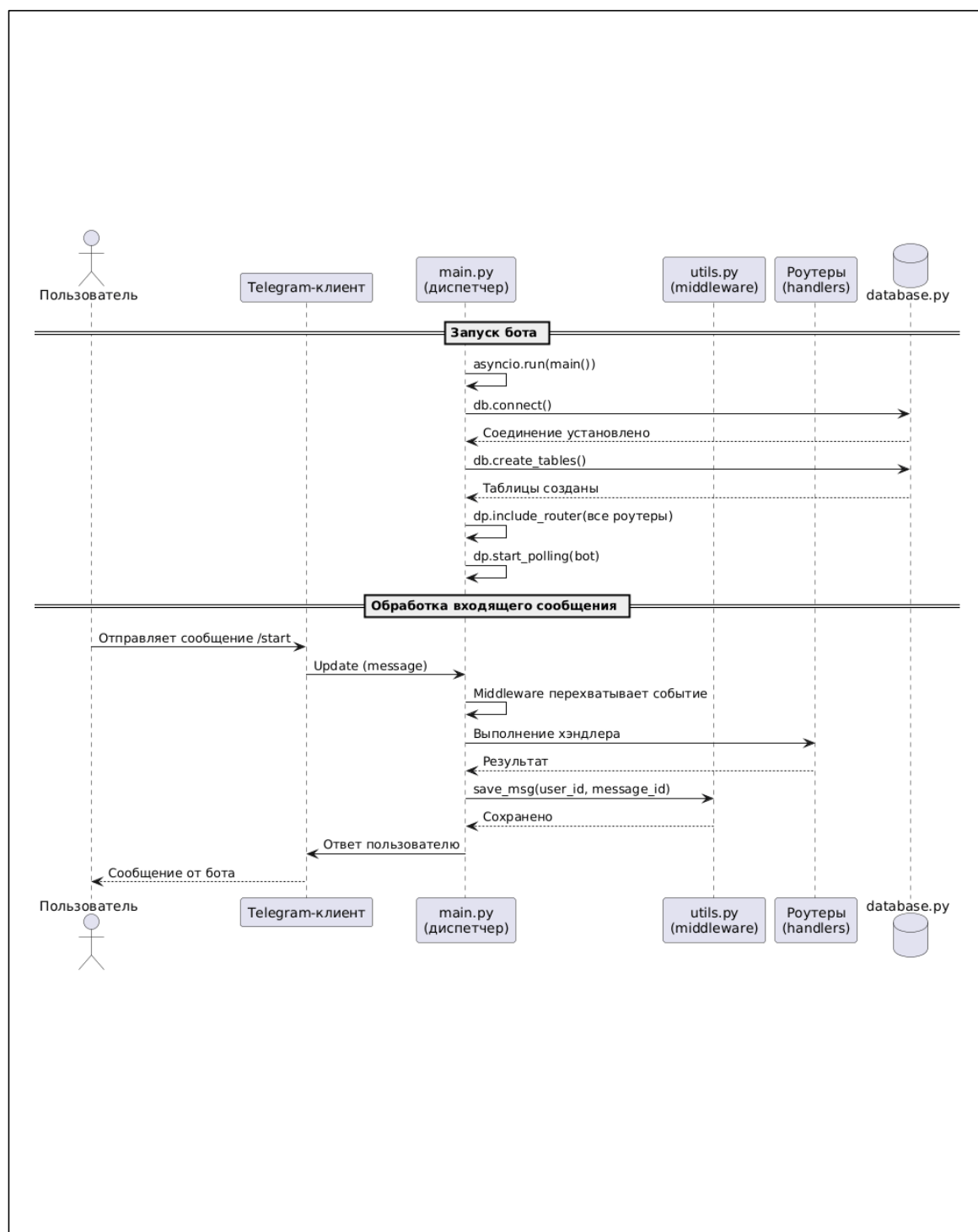
					ПГУ1.090301.09.013 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВІІ		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

Приложение А.13 – UML диаграмма модуля admin_orders.py



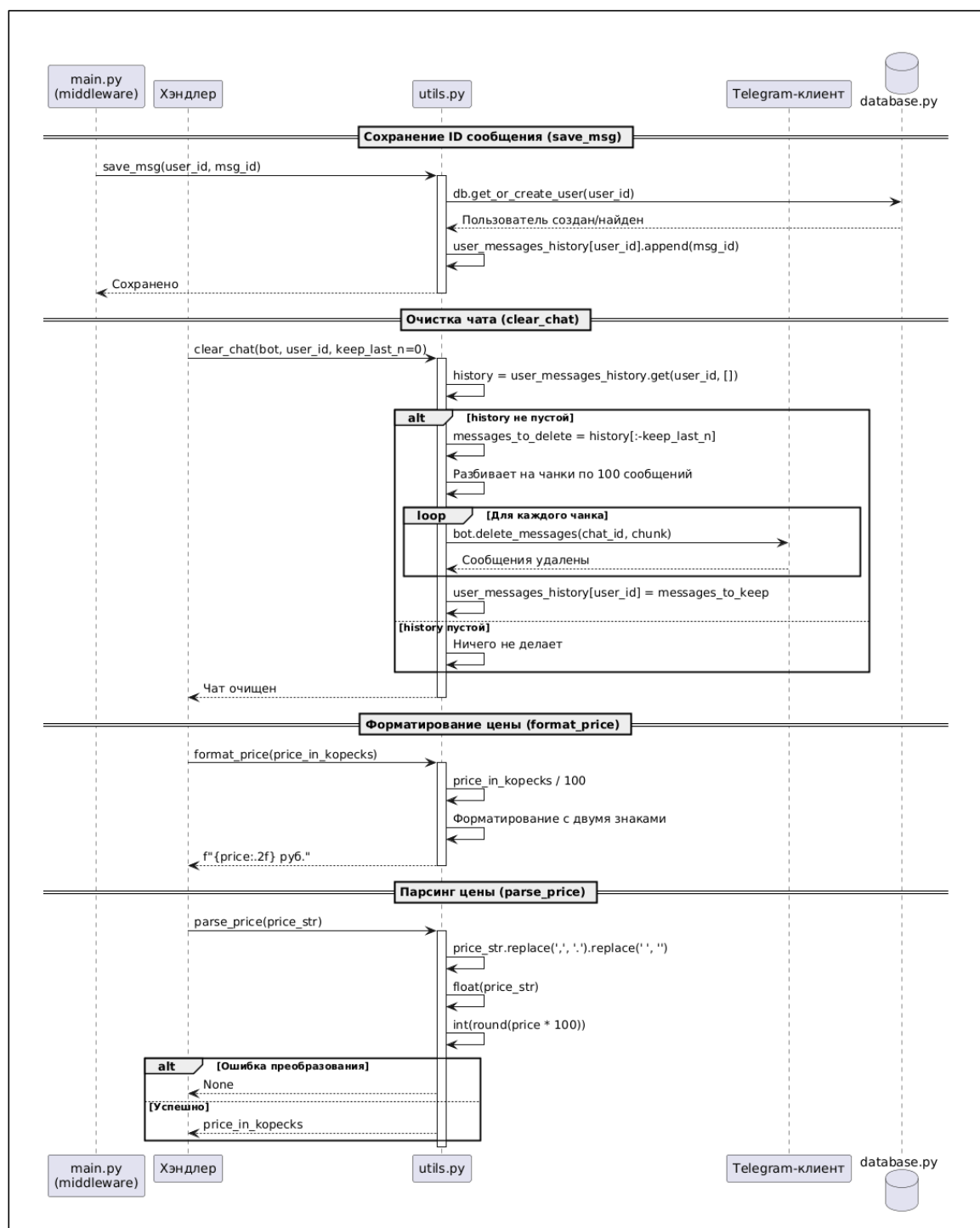
					ПГУ1.090301.09.014 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВП1		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

Приложение А.14 – UML диаграмма модуля main.py



					ПГУ1.090301.09.015 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВІІ		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

Приложение А.15 – UML диаграмма модуля utils.py



					ПГУ1.090301.09.016 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВП1		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

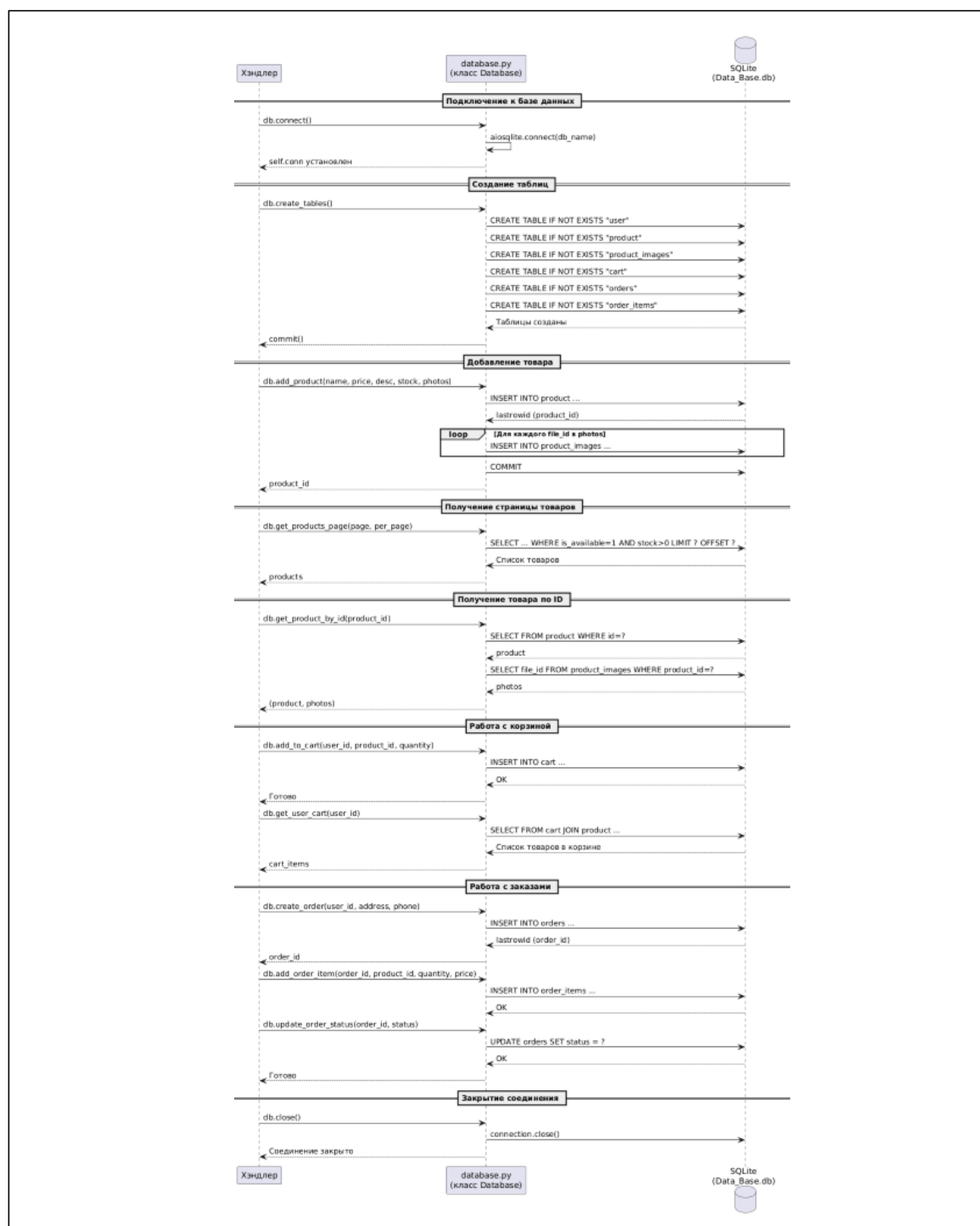
Приложение А.16 – UML диаграмма модуля keyboards.py



ПГУ1.090301.09.017 Д14

					ПГУ1.090301.09.017 Д14									
Изм.	Лист	N докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности					Лит.		Масса	Масштаб	
Разраб.	Евдокимов Р.Е.													
Пров.	Юрова О.В.													
Т.контр.														
Реценз														
Н.контр.	Бычков А.С.									Лист 1		Листов 1		
Утв.	Митрохин М.А.											Гр. 22ВВІІІ		

Приложение А.17 – UML диаграмма модуля database.py



					ПГУ1.090301.09.018 Д14			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм последовательности	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВП1		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

Приложение А.18 – UML диаграмма компонентов



					ПГУ1.090301.09.019 Д18			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес проект: «Разработка интернет-магазина на основе Telegram-бота» Диаграмм компонентов	Лит.	Масса	Масштаб
Разраб.	Евдокимов Р.Е.							
Пров.	Юрова О.В.							
Т.контр.						Лист 1	Листов 1	
Реценз						Гр. 22ВВП1		
Н.контр.	Бычков А.С.							
Утв.	Митрохин М.А.							

Приложение Б

(обязательное)

Бизнес-проект «Разработка интернет-магазина на основе Telegram-бота»

Листинг программы

					ПГУ1.090301.09.001 ПЗ	Лист
						107
Изм.	Лист	№ докум.	Подпись	Дата		

Файл main.py

```
import asyncio
import logging
from aiogram import Bot, Dispatcher, types
from aiogram.fsm.storage.memory import MemoryStorage
from typing import Any, Awaitable, Callable, Dict
from utils import db, save_msg
from handlers.admin.admin_main import admin_router
from handlers.admin.add_product import add_product_router
from handlers.admin.edit_product import edit_product_router
from handlers.shared.catalog import catalog_router
from handlers.user.user_main import user_router
from handlers.user.cart import cart_router
from handlers.user.profile import profile_router
from handlers.admin.add_admin import add_admin_router
from handlers.user.orders import user_orders_router
from handlers.admin.admin_orders import admin_orders_router
BOT_TOKEN = "8645694054:AAEyVidBsflc9Y3E-H8A7Q5SaB1DS1bdTZc"
bot = Bot(token=BOT_TOKEN)
dp = Dispatcher(storage=MemoryStorage())
@dp.update.outer_middleware()
async def save_user_messages_middleware(
    handler: Callable[[types.Update, Dict[str, Any]], Awaitable[Any]],
    event: types.Update,
    data: Dict[str, Any]
) -> Any:
    result = await handler(event, data)
    if event.message and event.message.from_user:
        await save_msg(event.message.from_user.id, event.message.message_id)
    return result
async def main():
    await db.connect()
    await db.create_tables()
    dp.include_router(admin_router)
    dp.include_router(admin_orders_router)
    dp.include_router(add_product_router)
    dp.include_router(edit_product_router)
    dp.include_router(catalog_router)
    dp.include_router(user_router)
    dp.include_router(user_orders_router)
    dp.include_router(cart_router)
    dp.include_router(profile_router)
    dp.include_router(add_admin_router)
    logging.info("Бот запущен и готов к работе")
    try:
        await dp.start_polling(bot)
    finally:
        await db.close()
if __name__ == "__main__":
    try:
        asyncio.run(main())
```

					ПГУ1.090301.09.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		

```
except (KeyboardInterrupt, SystemExit):
    logging.info("Бот остановлен")
```

Файл keyboards.py

```
from aiogram.utils.keyboard import InlineKeyboardBuilder
from aiogram.types import InlineKeyboardButton

class UserKeyboards:
    @staticmethod
    def main_menu(is_admin=False):
        builder = InlineKeyboardBuilder()

        builder.button(text="📖 Каталог", callback_data="catalog")

        if not is_admin:
            builder.button(text="🛒 Корзина", callback_data="cart")
            builder.button(text="👤 Профиль", callback_data="profile")
            builder.button(text="👤 Поддержка", callback_data="support_admins")
        else:
            builder.button(text="⚙️ Админ панель", callback_data="admin_panel")

        builder.adjust(2 if not is_admin else 1)
        return builder.as_markup()

    @staticmethod
    def product_preview_kb(product_id, is_admin=False, in_cart=False, stock=0):
        builder = InlineKeyboardBuilder()

        builder.button(text="ℹ️ Подробнее", callback_data=f"view_{product_id}")

        if is_admin:
            builder.button(text="✎ Редактировать", callback_data=f"admin_edit_{product_id}")
        else:
            if in_cart:
                builder.button(text="❌ Удалить из корзины",
callback_data=f"rm_prev_catalog_{product_id}")
            else:
                builder.button(text="🛒 В корзину",
callback_data=f"add_prev_catalog_{product_id}")
                builder.adjust(1) # Все кнопки в один столбец

        return builder.as_markup()

    @staticmethod
    def product_full_card_kb(product_id, is_admin=False, in_cart=False, source="catalog"):
        builder = InlineKeyboardBuilder()

        back_data = "cart" if source == "cart" else "catalog"
        builder.button(text="⬅️ Назад", callback_data=back_data)

        if not is_admin:
            if in_cart:
                builder.button(text="❌ Удалить из корзины",
callback_data=f"rm_full_{source}_{product_id}")
            else:
                builder.button(text="🛒 В корзину",
callback_data=f"add_full_{source}_{product_id}")
                builder.adjust(2)

        return builder.as_markup()

    @staticmethod
    def catalog_pagination(page, total_pages, prefix="catalogpage"):
```

```

builder = InlineKeyboardBuilder()
buttons = []
if page > 1:
    buttons.append(InlineKeyboardButton(text="⬅ Пред.",
callback_data=f"{prefix}_{page-1}"))
    if page < total_pages:
        buttons.append(InlineKeyboardButton(text="След. ➡",
callback_data=f"{prefix}_{page+1}"))
    if buttons:
        builder.row(*buttons)

builder.row(InlineKeyboardButton(text="🏠 В меню", callback_data="main_menu"))
return builder.as_markup()

@staticmethod
def cart_item_kb(product_id, quantity=1, is_selected=False, is_oos=False):
    builder = InlineKeyboardBuilder()
    if is_oos:
        builder.button(text="❌ Удалить из корзины",
callback_data=f"rm_cart_{product_id}")
    else:
        builder.button(text="ℹ Подробнее", callback_data=f"view_cart_{product_id}")
        builder.button(text="—", callback_data=f"cart_minus_{product_id}")
        builder.button(text=f"{quantity} шт.", callback_data="ignore")
        builder.button(text="+", callback_data=f"cart_plus_{product_id}")
        if is_selected:
            builder.button(text="— Убрать из оплаты",
callback_data=f"unselect_pay_{product_id}")
        else:
            builder.button(text="+ Добавить к оплате",
callback_data=f"select_pay_{product_id}")
        builder.button(text="❌ Удалить из корзины",
callback_data=f"rm_cart_{product_id}")
        builder.adjust(1, 3, 1, 1)
    return builder.as_markup()

@staticmethod
def cart_menu(page=1, total_pages=1):
    builder = InlineKeyboardBuilder()
    nav_buttons = []
    if page > 1:
        nav_buttons.append(InlineKeyboardButton(text="⬅ Пред.",
callback_data=f"cartpage_{page-1}"))
        if page < total_pages:
            nav_buttons.append(InlineKeyboardButton(text="След. ➡",
callback_data=f"cartpage_{page+1}"))
        if nav_buttons:
            builder.row(*nav_buttons)

builder.row(InlineKeyboardButton(text="💳 Оплатить", callback_data="pay_cart"))
builder.row(InlineKeyboardButton(text="🏠 Назад в меню", callback_data="main_menu"))
return builder.as_markup()

@staticmethod

```

```

def empty_cart_kb():
    builder = InlineKeyboardBuilder()

    builder.button(text="🏠 На главную", callback_data="main_menu")

    return builder.as_markup()

@staticmethod
def delivery_service_kb():
    builder = InlineKeyboardBuilder()

    builder.button(text="📦 СДЕК", callback_data="delivery_cdek")

    builder.button(text="📮 Почта России", callback_data="delivery_post")

    builder.button(text="❌ Отмена", callback_data="cancel_profile")

    builder.adjust(2, 1)

    return builder.as_markup()

@staticmethod
def profile_menu():
    builder = InlineKeyboardBuilder()

    builder.button(text="📦 Активные заказы", callback_data="my_orders")

    builder.button(text="✅ Доставленные", callback_data="my_orders_done")

    builder.button(text="📄 Изменить ФИО", callback_data="prof_edit_fio")

    builder.button(text="📞 Изменить телефон", callback_data="update_phone")

    builder.button(text="🚚 Выбрать адрес доставки", callback_data="update_address")

    builder.button(text="🏠 Главное меню", callback_data="main_menu")

    builder.adjust(1)

    return builder.as_markup()

@staticmethod
def delivery_map_kb(service: str):
    builder = InlineKeyboardBuilder()

    if service == "cdek":

        builder.button(text="📍 Открыть карту СДЭК",
url="https://yandex.ru/maps/?text=Отделения+СДЕК+Москва")

        elif service == "post":

            builder.button(text="📍 Открыть карту Почты России",
url="https://yandex.ru/maps/?text=Отделения+Почты+России+Москва")

            builder.button(text="❌ Отмена", callback_data="cancel_profile")

            builder.adjust(1)

            return builder.as_markup()

    @staticmethod
def cancel_profile():
    """Простая клавиатура с кнопкой отмены (для операций в профиле)."""
    builder = InlineKeyboardBuilder()

    builder.button(text="❌ Отмена", callback_data="cancel_profile")

    return builder.as_markup()

@staticmethod
def user_orders_list(orders, page, total_pages, status_filter="all"):
    builder = InlineKeyboardBuilder()

    if not orders:

        builder.button(text="👤 Назад в профиль", callback_data="profile")

        return builder.as_markup()

    for order in orders:
        o_id, status, date = order

```

```

        short_date = str(date).split()[0] if date else ""
        builder.button(text=f"#{o_id} | {status} | {short_date}",
callback_data=f"my_order_info_{o_id}_{status_filter}")
        builder.adjust(1)
        nav_buttons = []
        if page > 1:
            nav_buttons.append(InlineKeyboardButton(text="⬅ Пред.",
callback_data=f"my_orders_p_{status_filter}_{page-1}"))
            if page < total_pages:
                nav_buttons.append(InlineKeyboardButton(text="След. ➡",
callback_data=f"my_orders_p_{status_filter}_{page+1}"))
            if nav_buttons:
                builder.row(*nav_buttons)

        builder.row(InlineKeyboardButton(text="👤 Назад в профиль", callback_data="profile"))
        return builder.as_markup()

class AdminKeyboards:
    @staticmethod
    def admin_panel():
        """Главное меню админ-панели."""
        builder = InlineKeyboardBuilder()

        builder.button(text="📋 Управление заказами", callback_data="admin_orders_menu")
        builder.button(text="✚ Добавить товар", callback_data="admin_add_product")
        builder.button(text="🗑 Закончившиеся товары", callback_data="catalog_oos")
        builder.button(text="👤 Добавить администратора", callback_data="add_admin")
        builder.button(text="🏠 Главное меню", callback_data="main_menu")
        builder.adjust(1)
        return builder.as_markup()

    @staticmethod
    def confirm_admin_kb():
        builder = InlineKeyboardBuilder()

        builder.button(text="✅ Да, уверен", callback_data="confirm_add_admin_yes")
        builder.button(text="❌ Отмена", callback_data="confirm_add_admin_no")
        builder.adjust(2)
        return builder.as_markup()

    @staticmethod
    def back_to_admin_panel():
        """Простая кнопка "Назад в панель"."""
        builder = InlineKeyboardBuilder()

        builder.button(text="⬅ Назад в панель", callback_data="admin_panel")
        return builder.as_markup()

    @staticmethod
    def edit_product_menu(product_id, source="catalog"):
        builder = InlineKeyboardBuilder()

        builder.button(text="🖼 Заменить фото", callback_data=f"edit_photos_{product_id}")
        builder.button(text="📄 Изменить название", callback_data=f"edit_name_{product_id}")
        builder.button(text="📖 Изменить описание", callback_data=f"edit_desc_{product_id}")
        builder.button(text="💰 Изменить цену", callback_data=f"edit_price_{product_id}")
        builder.button(text="📦 Изменить кол-во", callback_data=f"edit_stock_{product_id}")

```

```

        builder.button(text="🗑 Удалить товар", callback_data=f"confirm_delete_{product_id}")
        back_callback = "catalog_oos" if source == "oos" else "catalog"
        builder.button(text="⬅ Назад", callback_data=back_callback)
        builder.adjust(1)
        return builder.as_markup()
    @staticmethod
    def add_product_navigation(step):

        builder = InlineKeyboardBuilder()
        builder.button(text="⬅ Назад", callback_data=f"back_to_{step}")
        builder.button(text="❌ Отмена", callback_data="cancel_add_product")
        builder.adjust(2)
        return builder.as_markup()
    @staticmethod
    def admin_orders_menu():
        """Меню категорий заказов в админке (по статусам)."""
        builder = InlineKeyboardBuilder()
        builder.button(text="🆕 Новые", callback_data="admin_orders_list_оформлен_1")
        builder.button(text="📦 Собранные", callback_data="admin_orders_list_собран_1")
        builder.button(text="🚚 Отправленные", callback_data="admin_orders_list_отправлен_1")
        builder.button(text="⬅ Назад", callback_data="admin_panel")
        builder.adjust(1)
        return builder.as_markup()
    @staticmethod
    def admin_order_details(order_id, current_status, back_status, back_page):
        builder = InlineKeyboardBuilder()

        if current_status == "оформлен":
            builder.button(text="⚙ Собрать заказ",
callback_data=f"admin_confirm_st_{order_id}_собран_{back_status}_{back_page}")
            elif current_status == "собран":
                builder.button(text="🚚 Отправить (ввести трек)",
callback_data=f"admin_order_settrack_{order_id}_{back_status}_{back_page}")
                elif current_status == "отправлен":
                    builder.button(text="✅ Завершить (Доставлен)",
callback_data=f"admin_confirm_st_{order_id}_доставлен_{back_status}_{back_page}")
                    builder.button(text="✏ Изменить трек-номер",
callback_data=f"admin_order_settrack_{order_id}_{back_status}_{back_page}")
                    builder.button(text="⬅ Назад",
callback_data=f"admin_orders_list_{back_status}_{back_page}")
                    builder.adjust(1)
                    return builder.as_markup()
    @staticmethod
    def admin_confirm_status_kb(order_id, next_status, back_status, back_page):
        builder = InlineKeyboardBuilder()
        builder.button(text="✅ Да, изменить",
callback_data=f"admin_order_setst_{order_id}_{next_status}_{back_status}_{back_page}")
        builder.button(text="❌ Отмена",
callback_data=f"admin_order_info_{order_id}_{back_status}_{back_page}")

```

```

builder.adjust(1)
return builder.as_markup()

```

Файл database.py

```

import aiosqlite
import datetime

class Database:

    def __init__(self, db_name="Data_Base.db"):
        self.db_name = db_name
        self.conn = None

    async def connect(self):
        if not self.conn:
            self.conn = await aiosqlite.connect(self.db_name)

    async def close(self):
        if self.conn:
            await self.conn.close()

    async def create_tables(self):

        await self.conn.execute('''
            CREATE TABLE IF NOT EXISTS "user" (
                "id" INTEGER PRIMARY KEY AUTOINCREMENT,
                "user_id" INTEGER NOT NULL UNIQUE,
                "username" TEXT,
                "address" TEXT,
                "is_admin" INTEGER NOT NULL DEFAULT 0,
                "phone" TEXT,
                "last_name" TEXT,
                "first_name" TEXT,
                "middle_name" TEXT,
                "messages_history" TEXT DEFAULT '[]'
            )''')

        await self.conn.execute('''
            CREATE TABLE IF NOT EXISTS "product" (
                "id" INTEGER PRIMARY KEY AUTOINCREMENT,
                "name" TEXT NOT NULL,
                "price" INTEGER NOT NULL,
                "description" TEXT,
                "stock" INTEGER DEFAULT 0,
                "is_available" INTEGER DEFAULT 1
            )''')

        await self.conn.execute('''
            CREATE TABLE IF NOT EXISTS "product_images" (
                "id" INTEGER PRIMARY KEY AUTOINCREMENT,
                "product_id" INTEGER NOT NULL,
                "file_id" TEXT NOT NULL,
                FOREIGN KEY("product_id") REFERENCES "product"("id") ON DELETE CASCADE
            )''')

        await self.conn.execute('''
            CREATE TABLE IF NOT EXISTS "cart" (
                "id" INTEGER PRIMARY KEY AUTOINCREMENT,
                "user_id" INTEGER NOT NULL,
                "product_id" INTEGER NOT NULL,

```

					ПГУ1.090301.09.001 ПЗ	Лист
						114
Изм.	Лист	№ докум.	Подпись	Дата		

```

        "quantity" INTEGER NOT NULL DEFAULT 1
    )'''
await self.conn.execute('''
    CREATE TABLE IF NOT EXISTS "orders" (
        "id" INTEGER PRIMARY KEY AUTOINCREMENT,
        "user_id" INTEGER NOT NULL,
        "status" TEXT DEFAULT 'оформлен',
        "address" TEXT NOT NULL,
        "phone" TEXT,
        "track_number" TEXT,
        "date_create" TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )'''
await self.conn.execute('''
    CREATE TABLE IF NOT EXISTS "order_items" (
        "id" INTEGER PRIMARY KEY AUTOINCREMENT,
        "order_id" INTEGER NOT NULL,
        "product_id" INTEGER NOT NULL,
        "quantity" INTEGER NOT NULL,
        "price_at_purchase" INTEGER NOT NULL,
        FOREIGN KEY("order_id") REFERENCES "orders"("id") ON DELETE CASCADE
    )'''
await self.conn.commit()
async def add_product(self, name, price, description, stock, photos):
    cursor = await self.conn.execute(
        "INSERT INTO product (name, price, description, stock) VALUES (?, ?, ?, ?)",
        (name, price, description, stock)
    )
    product_id = cursor.lastrowid
    for f_id in photos:
        await self.conn.execute("INSERT INTO product_images (product_id, file_id) VALUES
(?, ?)", (product_id, f_id))
        await self.conn.commit()
    return product_id
async def update_product_field(self, product_id, field, value):
    await self.conn.execute(f"UPDATE product SET {field} = ? WHERE id = ?", (value,
product_id))
    await self.conn.commit()
async def delete_product(self, product_id):
    await self.conn.execute("UPDATE product SET is_available = 0 WHERE id = ?",
(product_id,))
    await self.conn.commit()
async def clear_product_photos(self, product_id):
    await self.conn.execute("DELETE FROM product_images WHERE product_id = ?",
(product_id,))
    await self.conn.commit()
async def add_product_photo(self, product_id, file_id):
    await self.conn.execute("INSERT INTO product_images (product_id, file_id) VALUES (?,
?)", (product_id, file_id))
    await self.conn.commit()
async def get_products_page(self, page, per_page=4):
    offset = (page - 1) * per_page
    async with self.conn.execute("""

```

```

        SELECT p.id, p.name, p.price, p.description, p.stock,
        (SELECT file_id FROM product_images WHERE product_id = p.id LIMIT 1) as photo
        FROM product p
        WHERE p.is_available = 1 AND p.stock > 0
        ORDER BY p.id DESC
        LIMIT ? OFFSET ? (per_page, offset)) as cursor:
        return await cursor.fetchall()

    async def get_product_by_id(self, product_id):
        async with self.conn.execute("SELECT id, name, price, description, stock FROM product
        WHERE id = ?", (product_id,)) as cursor:
            product = await cursor.fetchone()

            async with self.conn.execute("SELECT file_id FROM product_images WHERE product_id =
            ?", (product_id,)) as cursor:
                rows = await cursor.fetchall()
                photos = [row[0] for row in rows]

            return product, photos

    async def get_products_count(self):

        async with self.conn.execute("SELECT COUNT(*) FROM product WHERE is_available = 1 AND
        stock > 0") as cursor:
            res = await cursor.fetchone()
            return res[0]

    async def get_oos_products_page(self, page, per_page=4):
        offset = (page - 1) * per_page
        async with self.conn.execute("""
            SELECT p.id, p.name, p.price, p.description, p.stock,
            (SELECT file_id FROM product_images WHERE product_id = p.id LIMIT 1) as photo
            FROM product p
            WHERE p.is_available = 1 AND p.stock = 0
            ORDER BY p.id DESC
            LIMIT ? OFFSET ?

            async with self.conn.execute("SELECT COUNT(*) FROM product WHERE is_available = 1 AND
            stock = 0") as cursor:
                res = await cursor.fetchone()
                return res[0]

        async def is_admin(self, user_id):
            async with self.conn.execute("SELECT is_admin FROM user WHERE user_id = ?",
            (user_id,)) as cursor:
                res = await cursor.fetchone()
                return res and res[0] == 1

        async def get_or_create_user(self, user_id, username=None):
            async with self.conn.execute("SELECT * FROM user WHERE user_id = ?", (user_id,)) as
            cursor:
                user = await cursor.fetchone()

                if not user:
                    await self.conn.execute("INSERT INTO user (user_id, username) VALUES (?, ?)",
                    (user_id, username))
                    await self.conn.commit()

                    async with self.conn.execute("SELECT * FROM user WHERE user_id = ?", (user_id,))
                    as cursor:
                        user = await cursor.fetchone()

                return user

```

```

async def update_user_fio(self, user_id, last_name, first_name, middle_name):
    await self.conn.execute(
        "UPDATE user SET last_name = ?, first_name = ?, middle_name = ? WHERE user_id =
?",
        (last_name, first_name, middle_name, user_id)
    )
    await self.conn.commit()

async def get_user_data(self, user_id):
    async with self.conn.execute(
        "SELECT username, phone, address, last_name, first_name, middle_name FROM user
WHERE user_id = ?",
        (user_id,)
    ) as cursor:
        return await cursor.fetchone()

async def update_user_phone(self, user_id, phone):

    await self.conn.execute("UPDATE user SET phone = ? WHERE user_id = ?", (phone,
user_id))
    await self.conn.commit()

async def update_user_address(self, user_id, address):
    await self.conn.execute("UPDATE user SET address = ? WHERE user_id = ?", (address,
user_id))
    await self.conn.commit()

async def get_user_by_username(self, username):
    async with self.conn.execute(
        "SELECT user_id, username FROM user WHERE username = ?", (username,)
    ) as cursor:
        res = await cursor.fetchone()
        if res:
            return {"user_id": res[0], "username": res[1]}
        return None

async def set_admin_status(self, user_id, status: int):
    await self.conn.execute(
        "UPDATE user SET is_admin = ? WHERE user_id = ?", (status, user_id)
    )
    await self.conn.commit()

async def get_admins_list(self):
    async with self.conn.execute("SELECT username FROM user WHERE is_admin = 1") as
cursor:
        rows = await cursor.fetchall()
        return [row[0] for row in rows if row[0]]

async def add_to_cart(self, user_id, product_id, quantity=1):
    in_cart = await self.check_in_cart(user_id, product_id)
    if not in_cart:
        await self.conn.execute("INSERT INTO cart (user_id, product_id, quantity) VALUES
(?, ?, ?)", (user_id, product_id, quantity))
        await self.conn.commit()

async def remove_from_cart(self, user_id, product_id):
    await self.conn.execute("DELETE FROM cart WHERE user_id = ? AND product_id = ?",
(user_id, product_id))

```

```

        await self.conn.commit()
    async def check_in_cart(self, user_id, product_id):
        async with self.conn.execute("SELECT 1 FROM cart WHERE user_id = ? AND product_id =
?", (user_id, product_id)) as cursor:
            res = await cursor.fetchone()
            return res is not None
    async def get_user_cart(self, user_id):

        query = """
            SELECT p.id, p.name, p.price, c.quantity, p.stock
            FROM cart c
            JOIN product p ON c.product_id = p.id
            WHERE c.user_id = ?
        """
        async with self.conn.execute(query, (user_id,)) as cursor:
            return await cursor.fetchall()
    async def get_cart_count(self, user_id):
        async with self.conn.execute("SELECT COUNT(*) FROM cart WHERE user_id = ?",
(user_id,)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res else 0

    async def get_cart_page(self, user_id, page, limit):
        offset = (page - 1) * limit
        sql = """
            SELECT p.id, p.name, p.price, p.description,
                (SELECT file_id FROM product_images WHERE product_id = p.id LIMIT 1) as
photo,
                p.stock, c.quantity
            FROM cart c
            JOIN product p ON c.product_id = p.id
            WHERE c.user_id = ?
            LIMIT ? OFFSET ?
        """
        async with self.conn.execute(sql, (user_id, limit, offset)) as cursor:
            return await cursor.fetchall()
    async def update_cart_quantity(self, user_id, product_id, quantity):
        await self.conn.execute(
            "UPDATE cart SET quantity = ? WHERE user_id = ? AND product_id = ?",
            (quantity, user_id, product_id))
        await self.conn.commit()
    async def get_cart_item_quantity(self, user_id, product_id):
        async with self.conn.execute("SELECT quantity FROM cart WHERE user_id = ? AND
product_id = ?", (user_id, product_id)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res else 1
    async def create_order(self, user_id, address, phone):

        now_str = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        cursor = await self.conn.execute(
            "INSERT INTO orders (user_id, address, phone, status, date_create) VALUES (?, ?,
?, 'оформлен', ?)",

```

					ПГУ1.090301.09.001 ПЗ	Лист
						118
Изм.	Лист	№ докум.	Подпись	Дата		

```

        (user_id, address, phone, now_str)
    )
    await self.conn.commit()
    return cursor.lastrowid
async def add_order_item(self, order_id, product_id, quantity, price_at_purchase):
    await self.conn.execute(
        "INSERT INTO order_items (order_id, product_id, quantity, price_at_purchase)
VALUES (?, ?, ?, ?)",
        (order_id, product_id, quantity, price_at_purchase)
    )
    await self.conn.commit()

async def decrease_product_stock(self, product_id, quantity=1):
    await self.conn.execute(
        "UPDATE product SET stock = stock - ? WHERE id = ? AND stock >= ?",
        (quantity, product_id, quantity)
    )
    await self.conn.commit()

async def get_user_orders_count(self, user_id, status=None, exclude_status=None):
    query = "SELECT COUNT(*) FROM orders WHERE user_id = ?"
    params = [user_id]
    if status:
        query += " AND status = ?"
        params.append(status)
    if exclude_status:
        query += " AND status != ?"
        params.append(exclude_status)
    async with self.conn.execute(query, params) as cursor:
        res = await cursor.fetchone()
        return res[0] if res else 0

async def get_user_orders_page(self, user_id, status=None, exclude_status=None, page=1,
limit=5):
    offset = (page - 1) * limit
    query = "SELECT id, status, date_create FROM orders WHERE user_id = ?"
    params = [user_id]
    if status:
        query += " AND status = ?"
        params.append(status)
    if exclude_status:
        query += " AND status != ?"
        params.append(exclude_status)
    query += " ORDER BY id DESC LIMIT ? OFFSET ?"
    params.extend([limit, offset])
    async with self.conn.execute(query, params) as cursor:
        return await cursor.fetchall()

async def get_order_details(self, order_id):

    async with self.conn.execute("""
        SELECT o.id, o.status, o.address, o.phone, o.track_number, o.date_create,
        u.username, u.user_id, u.last_name, u.first_name, u.middle_name

```

```

        FROM orders o
        JOIN user u ON o.user_id = u.user_id
        WHERE o.id = ?
    """
    (order_id,)) as cursor:
        order_info = await cursor.fetchone()
    async with self.conn.execute("""
        SELECT p.name, oi.quantity, oi.price_at_purchase
        FROM order_items oi
        JOIN product p ON oi.product_id = p.id
        WHERE oi.order_id = ?
    """, (order_id,)) as cursor:
        items = await cursor.fetchall()
    return order_info, items

    async def get_all_orders_count(self, status=None):
        query = "SELECT COUNT(*) FROM orders"
        params = []
        if status and status != "all":
            query += " WHERE status = ?"
            params.append(status)
        async with self.conn.execute(query, params) as cursor:
            res = await cursor.fetchone()
            return res[0] if res else 0

    async def get_all_orders_page(self, status=None, page=1, limit=6):
        offset = (page - 1) * limit
        query = "SELECT o.id, o.status, o.date_create, u.username FROM orders o JOIN user u ON
o.user_id = u.user_id"
        params = []
        if status and status != "all":
            query += " WHERE o.status = ?"
            params.append(status)
        query += " ORDER BY o.id DESC LIMIT ? OFFSET ?"
        params.extend([limit, offset])
        async with self.conn.execute(query, tuple(params)) as cursor:
            return await cursor.fetchall()

    async def update_order_status(self, order_id, status):
        await self.conn.execute("UPDATE orders SET status = ? WHERE id = ?", (status,
order_id))
        await self.conn.commit()

    async def update_order_track(self, order_id, track):
        await self.conn.execute("UPDATE orders SET track_number = ? WHERE id = ?", (track,
order_id))
        await self.conn.commit()

```

Файл utils.py

```

import logging

from database import Database

db = Database()

user_messages_history = {}

    async def save_msg(user_id: int, msg_id: int):
        # Создаем/обновляем пользователя в БД (необходимо для корректных проверок)
        await db.get_or_create_user(user_id)

```

					ПГУ1.090301.09.001 ПЗ	Лист
						120
Изм.	Лист	№ докум.	Подпись	Дата		

```

if user_id not in user_messages_history:
    user_messages_history[user_id] = []
    if msg_id not in user_messages_history[user_id]:
        user_messages_history[user_id].append(msg_id)
async def clear_chat(bot, user_id: int, keep_last_n: int = 0):
    history = user_messages_history.get(user_id, [])
    if not history:
        return
    messages_to_delete = history[:-keep_last_n] if keep_last_n > 0 else history
    messages_to_keep = history[-keep_last_n:] if keep_last_n > 0 else []
    if messages_to_delete:
        for i in range(0, len(messages_to_delete), 100):
            chunk = messages_to_delete[i:i + 100]
            try:
                await bot.delete_messages(chat_id=user_id, message_ids=chunk)
            except Exception as e:
                logging.error(f"Ошибка при удалении сообщений у {user_id}: {e}")
    user_messages_history[user_id] = messages_to_keep
def format_price(price_in_kopecks):
    return f"{price_in_kopecks/100:.2f}"
def parse_price(price_str):
    try:
        price_str = price_str.replace(',', '.').replace(' ', '')
        price = float(price_str)
        return int(round(price * 100))
    except ValueError:
        return None

```

Файл catalog.py

```

from aiogram import Router, F, types
from aiogram.utils.media_group import MediaGroupBuilder
from utils import db, clear_chat, save_msg, format_price
from keyboards import UserKeyboards, AdminKeyboards
from aiogram.fsm.context import FSMContext
catalog_router = Router()
PRODUCTS_PER_PAGE = 4 # Количество карточек товаров на одной странице каталога
@catalog_router.callback_query(F.data == "catalog")
@catalog_router.callback_query(F.data.startswith("catalogpage_"))
async def show_catalog(event: types.Message | types.CallbackQuery, state: FSMContext):
    user_id = event.from_user.id
    bot = event.bot
    message = event.message if isinstance(event, types.CallbackQuery) else event
    data = await state.get_data()
    page = data.get("catalog_page", 1)
    if isinstance(event, types.CallbackQuery) and event.data.startswith("catalogpage_"):
        page = int(event.data.split("_")[1])
        await state.update_data(catalog_page=page)
    await clear_chat(bot, user_id)
    is_admin = await db.is_admin(user_id)
    total_products = await db.get_products_count()
    if total_products == 0:

```

```

msg = await message.answer("🛒 Каталог пока пуст.",
reply_markup=UserKeyboards.main_menu(is_admin))
    await save_msg(user_id, msg.message_id)
else:
    products = await db.get_products_page(page, PRODUCTS_PER_PAGE)
    total_pages = (total_products + PRODUCTS_PER_PAGE - 1)
    for p_id, name, price, desc, stock, photo in products:
        caption = f"<b>{name}</b>\n💰 {format_price(price)} руб."
        in_cart = await db.check_in_cart(user_id, p_id)
        kb = UserKeyboards.product_preview_kb(product_id=p_id, is_admin=is_admin,
in_cart=in_cart, stock=stock)
        if photo:
            msg = await message.answer_photo(photo, caption=caption, parse_mode="HTML",
reply_markup=kb)
        else:
            msg = await message.answer(caption, parse_mode="HTML", reply_markup=kb)
            await save_msg(user_id, msg.message_id)
    nav_msg = await message.answer(
        f"📖 Страница {page} из {total_pages}",
        reply_markup=UserKeyboards.catalog_pagination(page, total_pages)
    )
    await save_msg(user_id, nav_msg.message_id)
if isinstance(event, types.CallbackQuery):
    await event.answer()

async def send_product_card(bot, user_id, product_id, is_admin=False, is_editing=False,
source="catalog"):
    res = await db.get_product_by_id(product_id)
    if not res or not res[0]:
        return None
    product, photos = res[0], res[1]
    p_id, name, price, desc, stock = product
    text = (f"🛒 <b>{name}</b>\n\n"
        f"{desc}\n\n"
        f"💰 Цена: {format_price(price)} руб.\n"
        f"📦 На складе: {stock} шт.")
    if is_editing:
        kb = AdminKeyboards.edit_product_menu(p_id, source=source)
    else:
        in_cart = await db.check_in_cart(user_id, p_id)
        kb = UserKeyboards.product_full_card_kb(p_id, is_admin, in_cart, source)
    if photos:
        media = []
        media.append(types.InputMediaPhoto(media=photos[0], caption=text, parse_mode="HTML"))
        for ph in photos[1:]:
            media.append(types.InputMediaPhoto(media=ph))
        msgs = await bot.send_media_group(chat_id=user_id, media=media)
        for m in msgs:
            await save_msg(user_id, m.message_id)
        kb_msg = await bot.send_message(user_id, "<b>Выберите действие:</b>",
parse_mode="HTML", reply_markup=kb)
        await save_msg(user_id, kb_msg.message_id)

```

```

else:
    msg = await bot.send_message(user_id, text, parse_mode="HTML", reply_markup=kb)
    await save_msg(user_id, msg.message_id)
@catalog_router.callback_query(F.data.startswith("view_"))
async def view_product(callback: types.CallbackQuery):
    user_id = callback.from_user.id
    parts = callback.data.split("_")
    if "cart" in parts:
        product_id = int(parts[2])
        source = "cart"
    elif "oos" in parts:
        product_id = int(parts[2])
        source = "oos"
    else:
        product_id = int(parts[1])
        source = "catalog"

    await clear_chat(callback.bot, user_id)
    is_admin = await db.is_admin(user_id)
    await send_product_card(callback.bot, user_id, product_id, is_admin, source=source)
    await callback.answer()
@catalog_router.callback_query(F.data == "catalog_oos")
@catalog_router.callback_query(F.data.startswith("oospage_"))
async def show_oos_catalog(event: types.Message | types.CallbackQuery, state: FSMContext):
    user_id = event.from_user.id
    bot = event.bot
    message = event.message if isinstance(event, types.CallbackQuery) else event
    data = await state.get_data()
    page = data.get("oos_page", 1)
    if isinstance(event, types.CallbackQuery) and event.data.startswith("oospage_"):
        page = int(event.data.split("_")[1])
        await state.update_data(oos_page=page)
    await clear_chat(bot, user_id)
    is_admin = await db.is_admin(user_id)

    if not is_admin:
        return
    total_products = await db.get_oos_products_count()
    if total_products == 0:
        msg = await message.answer("✅ Закончившихся товаров нет.",
reply_markup=AdminKeyboards.back_to_admin_panel())
        await save_msg(user_id, msg.message_id)
    else:
        products = await db.get_oos_products_page(page, PRODUCTS_PER_PAGE)
        total_pages = (total_products + PRODUCTS_PER_PAGE - 1) // PRODUCTS_PER_PAGE
        for p_id, name, price, desc, stock, photo in products:
            caption = f"⚠️ <b>{name}</b>\n💰 {format_price(price)} руб.\n🚫 Нет в наличии"
            kb = UserKeyboards.product_preview_kb(p_id, is_admin, in_cart=False, stock=stock)
            if photo:
                msg = await message.answer_photo(photo, caption=caption, parse_mode="HTML",
reply_markup=kb)

```

```

        else:
            msg = await message.answer(caption, parse_mode="HTML", reply_markup=kb)
            await save_msg(user_id, msg.message_id)
            nav_msg = await message.answer(
                f"📄 Страница {page} из {total_pages} (Закончившиеся)",
                reply_markup=UserKeyboards.catalog_pagination(page, total_pages, prefix="oospace")
            )
            await save_msg(user_id, nav_msg.message_id)
    if isinstance(event, types.CallbackQuery):
        await event.answer()

```

Файл cart.py

```

from aiogram import Router, F, types
from utils import db, clear_chat, save_msg, format_price
from keyboards import UserKeyboards
from aiogram.fsm.context import FSMContext
from aiogram.types import LabeledPrice
from aiogram.utils.keyboard import InlineKeyboardBuilder

cart_router = Router()
CART_PER_PAGE = 4

@cart_router.callback_query(F.data.startswith("add_prev_"))
@cart_router.callback_query(F.data.startswith("add_full_"))
@cart_router.callback_query(F.data.startswith("rm_prev_"))
@cart_router.callback_query(F.data.startswith("rm_full_"))
async def toggle_cart_item(callback: types.CallbackQuery):
    parts = callback.data.split("_")
    action = parts[0]
    context = parts[1]
    source = parts[2]
    p_id = int(parts[3])
    user_id = callback.from_user.id
    if action == "add":
        await db.add_to_cart(user_id, p_id)
        await callback.answer("✅ Товар добавлен в корзину!")
        in_cart = True
    else:
        await db.remove_from_cart(user_id, p_id)
        await callback.answer("❌ Товар удален из корзины!")
        in_cart = False
    is_admin = await db.is_admin(user_id)
    if context == "prev":
        kb = UserKeyboards.product_preview_kb(p_id, is_admin, in_cart)
    else:
        kb = UserKeyboards.product_full_card_kb(p_id, is_admin, in_cart, source)
    try:
        await callback.message.edit_reply_markup(reply_markup=kb)
    except Exception:
        pass

@cart_router.callback_query(F.data.startswith("cart_plus_"))
async def cart_plus(callback: types.CallbackQuery, state: FSMContext):
    p_id = int(callback.data.split("_")[2])
    user_id = callback.from_user.id

```

```

product, _ = await db.get_product_by_id(p_id)
if not product:
    return await callback.answer("Товар не найден")
name, price, stock = product[1], product[2], product[4]
current_qty = await db.get_cart_item_quantity(user_id, p_id)
if current_qty >= stock:
    return await callback.answer(f"⚠ Больше нет! На складе всего {stock} шт.",
show_alert=True)
new_qty = current_qty + 1
await db.update_cart_quantity(user_id, p_id, new_qty)
await update_cart_item_message(callback, state, p_id, name, price, stock, new_qty)
await callback.answer()
@cart_router.callback_query(F.data.startswith("cart_minus_"))
async def cart_minus(callback: types.CallbackQuery, state: FSMContext):
    p_id = int(callback.data.split("_")[2])
    user_id = callback.from_user.id
    product, _ = await db.get_product_by_id(p_id)
    if not product:
        return
    name, price, stock = product[1], product[2], product[4]
    current_qty = await db.get_cart_item_quantity(user_id, p_id)
    if current_qty > 1:
        new_qty = current_qty - 1
        await db.update_cart_quantity(user_id, p_id, new_qty)
        await update_cart_item_message(callback, state, p_id, name, price, stock, new_qty)
        await callback.answer()
    else:
        await callback.answer("Минимум 1 шт. Нажмите 'Удалить', чтобы убрать товар.")
async def update_cart_item_message(callback, state, p_id, name, price, stock, quantity):
    data = await state.get_data()
    selected_items = data.get("selected_items", []) # Товары, выбранные к оплате
    is_selected = p_id in selected_items
    caption = (
        f"🛒 <b>{name}</b>\n"
        f"💰 Цена: {format_price(price)} руб.\n"
        f"🛒 В корзине: {quantity} шт. (Склад: {stock})\n"
        f"💰 Итого за позицию: {format_price(price * quantity)} руб."
    )
    if is_selected:
        caption += "\n✅ <b>Выбран для оплаты</b>"
    kb = UserKeyboards.cart_item_kb(p_id, quantity=quantity, is_selected=is_selected)
    try:
        # Редактируем caption (для фото) или текст (для обычного сообщения)
        if callback.message.photo:
            await callback.message.edit_caption(caption=caption, parse_mode="HTML",
reply_markup=kb)
        else:
            await callback.message.edit_text(text=caption, parse_mode="HTML", reply_markup=kb)
    except Exception:
        pass # Игнорируем ошибку, если текст не изменился

```

```

@cart_router.callback_query(F.data == "cart")
@cart_router.callback_query(F.data.startswith("cartpage_"))
async def show_cart(event: types.CallbackQuery, state: FSMContext):
    user_id = event.from_user.id
    bot = event.bot
    data = await state.get_data()
    page = data.get("cart_page", 1)
    selected_items = data.get("selected_items", [])
    if event.data.startswith("cartpage_"):
        page = int(event.data.split("_")[1])
        await state.update_data(cart_page=page)
    await clear_chat(bot, user_id)
    total_items = await db.get_cart_count(user_id)
    if total_items == 0:
        msg = await event.message.answer(
            "🛒 <b>Ваша корзина пуста.</b>",
            parse_mode="HTML",
            reply_markup=UserKeyboards.empty_cart_kb()
        )
        await save_msg(user_id, msg.message_id)
        return await event.answer()
    total_pages = max((total_items + CART_PER_PAGE - 1) // CART_PER_PAGE, 1)
    if page > total_pages:
        page = total_pages
        await state.update_data(cart_page=page)
    products = await db.get_cart_page(user_id, page, CART_PER_PAGE)
    for p_id, name, price, desc, photo, stock, quantity in products:
        is_oos = stock <= 0 # Флаг "нет в наличии"
        if not is_oos and quantity > stock:
            quantity = stock
            await db.update_cart_quantity(user_id, p_id, quantity)
        is_selected = p_id in selected_items and not is_oos
        if is_oos and p_id in selected_items:
            selected_items.remove(p_id)
            await state.update_data(selected_items=selected_items)
        caption = (
            f"🛒 <b>{name}</b>\n"
            f"💰 Цена: {format_price(price)} руб.\n"
            f"📦 В корзине: {quantity} шт. (Склад: {stock})\n"
            f"💵 Итого за позицию: {format_price(price * quantity)} руб."
        )
        if is_oos:
            caption += "\n⚠️ <b>ТОВАР ЗАКОНЧИЛСЯ</b>"
        elif is_selected:
            caption += "\n✅ <b>Выбран для оплаты</b>"
        kb = UserKeyboards.cart_item_kb(p_id, quantity=quantity, is_selected=is_selected,
is_oos=is_oos)
        if photo:
            msg = await event.message.answer_photo(photo, caption=caption, parse_mode="HTML",
reply_markup=kb)

```

```

        else:
            msg = await event.message.answer(caption, parse_mode="HTML", reply_markup=kb)
            await save_msg(user_id, msg.message_id)
        bottom_text = f"🛒 Корзина: Страница {page} из {total_pages}\n"
        if selected_items:
            bottom_text += f"💎 Выбрано для оплаты: {len(selected_items)} поз."
        else:
            bottom_text += "Выберите товары для оплаты:"
        bot_msg = await event.message.answer(
            bottom_text,
            reply_markup=UserKeyboards.cart_menu(page, total_pages)
        )
        await save_msg(user_id, bot_msg.message_id)
        await event.answer()
@cart_router.callback_query(F.data.startswith("rm_cart_"))
async def remove_from_cart_immediately(callback: types.CallbackQuery, state: FSMContext):
    product_id = int(callback.data.split("_")[2])
    user_id = callback.from_user.id
    data = await state.get_data()
    selected_items = data.get("selected_items", [])
    if product_id in selected_items:
        selected_items.remove(product_id)
        await state.update_data(selected_items=selected_items)
    await db.remove_from_cart(user_id, product_id)
    try:
        await callback.message.delete()
    except Exception:
        pass
    await callback.answer("❌ Удалено")
    await show_cart(callback, state)
@cart_router.callback_query(F.data == "pay_cart")
async def process_payment(callback: types.CallbackQuery, state: FSMContext):
    user_id = callback.from_user.id
    data = await state.get_data()
    selected_items = data.get("selected_items", [])
    if not selected_items:
        return await callback.answer("⚠️ Вы не выбрали товары!", show_alert=True)
    user_data = await db.get_user_data(user_id)
    _, phone, address, l_name, f_name, m_name = user_data
    missing = []
    if not phone or phone == "❌ Не указан": missing.append("📞 Телефон")
    if not address or address == "❌ Не указан": missing.append("📍 Адрес")
    if not all([l_name, f_name, m_name]): missing.append("👤 ФИО")
    if missing:
        await callback.answer(
            f"⚠️ Заполните данные в профиле:\n" + "\n".join(missing),
            show_alert=True
        )
    return

```

```

prices = []
out_of_stock_names = []
cart_items = await db.get_user_cart(user_id)
for p_id, name, price, quantity, stock in cart_items:
    if p_id in selected_items:
        if quantity > stock:
            out_of_stock_names.append(name)
        else:
            prices.append(LabeledPrice(label=f"{name} x{quantity}", amount=price *
quantity))
    if out_of_stock_names:
        await callback.answer(
            "⚠ Недостаточно товара на складе. Корзина обновлена.",
            show_alert=True
        )
    return await show_cart(callback, state)
if not prices:
    return await callback.answer("⚠ Нет товаров для оплаты.", show_alert=True)
await callback.answer("🧮 Генерируем счет...")
payload = f"order_{user_id}_" + "_".join(map(str, selected_items))
PAYMENT_TOKEN = "1744374395:TEST:00f5f5ce6eb6065d7c0d" # Тестовый токен
try:
    msg = await callback.bot.send_invoice(
        chat_id=user_id,
        title="Оплата заказа",
        description=f"Оплата товаров ({len(selected_items)} поз.)",
        payload=payload,
        provider_token=PAYMENT_TOKEN,
        currency="RUB",
        prices=prices,
        start_parameter="cart_payment"
    )
    await save_msg(user_id, msg.message_id)
except Exception:
    msg = await callback.message.answer("❌ Ошибка платежной системы.")
    await save_msg(user_id, msg.message_id)
@cart_router.pre_checkout_query()
async def process_pre_checkout_query(pre_checkout_query: types.PreCheckoutQuery):
    payload = pre_checkout_query.invoice_payload
    parts = payload.split("_")
    user_id = int(parts[1])
    paid_ids = [int(i) for i in parts[2:]]
    cart_items = await db.get_user_cart(user_id)
    cart_dict = {p_id: (quantity, stock) for p_id, _, _, quantity, stock in cart_items}
    for p_id in paid_ids:
        if p_id not in cart_dict or cart_dict[p_id][0] > cart_dict[p_id][1]:
            return await pre_checkout_query.answer(
                ok=False,
                error_message="Ой! Один из товаров закончился или его количество изменилось."
            )
    await pre_checkout_query.answer(ok=True)

```

```

@cart_router.message(F.successful_payment)
async def on_success_payment(message: types.Message, state: FSMContext):
    user_id = message.from_user.id
    payment = message.successful_payment
    parts = payment.invoice_payload.split("_")
    paid_ids = [int(i) for i in parts[2:]]
    user_info = await db.get_user_data(user_id)
    address = user_info[2] if user_info[2] else 'Не указан'
    phone = user_info[1] if user_info[1] else 'Не указан'
    order_id = await db.create_order(user_id, address, phone)
    cart_items = await db.get_user_cart(user_id)
    cart_dict = {p_id: quantity for p_id, _, quantity, _ in cart_items}
    for p_id in paid_ids:
        product, _ = await db.get_product_by_id(p_id)
        if product:
            qty = cart_dict.get(p_id, 1)
            await db.add_order_item(order_id, p_id, qty, product[2])
            await db.decrease_product_stock(p_id, qty)
            await db.remove_from_cart(user_id, p_id)
    await clear_chat(message.bot, user_id)

    builder = InlineKeyboardBuilder()
    builder.button(text="🏠 В меню", callback_data="main_menu")

    msg = await message.answer(
        f"👋 <b>Оплачено!</b>\nСумма: {payment.total_amount / 100:.2f} руб.",
        parse_mode="HTML",
        reply_markup=builder.as_markup()
    )
    await save_msg(user_id, msg.message_id)

@cart_router.callback_query(F.data.startswith("select_pay_"))
@cart_router.callback_query(F.data.startswith("unselect_pay_"))
async def toggle_pay_selection(callback: types.CallbackQuery, state: FSMContext):
    parts = callback.data.split("_")
    action = parts[0] # "select" или "unselect"
    p_id = int(parts[2])
    data = await state.get_data()
    selected_items = data.get("selected_items", [])
    if action == "select" and p_id not in selected_items:
        selected_items.append(p_id)
        await callback.answer("✅ Добавлено к оплате")
    elif action == "unselect" and p_id in selected_items:
        selected_items.remove(p_id)
        await callback.answer("❌ Убрано")
    await state.update_data(selected_items=selected_items)
    qty = await db.get_cart_item_quantity(callback.from_user.id, p_id)
    kb = UserKeyboards.cart_item_kb(p_id, quantity=qty, is_selected=(p_id in selected_items))
    try:
        await callback.message.edit_reply_markup(reply_markup=kb)
    except Exception:

```

pass

Файл orders.py

```
from aiogram import Router, F, types
from utils import db, clear_chat, save_msg, format_price
from keyboards import UserKeyboards
from aiogram.utils.keyboard import InlineKeyboardBuilder
user_orders_router = Router()
ORDERS_PER_PAGE = 4
@user_orders_router.callback_query(F.data.in_(["my_orders", "my_orders_done"]))
@user_orders_router.callback_query(F.data.startswith("my_orders_p_"))
async def show_my_orders(callback: types.CallbackQuery):
    user_id = callback.from_user.id
    status_filter = "active" # По умолчанию активные
    page = 1
    if callback.data == "my_orders_done":
        status_filter = "доставлен"
    elif callback.data.startswith("my_orders_p_"):
        parts = callback.data.split("_")
        status_filter = parts[3]
        page = int(parts[4])
    await clear_chat(callback.bot, user_id)
    if status_filter == "доставлен":
        db_status = "доставлен"
        db_exclude = None
    else:
        db_status = None
        db_exclude = "доставлен" # Все, кроме доставленных
    total_orders = await db.get_user_orders_count(user_id, status=db_status,
exclude_status=db_exclude)
    total_pages = max((total_orders + ORDERS_PER_PAGE - 1) // ORDERS_PER_PAGE, 1)
    orders = await db.get_user_orders_page(user_id, status=db_status,
exclude_status=db_exclude, page=page, limit=ORDERS_PER_PAGE)
    if not orders:
        text = "🚫 У вас пока нет заказов в этой категории."
    else:
        title = "🛒 Ваши активные заказы" if status_filter == "active" else "✅ Доставленные заказы"
        text = f"<b>{title}</b>\nВыберите заказ для просмотра деталей."
        msg = await callback.message.answer(
            text,
            parse_mode="HTML",
            reply_markup=UserKeyboards.user_orders_list(orders, page, total_pages, status_filter)
        )
        await save_msg(user_id, msg.message_id)
        await callback.answer()
@user_orders_router.callback_query(F.data.startswith("my_order_info_"))
async def show_order_info(callback: types.CallbackQuery):
    user_id = callback.from_user.id
    parts = callback.data.split("_")
    order_id = int(parts[3])
    back_status = parts[4] if len(parts) > 4 else "active"
```

Изм.	Лист	№ докум.	Подпись	Дата

```

order_info, items = await db.get_order_details(order_id)
if not order_info:
    return await callback.answer("Заказ не найден.")
_, status, address, phone, track_number, date_create, *_ = order_info
await clear_chat(callback.bot, user_id)
text = f"📄 <b>Заказ #{order_id}</b>\n"
text += f"📅 Дата: {date_create}\n"
text += f"📌 Статус: <b>{status}</b>\n"
if status == "отправлен" and track_number:
    text += f"🚚 <b>Трек-номер:</b> <code>{track_number}</code>\n"
text += f"📍 Адрес доставки:\n{address}\n\n"
text += f"📦 <b>Товары:</b>\n"
total_sum = 0
for name, qty, price in items:
    total = qty * price
    total_sum += total
    text += f"📦 {name} x{qty} – {format_price(total)} руб.\n"
text += f"\n💰 <b>Итого: {format_price(total_sum)} руб.</b>"
builder = InlineKeyboardBuilder()
if back_status == "доставлен":
    back_callback = "my_orders_done"
else:
    back_callback = "my_orders"
builder.button(text="⬅️ Назад к списку", callback_data=back_callback)
msg = await callback.message.answer(
    text,
    parse_mode="HTML",
    reply_markup=builder.as_markup()
)
await save_msg(user_id, msg.message_id)
await callback.answer()

```

Файл profile.py

```

import re
import asyncio
from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from utils import db, clear_chat, save_msg
from keyboards import UserKeyboards
profile_router = Router()
class ProfileStates(StatesGroup):
    waiting_for_last_name = State()
    waiting_for_first_name = State()
    waiting_for_middle_name = State()
    waiting_for_phone = State()
    waiting_for_delivery_service = State()
    waiting_for_address = State()
async def render_profile(event: types.Message | types.CallbackQuery, user_id: int):
    user_data = await db.get_user_data(user_id)

```

```

u_obj = event.from_user
phone = user_data[1] if user_data and user_data[1] else "✗ Не указан"
address = user_data[2] if user_data and user_data[2] else "✗ Не указан"
l_name = user_data[3] if user_data and user_data[3] else ""
f_name = user_data[4] if user_data and user_data[4] else ""
m_name = user_data[5] if user_data and user_data[5] else ""
fio = f"{l_name} {f_name} {m_name}".strip()
if not fio:
    fio = "✗ Не указано"
text = (
    f"👤 <b>Ваш профиль</b>\n\n"
    f"<b>ФИО:</b> {fio}\n"
    f"<b>Телефон:</b> {phone}\n"
    f"<b>Адрес доставки:</b> {address}"
)
await clear_chat(event.bot, user_id)
msg = await event.bot.send_message(
    chat_id=user_id,
    text=text,
    parse_mode="HTML",
    reply_markup=UserKeyboards.profile_menu()
)
await save_msg(user_id, msg.message_id)
@profile_router.callback_query(F.data == "cancel_profile")
async def cancel_profile_action(callback: types.CallbackQuery, state: FSMContext):
    await callback.answer("Действие отменено")
    await state.clear()
    await render_profile(callback, callback.from_user.id)
@profile_router.callback_query(F.data == "profile")
async def show_profile(callback: types.CallbackQuery, state: FSMContext):
    await callback.answer()
    await state.clear()
    await render_profile(callback, callback.from_user.id)
@profile_router.callback_query(F.data == "prof_edit_fio")
async def start_fio_edit(callback: types.CallbackQuery, state: FSMContext):
    await clear_chat(callback.bot, callback.from_user.id)
    msg = await callback.message.answer("Введите вашу Фамилию:",
    reply_markup=UserKeyboards.cancel_profile())
    await save_msg(callback.from_user.id, msg.message_id)
    await state.set_state(ProfileStates.waiting_for_last_name)
@profile_router.message(ProfileStates.waiting_for_last_name)
async def process_last_name(message: types.Message, state: FSMContext):
    await state.update_data(last_name=message.text.strip())
    msg = await message.answer("Введите ваше Имя:")
    await save_msg(message.from_user.id, msg.message_id)
    await state.set_state(ProfileStates.waiting_for_first_name)
@profile_router.message(ProfileStates.waiting_for_first_name)
async def process_first_name(message: types.Message, state: FSMContext):
    await state.update_data(first_name=message.text.strip())
    msg = await message.answer("Введите ваше Отчество (если нет, поставьте прочерк):")
    await save_msg(message.from_user.id, msg.message_id)

```

```

        await state.set_state(ProfileStates.waiting_for_middle_name)
@profile_router.message(ProfileStates.waiting_for_middle_name)
async def process_middle_name(message: types.Message, state: FSMContext):
    try:
        await message.delete() # Удаляем сообщение пользователя с отчеством
    except:
        pass
    data = await state.get_data()
    l_name = data['last_name']
    f_name = data['first_name']
    m_name = message.text.strip()
    await db.update_user_fio(message.from_user.id, l_name, f_name, m_name)
    await state.clear()
    await render_profile(message, message.from_user.id)
@profile_router.callback_query(F.data == "update_phone")
async def edit_phone_start(callback: types.CallbackQuery, state: FSMContext):
    await callback.answer()
    await state.set_state(ProfileStates.waiting_for_phone)
    await callback.message.edit_text(
        "📞 Введите номер телефона (напр. +79991234567):",
        reply_markup=UserKeyboards.cancel_profile()
    )
    await state.update_data(prompt_msg_id=callback.message.message_id)
@profile_router.message(ProfileStates.waiting_for_phone)
async def process_phone(message: types.Message, state: FSMContext):
    user_id = message.from_user.id
    phone_text = message.text.strip()
    data = await state.get_data()
    prompt_msg_id = data.get("prompt_msg_id")
    try:
        await message.delete()
    except:
        pass
    if not re.match(r'^\+?[\d\s\-\]{10,20}$', phone_text):
        if prompt_msg_id:
            await message.bot.edit_message_text(
                "❌ Неверный формат. Попробуйте еще раз:\n📞 Введите номер телефона (напр. +79991234567):",
                chat_id=user_id,
                message_id=prompt_msg_id,
                reply_markup=UserKeyboards.cancel_profile()
            )
        return
    await db.update_user_phone(user_id, phone_text)
    await state.clear()
    if prompt_msg_id:
        await message.bot.edit_message_text(
            "✅ Телефон сохранен!",
            chat_id=user_id,
            message_id=prompt_msg_id
        )

```

```

        await render_profile(message, user_id)
@profile_router.callback_query(F.data == "update_address")
async def choose_delivery_method(callback: types.CallbackQuery, state: FSMContext):
    """Первый шаг: выбор службы доставки."""
    await callback.answer()
    await state.set_state(ProfileStates.waiting_for_delivery_service)
    await callback.message.edit_text(
        """📦 <b>Выбор доставки</b>\n\n
        Выберите службу доставки:\n\n
        📦 СДЕК\n
        📮 Почта России\n\n
        После выбора откроется карта с пунктами выдачи.",
        parse_mode="HTML",
        reply_markup=UserKeyboards.delivery_service_kb()
    )

@profile_router.callback_query(ProfileStates.waiting_for_delivery_service, F.data ==
"delivery_cdek")
async def delivery_cdek(callback: types.CallbackQuery, state: FSMContext):
    """Выбран СДЕК: предлагаем открыть карту и вставить адрес."""
    await callback.answer()
    await state.set_state(ProfileStates.waiting_for_address)
    await state.update_data(delivery_type="📦 СДЭК")
    await callback.message.edit_text(
        """📦 <b>СДЕК</b>\n\n
        1. Нажмите кнопку ниже\n
        2. Найдите удобный пункт выдачи\n
        3. Скопируйте его адрес\n
        4. Вставьте сюда\n\n
        Пример:\n
        Москва, ул. Ленина 10 (СДЭК)\n\n
        📍 Откройте карту:",
        parse_mode="HTML",
        reply_markup=UserKeyboards.delivery_map_kb("cdek")
    )

@profile_router.callback_query(ProfileStates.waiting_for_delivery_service, F.data ==
"delivery_post")
async def delivery_post(callback: types.CallbackQuery, state: FSMContext):
    """Выбрана Почта России: предлагаем открыть карту и вставить адрес."""
    await callback.answer()
    await state.set_state(ProfileStates.waiting_for_address)
    await state.update_data(delivery_type="📮 Почта России")
    await callback.message.edit_text(
        """📮 <b>Почта России</b>\n\n
        1. Нажмите кнопку ниже\n
        2. Найдите удобное отделение\n
        3. Скопируйте адрес\n
        4. Вставьте сюда\n\n
        Пример:\n
        Москва, ул. Пушкина 5 (Почта России)\n\n
        📍 Откройте карту:",

```

```

        parse_mode="HTML",
        reply_markup=UserKeyboards.delivery_map_kb("post")
    )
@profile_router.message(ProfileStates.waiting_for_address)
async def save_address(message: types.Message, state: FSMContext):
    user_id = message.from_user.id
    address_text = message.text.strip()
    data = await state.get_data()
    delivery_type = data.get("delivery_type", "")
    try:
        await message.delete()
    except:
        pass
    full_address = f"{delivery_type}: {address_text}"
    await db.update_user_address(user_id, full_address)
    await state.clear()
    msg = await message.answer(
        f"✅ <b>Адрес сохранен!</b>\n\n{full_address}",
        parse_mode="HTML"
    )
    await asyncio.sleep(2)
    try:
        await msg.delete()
    except:
        pass
    await render_profile(message, user_id)

```

Файл user_main.py

```

from aiogram import Router, F, types
from aiogram.filters import Command
from aiogram.fsm.context import FSMContext
from utils import db, clear_chat, save_msg
from keyboards import UserKeyboards
from aiogram.utils.keyboard import InlineKeyboardBuilder
user_router = Router()
@user_router.message(Command("start"))
async def cmd_start(message: types.Message):
    user_id = message.from_user.id
    await db.get_or_create_user(user_id, message.from_user.username)
    await clear_chat(message.bot, user_id)
    is_admin = await db.is_admin(user_id)
    msg = await message.answer(
        f"👋 Привет, {message.from_user.first_name}!",
        reply_markup=UserKeyboards.main_menu(is_admin)
    )
    await save_msg(user_id, msg.message_id)
@user_router.callback_query(F.data == "main_menu")
async def back_to_menu(callback: types.CallbackQuery, state: FSMContext = None):
    if state:
        await state.clear()
    user_id = callback.from_user.id
    await clear_chat(callback.message.bot, user_id)

```

```

is_admin = await db.is_admin(user_id)
msg = await callback.message.answer("🏠 Главное меню:",
reply_markup=UserKeyboards.main_menu(is_admin))
await save_msg(user_id, msg.message_id)
await callback.answer() # Закрываем уведомление о callback
@user_router.callback_query(F.data == "support_admins")
async def show_admins_list(callback: types.CallbackQuery):
    user_id = callback.from_user.id
    admins = await db.get_admins_list()

    if not admins:
        text = "😞 К сожалению, сейчас нет доступных администраторов."
    else:
        admin_links = "\n".join([f"💎 @{admin}" for admin in admins])
        text = (
            "<b>Служба поддержки</b> 🧑‍💻\n\n"
            "Если у вас возникли какие-то вопросы или проблемы, "
            "вы можете написать администраторам напрямую:\n\n"
            f"{admin_links}"
        )
    await clear_chat(callback.message.bot, user_id)
    builder = InlineKeyboardBuilder()
    builder.button(text="⬅️ Назад", callback_data="main_menu")
    msg = await callback.message.answer(
        text,
        parse_mode="HTML",
        reply_markup=builder.as_markup()
    )
    await save_msg(user_id, msg.message_id)
    await callback.answer()

```

Файл add_admin.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from utils import db, save_msg
from keyboards import AdminKeyboards
add_admin_router = Router()
class AddAdminStates(StatesGroup):
    waiting_for_username = State()
    confirm_add = State()
@add_admin_router.callback_query(F.data == "add_admin")
async def start_add_admin(callback: types.CallbackQuery, state: FSMContext):
    await state.set_state(AddAdminStates.waiting_for_username)
    await state.update_data(main_msg_id=callback.message.message_id)
    await callback.message.edit_text(
        "📄 <b>Добавление администратора</b>\n\nВведите @username пользователя:",
        parse_mode="HTML",
        reply_markup=AdminKeyboards.back_to_admin_panel()
    )
    await callback.answer()

```

```

@add_admin_router.message(AddAdminStates.waiting_for_username)
async def process_username(message: types.Message, state: FSMContext):
    username = message.text.replace("@", "").strip()
    data = await state.get_data()
    main_msg_id = data.get("main_msg_id")
    try: await message.delete()
    except: pass
    user_found = await db.get_user_by_username(username)
    if user_found:
        await state.update_data(target_id=user_found['user_id'], target_username=username)
        text = f"❓ Вы точно уверены, что хотите сделать <b>@{username}</b> администратором?"
        kb = AdminKeyboards.confirm_admin_kb()
        await state.set_state(AddAdminStates.confirm_add)
    else:
        # Пользователь не найден - остаемся на том же шаге
        text = f"❌ Пользователь <b>@{username}</b> не найден.\n\nПопробуйте еще раз или
нажмите «Назад»:"
        kb = AdminKeyboards.back_to_admin_panel()
    if main_msg_id:
        await message.bot.edit_message_text(
            chat_id=message.from_user.id,
            message_id=main_msg_id,
            text=text,
            parse_mode="HTML",
            reply_markup=kb
        )
@add_admin_router.callback_query(AddAdminStates.confirm_add, F.data ==
"confirm_add_admin_yes")
async def confirm_yes(callback: types.CallbackQuery, state: FSMContext):
    data = await state.get_data()
    await db.set_admin_status(data['target_id'], 1)
    await callback.message.edit_text(
        f"✅ Пользователь <b>@{data['target_username']}</b> теперь администратор!",
        parse_mode="HTML",
        reply_markup=AdminKeyboards.back_to_admin_panel()
    )
    await state.clear()
    await callback.answer()
@add_admin_router.callback_query(F.data == "confirm_add_admin_no")
async def confirm_no(callback: types.CallbackQuery, state: FSMContext):
    await state.clear()
    from handlers.admin.admin_main import admin_panel
    await admin_panel(callback, state)

```

Файл add_product.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.utils.keyboard import InlineKeyboardBuilder
from utils import db, clear_chat, save_msg, format_price, parse_price
from keyboards import AdminKeyboards
add_product_router = Router()

```

					ПГУ1.090301.09.001 ПЗ	Лист
						137
Изм.	Лист	№ докум.	Подпись	Дата		

```

class AddProductStates(StatesGroup):
    waiting_for_photos = State
    waiting_for_name = State
    waiting_for_price = State
    waiting_for_desc = State
    waiting_for_stock = State

def get_top_actions_kb():
    builder = InlineKeyboardBuilder()

    builder.button(text="✅ Завершить загрузку", callback_data="finish_photos")

    builder.button(text="❌ Отмена", callback_data="cancel_add_product_admin")

    builder.adjust(2)

    return builder.as_markup()

def get_photo_delete_kb(photo_index):
    builder = InlineKeyboardBuilder()

    builder.button(text="❌ Удалить", callback_data=f"photo_delete_{photo_index}")

    return builder.as_markup()

@add_product_router.callback_query(F.data == "admin_add_product")
async def add_start(callback: types.CallbackQuery, state: FSMContext):
    user_id = callback.from_user.id

    await clear_chat(callback.bot, user_id)

    await state.set_state(AddProductStates.waiting_for_photos)

    await state.update_data(photos=[], photo_msg_pairs=[])

    msg = await callback.message.answer(
        "📷 Пришлите фото товара (можно несколько).",
        reply_markup=get_top_actions_kb()
    )

    await state.update_data(prompt_msg_id=msg.message_id)

    await save_msg(user_id, msg.message_id)

    await callback.answer()

@add_product_router.callback_query(F.data.in(["cancel_add_product",
"cancel_add_product_admin"]))
async def cancel_add_process(callback: types.CallbackQuery, state: FSMContext):
    await state.clear()

    await callback.answer("❌ Добавление товара отменено")

    from handlers.admin.admin_main import admin_panel

    await admin_panel(callback, state)

@add_product_router.message(AddProductStates.waiting_for_photos, F.photo)
async def handle_photos(message: types.Message, state: FSMContext):
    user_id = message.from_user.id

    data = await state.get_data()

    photos = data.get("photos", []) # Список file_id
    photo_msg_pairs = data.get("photo_msg_pairs", []) # Пары (user_msg_id, bot_msg_id)
    photos.append(message.photo[-1].file_id)
    photo_index = len(photos) - 1

    msg = await message.answer_photo(
        message.photo[-1].file_id,
        caption=f"✅ Фото {len(photos)}",
        reply_markup=get_photo_delete_kb(photo_index)
    )

    photo_msg_pairs.append((message.message_id, msg.message_id))

    await state.update_data(photos=photos, photo_msg_pairs=photo_msg_pairs)

```

```

await save_msg(user_id, msg.message_id)
await save_msg(user_id, message.message_id)

@add_product_router.callback_query(F.data.startswith("photo_delete_"),
AddProductStates.waiting_for_photos)
async def delete_photo(callback: types.CallbackQuery, state: FSMContext):
    photo_index = int(callback.data.split("_")[2])
    data = await state.get_data()
    photos = data.get("photos", [])
    photo_msg_pairs = data.get("photo_msg_pairs", [])
    if photo_index < len(photos):
        photos.pop(photo_index)
        pair = photo_msg_pairs.pop(photo_index)
        await state.update_data(photos=photos, photo_msg_pairs=photo_msg_pairs)
        user_msg_id, bot_msg_id = pair
        chat_id = callback.message.chat.id
        if user_msg_id:
            try: await callback.bot.delete_message(chat_id, user_msg_id)
            except: pass
            try: await callback.bot.delete_message(chat_id, bot_msg_id)
            except: pass
        await callback.answer("✅ Фото удалено")
    await callback.answer()

@add_product_router.callback_query(F.data == "finish_photos",
AddProductStates.waiting_for_photos)
async def photos_done(callback: types.CallbackQuery, state: FSMContext):
    user_id = callback.from_user.id
    data = await state.get_data()
    photos = data.get("photos", [])
    if not photos:
        try:
            await callback.message.edit_text(
                "❌ Вы не загрузили ни одного фото!\nПришлите хотя бы одно фото.",
                reply_markup=get_top_actions_kb()
            )
        except: pass
        return await callback.answer()
    photo_msg_pairs = data.get("photo_msg_pairs", [])
    for user_msg_id, bot_msg_id in photo_msg_pairs:
        if user_msg_id:
            try: await callback.bot.delete_message(callback.message.chat.id, user_msg_id)
            except: pass
            try: await callback.bot.delete_message(callback.message.chat.id, bot_msg_id)
            except: pass
    await clear_chat(callback.bot, user_id)
    await state.set_state(AddProductStates.waiting_for_name)
    msg = await callback.message.answer(
        f"✅ Загружено фото: {len(photos)} шт.\n📄 Введите название товара:",
        reply_markup=AdminKeyboards.add_product_navigation("photos")
    )
    await state.update_data(prompt_msg_id=msg.message_id)

```

```

        await save_msg(user_id, msg.message_id)
        await callback.answer()
    @add_product_router.message(AddProductStates.waiting_for_name)
    async def add_name(message: types.Message, state: FSMContext):
        user_id = message.from_user.id
        try: await message.delete()
        except: pass
        name = message.text.strip() if message.text else ""
        data = await state.get_data()
        prompt_msg_id = data.get("prompt_msg_id")
        if not name:
            if prompt_msg_id:
                try:
                    await message.bot.edit_message_text(
                        chat_id=user_id, message_id=prompt_msg_id,
                        text="❌ Название не может быть пустым!\nВведите название товара:",
                        reply_markup=AdminKeyboards.add_product_navigation("photos")
                    )
                except: pass
            await state.update_data(name=name)
            await state.set_state(AddProductStates.waiting_for_price)
            await clear_chat(message.bot, user_id)
            msg = await message.answer(
                "💰 Введите цену в рублях (например: 23.03):",
                reply_markup=AdminKeyboards.add_product_navigation("name")
            )
            await state.update_data(prompt_msg_id=msg.message_id)
            await save_msg(user_id, msg.message_id)
    @add_product_router.message(AddProductStates.waiting_for_price)
    async def add_price(message: types.Message, state: FSMContext):
        user_id = message.from_user.id
        try: await message.delete()
        except: pass
        price = parse_price(message.text) if message.text else None
        data = await state.get_data()
        prompt_msg_id = data.get("prompt_msg_id")
        if price is None or price <= 0:
            if prompt_msg_id:
                try:
                    await message.bot.edit_message_text(
                        chat_id=user_id, message_id=prompt_msg_id,
                        text="❌ Ошибка! Введите корректное число больше 0:",
                        reply_markup=AdminKeyboards.add_product_navigation("name")
                    )
                except: pass
            await state.update_data(price=price)
            await state.set_state(AddProductStates.waiting_for_desc)
            await clear_chat(message.bot, user_id)
            msg = await message.answer(
                f"✅ Цена: {format_price(price)} руб.\n\n📄 Введите описание товара:",
                reply_markup=AdminKeyboards.add_product_navigation("price")
            )

```

```

    )
    await state.update_data(prompt_msg_id=msg.message_id)
    await save_msg(user_id, msg.message_id)
@add_product_router.message(AddProductStates.waiting_for_desc)
async def add_desc(message: types.Message, state: FSMContext):
    user_id = message.from_user.id
    try: await message.delete()
    except: pass
    desc = message.text.strip() if message.text else ""
    data = await state.get_data()
    prompt_msg_id = data.get("prompt_msg_id")
    if not desc:
        if prompt_msg_id:
            try:
                await message.bot.edit_message_text(
                    chat_id=user_id, message_id=prompt_msg_id,
                    text="❌ Описание не может быть пустым!",
                    reply_markup=AdminKeyboards.add_product_navigation("price")
                )
            except: pass
        return
    await state.update_data(desc=desc)
    await state.set_state(AddProductStates.waiting_for_stock)
    await clear_chat(message.bot, user_id)
    msg = await message.answer(
        "📦 Введите количество товара на складе:",
        reply_markup=AdminKeyboards.add_product_navigation("desc")
    )
    await state.update_data(prompt_msg_id=msg.message_id)
    await save_msg(user_id, msg.message_id)
@add_product_router.message(AddProductStates.waiting_for_stock)
async def add_stock(message: types.Message, state: FSMContext):
    user_id = message.from_user.id
    try: await message.delete()
    except: pass
    data = await state.get_data()
    prompt_msg_id = data.get("prompt_msg_id")
    if not message.text or not message.text.isdigit():
        if prompt_msg_id:
            try:
                await message.bot.edit_message_text(
                    chat_id=user_id, message_id=prompt_msg_id,
                    text="❌ Введите целое число!",
                    reply_markup=AdminKeyboards.add_product_navigation("desc")
                )
            except: pass
        return
    stock = int(message.text)
    name, price, desc, photos = data.get('name'), data.get('price'), data.get('desc'),
    data.get('photos', [])
    await db.add_product(name, price, desc, stock, photos)

```

```

await state.clear()
await clear_chat(message.bot, user_id)
confirmation_text = (
    f"✅ <b>Товар успешно добавлен!</b>\n\n"
    f"📄 <b>Название:</b> {name}\n"
    f"💰 <b>Цена:</b> {format_price(price)} руб.\n"
    f"📦 <b>На складе:</b> {stock} шт."
)
msg_confirm = await message.answer(confirmation_text, parse_mode="HTML")
await save_msg(user_id, msg_confirm.message_id)
msg_panel = await message.answer(
    "⚙️ <b>Панель администратора</b>\nВыберите действие:",
    parse_mode="HTML",
    reply_markup=AdminKeyboards.admin_panel()
)
await save_msg(user_id, msg_panel.message_id)
@add_product_router.callback_query(F.data == "back_to_photos")
async def back_to_photos(callback: types.CallbackQuery, state: FSMContext):
    user_id = callback.from_user.id
    await clear_chat(callback.bot, user_id)
    await state.set_state(AddProductStates.waiting_for_photos)
    data = await state.get_data()
    msg = await callback.message.answer("📷 Пришлите фото товара (можно несколько).",
    reply_markup=get_top_actions_kb())
    await state.update_data(prompt_msg_id=msg.message_id)
    await save_msg(user_id, msg.message_id)
    new_pairs = []
    for i, photo in enumerate(data.get("photos", [])):
        p_msg = await callback.message.answer_photo(photo, caption=f"✅ Фото {i + 1}",
        reply_markup=get_photo_delete_kb(i))
        new_pairs.append((None, p_msg.message_id))
        await state.update_data(photo_msg_pairs=new_pairs)
        await callback.answer()
@add_product_router.callback_query(F.data == "back_to_name")
async def back_to_name(callback: types.CallbackQuery, state: FSMContext):
    user_id = callback.from_user.id
    await clear_chat(callback.bot, user_id)
    await state.set_state(AddProductStates.waiting_for_name)
    msg = await callback.message.answer("📄 Введите название товара:",
    reply_markup=AdminKeyboards.add_product_navigation("photos"))
    await state.update_data(prompt_msg_id=msg.message_id)
    await save_msg(user_id, msg.message_id)
    await callback.answer()
@add_product_router.callback_query(F.data == "back_to_price")
async def back_to_price(callback: types.CallbackQuery, state: FSMContext):
    user_id = callback.from_user.id
    await clear_chat(callback.bot, user_id)
    await state.set_state(AddProductStates.waiting_for_price)
    msg = await callback.message.answer("💰 Введите цену в рублях:",
    reply_markup=AdminKeyboards.add_product_navigation("name"))

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

142

```

        await state.update_data(prompt_msg_id=msg.message_id)
        await save_msg(user_id, msg.message_id)
        await callback.answer()
    @add_product_router.callback_query(F.data == "back_to_desc")
    async def back_to_desc(callback: types.CallbackQuery, state: FSMContext):
        user_id = callback.from_user.id
        await clear_chat(callback.bot, user_id)
        await state.set_state(AddProductStates.waiting_for_desc)
        msg = await callback.message.answer("📝 Введите описание товара:",
        reply_markup=AdminKeyboards.add_product_navigation("price"))
        await state.update_data(prompt_msg_id=msg.message_id)
        await save_msg(user_id, msg.message_id)
        await callback.answer()

```

Файл admin_main.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from utils import clear_chat, save_msg, db
from keyboards import AdminKeyboards
admin_router = Router()
@admin_router.callback_query(F.data == "admin_panel")
async def admin_panel(event: types.Message | types.CallbackQuery, state: FSMContext):
    if state:
        await state.clear() # Сбрасываем все текущие FSM-состояния
    user_id = event.from_user.id
    bot = event.bot
    message = event.message if isinstance(event, types.CallbackQuery) else event
    await clear_chat(bot, user_id)
    msg = await message.answer(
        "⚙️ <b>Панель администратора</b>\nВыберите действие:",
        parse_mode="HTML",
        reply_markup=AdminKeyboards.admin_panel()
    )
    await save_msg(user_id, msg.message_id)
    if isinstance(event, types.CallbackQuery):
        await event.answer()

```

Файл admin_orders.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from utils import db, clear_chat, save_msg, format_price
from keyboards import AdminKeyboards
admin_orders_router = Router()
ADMIN_ORDERS_PER_PAGE = 4 # Заказов на странице
class AdminOrderStates(StatesGroup):
    waiting_for_track = State() # Ожидание ввода трек-номера
    async def render_order_details(message: types.Message, order_id: int, back_status: str,
    back_page: str):
        order_info, items = await db.get_order_details(order_id)
        if not order_info:
            return await message.answer("Заказ не найден.")

```

```

_, status, address, phone, track_number, date_create, username, buyer_id, l_name, f_name,
m_name = order_info
user_link = f"@{username}" if username else f"ID: {buyer_id}"
fio = f"{l_name or ''} {f_name or ''} {m_name or ''}".strip()
text = (f"📄 <b>Заказ #{order_id}</b>\n"
        f"👤 Покупатель: {fio}{user_link}\n"
        f"☎ Телефон: {phone}\n"
        f"📅 Оформлен: {date_create}\n"
        f"📌 <b>Статус:</b> {status.upper()}\n")
if track_number:
    text += f"📍 <b>Трек-номер:</b> <code>{track_number}</code>\n"
text += f"\n📍 Адрес:\n{address}\n\n📦 <b>Товары:</b>\n"
total_sum = sum(qty * price for name, qty, price in items)
for name, qty, price in items:
    text += f"▪ {name} x{qty} – {format_price(qty * price)} руб.\n"
text += f"\n💰 <b>Итого: {format_price(total_sum)} руб.</b>"
msg = await message.answer(
    text,
    parse_mode="HTML",
    reply_markup=AdminKeyboards.admin_order_details(order_id, status, back_status,
back_page)
)
await save_msg(message.chat.id, msg.message_id)
async def send_admin_orders_list(bot, chat_id: int, status: str, page: int):
    total_count = await db.get_all_orders_count(status)
    total_pages = max((total_count + ADMIN_ORDERS_PER_PAGE - 1) // ADMIN_ORDERS_PER_PAGE, 1)
    orders = await db.get_all_orders_page(status, page, ADMIN_ORDERS_PER_PAGE)
    from aiogram.utils.keyboard import InlineKeyboardBuilder
    builder = InlineKeyboardBuilder()
    if not orders:
        builder.button(text="⬅ Назад", callback_data="admin_orders_menu")
        msg = await bot.send_message(
            chat_id=chat_id,
            text=f"📦 Заказов со статусом '{status}' пока нет.",
            reply_markup=builder.as_markup()
        )
        await save_msg(chat_id, msg.message_id)
        return
    for o_id, o_status, o_date, o_user in orders:
        short_date = str(o_date).split()[0] if o_date else ""
        builder.button(text=f"#{o_id} | {short_date}",
callback_data=f"admin_order_info_{o_id}_{status}_{page}")
        builder.adjust(1)
    nav_buttons = []
    if page > 1:
        nav_buttons.append(types.InlineKeyboardButton(text="⬅ Пред.",
callback_data=f"admin_orders_list_{status}_{page-1}"))
    if page < total_pages:
        nav_buttons.append(types.InlineKeyboardButton(text="След. ➡",
callback_data=f"admin_orders_list_{status}_{page+1}"))

```

```

if nav_buttons:
    builder.row(*nav_buttons)

builder.row(types.InlineKeyboardButton(text="⬅ Назад",
callback_data="admin_orders_menu"))

msg = await bot.send_message(
    chat_id=chat_id,
    text=f"📦 Заказы ({status}) [Стр. {page}/{total_pages}]:",
    reply_markup=builder.as_markup()
)

await save_msg(chat_id, msg.message_id)

@admin_orders_router.callback_query(F.data == "admin_orders_menu"):
async def show_orders_menu(callback: types.CallbackQuery, state: FSMContext):
    """Показывает меню категорий заказов (по статусам)."""
    await state.clear()
    await clear_chat(callback.bot, callback.from_user.id)
    msg = await callback.message.answer(
        "📋 <b>Категории заказов</b>",
        parse_mode="HTML",
        reply_markup=AdminKeyboards.admin_orders_menu()
    )
    await save_msg(callback.from_user.id, msg.message_id)
    await callback.answer()

@admin_orders_router.callback_query(F.data.startswith("admin_orders_list_"))
async def show_admin_orders_list(callback: types.CallbackQuery):
    """Показывает список заказов по выбранному статусу."""
    parts = callback.data.split("_")
    status, page = parts[3], int(parts[4])
    await clear_chat(callback.bot, callback.from_user.id)
    await send_admin_orders_list(callback.bot, callback.from_user.id, status, page)
    await callback.answer()

@admin_orders_router.callback_query(F.data.startswith("admin_order_info_"))
async def show_admin_order_info(callback: types.CallbackQuery):
    """Показывает детали конкретного заказа."""
    parts = callback.data.split("_")
    order_id, back_status, back_page = int(parts[3]), parts[4], parts[5]
    await clear_chat(callback.bot, callback.from_user.id)
    await render_order_details(callback.message, order_id, back_status, back_page)
    await callback.answer()

@admin_orders_router.callback_query(F.data.startswith("admin_confirm_st_"))
async def confirm_status_change(callback: types.CallbackQuery):
    parts = callback.data.split("_")
    order_id, next_status, back_status, back_page = parts[3], parts[4], parts[5], parts[6]
    await callback.message.edit_text(
        f"❓ Вы точно хотите изменить статус заказа <b>#{order_id}</b> на
<b>{next_status}</b>?",
        parse_mode="HTML",
        reply_markup=AdminKeyboards.admin_confirm_status_kb(order_id, next_status,
back_status, back_page)
    )

@admin_orders_router.callback_query(F.data.startswith("admin_order_setst_"))
async def set_order_status(callback: types.CallbackQuery):

```

```

        """Меняет статус заказа после подтверждения."""
        parts = callback.data.split("_")
        order_id, new_status, back_status, back_page = int(parts[3]), parts[4], parts[5],
int(parts[6])
        await db.update_order_status(order_id, new_status)
        await callback.answer(f"Статус #{order_id} изменен")
        await clear_chat(callback.bot, callback.from_user.id)
        await send_admin_orders_list(callback.bot, callback.from_user.id, back_status, back_page)
@admin_orders_router.callback_query(F.data.startswith("admin_order_settrack_"))
async def prompt_for_track_number(callback: types.CallbackQuery, state: FSMContext):
    """Запрашивает у админа трек-номер для заказа."""
    parts = callback.data.split("_")
    order_id, back_status, back_page = int(parts[3]), parts[4], parts[5]
    await state.set_state(AdminOrderStates.waiting_for_track)
    await state.update_data(order_id=order_id, back_status=back_status, back_page=back_page)
    await clear_chat(callback.bot, callback.from_user.id)

    msg = await callback.message.answer(f"📄 Введите трек-номер для заказа #{order_id}:")
    await save_msg(callback.from_user.id, msg.message_id)
@admin_orders_router.message(AdminOrderStates.waiting_for_track)
async def process_track_number(message: types.Message, state: FSMContext):
    """Сохраняет трек-номер и меняет статус заказа на 'отправлен'."""
    track_number = message.text.strip()
    data = await state.get_data()
    order_id = data['order_id']
    back_status = data['back_status']
    back_page = int(data['back_page'])
    wait db.update_order_track(order_id, track_number)
    await db.update_order_status(order_id, "отправлен")
    await clear_chat(message.bot, message.from_user.id)
    await send_admin_orders_list(message.bot, message.from_user.id, back_status, back_page)
    await state.clear()

```

Файл edit_product.py

```

from aiogram import Router, F, types
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.utils.keyboard import InlineKeyboardBuilder
from utils import db, clear_chat, save_msg, parse_price
from keyboards import AdminKeyboards
from handlers.shared.catalog import send_product_card, show_catalog
edit_product_router = Router()
class EditStates(StatesGroup):
    waiting_for_text = State
    waiting_for_new_photos = State
def get_edit_top_actions_kb(product_id):
    builder = InlineKeyboardBuilder()
    builder.button(text="✅ Сохранить изменения", callback_data=f"save_ph_{product_id}")
    builder.button(text="❌ Отмена", callback_data=f"admin_edit_{product_id}")
    builder.adjust(1)
    return builder.as_markup()
def get_edit_photo_delete_kb(photo_index):
    builder = InlineKeyboardBuilder()

```

```

        builder.button(text="❌ Удалить фото", callback_data=f"ed_ph_del_{photo_index}")
        return builder.as_markup()
    async def open_edit_menu(event: types.Message | types.CallbackQuery, state: FSMContext,
                             product_id: int):
        await state.clear()
        user_id = event.from_user.id
        bot = event.bot
        await clear_chat(bot, user_id)
        await send_product_card(bot, user_id, product_id, is_admin=True, is_editing=True)
        if isinstance(event, types.CallbackQuery):
            await event.answer()
    @edit_product_router.callback_query(F.data.startswith("admin_edit_"))
    async def edit_product_handler(callback: types.CallbackQuery, state: FSMContext):
        p_id = int(callback.data.split("_")[2])
        await open_edit_menu(callback, state, p_id)
    @edit_product_router.callback_query(F.data.startswith("edit_photos_"))
    async def start_replace_photos(callback: types.CallbackQuery, state: FSMContext):
        """Начинает процесс замены фотографий товара."""
        p_id = int(callback.data.split("_")[2])
        res = await db.get_product_by_id(p_id)
        if not res:
            return
        await state.set_state(EditStates.waiting_for_new_photos)
        await state.update_data(edit_p_id=p_id, new_photos=list(res[1]))
        await clear_chat(callback.bot, callback.from_user.id)
        msg = await callback.message.answer(
            "📷 <b>Редактирование фото</b>\nПришлите новые фото или удалите старые:",
            parse_mode="HTML",
            reply_markup=get_edit_top_actions_kb(p_id)
        )
        await save_msg(callback.from_user.id, msg.message_id)
        for i, f_id in enumerate(res[1]):
            m = await callback.message.answer_photo(
                f_id,
                caption=f"✅ Фото {i+1}",
                reply_markup=get_edit_photo_delete_kb(i)
            )
            await save_msg(callback.from_user.id, m.message_id)
        await callback.answer()
    @edit_product_router.message(EditStates.waiting_for_new_photos, F.photo)
    async def handle_new_edit_photos(message: types.Message, state: FSMContext):
        user_id = message.from_user.id
        await save_msg(user_id, message.message_id)
        data = await state.get_data()
        photos = data.get("new_photos", [])
        photos.append(message.photo[-1].file_id)
        await state.update_data(new_photos=photos)
        msg = await message.answer_photo(
            message.photo[-1].file_id,
            caption=f"✅ Фото {len(photos)}",
            reply_markup=get_edit_photo_delete_kb(len(photos)-1)
        )

```

```

    )
    await save_msg(user_id, msg.message_id)
@edit_product_router.callback_query(F.data.startswith("ed_ph_del_"),
EditStates.waiting_for_new_photos)
async def delete_edit_photo(callback: types.CallbackQuery, state: FSMContext):
    photo_index = int(callback.data.split("_")[3])
    data = await state.get_data()
    photos = data.get("new_photos", [])
    if photo_index < len(photos):
        photos.pop(photo_index)
        await state.update_data(new_photos=photos)
        try:
            await callback.message.delete()
        except:
            pass
        await callback.answer("❌ Фото удалено из временного списка")
    await callback.answer()
@edit_product_router.callback_query(F.data.startswith("save_ph_"))
async def save_replaced_photos(callback: types.CallbackQuery, state: FSMContext):
    data = await state.get_data()
    p_id = data['edit_p_id']
    new_photos = data.get('new_photos', [])
    if not new_photos:
        return await callback.answer("❌ Нужно загрузить хотя бы одно фото!", show_alert=True)
    await db.clear_product_photos(p_id)
    for f_id in new_photos:
        await db.add_product_photo(p_id, f_id)
    await callback.answer("✅ Фотографии обновлены")
    await open_edit_menu(callback, state, p_id)
@edit_product_router.callback_query(F.data.startswith("edit_"))
async def start_edit_text(callback: types.CallbackQuery, state: FSMContext):
    if "photos" in callback.data:
        return # Пропускаем, это не текстовое поле
    field, p_id = callback.data.split("_")[1], int(callback.data.split("_")[2])
    await state.set_state(EditStates.waiting_for_text)
    await state.update_data(edit_p_id=p_id, edit_field=field)
    prompts = {
        "name": "📄 Введите название:",
        "desc": "📖 Введите описание:",
        "price": "💰 Введите цену:",
        "stock": "📊 Введите остаток:"
    }
    }
    await clear_chat(callback.bot, callback.from_user.id)
    kb = InlineKeyboardBuilder().button(text="❌ Отмена",
callback_data=f"admin_edit_{p_id}").as_markup()
    msg = await callback.message.answer(prompts[field], reply_markup=kb)
    await save_msg(callback.from_user.id, msg.message_id)
    await callback.answer()
@edit_product_router.message(EditStates.waiting_for_text)
async def process_edit_text(message: types.Message, state: FSMContext):

```

```

"""Обрабатывает введенное текстовое значение и сохраняет в БД."""
user_id = message.from_user.id
await save_msg(user_id, message.message_id)
data = await state.get_data()
p_id, field = data['edit_p_id'], data['edit_field']
# Сопоставление: callback_field -> колонка в БД
db_map = {"name": "name", "desc": "description", "price": "price", "stock": "stock"}

val = message.text.strip()
error_text = None
if field == "price":
    val = parse_price(val)
    if val is None or val <= 0:
        error_text = "❌ Ошибка! Введите корректное число больше 0:"
elif field == "stock":
    if not val.isdigit():
        error_text = "❌ Введите целое число!"
    else:
        val = int(val)
elif not val:
    error_text = "❌ Поле не может быть пустым!"
if error_text:
    kb = InlineKeyboardBuilder().button(text="❌ Отмена",
callback_data=f"admin_edit_{p_id}").as_markup()
    msg = await message.answer(error_text, reply_markup=kb)
    await save_msg(user_id, msg.message_id)
    return # Остаемся в состоянии waiting_for_text
    await db.update_product_field(p_id, db_map[field], val)
    await open_edit_menu(message, state, p_id)
@edit_product_router.callback_query(F.data.startswith("confirm_delete_"))
async def confirm_delete(callback: types.CallbackQuery):
    p_id = int(callback.data.split("_")[2])
    await clear_chat(callback.bot, callback.from_user.id)
    res = await db.get_product_by_id(p_id)
    if not res or not res[0]:
        return await callback.answer("Ошибка: Товар не найден")
    product_name = res[0][1]
    kb = InlineKeyboardBuilder()
    kb.button(text="✅ ДА, УДАЛИТЬ", callback_data=f"delete_prod_{p_id}")
    kb.button(text="❌ ОТМЕНА", callback_data=f"admin_edit_{p_id}")
    msg = await callback.message.answer(
        f"⚠️ <b>Вы уверены, что хотите удалить товар:</b>\n📦 {product_name}?",
        parse_mode="HTML",
        reply_markup=kb.adjust(1).as_markup()
    )
    await save_msg(callback.from_user.id, msg.message_id)
    await callback.answer()
@edit_product_router.callback_query(F.data.startswith("delete_prod_"))
async def delete_product_exec(callback: types.CallbackQuery, state: FSMContext):
    p_id = int(callback.data.split("_")[2])

```

```
await db.delete_product(p_id)
await db.conn.commit()
await callback.answer("🗑 Товар успешно удален (архивирован)")
await show_catalog(callback, state)
```

Приложение В
(обязательное)

Бизнес-проект «Разработка интернет-магазина на основе Telegram-бота»
Презентация

					<i>ПГУ1.090301.09.001 ПЗ</i>	Лист
						151
Изм.	Лист	№ докум.	Подпись	Дата		

Бизнес-проект «Разработка интернет-магазина на основе Telegram-бота»

Бакалаврская работа | Направление 09.03.01

Выполнил: студент гр. 22ВВП1 Евдокимов Р.Е.

Руководитель: к.т.н., доцент Юрова О.В.

Пензенский государственный университет, 2026

1

АКТУАЛЬНОСТЬ ТЕМЫ

Telegram трансформируется из мессенджера в полноценную **бизнес-экосистему**. Сегодня это идеальное решение для малого бизнеса:

- ✓ **Рост аудитории:** Более 102 млн пользователей в РФ (2025).
- ✓ **Экономия:** Запуск магазина в 3-5 раз дешевле разработки сайта.



2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист
152

ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ



Цель проекта

Разработка отказоустойчивого ПО для автоматизации продаж и управления коммерческими процессами внутри мессенджера.



Проектирование

Создание асинхронной архитектуры на базе Python и реляционной схемы базы данных для хранения товаров и заказов.



Реализация

Интеграция платежных шлюзов, разработка FSM-сценариев и проведение комплексного функционального тестирования.

3

ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

Критерий сравнения	@IMP_egezebot	@SpaceCosmetics_bot	Разрабатываемое ПО
<u>Наличие корзины</u>	✗	✓	✓
<u>Автоматизация оплаты</u>	✓	✓	✓
Товары	Электронные	Физические	Физические
<u>Интеграция с СУБД</u>	✓	✓	✓

4

МНОГОУРОВНЕВАЯ АРХИТЕКТУРА MVC



Presentation Layer

Роутеры и диспетчер, обрабатывающие входящие события (сообщения, нажатия кнопок) от пользователя.



Business Logic

Хэндлеры и Машина состояний (FSM). Реализация пошаговых сценариев диалога и валидация данных.



Data Access Layer

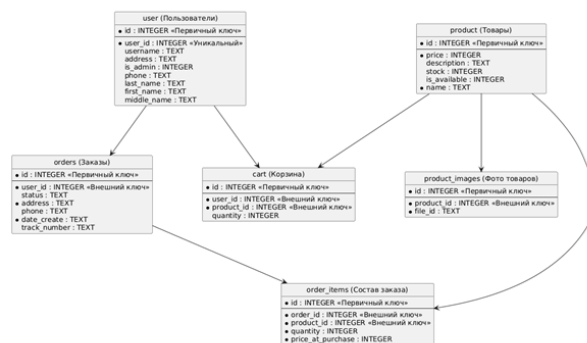
Модуль асинхронного взаимодействия с БД (aiosqlite) и паттерн «Репозиторий» для CRUD-операций.

5

ОРГАНИЗАЦИЯ БАЗЫ ДАННЫХ

Используется реляционная СУБД SQLite. Схема состоит из 6 нормализованных таблиц:

- ✓ user
- ✓ product
- ✓ cart
- ✓ orders
- ✓ product_images
- ✓ order_items



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

154

ТЕХНОЛОГИЧЕСКИЙ СТЕК ПРОЕКТА

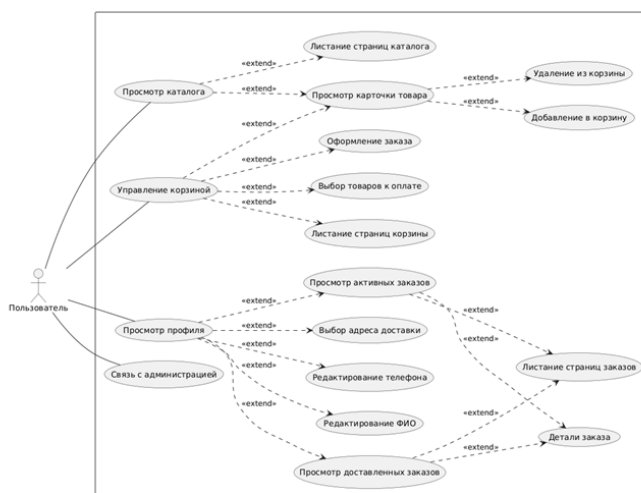
- ✓ **Python:** Основной язык разработки.
- ✓ **aiogram:** Современный асинхронный фреймворк для взаимодействия с Bot API.
- ✓ **aiosqlite:** Неблокирующий интерфейс к БД.
- ✓ **Visual Studio Code:** Основная среда разработки и отладки.



7

ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

- ✓ **Главное меню:** Навигация по разделам.
- ✓ **Каталог:** Постраничная пагинация.
- ✓ **Корзина:** Управление количеством.
- ✓ **Оплата:** Интеграция с API.



8

Изм.	Лист	№ докум.	Подпись	Дата

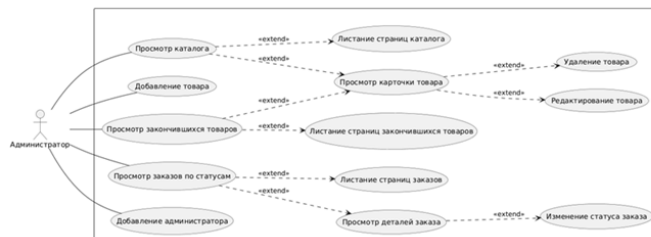
ПГУ1.090301.09.001 ПЗ

Лист

155

ПАНЕЛЬ АДМИНИСТРАТОРА

- ✓ **Управление товарами:** Мягкое удаление, редактирование цен и медиа.
- ✓ **Статусы заказов:** Обработка жизненного цикла (Оформлен → Собран → Отправлен).
- ✓ **Трекинг:** Ввода трек-номеров для уведомления клиентов.
- ✓ **Права доступа:** Динамическое назначение новых модераторов.



9

ИСТОЧНИКИ ДОХОДА И ЭФФЕКТИВНОСТЬ

Целевая аудитория

- ✓ Владельцы малого бизнеса
- ✓ Начинающие предприниматели

Услуги реализации

- ✓ Разработка программного комплекса от 60 000 рублей
- ✓ Подписка за администрирование и поддержку от 3 000 рублей в месяц.
- ✓ Развертывание облачной инфраструктуры, домены и аренда серверов ~10 000 рублей.

10

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.09.001 ПЗ

Лист

156

Итоги

Выполнены поставленные задачи

Подробно изучен стек технологий

Проанализированы аналоги

Построена бизнес модель

11

					ПГУ1.090301.09.001 ПЗ	Лист
						157
Изм.	Лист	№ докум.	Подпись	Дата		