

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление 09.03.02 «Информационные системы и технологии»
подготовки

БАКАЛАВРСКАЯ РАБОТА
на тему
**«Информационная
система для курьерской службы
доставки продуктов питания»**

Студент	_____	Быстров Алексей Евгеньевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Козлов А.Ю
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Быстров Алексей Евгеньевич гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система для курьерской службы доставки продуктов питания

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы Профессор Козлов А.Ю.

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

Разработать и отладить чат-бота с административной панелью, предназначенного для работы курьерской службы доставки. Система должна быть разработана на языке программирования Python. Система должна позволять клиентам оформлять заказы с доставкой по указанному адресу через чат-бота. Администратор должен иметь возможность добавлять, редактировать, удалять сущности базы данных через административную панель. Курьеры должны иметь возможность выбирать для дальнейшей доставки заказ клиента через чат-бота. Система должна запускаться в операционной системе Linux Ubuntu. Для запуска системы должны быть установлены Docker и Docker-compose.

5. Перечень подлежащих разработке вопросов

1. Анализ предметной области 1.1. Описание предметной области. 1.2. Обзор существующих решений. 1.3. Актуальность использования чат – ботов. 2. Описание требований к системе. 2.1 Функциональные требования 2.2 Требования к надежности 2.3. Требования к пользователю информационной системы 2.3. Требования к информационной и программной совместимости. 3. Выбор технологий и средств для разработки приложения. 3.1 Язык программирования Python. 3.2 Среда разработки PyCharm. 3.3 Django Framework. 3.4 Aiogram Framework. 3.5 Docker. 4. Проектирование информационной системы. 4.1. Проектирование административной панели. 4.2. Проектирование телеграм – бота. 5. Реализация. 5.1. Реализация административной панели. 5.2. Реализация телеграм – бота. 6. Тестирование основных функций для работы с базой данных. 6.1. Выбор метода тестирования. 6.2. Юнит-тестирование. 7. Планирование и расчет бюджета.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова

Реферат

Пояснительная записка содержит 140 листов, 45 рисунков, 12 использованных источников, 11 таблиц и 3 приложения.

ТЕЛЕГРАМ БОТ, AIOGRAM, PYTHON, DJANGO, PYCHARM, PYTHON,
ИНФОРМАЦИОННАЯ СИСТЕМА, КЛИЕНТ-СЕРВЕР.

Объектом исследования является информационная система для курьерской службы доставки продуктов питания.

Предмет исследования – технология разработки чат-бота и административной панели.

Цель работы: разработка информационной системы для курьерской службы доставки. Система должна включать в себя чат-бота и административную панель, реализованную в виде веб-приложения.

Технологии разработки: Python, MVC, Django, Aiogram.

Результат работы: спроектирована и разработана информационная система для курьерской службы доставки, которая включает в себя чат-бота и административную панель, реализованную в виде веб-приложения.

					ПГУ 09.03.02 – 08БР191.04 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.	Быстров А.Е.				«Информационная система для курьерской службы доставки продуктов питания» Пояснительная записка	Лит.	Лист	Листов
Провер.	Козлов А.Ю.						4	140
Реценз.						группа 19ВИ1		
Н. Контр.	Такташкин Д.В.							
Утверд.	Козлов А.Ю.							

СОДЕРЖАНИЕ

Введение.....	7
1 Анализ предметной области	9
1.1 Описание предметной области.....	9
1.2 Обзор существующих решений	10
1.3 Актуальность использования чат-ботов.....	19
2 Описание требований к системе.....	22
2.1 Функциональные требования	22
2.2 Требования к надежности	23
2.3 Требования к пользователю информационной системы	24
2.4 Требования к информационной и программной совместимости	24
3 Выбор технологий и средств для разработки приложения	25
3.1 Язык программирования Python	25
3.2 Среда разработки PyCharm.....	25
3.3 Django Framework	26
3.4 Aiogram Framework.....	27
3.5 Docker.....	27
4 Проектирование информационной системы.....	29
4.1 Выделение компонентов архитектуры	29
4.2 Проектирование чат-бота	30
4.3 Проектирование административной панели.....	31
4.4 Проектирование базы данных	32
5 Реализация информационной системы.....	39
5.1 Реализация архитектуры	39
5.2 Реализация административной панели.....	41
5.3 Реализация телеграм бота	43
5.4 Реализация базы данных	54
6 Тестирование основных функций для работы с базой данных.....	57
6.1 Выбор метода тестирования	57
6.2 Юнит-тестирование	60

7	Планирование и расчет бюджета	61
	Заключение	68
	Список использованных источников	70
	Приложение А. ИСХОДНЫЙ КОД АДМИНИСТРАТИВНОЙ ПАНЕЛИ.....	72
	Приложение Б. ИСХОДНЫЙ КОД ЧАТ-БОТА	86
	Приложение В. ИСХОДНЫЙ КОД ЮНИТ–ТЕСТОВ.....	134

Введение

В настоящее время все большую популярность приобретает использование чат-ботов в различных сферах деятельности, начиная от электронной коммерции и заканчивая промышленностью и образованием. Этот инструмент прост в интеграции в существующую экосистему организации, доступен по цене и легок в использовании, что объясняет его растущую популярность.

За последние несколько лет все больше компаний и организаций, независимо от их масштабов – от небольших частных проектов до крупнейших корпораций, начали использовать чат-ботов в своей работе. Это решение становится все более актуальным, так как социальные сети привлекают огромную и активную аудиторию, которая может стать потенциальными потребителями [1].

Пользователи уже привыкли к тому, что на их вопросы к бренду или ритейлеру в социальных сетях и мессенджерах отвечают чат-боты. Некоторые удивляют своей сообразительностью, в то время как другие могут раздражать своей ограниченностью. Однако, если настройка бота проведена грамотно, то он может решить множество вопросов, сэкономив время компании на более важные задачи.

Чат-боты могут быть очень полезны в сфере услуг. Они могут помочь автоматизировать процессы, ускорить ответы на запросы клиентов и улучшить качество обслуживания. Одним из вариантов применения чат-ботов в сфере услуг является автоматизация процессов курьерской службы доставки.

Для гибкой настройки чат-ботов разработчики довольно часто создают административную панель, которая является интерфейсом, позволяющим управлять и настраивать работу чат-бота. Она может быть реализована в виде веб-приложения или в виде отдельного приложения для настольных компьютеров, или мобильных устройств.

Целью представленной работы выступает разработка информационной системы для курьерской службы доставки. Система включает в себя чат-бота в

мессенджере телеграм и административную панель, реализованную в виде веб-приложения.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- выполнить анализ предметной области;
- рассмотреть существующие решения;
- определить требования к системе;
- спроектировать информационную систему с помощью языка UML;
- создать продукт и протестировать его.

Использование продукта позволит улучшить опыт покупки для клиентов и повысить эффективность работы курьерской службы. Даже если курьерская служба уже имеет информационную систему, продукт может быть полезен, так как его можно использовать для увеличения охвата клиентов, которые будут приходить из мессенджера телеграм.

1 Анализ предметной области

1.1 Описание предметной области

Доставка продуктов на дом – это услуга, которая позволяет заказать продукты питания и другие товары через интернет или по телефону и получить их доставку прямо на дом. Клиент выбирает нужные товары из каталога, оформляет заказ и указывает адрес доставки. Затем курьер доставляет заказанные товары в указанное место. Эта услуга позволяет сэкономить время и удобна для людей, которые не могут или не хотят посещать магазины самостоятельно.

Сегодня, благодаря развитию интернета, доставка продуктов в офис или на дом уже не является чем-то удивительным. В сети можно заказать практически все, что угодно и куда угодно. Современные покупатели ценят свое личное время и готовы заплатить за доставку, чтобы не тратить его на походы в магазины и не стоять в очередях. Кроме того, заказывая доставку, покупатель может приобрести товары напрямую у производителя или оптового поставщика, которые предлагают товары по более низким ценам. Доплатив немного за доставку, покупатель может получить товары по цене, сравнимой с ценой в розничном магазине, где на товары установлена значительная наценка. Все это делает доставку продуктов одним из наиболее востребованных сегментов рынка электронной коммерции в России.

Всего два года назад доставка продуктов на дом считалась роскошью или экстренной мерой. В Москве и других крупных городах доставка продуктов была нормой, но в более провинциальных регионах ее использовали единицы. Однако в 2020 году произошел глобальный переворот в индустрии онлайн-покупок. В мире и, в частности, в России началась пандемия, и большая часть населения вынуждена была находиться на самоизоляции [2]. Это привело к стремительному развитию доставки продуктов на дом и онлайн торговли в целом, что можно увидеть на диаграмме, построенной по данным ассоциации компаний интернет-торговли и сбербанка, которая показана на рисунке 1.

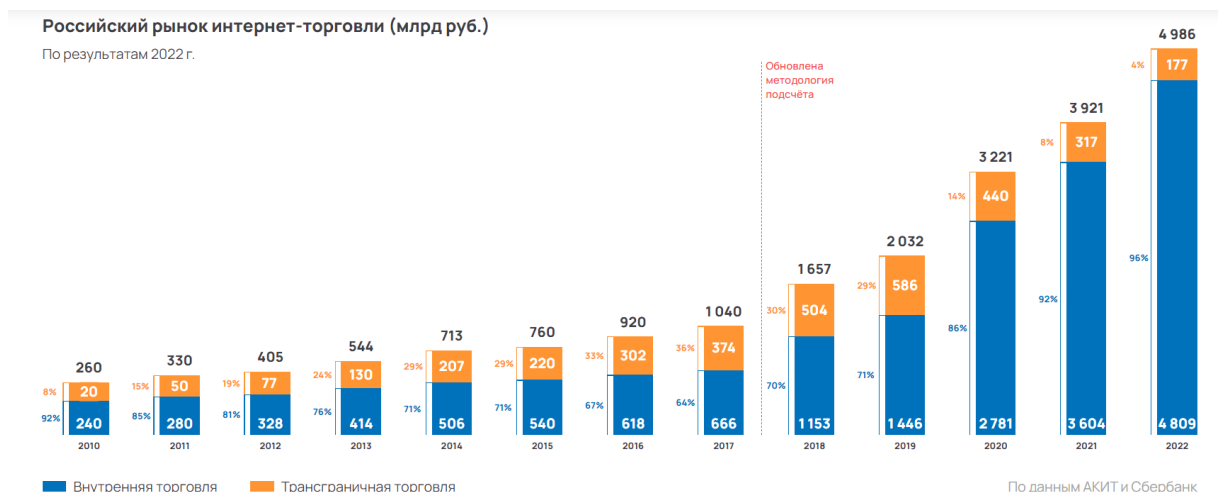


Рисунок 1 – История развития рынка интернет-торговли

В настоящее время доставка продуктов питания стала очень популярной услугой, которая является неотъемлемой частью жизни многих людей. Именно поэтому доставка продуктов является одним из наиболее быстрорастущих сегментов электронной коммерции на российском рынке.

1.2 Обзор существующих решений

Существует несколько типов сервисов предназначенных для доставки продуктов [3]:

- продуктовые сети, имеющие собственную доставку;
- онлайн гипермаркеты, которые привозят продукты с собственных складов;
- сервисы доставки, которые привозят продукты из нескольких сетевых гипермаркетов.

Сервисы могут использовать различные типы приложений для работы с клиентами. Рассмотрим некоторые варианты:

- мобильные приложения;
- веб-приложения;
- чат-боты.

Мобильные приложения можно загрузить на мобильные устройства, такие как смартфоны и планшеты. Они могут быть доступны как для операционных систем Android, так и для iOS.

Веб-приложения можно посетить через веб-браузер. Они могут предлагать те же функции, что и мобильные приложения, но без необходимости загружать их на устройство.

Чат-боты работают через мессенджеры, такие как Facebook, Telegram, Viber.

Онлайн-гипермаркеты и сервисы доставки более узконаправлены на доставку, в отличие от продуктовых сетей. Наиболее популярные онлайн-гипермаркеты:

- «Самокат»;
- «Утконос».

Наиболее популярные сервисы доставки:

- «Яндекс.Лавка»;
- «СберМаркет».

Далее можно выделить «Самокат» из категории онлайн – гипермаркеты, и «Сбермаркет» из сервисов доставки, так как именно эти сервисы распространены в большем количестве городов России.

Для начала, стоит отметить, что сервис доставки продуктов «Самокат» не имеет своего веб-сайта. Единственный способ воспользоваться услугами данного сервиса – загрузить приложение на устройство с операционной системой Android или IOS. Приложение «Самокат» представляет собой каталог продуктов, в котором содержится информация о товарах, акциях, заказах и личном профиле. Основная цель сервиса – обеспечить доступную и бесплатную доставку продуктов.

Структура приложения:

- главная страница;
- каталог товаров;
- расширенный поиск;

– профиль пользователя.

На рисунке 2 можно увидеть современный и приятный интерфейс приложения. Сразу же предлагаются товары со скидками. Далее, при прокрутке страницы вниз, можно увидеть категории товаров. В самом верху находится строка поиска.

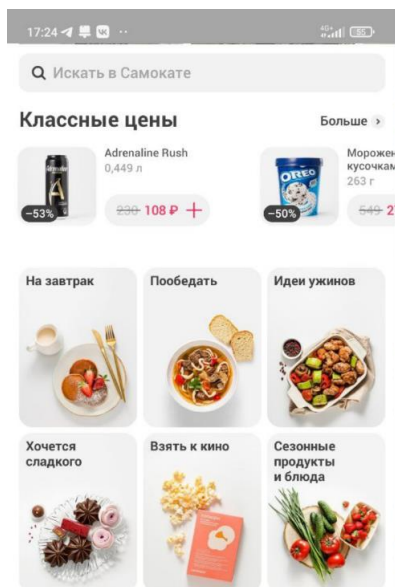


Рисунок 2 – Интерфейс приложения «Самокат»

При нажатии на строку поиска открываются все категории и различные подборки товаров для конкретных целей. Страница со строкой поиска показана на рисунке 3.

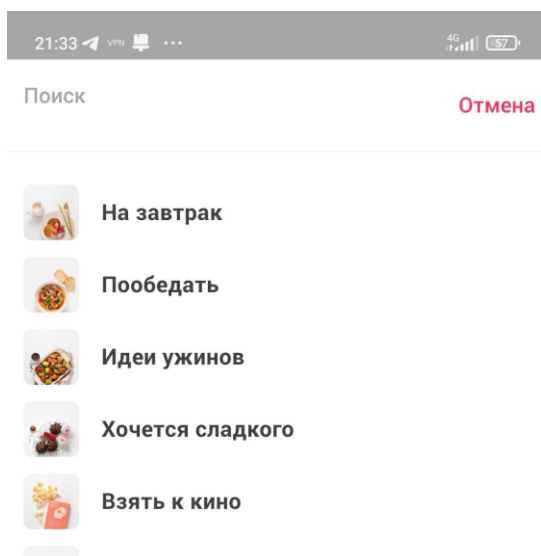


Рисунок 3 – Строка поиска в приложении «Самокат»

Когда пользователь вводит название нужного товара, сервис предлагает различные категории и товары, соответствующие запросу пользователя, чтобы уменьшить количество вариантов и ускорить процесс поиска нужного товара. Пример поиска товара показан на рисунке 4.

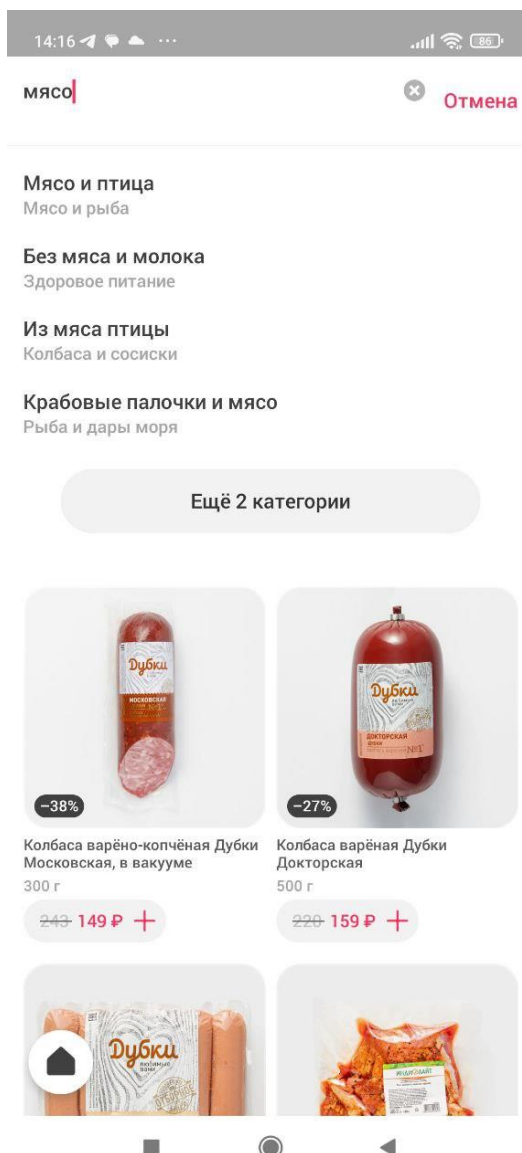


Рисунок 4 – Пример поиска в приложении «Самокат»

После добавления необходимых товаров в корзину, пользователь оформляет заказ и ему остается только дожидаться курьера. В случае необходимости в приложении имеется возможность просмотра истории заказов.

В заключении можно сказать, что интерфейс приложения очень удобен и прост, он позволяет максимально быстро найти необходимые товары и оформить заказ.

СберМаркет имеет приложения для Android и IOS, а также веб-версию своего сервиса. Далее будет рассмотрена веб-версия СберМаркета.

Структура сайта:

- главная страница;
- каталог магазинов;
- каталог товаров;
- поддержка;
- личный профиль;
- информация о сервисе.

При переходе на главную страницу покупателю предлагают выбрать магазин из списка найденных в его городе, как это показано на рисунке 5.

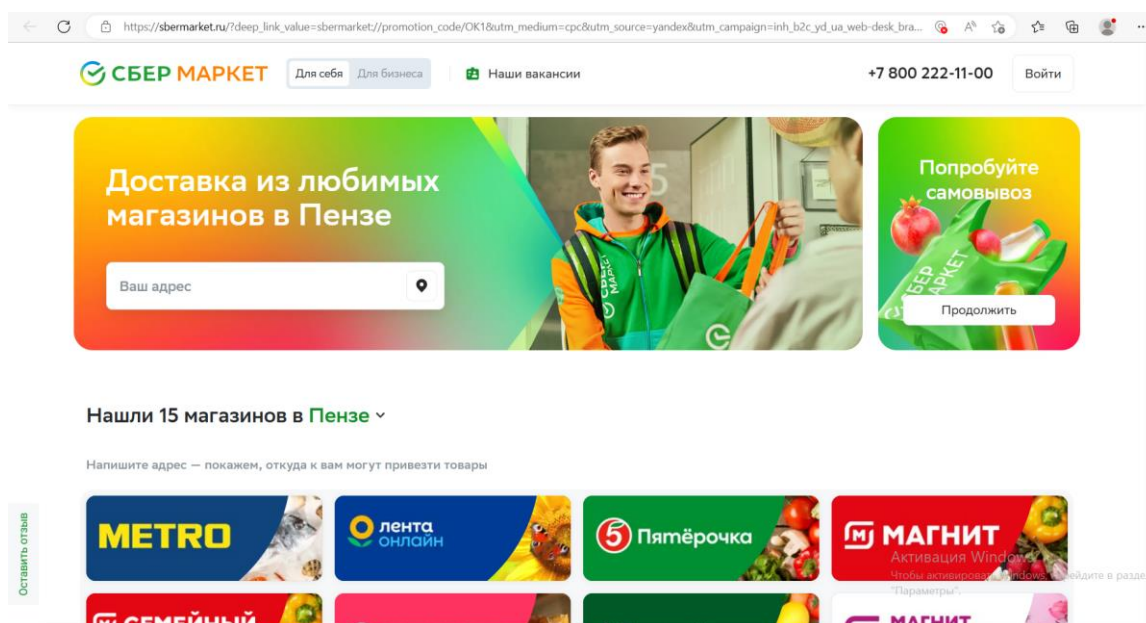


Рисунок 5 – Главная страница сайта «Сбермаркет»

После выбора магазина, сайт предлагает популярные товары выбранного магазина и товары со скидками. Предложения сайта показаны на рисунке 6.

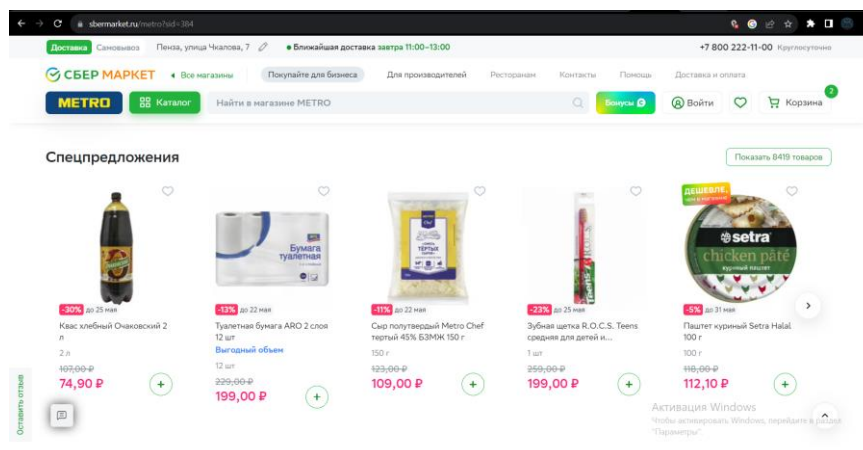


Рисунок 6 – Страница после выбора магазина на сайте «СберМаркет»

Под логотипом сайта появляется логотип выбранного магазина и кнопка «Каталог», по клику на которую пользователь может выбрать категорию продуктов, которая его интересует. Страницы выбора категории показана на рисунке 7.

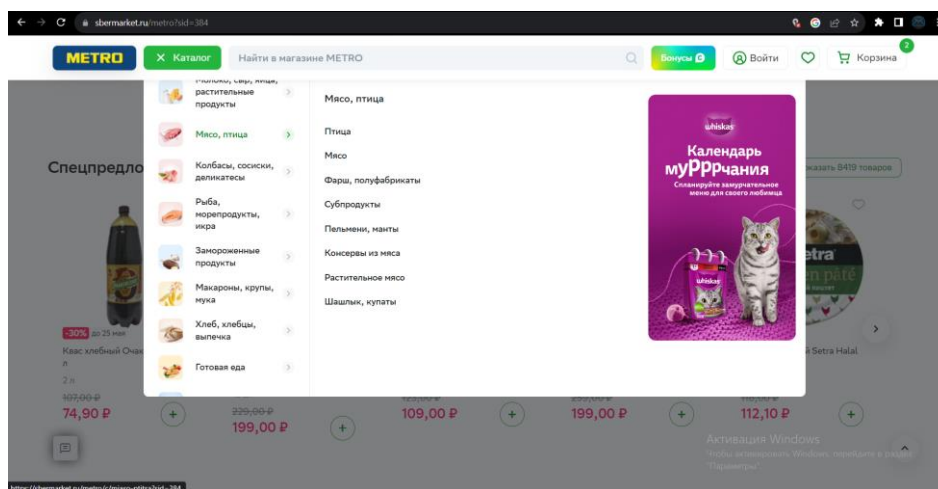


Рисунок 7 – Выбор категории товаров на сайте «СберМаркет»

Далее пользователь добавляет необходимые товары и оформляет заказ, который доставит курьер в указанное пользователем время.

Приложение довольно удобно и предлагает обширный каталог товаров из самых популярных гипермаркетов России.

Доставкой продуктов могут заниматься не только специальные сервисы, но и собственные интернет-магазины сетей («Ашан», «Метро», «Мираторг»), и даже фермерские хозяйства, например ферма «Для своих».

Ферма «Для своих» осуществляет доставку продуктов в пределах Московской области. Заказать продукты можно на сайте, либо через чат-бота в мессенджере телеграм. Далее будет рассмотрен чат-бот.

После открытия диалога с ботом пользователь видит приветственное сообщение с просьбой нажать кнопку «Start» для продолжения (рисунок 8).

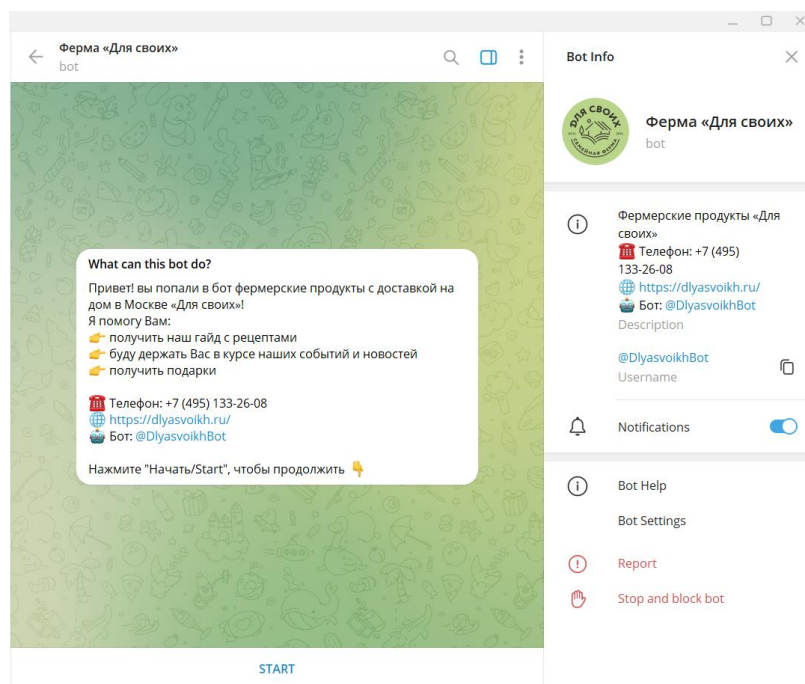


Рисунок 8 – Начало диалога с чат-ботом фермы «Для своих»

После нажатия на кнопку пользователь получает сообщение с краткой информацией о ферме, появляется основное меню бота, которое показано на рисунке 9.

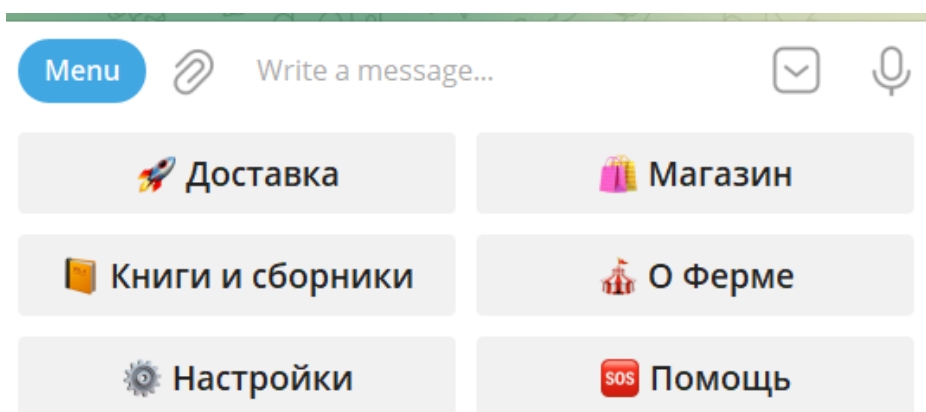


Рисунок 9 – Основное меню чат-бота фермы «Для своих»

Чтобы открыть каталог продуктов, нужно перейти в раздел «Магазин». Когда пользователь переходит в этот раздел, ему отправляется сообщение с категориями продуктов, как это показано на рисунке 10.



Рисунок 10 – Сообщение с категориями продуктов

После выбора категории бот отправляет пользователю сообщение с продуктами, которое показано на рисунке 11. Пользователь может просматривать меню продуктов нажимая на кнопки со стрелками. После нажатия на кнопку «Купить», продукт добавляется в корзину.



Рисунок 11 – Сообщение с продуктами

Для просмотра корзины нужно нажать на кнопку «Корзина». После нажатия кнопки пользователь получает сообщение с добавленными продуктами,

которое показано на рисунке 12. Далее пользователь может очистить корзину, изменить её содержимое, а также оформить заказ.

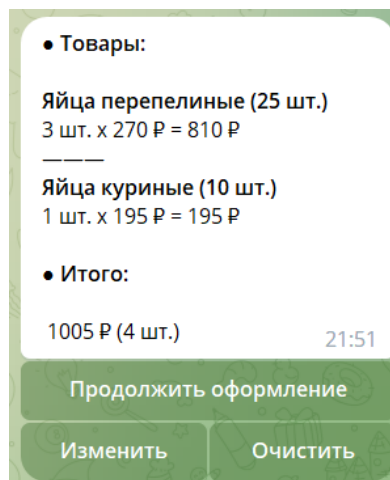


Рисунок 12 – Корзина чат-бота фермы «Для своих»

Далее идет этап выбора способа доставки и оплаты заказа. После пользователь ожидает получения заказа, либо забирает его сам, если выбрал вариант самовывоза.

Чат-бот фермы «Для своих» имеет не такой удобный интерфейс как у приложений «Самоката» и «СберМаркета», однако для заказа продуктов пользователям не нужно скачивать приложение, или открывать браузер. Можно заказать продукты прямо в мессенджере, что значительно упрощает процесс заказа продуктов для некоторых пользователей.

В результате анализа были выделены критерии, которые влияют на удобство и эффективность использования приложения:

- наличие поисковой строки;
- возможность добавить комментарий к заказу;
- интерактивный поиск по категориям;
- запоминание адреса доставки;
- возможность получить помощь в приложении.

Результаты сравнения показаны в таблице 1.

Таблица 1 - Сравнение по критериям

Критерий	«Сбермаркет»	«Самокат»	«Для своих»
Наличие поисковой строки	Есть	Есть	Отсутствует
Возможность добавить комментарий к заказу	Отсутствует	Отсутствует	Отсутствует
Интерактивный поиск по категориям	Есть	Есть	Отсутствует
Запоминание адреса доставки	Есть	Есть	Есть
Возможность получить помощь в приложении	Есть	Отсутствует	Есть

По результатам сравнения можно заключить, что каждое приложение имеет свой функционал, но не способно решить все проблемы, с которыми может столкнуться пользователь.

Таким образом, при разработке пользовательского приложения будут учтены все имеющиеся недостатки и реализуется полный набор необходимых функций.

1.3 Актуальность использования чат-ботов

Для того чтобы обеспечить высококачественное обслуживание клиентов, все больше компаний внедряют чат-ботов в свои бизнес-процессы. В некоторых отраслях, где массовое взаимодействие с клиентами является ключевым элементом, таких как банковское дело, страхование и здравоохранение, использование чат-ботов очень актуально [4].

Есть множество преимуществ использования чат-бота для повышения эффективности бизнеса.

Одним из способов улучшения опыта клиентов является предоставление возможности получить поддержку в любое удобное время. Для обеспечения круглосуточной поддержки в режиме реального времени наиболее эффективными инструментами являются онлайн-чаты и чат-боты.

При поступлении сотни запросов от клиентов ежедневно, обработка всех обращений вручную может быть сложной задачей. В этом случае чат-боты могут оказаться очень полезными.

Важным преимуществом использования чат-ботов для бизнеса является возможность мгновенного ответа клиенту. Скорость ответа может оказать значительное влияние на качество обслуживания клиентов.

Исследование, проведенное экспертом по развитию клиентского сервиса Джеффом Тойстером, показало, что главной причиной жалоб клиентов в Twitter является долгое ожидание ответа. Даже если продукт идеален, но клиенты вынуждены долго ждать, это может серьезно навредить репутации.

Особенно важно быстро отвечать на первый контакт с потенциальным клиентом. Если клиент пытается связаться, но не получает ответа в ближайшее время, то вероятнее всего он обратится к конкуренту. Чем быстрее он получит ответ, тем больше шансов заполучить клиента.

Основное преимущество чат-ботов – автоматизация, которая может помочь компании сократить и оптимизировать расходы.

Пока чат-боты берут на себя рутинные задачи, специалисты компании могут уделить больше внимания клиентам, имеющим более сложные вопросы. Это позволит сократить время ожидания ответа, ускорить процесс решения проблем клиентов и, как следствие, повысить их удовлетворенность.

Чат-бот имеет возможность активно предлагать свою помощь на веб-сайте и сопровождать пользователя на его пути через веб-сайт. Кроме того, он может посоветовать и помочь пользователю, например, помогая ему выбрать продукт или указывая подходящие предложения.

Чат-боты могут собирать данные о пользователях, их предпочтениях и поведении. Это позволяет улучшить качество обслуживания и адаптировать его

под потребности пользователей. Владелец бота сможет проанализировать, что именно движет клиентами и с какими проблемами они сталкиваются, о чём спрашивают и что ищут на сайте. Это позволяет создавать контент и продукты, которые отвечают запросам клиентов.

Чат-боты могут быть интегрированы в различные платформы, такие как социальные сети, мессенджеры, сайты. Это позволяет пользователям общаться с ботом в удобной для них форме, не выходя из своего любимого приложения.

Использование чат-ботов – это тенденция последних лет, которая по сей день остается актуальным. За 2020 год чат-ботов начали использовать множество сервисов, оказывающих услуги онлайн. На текущий момент они справляются с большинством задач различной сложности. Работают круглосуточно и оперативно.

2 Описание требований к системе

2.1 Функциональные требования

Анализ функциональных требований является важным этапом в разработке программного обеспечения. Он позволяет определить, какие функции должно выполнять программное обеспечение, какие данные оно должно обрабатывать и каким образом оно должно взаимодействовать с пользователем и другими системами.

Чат-бот должен отвечать следующим требованиям:

- пользователь должен иметь возможность просматривать весь каталог продуктов;
- пользователь должен иметь возможность просматривать продукты в выбранной категории;
- пользователь должен иметь возможность добавить продукт в корзину;
- пользователь должен иметь возможность изменять количество любого продукта в корзине;
- пользователь должен иметь возможность очистить корзину;
- пользователь должен иметь возможность оформить заказ;
- пользователь должен иметь возможность возвращаться на предыдущие этапы оформления заказа в случае необходимости;
- пользователь должен иметь возможность добавить комментарий к заказу;
- пользователь должен иметь возможность письменно задать вопрос администраторам и получить ответ;
- курьер должен иметь возможность просматривать все свободные оформленные заказы клиентов;
- курьер должен иметь возможность выбрать оформленный заказ клиента для дальнейшей доставки;

– администратор должен иметь возможность ответить на поступающий вопрос пользователя.

Основное назначение административной панели – управление данными сущностей базы данных. Она должна позволять администратору добавлять, удалять, редактировать, просматривать данные.

Учитывая назначение административной панели можно выделить следующие функциональные требования:

- возможность авторизоваться в административной панели;
- возможность управлять данными сущности «Клиент»;
- возможность управлять данными сущности «Курьер»;
- возможность управлять данными сущности «Заказ»;
- возможность управлять данными сущности «Продукт»;
- возможность управлять данными сущности «Категория»;
- возможность управлять данными сущности «Платежная система»;
- возможность управлять данными сущности «Пользователь административной панели».

В результате анализа были описаны основные функции, которые должна выполнять информационная система, что послужит основой для проектирования и реализации конечного продукта.

2.2 Требования к надежности

Основными требованиями к надёжности системы являются: бесперебойное круглосуточное функционирование, защита от неправильного ввода данных, ограничение доступа к административной панели для всех пользователей, кроме администраторов.

Для обеспечения бесперебойного круглосуточного функционирования системы запуск должен производиться на удаленном сервере.

Для обеспечения корректности вводимой информации в системе должна быть реализована проверка ввода. Если входные параметры содержат ошибки,

система должна выдавать соответствующие сообщения и отменять операцию ввода.

Для ограничения доступа к административной панели в системе должна быть реализована авторизация и регистрация администраторов.

2.3 Требования к пользователю информационной системы

Данная система имеет простой и понятный интерфейс, поэтому даже неопытные пользователи могут её использовать.

Система предназначена для 3 типов пользователей:

- клиенты;
- курьеры;
- администраторы.

Все пользователи информационной системы должны быть зарегистрированы в мессенджере телеграм и иметь доступ к интернету. Курьеры должны быть добавлены в систему администраторами через административную панель.

2.4 Требования к информационной и программной совместимости

Информационная система для курьерской службы доставки должна быть запущена в контейнерах docker на удаленном сервере с операционной системой linux ubuntu. Для запуска необходимо чтобы на сервере были установлены зависимости docker и docker-compose.

Все пользователи должны иметь доступ к чат-боту с любого устройства, на котором установлен клиент телеграм и имеется доступ к интернету.

Администраторы должны иметь доступ к административной панели с любого устройства через браузер.

3 Выбор технологий и средств для разработки приложения

3.1 Язык программирования Python

Python – это высокоуровневый язык программирования, который был создан в конце 1980-х годов Гвидо ван Россумом. Он был разработан для упрощения программирования и увеличения производительности разработчиков [5].

Python имеет простой и понятный синтаксис, который делает его легким для изучения и использования. Он также имеет множество библиотек и модулей, которые делают его мощным инструментом для разработки различных приложений, включая веб-приложения, игры и многое другое.

Python поддерживает множество парадигм программирования, включая объектно-ориентированное, функциональное и процедурное программирование. Он также имеет динамическую типизацию, что означает, что тип переменной определяется автоматически во время выполнения программы.

Python является кроссплатформенным языком программирования, что означает, что он может быть использован на различных операционных системах, таких как Windows, Linux и macOS. Он также имеет большое сообщество разработчиков, которые создают и поддерживают множество библиотек и модулей, что делает его еще более привлекательным для использования.

3.2 Среда разработки PyCharm

PyCharm – это интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains. Она предоставляет широкий спектр функций и инструментов для удобной и эффективной разработки приложений на Python.

Одной из главных особенностей PyCharm является его интеллектуальный анализ кода, который позволяет быстро находить ошибки и предлагать исправления. PyCharm также обладает мощным отладчиком, который позволяет

отслеживать выполнение программы и находить ошибки в режиме реального времени [6].

Среда разработки PyCharm интегрируется с системами контроля версий, такими как Git, Mercurial и SVN, что позволяет удобно работать с кодом в команде.

PyCharm также предоставляет возможность создавать виртуальные окружения Python, что позволяет изолировать зависимости и управлять версиями библиотек. Это особенно полезно при разработке приложений, которые используют различные версии библиотек.

В целом, PyCharm – это мощная и удобная среда разработки для Python, которая позволяет ускорить и упростить процесс разработки приложений на этом языке.

Компания JetBrains предлагает на выбор две версии PyCharm:

- PyCharm Professional Edition – платная версия с расширенным функционалом;
- PyCharm Community Edition – бесплатная версия с базовым функционалом.

Для написания программного кода информационной системы была использована версия с базовым функционалом, так как её возможности удовлетворяют требованиям на разработку.

3.3 Django Framework

Django – это высокоуровневый веб-фреймворк на языке Python, который позволяет быстро и легко создавать веб-приложения. Он используется для разработки многих известных сайтов, таких как Instagram, Pinterest и Mozilla.

Django предоставляет множество инструментов для управления базами данных, обработки форм, обработки изображений и многое другое. Он также имеет встроенную систему аутентификации и авторизации, что делает его идеальным выбором для создания безопасных веб-приложений [7].

Одной из главных особенностей Django является его модульность. Он состоит из множества небольших, но мощных компонентов, которые могут быть использованы в различных комбинациях для создания приложений. Это позволяет разработчикам быстро создавать и настраивать приложения, а также легко масштабировать их в будущем.

Django также имеет обширную документацию и активное сообщество разработчиков, что делает его легким в освоении и поддержке.

3.4 Aiogram Framework

Aiogram – это асинхронный фреймворк для создания телеграм ботов на языке Python. Он предоставляет удобный интерфейс для работы с API телеграм и позволяет создавать ботов с различными функциями, такими как обработка сообщений, команд, клавиатур и многое другое. Aiogram имеет множество преимуществ на фоне других фреймворков для разработки чат-ботов.

Aiogram предоставляет простой и интуитивно понятный интерфейс для работы с API телеграм.

Aiogram полностью асинхронный, что позволяет создавать быстрые и отзывчивые боты. Основным преимуществом использования асинхронных запросов является более эффективное использование процессорного времени.

Aiogram поддерживает многопоточность, что позволяет создавать ботов, которые могут обрабатывать несколько запросов одновременно;

Aiogram имеет встроенную машину состояний. Это механизм, который позволяет создавать ботов с состояниями, которые могут изменяться в зависимости от действий пользователя. В Aiogram машина состояний реализована с помощью класса FSMContext [8].

3.5 Docker

Docker – это платформа для контейнеризации приложений. Она позволяет упаковывать приложения и их зависимости в контейнеры, которые могут быть запущены на любой машине, где установлен Docker.

Контейнеры Docker представляют собой легковесные, автономные и переносимые окружения, которые могут быть запущены на любой операционной системе, поддерживаемой Docker. Контейнеры изолируют приложения друг от друга и от хост-системы, что позволяет запускать несколько приложений на одном хосте без конфликтов.

Docker использует файлы Dockerfile для создания образов контейнеров. Dockerfile – это текстовый файл, который содержит инструкции для создания образа контейнера. Образ контейнера – это шаблон, на основе которого создаются контейнеры [9].

Docker также предоставляет множество инструментов для управления контейнерами, включая Docker Compose для управления несколькими контейнерами, Docker Swarm для управления кластерами Docker и Docker Hub для хранения и обмена образами контейнеров.

Преимущества использования Docker включают упрощение развертывания приложений, улучшение портативности приложений, уменьшение затрат на инфраструктуру и улучшение безопасности приложений.

4 Проектирование информационной системы

4.1 Выделение компонентов архитектуры

Можно выделить три основных компонента архитектуры:

- чат-бот;
- административная панель;
- база данных.

Каждый компонент запускается в отдельном docker-сервисе, но при этом они работают вместе, как единое приложение.

Чат-бот связан с административной панелью через базу данных. Администраторы взаимодействуют с системой через веб-браузер и телеграм, а клиенты и курьеры только через телеграм. Наглядно компоненты архитектуры показаны на рисунке 13.

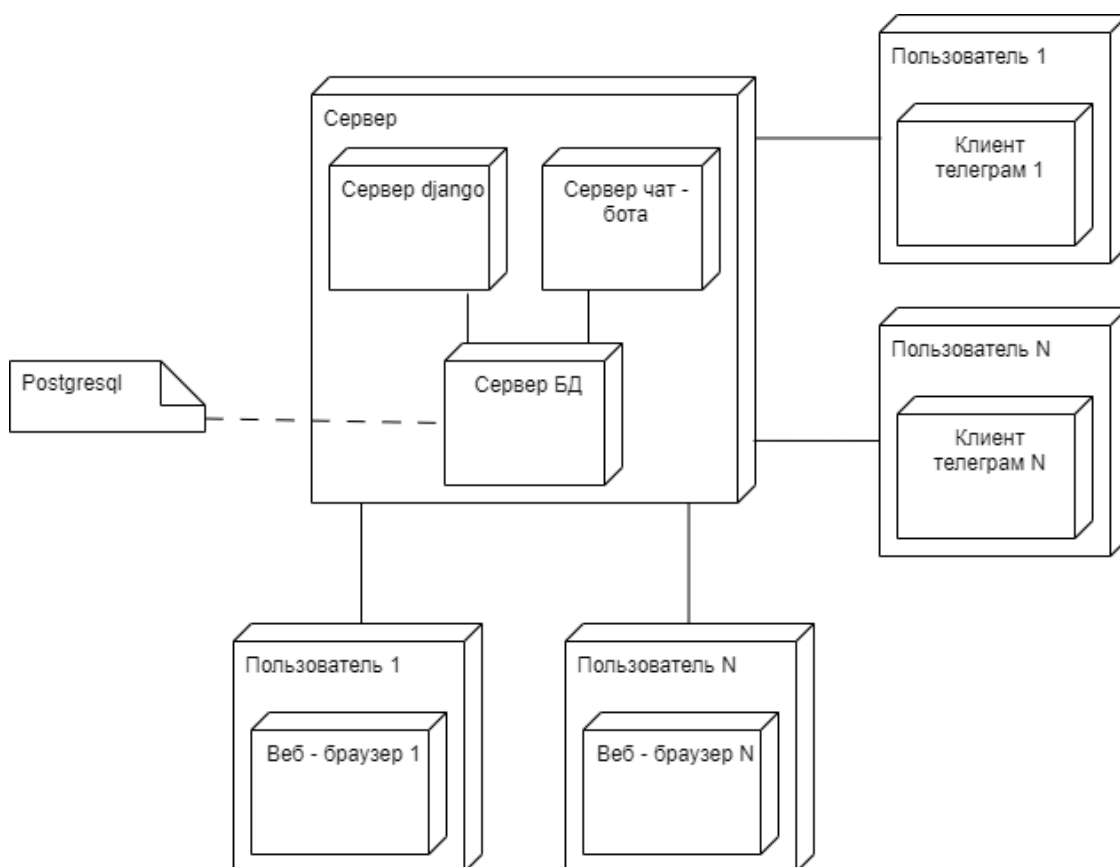


Рисунок 13 – Диаграмма развертывания компонентов архитектуры

Таким образом можно расположить все компоненты архитектуры на одном сервере, и наладить взаимодействие между ними.

4.2 Проектирование чат-бота

Первым архитектурным компонентом является чат-бот, в котором клиенты могут оформлять заказы на доставку продуктов и получать поддержку администраторов при необходимости. Заказ после оформления становится доступен для курьеров, которые могут выбрать его из списка, показываемого чат-ботом. Наглядно возможности клиента можно увидеть на диаграмме вариантов использования, которая показана на рисунке 14.

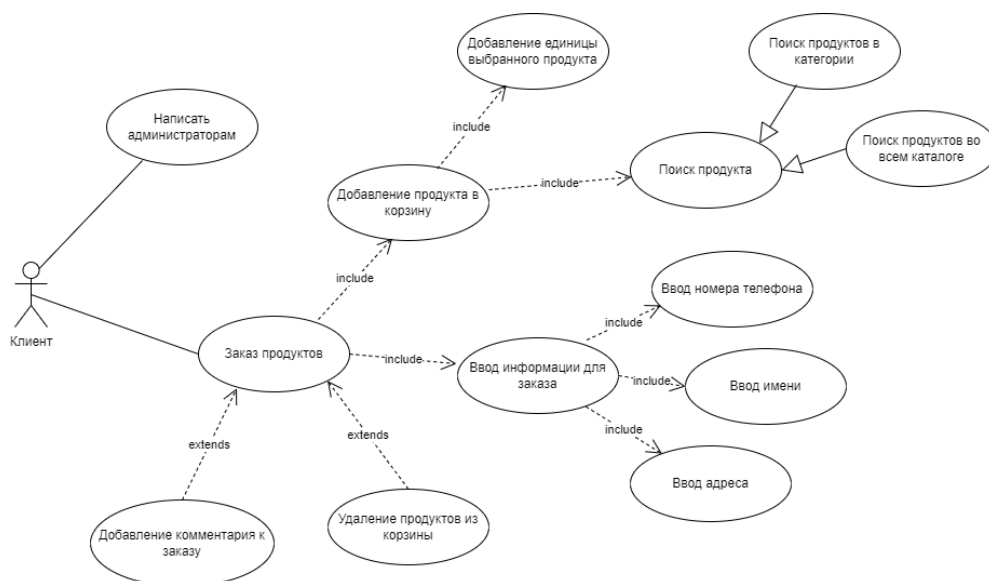


Рисунок 14 – Диаграмма вариантов использования для клиента

Возможности курьера представлены на диаграмме вариантов использования, которая показана на рисунке 15.

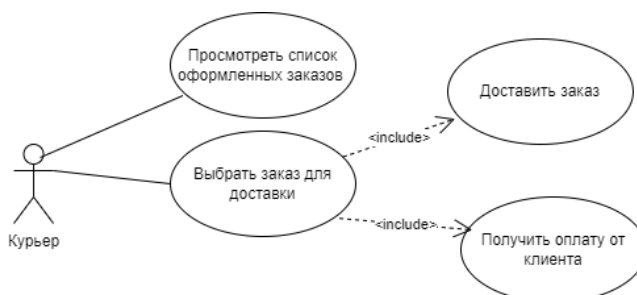


Рисунок 15 – Диаграмма вариантов использования для курьера

В результате были составлены uml-диаграммы вариантов использования для моделирования функциональности системы с точки зрения ее пользователей.

Это помогло наглядно определить, какие действия могут выполнять пользователи системы. На основе полученных диаграмм будет реализован чат-бот.

4.3 Проектирование административной панели

Вторым компонентом архитектуры является административная панель, доступ к которой имеют только администраторы. Она представляет собой веб-приложение, написанное на фреймворке Django. Административная панель позволяет администратору управлять сущностями информационной системы, более наглядно возможности администратора в административной панели показаны на рисунке 16.

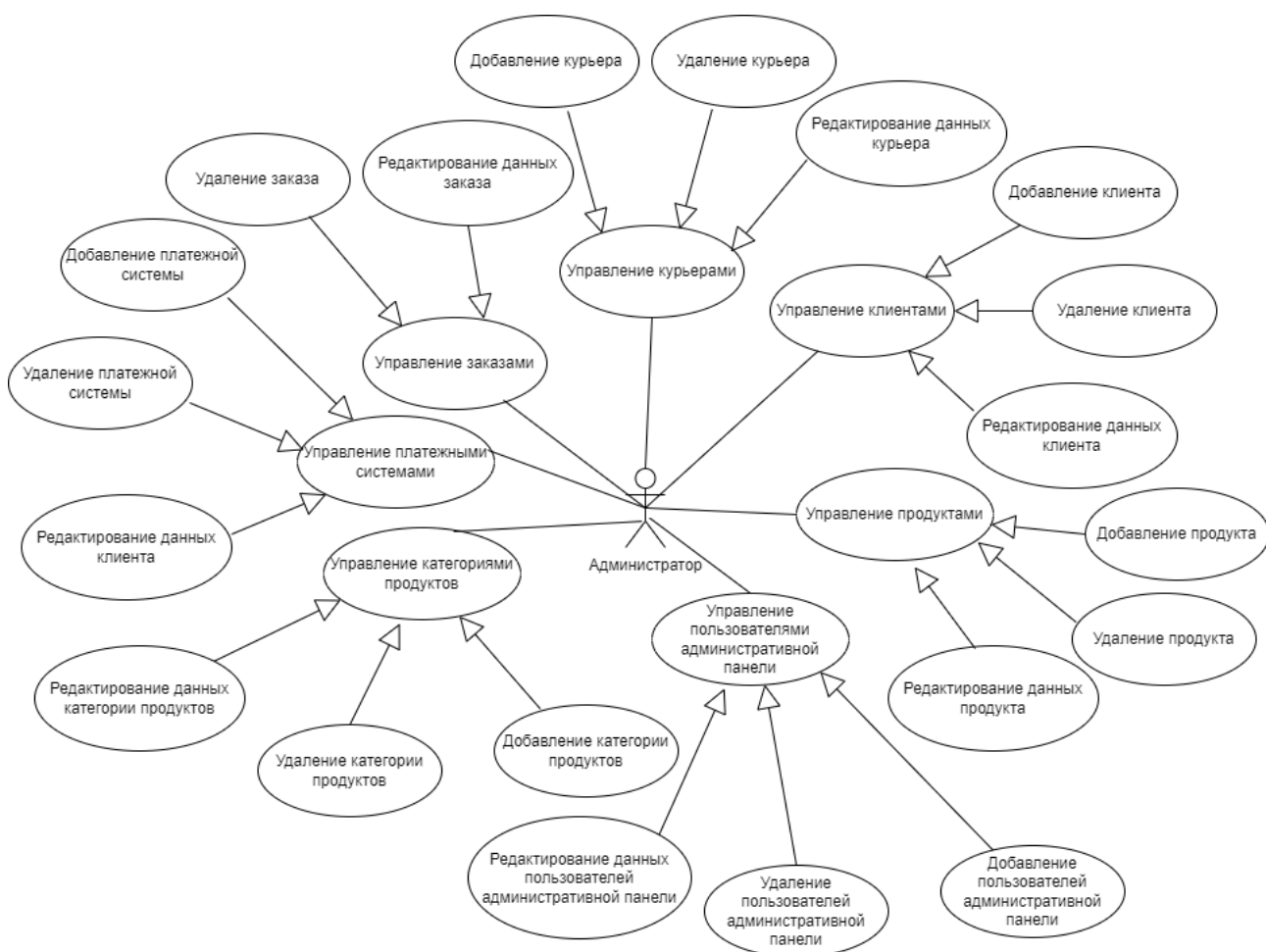


Рисунок 16 – Диаграмма вариантов использования для администратора

Административная панель позволит гибко настраивать систему. Также она позволит отслеживать её работу, например можно просматривать историю заказов или количество клиентов.

4.4 Проектирование базы данных

Последним компонентом архитектуры является база данных. В ней должны храниться данные о клиентах, курьерах, продуктах, платежных системах, категориях продуктов, заказах, адресах, заказанных продуктах. Логическая модель базы данных показана на рисунке 16.

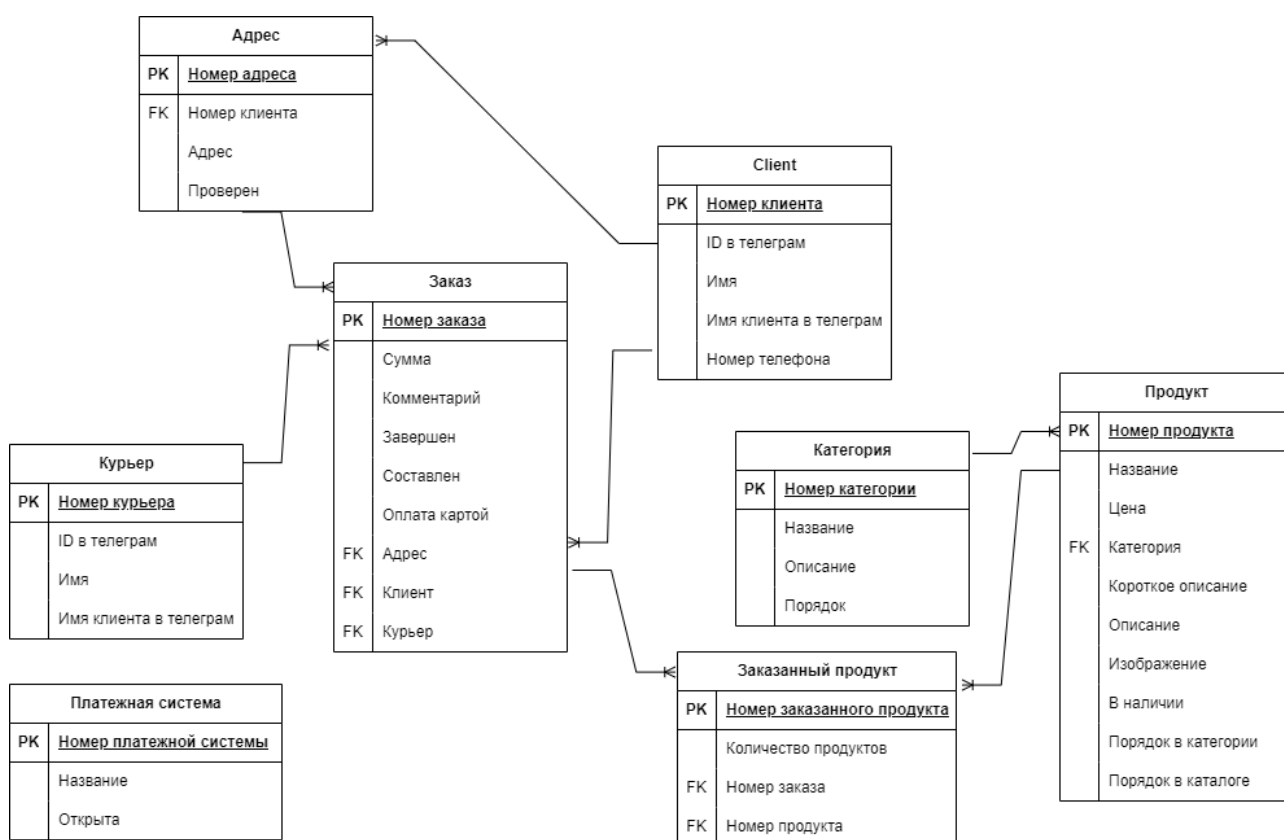


Рисунок 17 – Логическая модель базы данных

Далее необходимо рассмотреть подробнее атрибуты каждой сущности базы данных.

Описание атрибутов сущности «клиент» показано в таблице 2.

Таблица 2 – атрибуты сущности «клиент»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер клиента	Уникальный идентификатор	целый	10	нет	PK	1243
ID клиента в телеграм	Уникальный идентификатор пользователя телеграм	целый	19	нет		821893213
Имя	Имя клиента	символьный	250	да		Алексей
Имя клиента в телеграм	Уникальное имя пользователя телеграм	символьный	250	нет		R3kk3

Описание атрибутов сущности «адрес» показано в таблице 3.

Таблица 3 – атрибуты сущности «адрес»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер адреса	Уникальный идентификатор	целый	10	нет	PK	1243
Номер клиента	Уникальный идентификатор	целый	10	да	FK	1234
Адрес	Адрес доставки	символьный	500	да		Г. Пенза, ул. Чкалова, дом №55
Проверен	Хранит статус адреса. Истина, если адрес был подтвержден	логический	-	нет		True

Описание атрибутов сущности «категория» приведено в таблице 4.

Таблица 4 – атрибуты сущности «категория»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер категории	Уникальный идентификатор	целый	10	нет	РК	1243
Название	Название категории	символьный	250	нет		Мясо
Описание	Описание категории	символьный	4096	нет		Продукты из мяса
Порядок	Порядок показа категории в каталоге чат-бота	целый	10	да		1

Описание атрибутов сущности «продукт» приведено в таблице 5.

Таблица 5 – атрибуты сущности «продукт»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер продукта	Уникальный идентификатор	целый	10	нет	РК	1243
Название	Название продукта	символьный	250	нет		Сыр “Пестравка” 250гр.
Цена	Цена продукта	числовой	30	нет		400
Категория	Уникальный идентификатор	целый	10	нет	FK	1243

Продолжение таблицы 5

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Краткое описание	Описание продукта в каталоге чат-бота	символьный	2048	нет		Полутвердый сыр.
Описание	Описание продукта в карточке продукта в чат-боте	символьный	4096	нет		Полутвердый сыр с нежным слегка кисловатым вкусом и ажурной текстурой.
Изображение	Изображение продукта	двоичный	-	нет		-
В наличии	Количество единиц продукта в наличии	целый	10	нет		5
Порядок в категории	Порядок показа продукта в категории	целый	10	нет		3
Порядок в каталоге	Порядок показа продукта в каталоге	целый	10	нет		2

Описание атрибутов сущности «курьер» приведено в таблице 6.

Таблица 6 – атрибуты сущности «курьер»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер курьера	Уникальный идентификатор	целый	10	нет	РК	1243
ID курьера в телеграм	Уникальный идентификатор пользователя телеграм	целый	19	нет		821893213
Имя	Имя курьера	символьный	250	нет		Денис
Имя пользователя телеграм	Уникальное имя пользователя телеграм	символьный	250	нет		Den2000

Описание атрибутов сущности «продукт» приведено в таблице 7.

Таблица 7 – атрибуты сущности «заказ»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер заказа	Уникальный идентификатор	целый	10	нет	РК	1243
Сумма	Сумма заказа	числовой	30	нет		2300
Комментарий	Комментарий клиента к заказу	символьный	4096	да		Позвоните пожалуйста, когда подъедите к подъезду

Продолжение таблицы 7

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Завершен	Хранит статус заказа. Заказ завершен, если клиент получил его.	логический	-	нет		True
Составлен	Хранит статус заказа. Заказ составлен, если клиент оформил его и ожидает доставки.	логический	-	нет		False
Оплата картой	Хранит факт оплаты заказа картой.	логический	-	нет		False
Адрес	Уникальный идентификатор	целый	10	нет	FK	1243
Клиент	Уникальный идентификатор	целый	10	нет	FK	1243
Курьер	Уникальный идентификатор	целый	10	нет	FK	1243

Описание атрибутов сущности «платежная система» приведено в таблице

Таблица 8 – атрибуты сущности «платежная система»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер платежной системы	Уникальный идентификатор	целый	10	нет	PK	1243
Название	Название платежной системы	символьный	250	нет		QIWI
Открыта	Хранит статус платежной системы. Открыта, если доступна для клиентов	логический	-	нет		True

Описание атрибутов сущности «заказанный продукт» приведено в таблице

9.

Таблица 9 – атрибуты сущности «заказанный продукт»

Наименование	Описание	Тип данных	Размерность	Возможность NULL	Роль	Пример
Номер заказанного продукта	Уникальный идентификатор	целый	10	нет	PK	1243
Количество	Количество единиц продукта	целый	10	нет		5
Номер заказа	Уникальный идентификатор	целый	10	нет	FK	1243
Номер продукта	Уникальный идентификатор	целый	10	нет	FK	1243

Спроектированная база данных содержит в себе все необходимые для работы информационной системы сущности. Атрибуты каждой сущности были подробно описаны, составлена логическая схема базы данных.

5 Реализация информационной системы

5.1 Реализация архитектуры

Архитектура системы была реализована согласно ранее составленной диаграммы развертывания. Три основных компонента запускаются в docker-сервисах. Полная конфигурация сервисов находится в файле docker-compose.yml.

Листинг 1 – Файл docker-compose.yml

```
version: '3.1'

services:
  bot_delivery_products-db:
    container_name: bot_delivery_products-db
    image: postgres:alpine
    restart: always
    environment:
      POSTGRES_USER: delivery_bot
      POSTGRES_PASSWORD: sgsgd43324
      POSTGRES_DB: postgres
  networks:
    - bot_delivery_products
  volumes:
    - ./pgdata:/var/lib/postgresql/data

  delivery_products_bot:
    container_name: bot_delivery_products_bot
    build:
      context: .
    command: python3 app.py
    restart: always
    env_file:
      - .env
    environment:
      POSTGRES_HOST: bot_delivery_products-db
    depends_on:
      - bot_delivery_products-db
      - django_app
    networks:
      - bot_delivery_products

  django_app:
    container_name: django_app
    build:
      context: .
    command: python3 django_app.py runserver 0.0.0.0:80 --noreload
    ports:
      - 80:80
    networks:
      - bot_delivery_products
    restart: always
    env_file:
      - .env
```

```

    depends_on:
      - bot_delivery_products-db
networks:
  bot_delivery_products:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.48.7/24

```

В сервисе «bot_delivery_products-db» запускается база данных. Все данные хранятся в папке «pgdata» вне контейнера, чтобы не терять их при перезапуске системы. В параметре «image» указывается образ контейнера, который будет запущен внутри данного сервиса. В данном случае используется образ postgres:alpine, который содержит PostgreSQL базу данных и минимальную версию Alpine Linux.

В сервисе «delivery_products_bot» запускается чат-бот. В параметре «build» указывается путь к файлу «Dockerfile», который будет использоваться для сборки образа контейнера. В параметре «env_file» указывается, что для загрузки переменных окружения нужно использовать файл «.env». В параметре «command» указывается команда для запуска чат-бота, которая выполняется при запуске сервиса. В параметре «depends_on» указывается сервисы, после которых будет запущен текущий сервис. То есть сервис с чат-ботом запустится последним.

В сервисе «django_app» запускается Django-приложение. Все данные хранятся в папке «pgdata» вне контейнера, чтобы не терять их при перезапуске системы. Сервис также использует файл «.env» для загрузки переменных окружения. В параметре «command» указывается команда для запуска Django-приложения с портом 80. Этот сервис запускается вторым, после запуска сервиса с базой данных.

У всех сервисов в параметре «restart» указано значение «always», это означает, приложению всегда будет работать. Даже если сервис был остановлен по какой-то причине, например, из-за ошибки в приложении или из-за сбоя в системе, он автоматически повторно запустится.

5.2 Реализация административной панели

Административная панель встроена в высокоуровневый фреймворк. Это мощный инструмент управления, позволяющий сразу после установки фреймворка добавлять, удалять или редактировать любую модель базы данных через веб-интерфейс. Но добавив некоторое количество кода, можно настроить административную панель под себя и вывести возможности администрирования на новый уровень [10].

Административная панель доступна только администраторам, следовательно, доступ к ней ограничен. Чтобы его получить, нужно пройти авторизацию. Форма авторизации представлена на рисунке 18.

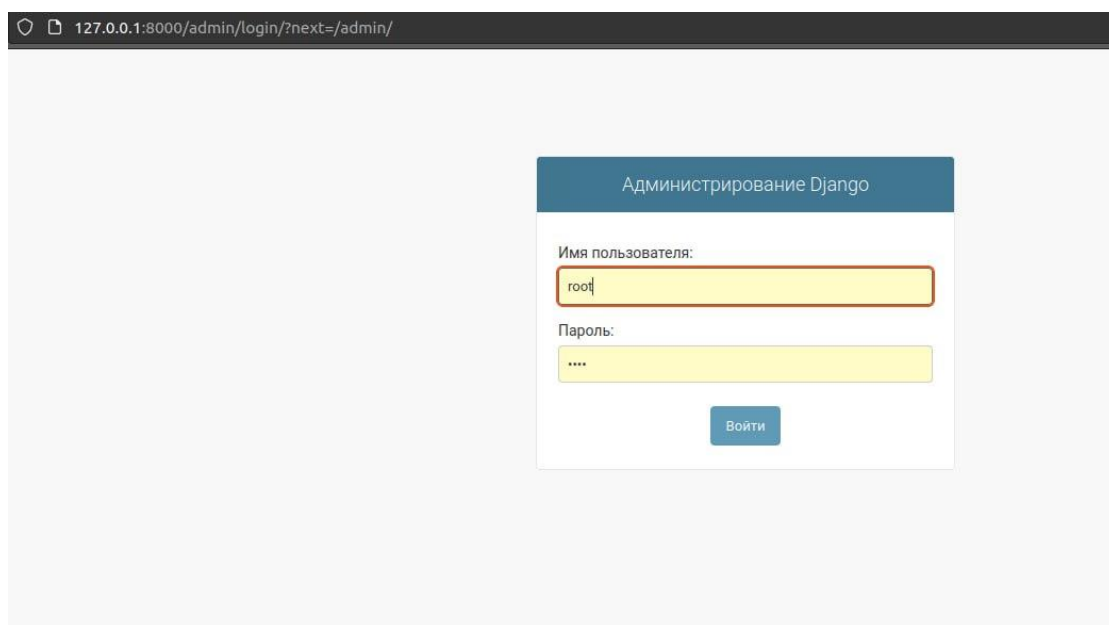


Рисунок 18 – Форма авторизации административной панели

После успешной авторизации пользователь административной панели попадает в главное меню, в котором в левой части страницы расположены 2 раздела – административная панель и телеграм бот. Главное меню показано на рисунке 19.

В разделе «административная панель» располагается только один подраздел – пользователи. Там можно просматривать, редактировать, удалять пользователей административной панели, иными словами администраторов.

В разделе «телеграм бот» расположено 6 подразделов, в каждом из них можно просматривать, добавлять, редактировать, удалять одноименную сущность базы данных.

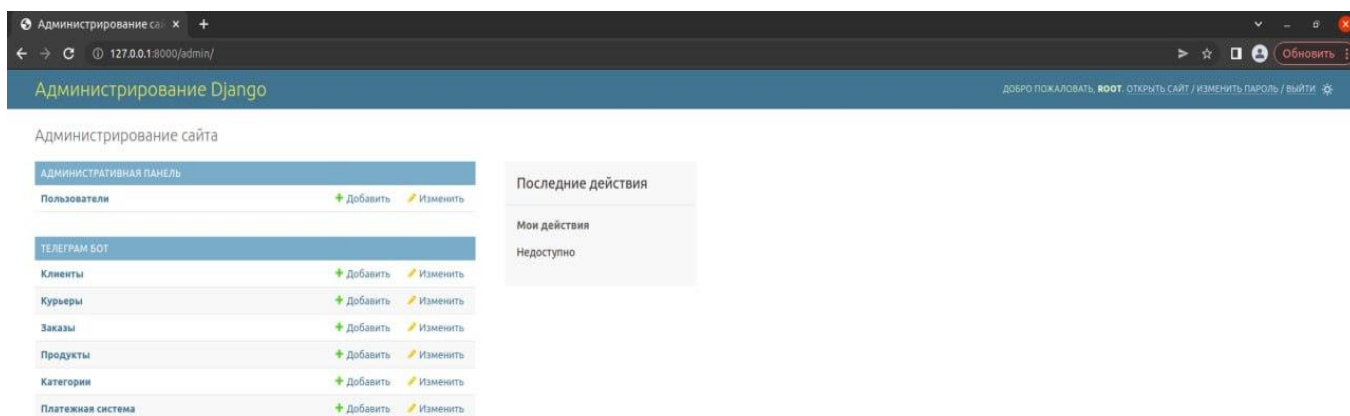


Рисунок 19 – Главное меню административной панели

Рассмотрим подробнее подраздел продукты, скриншот которого показан на рисунке 20. Открыв вкладку, пользователь видит список продуктов, которые на данный момент есть в базе данных. Сверху расположена строка поиска, которая служит для поиска конкретных продуктов. В правой верхней части экрана расположена кнопка для добавления новых продуктов и фильтр, который позволяет отфильтровать продукты по факту наличия на складе.

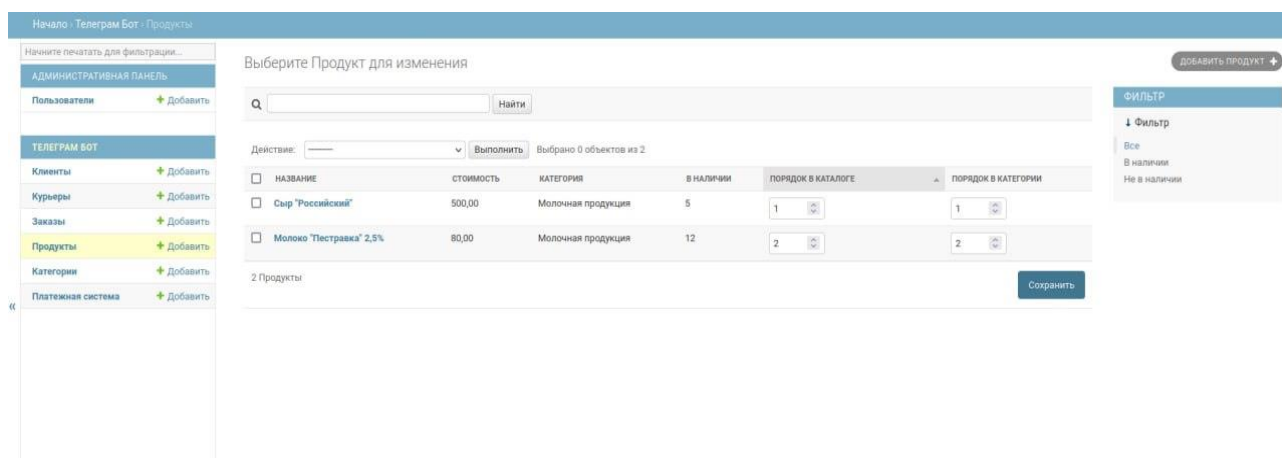


Рисунок 20 – Подраздел продукты

Рассмотрим форму добавления нового продукта, скриншот которой показан на рисунке 21. Форма содержит поля ввода для заполнения информации о продукте. Когда заполнение окончено, можно сохранить новый продукт по одному из трех возможных сценариев:

- сохранить и вернуться в раздел продуктов;
- сохранить и начать добавление нового продукта;
- сохранить и продолжить редактирование продукта.

Рисунок 21 – Форма добавления продукта

В результате удалось реализовать административную панель в виде веб-приложения для гибкой настройки системы. Доступ к панели имеют только администраторы, только у них имеются данные для авторизации в приложении.

5.3 Реализация телеграм бота

Телеграм боты создаются с помощью специального бота «BotFather». Через него можно создать нового бота, настроить его, добавить описание, иконку. Находится специальный бот в мессенджере телеграм, скриншот главной страницы бота представлен на рисунке 22.

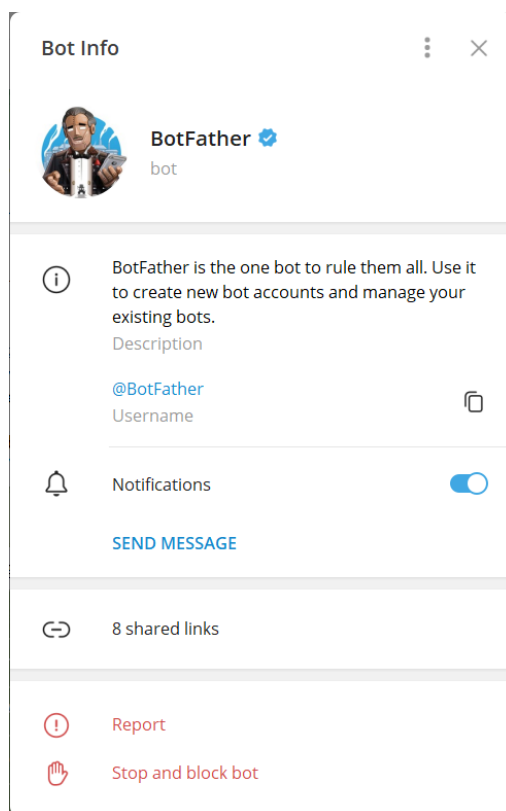


Рисунок 22 – Бот BotFather

После начала диалога с ботом «BotFather» необходимо ввести команду «/newbot» для добавления нового бота. Затем необходимо дать боту название и логин. В результате приходит сообщение с токеном, необходимым для связи бота с телеграм [11]. Процесс создания бота показан на рисунке 23.

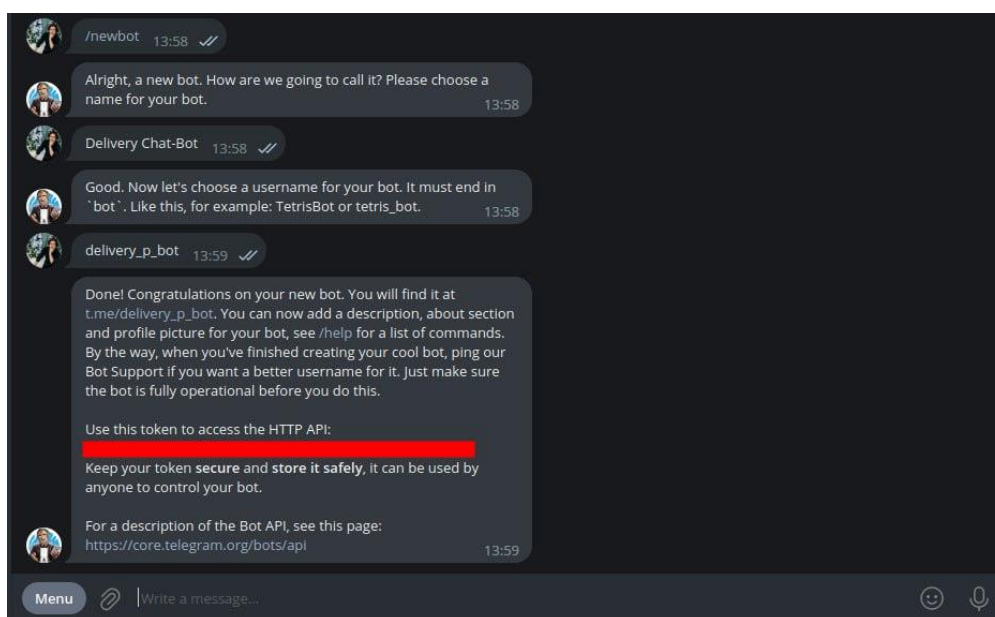


Рисунок 23 – Процесс создания бота

Telegram-бот является разновидностью чат-ботов. По своей сути – это те же пользовательские аккаунты, которыми вместо людей управляют программы. Созданный токен необходим для связи программы и аккаунта чат-бота. При первом открытии диалога с ботом, пользователя встречает окно с кнопкой «Start», которое показано на рисунке 24.

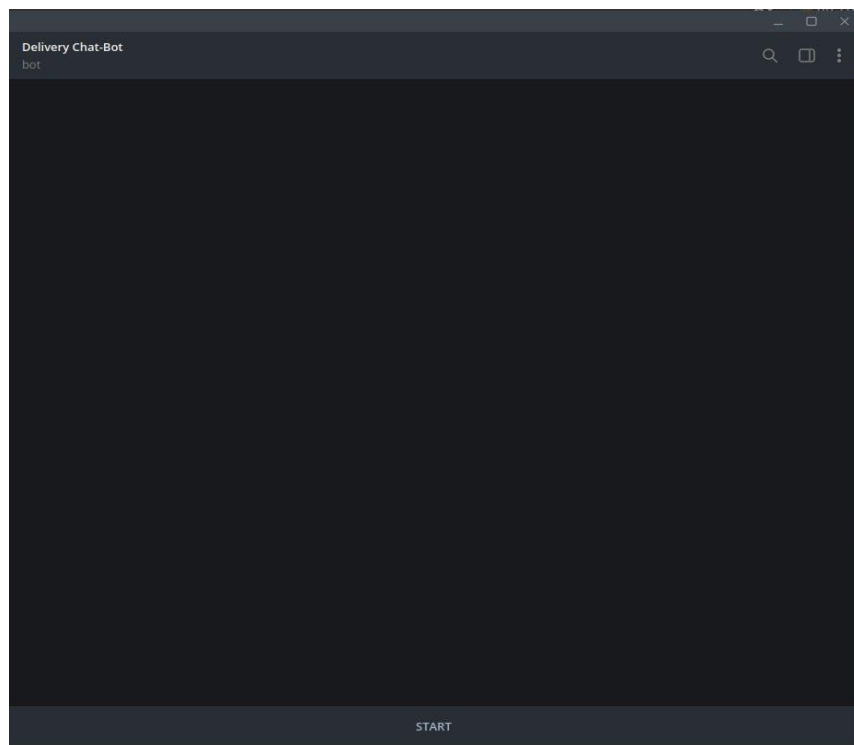


Рисунок 24 – Начальное окно телеграм бота

После нажатия кнопки пользователь попадает в главное меню бота, из которого он может попасть в каталог продуктов, корзину, либо получить помощь от администраторов. Главное меню показано на рисунке 25.

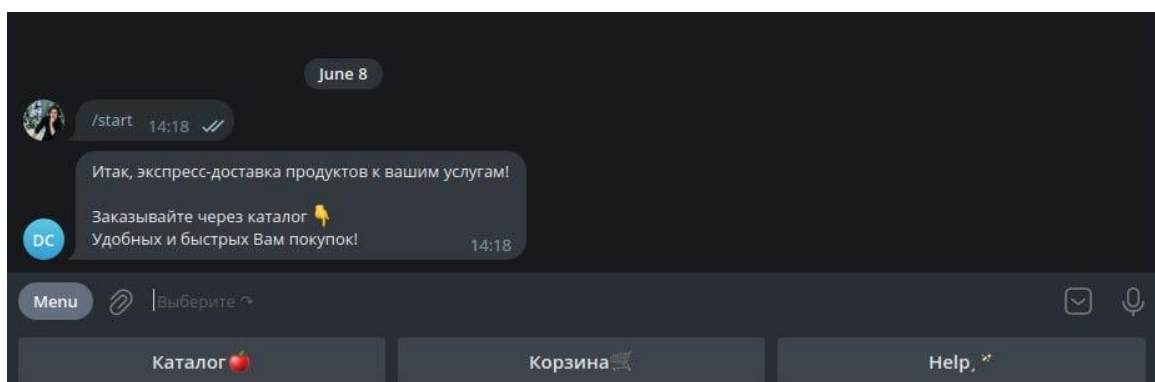


Рисунок 25 – Главное меню телеграм бота

Если пользователь выбрал раздел «Каталог», чат-бот присылает сообщение с категориями продуктов, скриншот которого показан на рисунке 26.

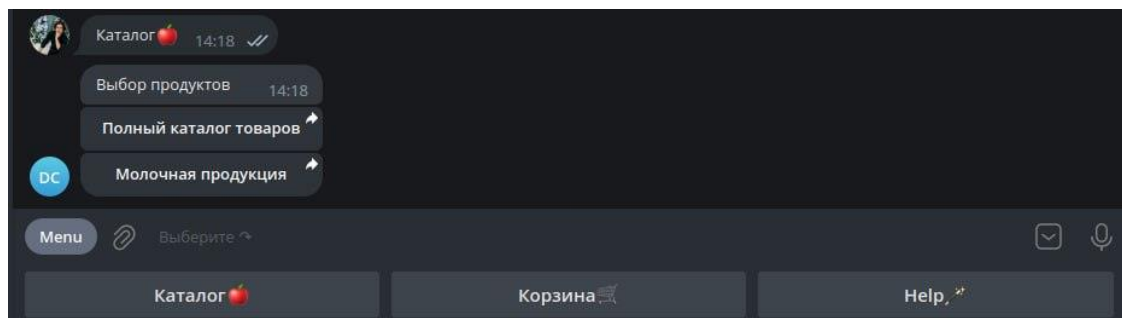


Рисунок 26 – Каталог продуктов телеграм бота

После выбора категории пользователь видит список продуктов этой категории. Список продуктов показан на рисунке 27. Он также может найти конкретный продукт, написав его название в строке ввода.

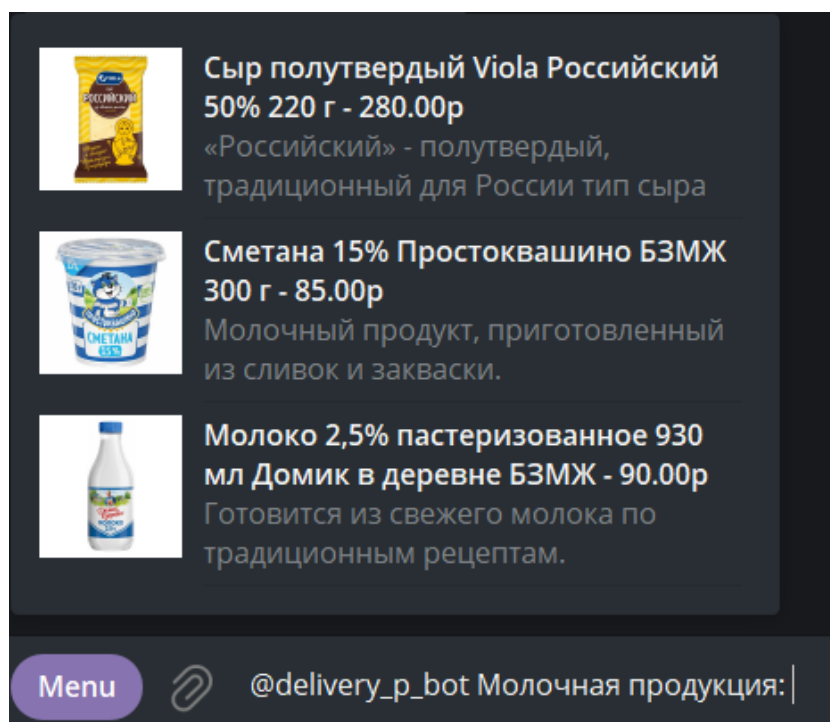


Рисунок 27 – Список продуктов выбранной категории

Когда пользователь выбирает продукт, он получает сообщение с подробной информацией, которое показано на рисунке 28. Затем он может добавить этот продукт в корзину, либо вернуться в каталог.



Рисунок 28 – Сообщение с подробной информацией о продукте

Если пользователь выбрал добавление продукта в корзину, он попадает в меню, в котором может редактировать количество единиц продукта, которое необходимо добавить в корзину. Меню показано на рисунке 29.

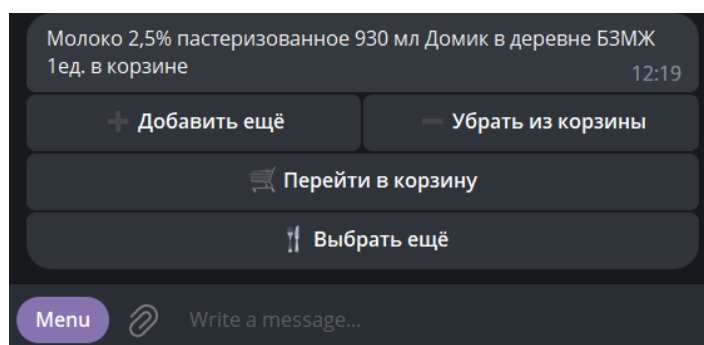


Рисунок 29 – Меню добавление продукта в корзину

Когда продукты в необходимом количестве добавлены, пользователь переходит в корзину, которая показана на рисунке 30. В корзине он может начать заказ продуктов, добавить комментарий к заказу, редактировать заказ, или отменить его.

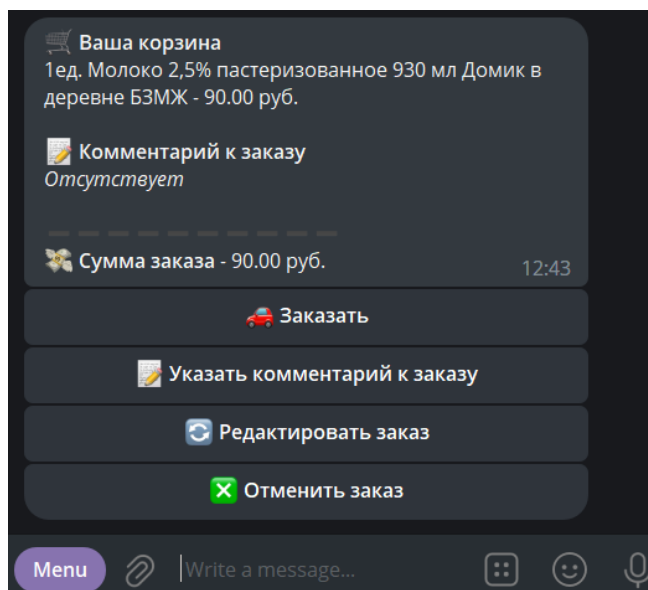


Рисунок 30 – Корзина

Если пользователь нажмет на кнопку «Заказать», чат-бот запросит необходимую для доставки заказа информацию как показано на рисунке 31. На каждом этапе ввода информации пользователь может вернуться на предыдущий этап.

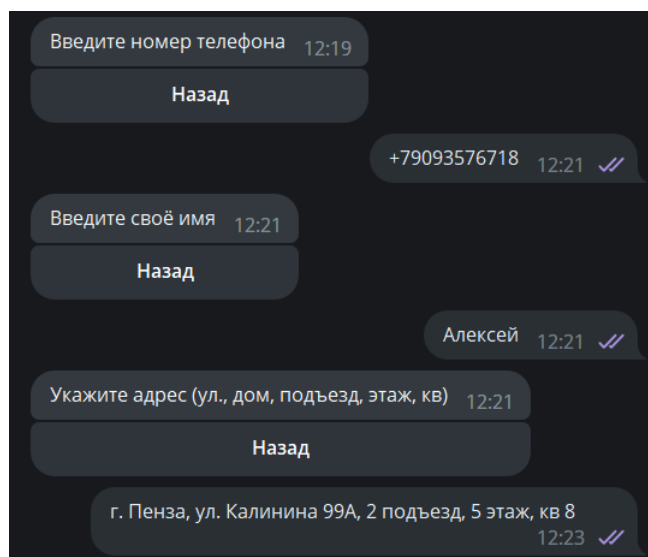


Рисунок 31 – Ввод необходимой для заказа информации

После ввода информации чат-бот присылает сообщение с информацией о заказе, которое показано на рисунке 32. На этом этапе пользователь проверяет правильность заказа, и принимает окончательное решение об оформлении заказа.

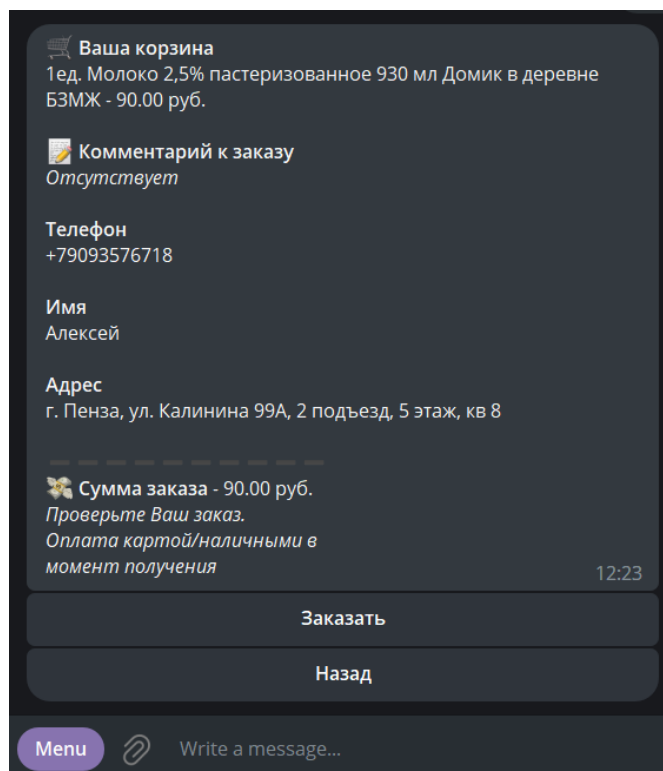


Рисунок 32 – Сообщение с информацией о заказе

Когда пользователь подтвердил оформление заказа, происходит проверка ранее введенного адреса. Если по адресу ранее не доставлялись заказы, чат-бот предлагает пользователю отправить адрес на проверку администратору, сообщение показано на рисунке 33.

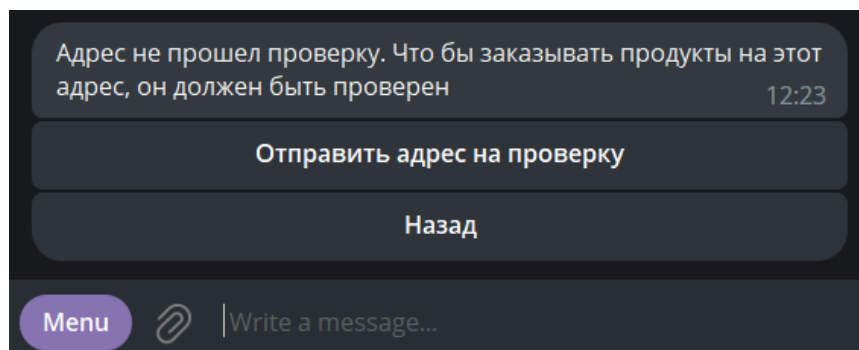


Рисунок 33 – Отправка адреса на проверку администратору

После отправки адреса на проверку, администраторы моментально получают сообщение с возможностью одобрить адрес, или отказать в обслуживании, как это показано на рисунке 34.

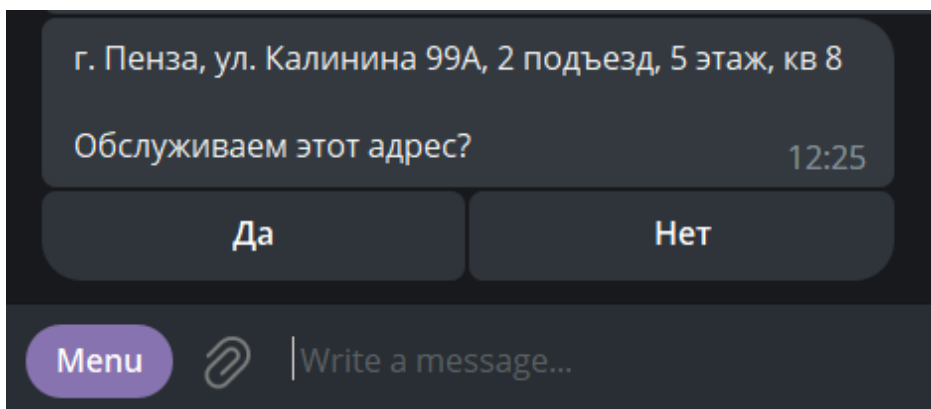


Рисунок 34 – Проверка адреса администраторами

После одобрения адреса заказ считается составленным и доступным для доставки курьерами. Пользователь получает сообщение с просьбой ожидания заказа, как показано на рисунке 35.

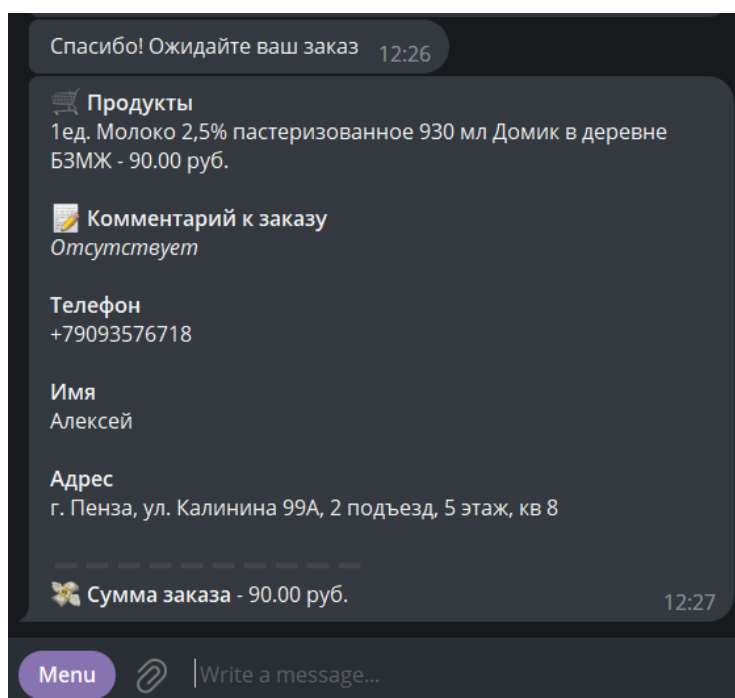


Рисунок 35 – Сообщение с просьбой ожидания заказа

Далее заказ попадает в общий список составленных заказов, в котором любой из курьеров может выбрать конкретный заказ и доставить его клиенту. Для получения списка заказов курьеру необходимо ввести в строку ввода команду «/delivery», как это показано на рисунке 36. Команда доступна только курьерам.

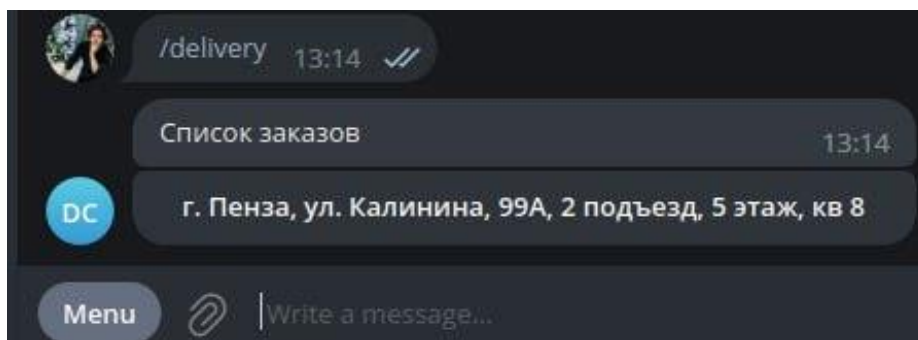


Рисунок 36 – Получение курьером списка составленных заказов

Когда курьер выбирает из списка заказ, он получает подробную информацию о заказе, как это показано на рисунке 37. Далее он может выбрать его для доставки, либо вернуться назад к списку заказов.

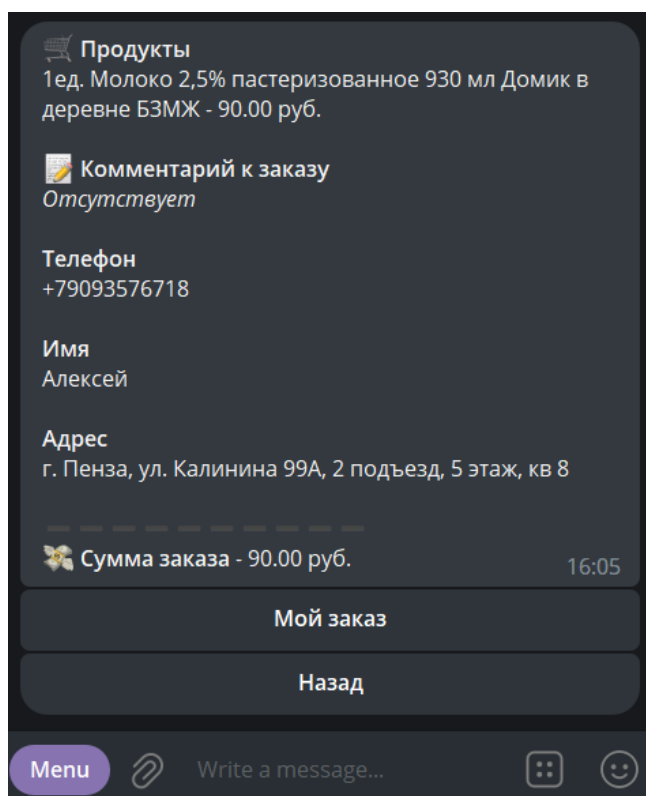


Рисунок 37 – Подробная информация о заказе для курьера

Если курьер выбрал заказ для доставки, под сообщением с информацией о заказе кнопки «Мой заказ» и «Назад» заменяются на кнопку «Доставлено», как это показано на рисунке 38. Эту кнопку курьеру необходимо нажать, когда он доставит заказ клиенту.

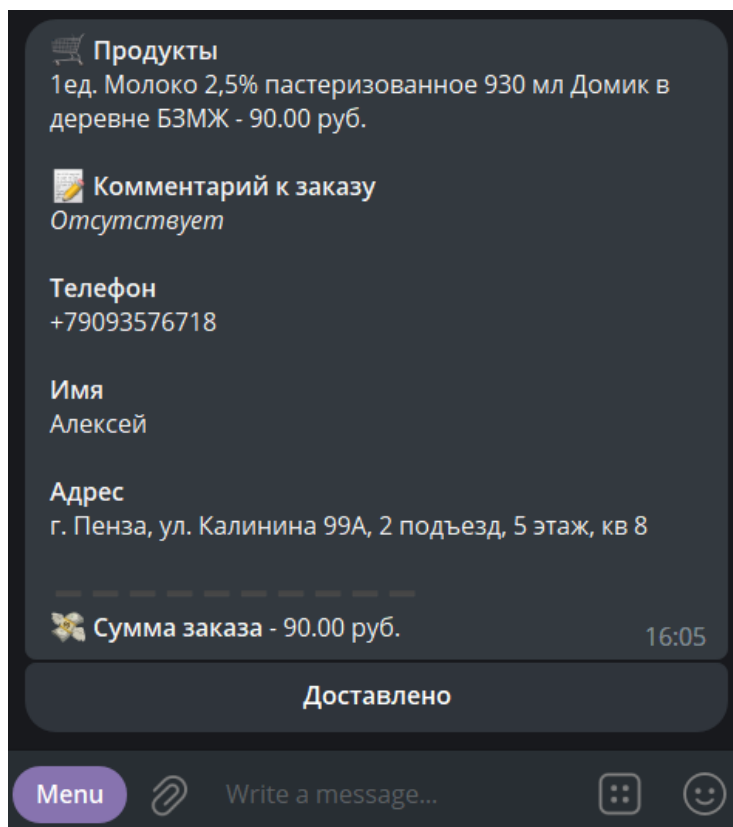


Рисунок 38 – Курьер выбрал заказ для доставки

Когда клиент получил заказ и курьер нажал кнопку «Доставлено», клиенту приходит сообщение для оплаты заказа, которое показано на рисунке 39. Клиент может выбрать один из двух способов оплаты.

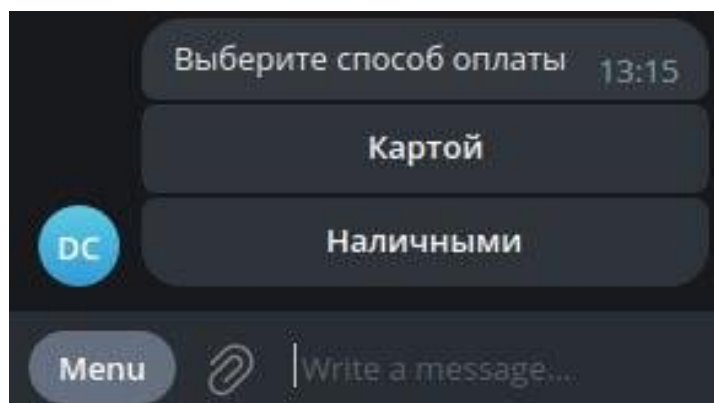


Рисунок 39 – Выбор способа оплаты заказа

Если пользователь выбрал оплату наличными, ему приходит сообщение для подтверждения передачи денежных средств, как показано на рисунке 40.

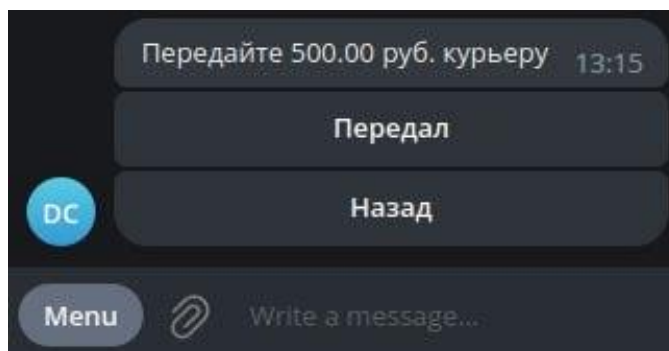


Рисунок 40 – Подтверждение передачи денежных средств

После подтверждения передачи курьер получает сообщение, под которым может подтвердить получение денежных средств. Это сообщение показано на рисунке 41.

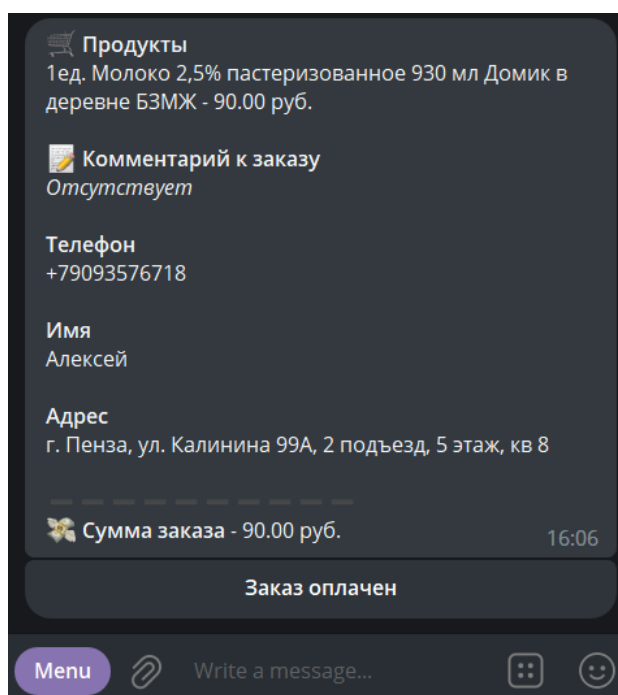


Рисунок 41 – Подтверждение курьером получения денежных средств от клиента

Когда курьер подтвердил оплату заказа, доставка окончена, заказ считается завершенным. Весь процесс оформления и доставки заказа отображен на диаграмме деятельности, которая показана на рисунке 42.

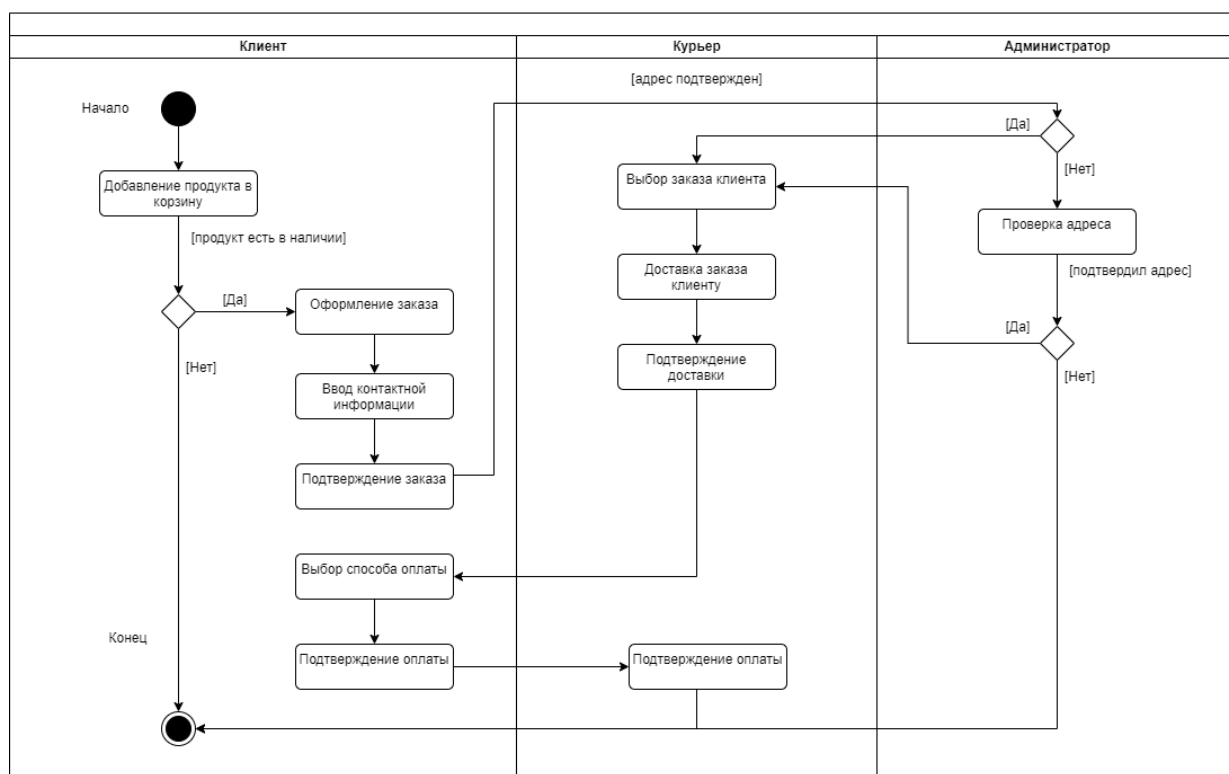


Рисунок 42 – Диаграмма деятельности

В результате был зарегистрирован новый чат-бот в мессенджере телеграм, затем он был разработан согласно требованиям, и подключен к мессенджеру.

5.4 Реализация базы данных

Для реализации базы данных была использована СУБД Postgresql и объектно-реляционное отображение Django.

Объектно-реляционное отображение – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Объектно-реляционное отображение является одной из самых мощных особенностей Django. Оно позволяет взаимодействовать с базой данных, используя код Python, а не SQL.

Для определения моделей базы данных необходимо было описать их в файле models.py на языке Python. Полное содержание файла находится в приложении А.

Затем были реализованы команды для работы с описанными моделями в файле `commands.py`. Полное содержание файла находится в приложении Б.

В результате на основе логической модели была реализована физическая модель базы данных. Построенная физическая модель показана на рисунке 43.

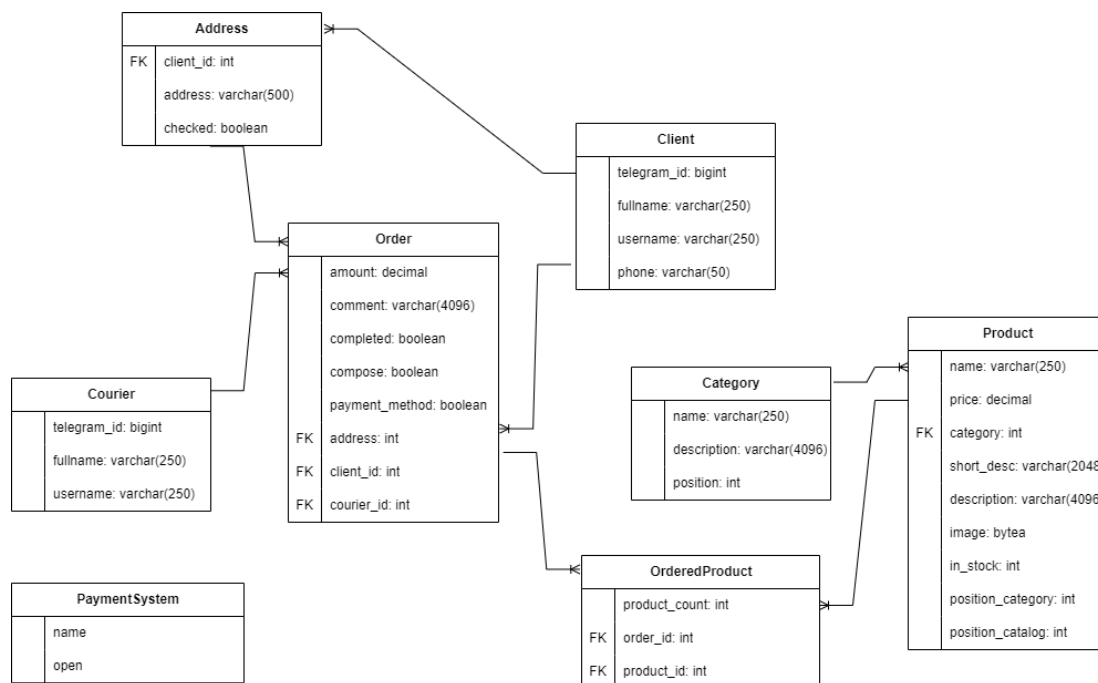


Рисунок 43 – Физическая модель базы данных

Чтобы подключить Django-проект к базе данных, необходимо было изменить содержание переменной `DATABASES` в файле `settings.py`, который находится в директории Django-приложения.

Листинг 2. Измененная переменная `DATABASES` в файле `settings.py`

```

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config.POSTGRES_DB,
        'USER': config.POSTGRES_USER,
        'PASSWORD': config.POSTGRES_PASSWORD,
        'HOST': config.DBHOST,
        'PORT': '5432'
    }
}
  
```

В итоге база данных была реализована на основе ранее составленной логической модели и подключена к Django-приложению. Были написаны все необходимые для работы системы команды для взаимодействия с базой данных.

6 Тестирование основных функций для работы с базой данных

6.1 Выбор метода тестирования

Прежде чем приступить к тестированию основных функций системы, необходимо рассмотреть существующие методы [12]:

- модульное тестирование (юнит-тестирование);
- функциональное тестирование;
- тестирование безопасности;
- регрессионное тестирование;
- тестирование производительности;
- системное тестирование программного обеспечения.

Модульное тестирование – это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу – «белого ящика», то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

Отсутствие модульного тестирования при написании кода значительно увеличивает уровень дефектов при тестировании. Качественное модульное тестирование на этапе разработки экономит время, а следовательно, в конечном итоге, и деньги.

Функциональное тестирование – это тип тестирования программного обеспечения, который проверяет программную систему на соответствие функциональным требованиям. Цель функциональных тестов состоит в том, чтобы проверить каждую функцию программного приложения, предоставляя соответствующий ввод, проверяя выход в соответствии с функциональными требованиями.

Функциональное тестирование в основном включает тестирование черного ящика и не касается исходного кода приложения. Это тестирование проверяет пользовательский интерфейс, API, базу данных, безопасность, связь клиент-сервер и другие функциональные возможности тестируемого приложения. Тестирование может проводиться либо вручную, либо с использованием автоматизации.

Тестирование безопасности – это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Тестирование безопасности может быть ручным или производится при помощи специализированно ПО. Оно включает следующие шесть мер для обеспечения защищенной среды:

- конфиденциальность – защищает от раскрытия информации непреднамеренным получателям;
- целостность – позволяет передавать точную и правильную требуемую информацию от отправителей к предназначенным приемникам;
- аутентификация – проверяет и подтверждает личность пользователя;
- авторизация – указывает права доступа к пользователям и ресурсам;
- доступность – это обеспечивает готовность информации о требованиях;
- безотказность – гарантирует, что от отправителя или получателя не будет отказано в отправке или получении сообщения.

Регрессионное тестирование – это не что иное, как полный или частичный отбор уже выполненных тестовых случаев, которые повторно выполняются для обеспечения нормальной работы существующих функций.

Оно проводится для того, чтобы убедиться, что новые изменения кода не будут иметь побочных эффектов на существующие функциональные возможности. Это гарантирует, что старый код все еще работает, как только последние изменения кода сделаны.

Регрессионное тестирование может быть использовано не только для проверки корректности программы, часто оно также используется для оценки качества полученного результата. Так, при разработке компилятора при прогоне регрессионных тестов рассматривается размер получаемого кода, скорость его выполнения и время компиляции каждого из тестовых примеров.

Тестирование производительности – тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов.

В зависимости от исследуемой характеристики программы, мы можем выделить такие типы тестирования производительности как:

- нагрузочное тестирование – производительность при нормальных условиях;
- стресс-тестирование – работоспособность, производительность и характеристики приложения при экстремальных нагрузках;
- тестирование стабильности – при длительной работе;
- объемное тестирование – при увеличенных объемах обрабатываемых данных.

Системное тестирование – это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы.

Поскольку системное тестирование – процесс, требующих значительных ресурсов, для его проведения часто выделяют отдельный коллектив тестировщиков, а зачастую системное тестирование выполняется организацией, которая не связана с коллективом разработчиков и тестировщиков, выполнявших работы на предыдущих этапах тестирования. При этом необходимо отметить, что при разработке некоторых типов программного

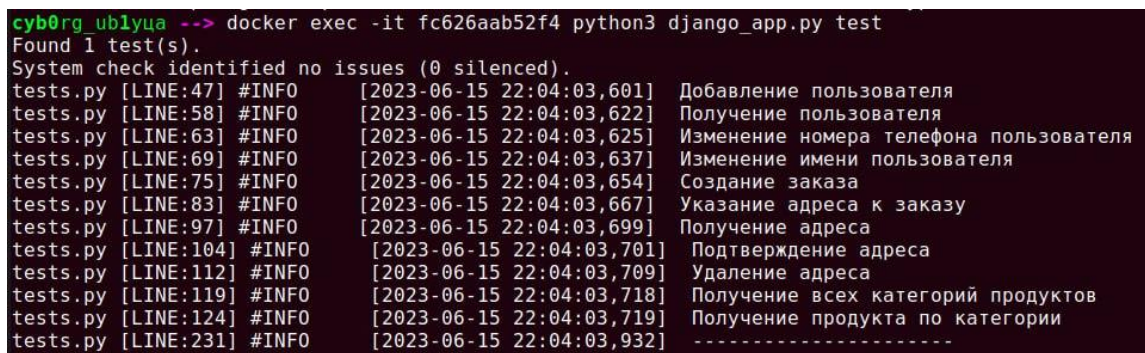
обеспечения (например, авиационного бортового) требование независимого тестирования на всех этапах разработки является обязательным.

Для тестирования функций системы было выбрано модульное тестирование, так как оно позволяло удобно тестировать постепенно добавляемые в ходе разработки функции для работы с базой данных. Также на выбор повлияло то, что Django позволяет очень легко внедрять юнит-тесты в приложения.

6.2 Юнит-тестирование

Чтобы провести тестирование функций для работы с базой данных методом модульного тестирования был создан файл `tests.py` в корне Django-приложения. Django использует юнит-тестовый модуль – встроенный «обнаружитель» тестов, который находит тесты в текущей рабочей директории, в файле с шаблонным именем `tests.py`. Затем в файле были написаны тест-кейсы для каждой функции. Листинг файла `tests.py` находится в приложении В.

В результате тестирования все тесты были успешно пройдены, результат тестирования показан на рисунке 44.



```

cyb0rg_ub1yua --> docker exec -it fc626aab52f4 python3 django_app.py test
Found 1 test(s).
System check identified no issues (0 silenced).
tests.py [LINE:47] #INFO [2023-06-15 22:04:03,601] Добавление пользователя
tests.py [LINE:58] #INFO [2023-06-15 22:04:03,622] Получение пользователя
tests.py [LINE:63] #INFO [2023-06-15 22:04:03,625] Изменение номера телефона пользователя
tests.py [LINE:69] #INFO [2023-06-15 22:04:03,637] Изменение имени пользователя
tests.py [LINE:75] #INFO [2023-06-15 22:04:03,654] Создание заказа
tests.py [LINE:83] #INFO [2023-06-15 22:04:03,667] Указание адреса к заказу
tests.py [LINE:97] #INFO [2023-06-15 22:04:03,699] Получение адреса
tests.py [LINE:104] #INFO [2023-06-15 22:04:03,701] Подтверждение адреса
tests.py [LINE:112] #INFO [2023-06-15 22:04:03,709] Удаление адреса
tests.py [LINE:119] #INFO [2023-06-15 22:04:03,718] Получение всех категорий продуктов
tests.py [LINE:124] #INFO [2023-06-15 22:04:03,719] Получение продукта по категории
tests.py [LINE:231] #INFO [2023-06-15 22:04:03,932] -----

```

Рисунок 44 – Результат модульного тестирования

Таким образом Django-проект был подготовлен для проведения тестирования, затем оно было проведено, в результате чего все тесты были успешно пройдены.

7 Планирование и расчет бюджета

Планирование времени и бюджета проекта поможет определить финансовые ресурсы и сроки выполнения работы, необходимые для успешного завершения проекта. Для такого плана назначаются задачи и ресурсы, которые будут потрачены на их выполнение. Это позволяет определить сроки и стоимость каждого этапа проекта, а также оценить потенциальные риски и проблемы.

Для планирования времени, необходимого на разработку информационной системы для курьерской службы доставки были определены задачи и сроки их выполнения, представленные в таблице 10.

Таблица 10 – задачи проекта

№	Наименование задачи	Дата начала	Дата окончания	Длительность (в часах)
1	Анализ предметной области	12.04.2023	18.04.2023	32
1.1	Описание предметной области	12.04.2023	13.04.2023	8
1.2	Обзор существующих решений	13.04.2023	16.04.2023	16
1.3	Актуальность использования чат-ботов	17.04.2023	18.04.2023	8
2	Описание требований к системе	18.04.2023	23.04.2023	8
2.1	Функциональные требования	18.04.2023	19.04.2023	2

Продолжение таблицы 10

№	Наименование задачи	Дата начала	Дата окончания	Длительность (в часах)
2.2	Требования к надежности	19.04.2023	20.04.2023	2
2.3	Требования к пользователю информационной системы	20.04.2023	21.04.2023	2
2.4	Требования к информационной программной совместимости	21.04.2023	23.04.2023	2
3	Выбор технологий и средств для разработки приложения	23.04.2023	25.05.2023	36
3.1	Язык программирования Python	23.04.2023	23.04.2023	2
3.2	Среда разработки PyCharm	23.04.2023	24.04.2023	2
3.3	Django Framework	24.04.2023	24.04.2023	12
3.4	Aiogam Framework	24.04.2023	25.04.2023	8
3.5	Docker	25.04.2023	25.04.2023	8
4	Проектирование системы	25.04.2023	3.05.2023	40

Окончание таблицы 10

№	Наименование задачи	Дата начала	Дата окончания	Длительность (в часах)
4.1	Выделение компонентов архитектуры	25.04.2023	26.04.2023	8
4.2	Проектирование чат-бота	26.04.2023	28.04.2023	8
4.2	Проектирование административной панели	28.04.2023	29.04.2023	8
4.3	Проектирование базы данных	29.04.2023	03.05.2023	16
5	Реализация	03.05.2023	15.05.2023	96
5.1	Реализация базы данных	03.05.2023	7.05.2023	24
5.2	Реализация чат-бота	07.05.2023	12.05.2023	56
5.3	Реализация административной панели	12.05.2023	15.05.2023	16
6	Тестирование	15.05.2023	17.05.2023	16
6.1	Выбор метода тестирования	15.05.2023	16.05.2023	4
6.2	Юнит-тестирование	16.05.2023	17.05.2023	12
7.	Оформление пояснительной записки	17.05.2023	03.06.2023	56

Для представления задач и их сроков в удобной форме была составлена диаграмма ганта. Такая диаграмма позволяет визуализировать проекты, структурировать их выполнение и видеть все задачи в удобном формате. Диаграмма представлена на рисунке 45.

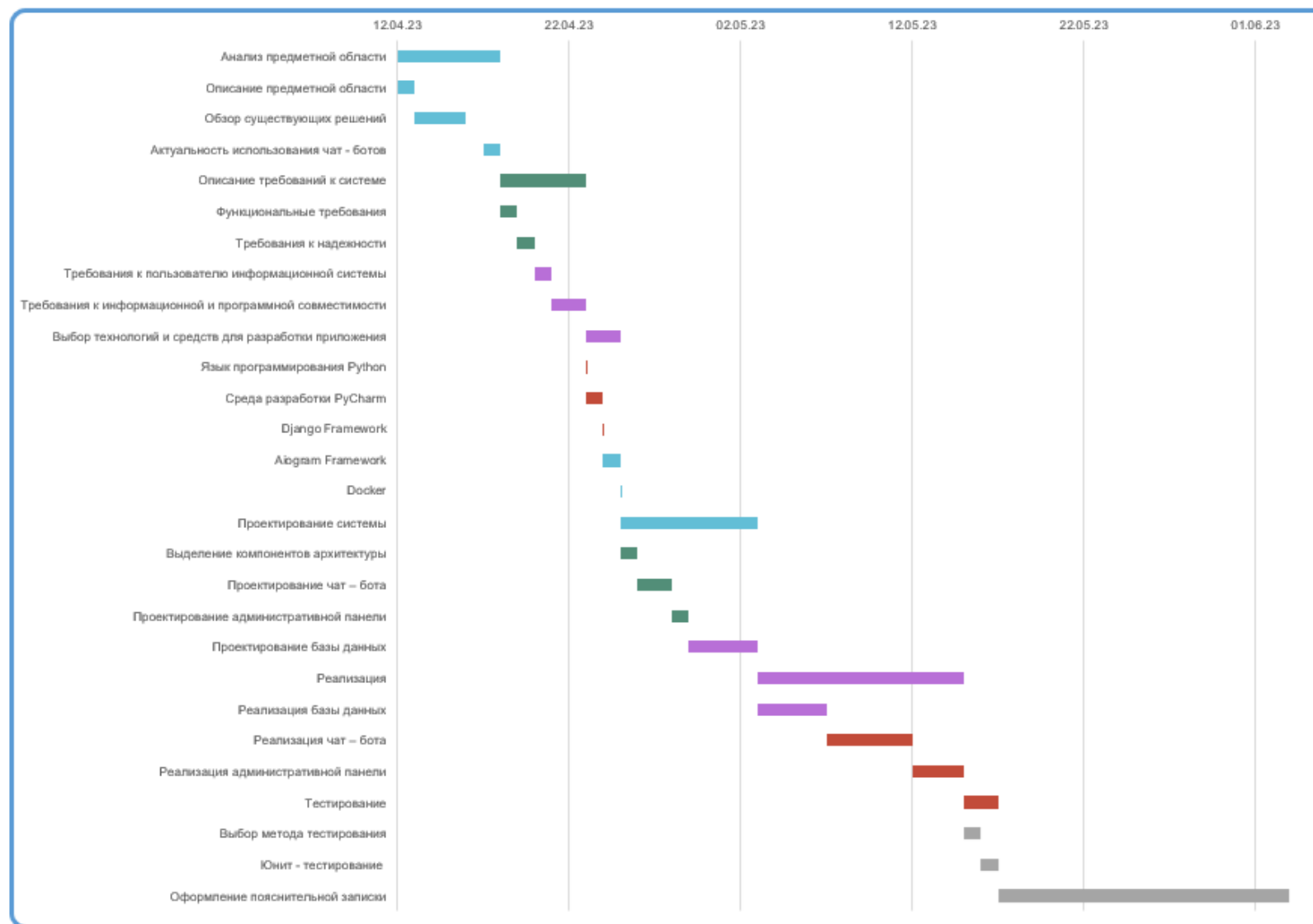


Рисунок 45 – Диаграмма Ганта

Также была подсчитана стоимость выполнения этапов разработки. Для этого были указаны исполнители каждого этапа, длительность их работы и зарплата, получаемая ими за час потраченного на выполнения поставленных задач времени. Результат представлен в таблице 11.

Таблица 11 – длительность и стоимость этапов проекта

№	Наименование этапа	Исполнитель	Длительность (в часах)	Зарплата (руб/ч)	Стоимость
1	Анализ предметной области	Аналитик	32	400	12800
2	Описание требований к системе	Аналитик	8	400	3200
3	Выбор технологий и средств для разработки приложения	Программист	36	700	25200
4	Проектирование системы	Проектировщик	40	350	14000
5	Реализация информационной системы	Программист	96	700	67200
6	Тестирование	Тестировщик	16	350	5600
7	Оформление пояснительной записки	Технический писатель	56	300	16800

В результате было определено время всех этапов и их стоимость. В сумме проект стоит 144 800 рублей, длительность выполнения составляет 284 часа.

Заключение

В ходе работы был проведен анализ предметной области, который показал, что использование чат-ботов на данный момент очень актуально. Все больше компаний внедряют чат-ботов в свои бизнес-процессы, в которых массовое взаимодействие с клиентами является ключевым элементом. Именно поэтому было принято решение о использовании чат-бота в качестве клиентского приложения.

Результатом работы является информационная система для доставки продуктов питания, которая включает в себя чат-бота и административную панель, реализованную в виде веб-приложения.

Чат-бот позволяет клиентам просматривать продукты питания из разных категорий каталога, добавлять продукты в корзину, оформлять доставку продуктов на указанный адрес. В случае возникновения проблем пользователь имеет возможность обратиться за помощью к администраторам через интерфейс бота. Также бот позволяет курьерам просматривать доступные для доставки заказы, доставлять их клиентам.

Административная панель позволяет администраторам управлять всеми необходимыми данными сущностей базы данных.

Для хранения всех необходимых данных и связи чат-бота с административной панелью была реализована база данных. Были написаны команды для взаимодействия с ней.

Компоненты системы запускаются в docker-контейнерах, что позволяет упростить запуск системы, обеспечить её безопасность и бесперебойную работу.

Таким образом разработанный продукт полностью удовлетворяет предъявленным функциональным и не функциональным требованиям.

Были рассмотрены основные методы тестирования программного обеспечения, в результате чего для тестирования основных функций для работы с базой данных был выбран метод модульного тестирования. В результате тестирования все тесты были успешно пройдены.

Работа над проектом позволила расширить знания и улучшить навыки создания чат-ботов и веб-приложений.

В будущем система может быть подвержена улучшению в соответствии с потребностями пользователей. Например, можно реализовать функционал уведомления для курьеров о новых заказах, которые доступны для доставки на данный момент.

Список использованных источников

1. Параскевов А.В. Перспективы и особенности разработки чат-ботов / А.В. Параскевов, А.А. Каденцева, С.И. Мороз // Научный журнал КубГАУ. – 2017. – №130(06).
2. Кудикова М.Г. Развитие интернет-торговли в России в условиях цифровизации / А.Г. Кудикова, Н.М. Сурай, Т.Г. Елистратова, Е.В. Уварова // Инновации и инвестиции. – 2022. – №4. – С. 238-243.
3. Королев Д.П. АНАЛИЗ СЕРВИСОВ, НАПРАВЛЕННЫХ НА ОРГАНИЗАЦИЮ ДОСТАВКИ ПРОДУКТОВ В РОССИИ / Д.П. Королев, М.Ю. Виноградская // Научные известия. – 2022. – №29. – С. 55-59.
4. Бушмелева К.Г. Бот в помощь. Книга-практикум по созданию чат-ботов и автоматизации бизнеса. / К.Г. Бушмелева. – 2022. – С. 106
5. История языка программирования Python [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/История_языка_программирования_Python (дата обращения: 15.05.2023).
6. PyCharm [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: <https://ru.wikipedia.org/wiki/PyCharm> (дата обращения: 15.05.2023).
7. Дронов В. Django: практика создания Web-сайтов на Python. / В. Дронов. – СПб.: БХВ-Петербург. – 2016. – С. 528
8. Aiogram [Электронный ресурс]: Aiogram documentation. Официальная документация. – Режим доступа: <https://docs.aiogram.dev/en/latest/> (дата обращения: 15.05.2023).
9. Моуэт Э. Использование Docker. / Э. Моуэт. – М.: ДМК-Пресс, 2017. – С. 354
10. Django [Электронный ресурс]: Django documentation. Официальная документация. – Режим доступа: <https://docs.djangoproject.com/en/> (дата обращения 20.05.2023).

11. Шумилина М.А. РАЗРАБОТКА ЧАТ-БОТА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON В МЕССЕНДЖЕРЕ «TELEGRAM» / М.А. Шумилина, А.В. Коробко // Научные известия. – 2022. – №28. – С. 51-52.

12. Секлетова Н.Н. МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ / Н.Н. Секлетова, А.И. Кондратьев // Экономика и социум. – 2018. – №10(53). – С. 538-540.

Приложение А
ИСХОДНЫЙ КОД АДМИНИСТРАТИВНОЙ ПАНЕЛИ
(рекомендуемое)

Листинг А.1 – Файл models.py

```

from django.db import models

class TimeBasedModel(models.Model):
    class Meta:
        abstract = True

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

class Client(TimeBasedModel):
    class Meta:
        verbose_name = "Клиент"
        verbose_name_plural = "Клиенты"

    id = models.AutoField(primary_key=True)
    telegram_id = models.BigIntegerField(unique=True,
verbose_name="ID клиента в телеграм")
    fullname = models.CharField(max_length=250, default="Не
указано", verbose_name="Имя клиента", null=True)
    username = models.CharField(max_length=250, default="Не
указано", verbose_name="Username клиента")
    phone = models.CharField(max_length=50, default="Не указано",
verbose_name="Номер телефона", null=True)

    def __str__(self):
        return self.username

class Address(TimeBasedModel):
    class Meta:
        verbose_name = "Адрес"
        verbose_name_plural = "Адреса"

    id = models.AutoField(primary_key=True)
    client_id = models.ForeignKey(Client,
on_delete=models.CASCADE, null=True)
    address = models.CharField(max_length=500, default="Не
указано", verbose_name="Адрес")
    checked = models.BooleanField(default=False,
verbose_name="Проверен")

    def __str__(self):
        return self.address

class Category(TimeBasedModel):
    class Meta:
        verbose_name = "Категория"
        verbose_name_plural = "Категории"
        ordering = ['position']

    id = models.AutoField(primary_key=True)

```

```

        name = models.CharField(max_length=250,
verbose_name="Название", unique=True)
        description = models.TextField(max_length=4096,
verbose_name="Описание")
        position = models.IntegerField(default=0,
verbose_name="Порядок")

    def __str__(self):
        return self.name

class Product(TimeBasedModel):
    class Meta:
        verbose_name = "Продукт"
        verbose_name_plural = "Продукты"
        ordering = ['position_catalog']

    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=250, default="Не указано",
verbose_name="Название")
    price = models.DecimalField(default=0,
verbose_name="Стоимость", max_digits=8, decimal_places=2)
    category = models.ForeignKey(Category, to_field="name",
db_column="category", on_delete=models.CASCADE,
verbose_name="Категория")
    short_desc = models.TextField(max_length=2048,
verbose_name="Краткое описание")
    description = models.TextField(max_length=4096,
verbose_name="Описание")
    image = models.ImageField(upload_to='images',
verbose_name="Изображение")
    in_stock = models.IntegerField(default=0, verbose_name="В
наличии")
    position_category = models.IntegerField(default=0,
verbose_name="Порядок в категории")
    position_catalog = models.IntegerField(default=0,
verbose_name="Порядок в каталоге")

    def __str__(self):
        return self.name

class Courier(TimeBasedModel):
    class Meta:
        verbose_name = "Курьер"
        verbose_name_plural = "Курьеры"

    id = models.AutoField(primary_key=True)
    telegram_id = models.BigIntegerField(unique=True,
verbose_name="ID Курьера")
    fullname = models.CharField(max_length=250, default="Не
указано", verbose_name="Имя курьера", null=True)
    username = models.CharField(max_length=250, default="Не
указано", verbose_name="Username курьера")

```

```

def __str__(self):
    return self.username

class Order(TimeBasedModel):
    class Meta:
        verbose_name = "Заказ"
        verbose_name_plural = "Заказы"

        id = models.AutoField(primary_key=True)
        amount = models.DecimalField(default=0, verbose_name="Сумма",
max_digits=8, decimal_places=2)
        comment = models.TextField(max_length=4096,
default="Отсутствует", verbose_name="Комментарий", null=True)
        completed = models.BooleanField(default=False,
verbose_name="Завершен")
        compose = models.BooleanField(default=False,
verbose_name="Составлен")
        payment_method = models.BooleanField(default=True,
verbose_name="Оплата картой")
        address = models.ForeignKey(Address, to_field="id",
db_column="address", on_delete=models.SET_NULL,
verbose_name="Адрес", null=True)
        client_id = models.ForeignKey(Client, to_field="telegram_id",
db_column="client_id", on_delete=models.CASCADE,
verbose_name="Клиент")
        courier_id = models.ForeignKey(Courier,
to_field="telegram_id", db_column="courier_id",
on_delete=models.SET_NULL, verbose_name="Курьер", null=True)

    def __str__(self):
        return f'№{self.id} - {self.address}'

class PaymentSystem(models.Model):
    class Meta:
        verbose_name = "Платежная система"
        verbose_name_plural = "Платежная система"

        id = models.AutoField(primary_key=True)
        name = models.CharField(max_length=250, default="QIWI",
verbose_name="Название")
        open = models.BooleanField(default=True,
verbose_name="Включить платежный шлюз")

    def __str__(self):
        return self.name

class OrderedProduct(TimeBasedModel):
    class Meta:
        verbose_name = "Продукт"
        verbose_name_plural = "Продукт"

```

```

    id = models.AutoField(primary_key=True)
    product_count = models.IntegerField(default=0,
verbose_name="Количество")
    order_id = models.ForeignKey(Order, on_delete=models.CASCADE)
    product_id = models.ForeignKey(Product,
on_delete=models.CASCADE)

    def __str__(self):
        return self.product_id.name

```

Листинг А.2 – Файл admin.py

```

from django.contrib import admin
from django.contrib.admin import AdminSite, SimpleListFilter
from django.contrib.auth.admin import GroupAdmin, UserAdmin
from django.contrib.auth.models import Group, User
from django.forms.models import BaseInlineFormSet
from django.http import HttpResponseRedirect

from data.config import BOT_TOKEN
from .models import Client, Address, Order, PaymentSystem,
Category, Product, OrderedProduct, Courier

class InStockFilter(SimpleListFilter):
    title = 'Фильтр' # a label for our filter
    parameter_name = 'in_stock' # you can put anything here

    def lookups(self, request, model_admin):
        # This is where you create filter options; we have two:
        return [
            ('в наличии', 'В наличии'),
            ('не в наличии', 'Не в наличии'),
        ]

    def queryset(self, request, queryset):
        # This is where you process parameters selected by use via
filter options:
        if self.value() == 'в наличии':

```

```

        return Product.objects.filter(
            in_stock__gt=0
        ).all()
    elif self.value() == 'не в наличии':
        return Product.objects.filter(
            in_stock__lte=0
        ).all()
    else:
        return Product.objects.all()

class OrderInline(admin.TabularInline):
    model = Order
    verbose_name = 'Заказ'
    verbose_name_plural = 'Заказы'
    ordering = ('updated_at',)
    sortable_by = ('updated_at',)
    fields = (
        'completed',
        'compose',
        'address',
        'courier_id',
        'client_id',
        'amount',
    )
    readonly_fields = (
        'completed',
        'compose',
        'address',
        'courier_id',
        'client_id',
        'amount',
    )
    extra = 0

```

```
class AddressInline(admin.TabularInline):
```

```
    model = Address
```

```
    verbose_name = 'Адрес'
```

```
    verbose_name_plural = 'Адреса'
```

```
    extra = 0
```

```
    can_delete = True
```

```
class OrderedProductInline(admin.TabularInline):
```

```
    model = OrderedProduct
```

```
    verbose_name = 'Продукт'
```

```
    verbose_name_plural = 'Продукты'
```

```
    extra = 0
```

```
    can_delete = True
```

```
class ClientAdmin(admin.ModelAdmin):
```

```
    class Meta():
```

```
        verbose_name_plural = '2. Клиенты'
```

```
    list_display = ('username', 'fullname', 'telegram_id')
```

```
    inlines = [
```

```
        AddressInline,
```

```
    ]
```

```
    readonly_fields = ('telegram_id', 'username', 'fullname')
```

```
class CourierAdmin(admin.ModelAdmin):
```

```
    list_display = ('username', 'fullname', 'telegram_id')
```

```
    inlines = [
```

```
        OrderInline,
```

```
    ]
```

```
class OrderAdmin(admin.ModelAdmin):
```

```
    list_display = ('client_id', 'courier_id', 'address',
```

```

'amount', 'completed')
    search_fields = ('courier_id__username',
'client_id__username', 'address__address', 'amount')
    list_filter = ('completed', 'compose')
    readonly_fields = (
        'completed',
        'compose',
        'address',
        'courier_id',
        'client_id',
        'amount',
        'comment',
    )
    inlines = [
        OrderedProductInline,
    ]

class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'price', 'category', 'in_stock',
'position_catalog', 'position_category')
    search_fields = ('name', 'price', 'category__name')
    list_filter = (InStockFilter, 'category')
    list_editable = ('position_catalog', 'position_category')

class CategoryAdmin(admin.ModelAdmin):
    list_display = ('name', 'position')
    list_editable = ('position',)
    search_fields = ('name',)

class PaymentSystemAdmin(admin.ModelAdmin):
    list_display = ('name', 'open')

```

```

class MyAdminSite(AdminSite):

    def get_app_list(self, request):
        """
        Return a sorted list of all the installed apps that have
        been
        registered in this site.
        """
        app_dict = self._build_app_dict(request)
        app_list = sorted(app_dict.values(), key=lambda x:
x['name'].lower())

        return app_list

admin.site = MyAdminSite()

admin.site.register(Client, ClientAdmin)
admin.site.register(Courier, CourierAdmin)
admin.site.register(Order, OrderAdmin)
admin.site.register(Product, ProductAdmin)
admin.site.register(Category, CategoryAdmin)
admin.site.register(PaymentSystem, PaymentSystemAdmin)
admin.site.register(Group, GroupAdmin)
admin.site.register(User, UserAdmin)

```

Листинг А.3 – Файл settings.py

```

import os
import sys
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.

BASE_DIR = Path(__file__).resolve().parent.parent

```

```

# Quick-start development settings - unsuitable for production
# See
https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = ""

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition
sys.path.append("/src")
from data import config

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'django_project.telegrambot.delivery_bot_manage.apps.DeliveryBotMa
nageConfig',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'django_project.telegrambot.telegrambot.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
    },
]

```

```

        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION =
'django_project.telegrambot.telegrambot.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config.POSTGRES_DB,
        'USER': config.POSTGRES_USER,
        'PASSWORD': config.POSTGRES_PASSWORD,
        'HOST': config.DBHOST,
        'PORT': '5432'

    }
}

TEMPLATE_DIRS = (
    os.path.join("django_project.telegrambot.settings",
'templates'),
)

# Password validation
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityVa
lidator',
    },

```

```

        {
            'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator'
        },
    ],
]

```

```

# Internationalization
# https://docs.djangoproject.com/en/4.0/topics/i18n/

```

```
LANGUAGE_CODE = 'ru-ru'
```

```
TIME_ZONE = 'Europe/Moscow'
```

```
USE_I18N = True
```

```
USE_TZ = True
```

```
STATIC_URL = 'static/'
```

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
MEDIA_URL = '/media/'
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

Листинг А.4 – Файл config.py

```

import logging
import os

```

```
from environs import Env
```

```

# Теперь используем вместо библиотеки python-dotenv библиотеку
environs
env = Env()
env.read_env()

```

```

POSTGRES_DB = env.str("POSTGRES_DB")
POSTGRES_USER = env.str("POSTGRES_USER")
POSTGRES_PASSWORD = env.str("POSTGRES_PASSWORD")
DBHOST = env.str("DBHOST")

QIWI_TOKEN = os.getenv("qiwi")
WALLET_QIWI = os.getenv("wallet")
QIWI_PUBKEY = os.getenv("qiwi_p_pub")

BOT_TOKEN = env.str("BOT_TOKEN") # Забираем значение типа str
ADMINS = env.list("ADMINS") # Тут у нас будет список из админов
IP = env.str("ip") # Тоже str, но для айпи адреса хоста

#POSTGRES_URI =
f"postgres://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{DBHOST}/{POSTGRES_DB}"

```

Листинг А.5 – Файл manage.py

```

import os
import sys

sys.path.append("/src")
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'django_project.telegrambot.telegrambot.settings')
    try:
        from django.core.management import
execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed
and "
            "available on your PYTHONPATH environment variable?
Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

```

```
if __name__ == '__main__':  
    main()
```

Приложение Б
ИСХОДНЫЙ КОД ЧАТ-БОТА
(рекомендуемое)

Листинг Б.1 – Файл loader.py

```
import os

import django
from aiogram import Bot, Dispatcher, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage

from data import config
from utils.misc.address_checker import AddressChecker
from utils.misc.message_sendler import MessageSendler

bot = Bot(token=config.BOT_TOKEN, parse_mode=types.ParseMode.HTML)
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)
address_checker = AddressChecker(bot)
message_sendler = MessageSendler(bot)

os.environ.setdefault(
    "DJANGO_SETTINGS_MODULE",
    "django_project.telegrambot.telegrambot.settings"
)
os.environ.update({"DJANGO_ALLOW_ASYNC_UNSAFE": "true"})
django.setup()
```

Листинг Б.2 – Файл commands.py для курьеров

```
import logging

from aiogram.types import Message, InlineKeyboardButton,
InlineKeyboardMarkup
from handlers.couriers.delivery import gey_orders_list
from loader import dp
from utils.db_api import db_commands

@dp.message_handler(commands=['delivery'], state="*")
async def show_compose_orders(message: Message):
    courier = await
db_commands.select_courier(message.from_user.id)
    if courier:
        keyboard = await gey_orders_list()
        logging.info(keyboard)
```

```

        if keyboard.inline_keyboard:
            await message.answer("Список заказов",
reply_markup=keyboard)
        else:
            await message.answer("На данный момент заказов нет")
    else:
        await message.answer("❌ Вы не инициализированы")

```

Листинг Б.3 – Файл delivery.py

```

from aiogram.types import CallbackQuery, InlineKeyboardMarkup,
InlineKeyboardButton

from loader import dp, bot
from utils.db_api import db_commands

async def gey_orders_list():
    buttons = []
    composed_orders = await db_commands.select_compose_orders()
    for order in composed_orders:
        buttons.append(
            [
                InlineKeyboardButton(
                    text=order.address.address,
                    callback_data=f"chosen_order:{order.id}"
                )
            ]
        )
    return InlineKeyboardMarkup(row_width=1,
inline_keyboard=buttons)

async def get_order_detail(order_id):
    order = await db_commands.select_order(order_id)
    ordered_products = await
db_commands.select_ordered_products(order)

    in_cart_list = [
        f'{p.product_count}ед. {p.product_id} - ' \
        f'{p.product_id.price * p.product_count} руб.\n' for p in
ordered_products
    ]

    in_cart = f"<b>🛒 Продукты</b>\n" + ''.join(in_cart_list)

```

```

order_detail = f"{in_cart}\n"
if order.comment:
    order_detail += f"<b>📝 Комментарий к заказу</b>\n" \
        f"<i>{order.comment}</i>\n"

order_amount = f"\n—————\n" \
    f"<b>💰 Сумма заказа</b> - {order.amount} руб."
phone = f"\n<b>Телефон</b>\n" + f"{order.client_id.phone}\n"
name = f"\n<b>Имя</b>\n" + f"{order.client_id.fullname}\n"
address = f"\n<b>Адрес</b>\n" + f"{order.address.address}\n"
order_detail += phone
order_detail += name
order_detail += address
order_detail += order_amount

return order_detail

@dp.callback_query_handler(text_contains="chosen_order",
state="*")
async def show_order_detail(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    order_detail = await get_order_detail(int(order_id))
    keyboard = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Мой заказ",
callback_data=f"my_order:{order_id}")
            ],
            [
                InlineKeyboardButton(text="Назад",
callback_data="show_compose_orders")
            ]
        ]
    )
    await call.message.edit_text(
        order_detail,
        reply_markup=keyboard
    )

@dp.callback_query_handler(text_contains="my_order", state="*")
async def select_order(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    await db_commands.set_courier(order_id, call.from_user.id)
    keyboard = InlineKeyboardMarkup(

```

```

        inline_keyboard=[
            [
                InlineKeyboardButton(text="Доставлено",
callback_data=f"put_order:{order_id}")
            ]
        ]
    )
    await call.message.edit_reply_markup(
        keyboard,
    )

@dp.callback_query_handler(text_contains="put_order", state="*")
async def o_put(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    await call.message.delete()
    user_id = await db_commands.get_user_id_by_order(order_id)

    payment_window = await db_commands.select_paysys("QIWI")
    if not payment_window.open:
        await db_commands.complete_order(order_id)
        await bot.send_message(
            int(user_id),
            "Платежный шлюз закрыт",
        )
    else:
        keyboard = InlineKeyboardMarkup(
            inline_keyboard=[
                [
                    InlineKeyboardButton(text="Картой",
callback_data=f"in_card:{order_id}")
                ],
                [
                    InlineKeyboardButton(text="Наличными",
callback_data=f"in_cash:{order_id}")
                ]
            ]
        )
        await bot.send_message(
            int(user_id),
            "Выберите способ оплаты",
            reply_markup=keyboard
        )

@dp.callback_query_handler(text_contains="delivery_finish",

```

```

state="*")
async def select_order(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    await db_commands.complete_order(order_id)
    await call.message.edit_text("✅ Заказ закрыт!")

@dp.callback_query_handler(text="show_compose_orders", state="*")
async def back_to_all_orders(call: CallbackQuery):
    keyboard = await get_orders_list()
    await call.message.edit_text("Список заказов",
reply_markup=keyboard)

```

Листинг Б.4 – Файл buy_product.py

```

import hashlib
import logging
import sys
import uuid

from aiogram import types
from aiogram.utils.exceptions import MessageNotModified,
BadRequest
from aiogram.dispatcher import FSMContext
from aiogram.types import CallbackQuery, InputTextMessageContent,
Message, \
    InlineKeyboardMarkup, InlineKeyboardButton
from django.conf.global_settings import MEDIA_ROOT

from keyboards.inline.buy_product_keyboards import
generate_pre_cart_menu
from loader import dp, bot
from states.states import Buy_product
from utils.db_api import db_commands
sys.path.append('/src')

async def get_menu_keyboard():
    buttons = []
    categorys = await db_commands.select_all_categorys()
    buttons.append(
        [
            InlineKeyboardButton(text="Полный каталог товаров",

switch_inline_query_current_chat=': ')
        ]

```

```

    )
    for category in categorys:
        buttons.append(
            [
                InlineKeyboardButton(text=category.name,

switch_inline_query_current_chat=f"{category.name}: ")
            ]
        )
    return InlineKeyboardMarkup(row_width=1,
inline_keyboard=buttons)

async def get_no_compose_order(user_id):
    order = await db_commands.select_no_compose_order(user_id)
    if order:
        return order
    logging.info("CREATE")
    client = await db_commands.select_client(user_id)
    new_order = await db_commands.create_new_order(client)
    return new_order

@dp.inline_handler(state="*")
async def choice_product(inline_query: types.InlineQuery):
    logging.info("Работает")
    try:
        category, name = inline_query.query.split(':')
    except:
        return
    items = await
db_commands.select_products_by_category(category, name)
    logging.info(items)
    result = []
    for item in items:
        logging.info(item.image)
        logging.info(item.image.url)
        randid: str = str(uuid.uuid4())
        result.append(types.InlineQueryResultArticle(
            id=randid,
            title=f"{item.name} - {item.price}p",
            #thumb_url="http://194-58-121-
51.cloudvps.regruhosting.ru/media/images/1617461.jpg",
            description=item.short_desc,
            thumb_url=f"http://176-99-12-
13.cloudvps.regruhosting.ru/{item.image.url}",

```

```

        input_message_content=InputTextMessageContent(
            message_text=f"🔑{item.name}🔑",
            parse_mode='HTML'
        )
    )
    )
    await bot.answer_inline_query(inline_query.id, results=result,
cache_time=1)

@dp.message_handler(text_contains="🔑", state="*")
async def show_product(message: Message):
    request = message.text.replace("🔑", '')
    product = await db_commands.select_products_by_name(request)
    if not product:
        await message.answer("Этого продукта нет в наличии")
        return

    photo = f"http://176-99-12-
13.cloudvps.regruhosting.ru{product.image.url} "

    choice = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Добавить в корзину",
callback_data=f"change_count:{product.id}:1")
            ],
            [
                InlineKeyboardButton(text="Выбрать другой товар",
callback_data="back_to_catalog")
            ]
        ]
    )
    await message.answer_photo(
        photo=photo,
        caption=f"<b>{product.name} - {product.price}</b>\n"
            f"{product.description}",
        reply_markup=choice
    )

@dp.callback_query_handler(text="back_to_catalog", state="*")
async def back_to_catalog(call: CallbackQuery):

```

```

        await Buy_product.wait_product.set()
        keyboard = await get_menu_keyboard()
        await call.message.answer("Выбор продуктов",
reply_markup=keyboard)
        await call.message.delete()

@dp.callback_query_handler(text_contains="change_count",
state="*")
async def change_count(call: CallbackQuery):
    data = call.data.split(':')
    order = await get_no_compose_order(call.from_user.id)
    product = await db_commands.select_products_by_id(data[1])
    logging.info(order)
    ordered_product, error_message = await
db_commands.change_count(order, product, data[-1])
    if ordered_product:
        menu = await generate_pre_cart_menu(data[1])
        try:
            await call.message.edit_text(f"{product.name}
{ordered_product.product_count}ед. в корзине", reply_markup=menu)
        except MessageNotModified:
            pass
        except BadRequest:
            await call.message.answer(f"{product.name}
{ordered_product.product_count}ед. в корзине", reply_markup=menu)
            await call.message.delete()
            logging.info(error_message)
            if error_message:
                await call.answer(error_message)
    else:
        await db_commands.remove_product_from_order(order,
product)
        await back_to_catalog(call)

```

Листинг Б.5 – Файл cart.py

```

from aiogram.dispatcher import FSMContext
from aiogram.types import CallbackQuery, Message,
InlineKeyboardButton, InlineKeyboardMarkup, MediaGroup

from handlers.users.buy_product import get_menu_keyboard
from keyboards.inline.cart_keyboards import save_comment_button,
generate_cart_menu
from loader import dp
from states.states import Buy_product, Cart

```

```

from utils.db_api import db_commands

async def get_cart_detail(upd: Message or CallbackQuery, step=1):
    order = await
    db_commands.select_no_compose_order(upd.from_user.id)
    ordered_products = await
    db_commands.select_ordered_products(order)

    in_cart_list = [
        f'{p.product_count}ед. {p.product_id} - ' \
        f'{p.product_id.price * p.product_count} руб.\n' for p in
ordered_products
    ]
    in_cart = f"<b>🛒 Ваша корзина</b>\n" + ''.join(in_cart_list)

    msg = f"{in_cart}\n"
    if order.comment:
        msg += f"<b>📝 Комментарий к заказу</b>\n" \
            f"<i>{order.comment}</i>\n"

    if step == 3:
        phone = f"\n<b>Телефон</b>\n" +
f"{order.client_id.phone}\n"
        name = f"\n<b>Имя</b>\n" + f"{order.client_id.fullname}\n"
        address = f"\n<b>Адрес</b>\n" +
f"{order.address.address}\n"
        msg += phone
        msg += name
        msg += address

    order_amount = f"\n—————\n" \
        f"<b>💰 Сумма заказа</b> - {order.amount} руб."
    msg += order_amount
    return msg

@dp.callback_query_handler(text_contains="to_cart", state="*")
async def show_cart(upd: Message or CallbackQuery):
    await Cart.menu.set()
    product_id = None
    if type(upd) == CallbackQuery:
        product_id = upd.data.split(':')[1]

    detail = await get_cart_detail(upd)

```

```

menu = await generate_cart_menu(product_id)
if type(upd) == CallbackQuery:
    await upd.message.edit_text(
        detail,
        reply_markup=menu,
        parse_mode="HTML"
    )
    return
await upd.answer(
    detail,
    reply_markup=menu,
    parse_mode="HTML"
)

@dp.callback_query_handler(text="edit_order", state=Cart.menu)
async def edit_order(call: CallbackQuery):
    order = await
db_commands.select_no_compose_order(call.from_user.id)
    ordered_products = await
db_commands.select_ordered_products(order)
    buttons = []
    for ordered_product in ordered_products:
        buttons.append(
            [

InlineKeyboardButton(text=f"{ordered_product.product_id.name} -
{ordered_product.product_count}ед.",

callback_data=f"change_count:{ordered_product.product_id.id}:none"
)
            ]
        )
        buttons.append(
            [
                InlineKeyboardButton(text=f"Назад",
                                     callback_data=f"back_to_cart")
            ]
        )
    keyboard = InlineKeyboardMarkup(row_width=1,
inline_keyboard=buttons)
    await call.message.edit_text(f"Сумма: {order.amount}",
reply_markup=keyboard)

@dp.callback_query_handler(text="back_to_cart", state=Cart.menu)

```

```

async def back_to_cart(call: CallbackQuery):
    await show_cart(call.message)

@dp.callback_query_handler(text="remove_order", state=Cart.menu)
async def remove_order_pre(call: CallbackQuery):
    choice = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Да",
callback_data=f"remove_order_approve"),
                InlineKeyboardButton(text="Нет",
callback_data=f"back_to_cart")
            ]
        ]
    )
    await call.message.edit_text(
        "Вы действительно хотите отменить заказ",
        reply_markup=choice
    )

@dp.callback_query_handler(text="remove_order_approve",
state=Cart.menu)
async def remove_order(call: CallbackQuery):
    client = await db_commands.select_client(call.from_user.id)
    await db_commands.delete_order(client)
    await Buy_product.wait_product.set()
    keyboard = await get_menu_keyboard()
    await call.message.edit_text("Выбор продуктов",
                                reply_markup=keyboard)

@dp.callback_query_handler(text="add_order_comment",
state=Cart.menu)
async def add_comment(call: CallbackQuery):
    await Cart.add_comment.set()
    await call.message.delete()
    await call.message.answer("Введите комментарий")

@dp.message_handler(content_types=['text'],
state=Cart.add_comment)
async def get_comment(message: Message, state: FSMContext):
    comment = message.text

```

```

        await state.update_data(add_comment=comment)
        await message.answer(f"Ваш комментарий\n"
                              f"{comment}",
                              reply_markup=save_comment_button)
        await Cart.save_comment.set()

@dp.callback_query_handler(text="save_comment",
                             state=Cart.save_comment)
async def save_comment(message: Message, state: FSMContext):
    data = await state.get_data()
    comment = data.get("add_comment")
    client = await db_commands.select_client(message.from_user.id)
    await db_commands.add_order_comment(client, comment)
    await show_cart(message)

```

Листинг Б.6 – Файл delivery.py

```

from aiogram.types import CallbackQuery, InlineKeyboardButton,
InlineKeyboardMarkup, Message
from aiogram.utils.markdown import hlink

from handlers.users.cart import get_cart_detail
from keyboards.inline.delivery_keyboards import pay_or_back
from loader import dp
from states.states import Cart, Lk
from utils.db_api import db_commands

@dp.callback_query_handler(text="delivery", state="*")
async def enter_phone(call: CallbackQuery):
    await Lk.edit_phone.set()
    back = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Назад",
callback_data=f"back_to_cart")
            ]
        ]
    )
    await call.message.edit_text(
        "Введите номер телефона",
        reply_markup=back
    )

```

```

@dp.callback_query_handler(text="nm_enter", state="*")
@dp.message_handler(content_types=['text'], state=Lk.edit_phone)
async def enter_name(upd: Message or CallbackQuery):
    await Lk.edit_name.set()
    back = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Назад",
callback_data=f"delivery")
            ]
        ]
    )
    if isinstance(upd, Message):
        await db_commands.change_client_phone(upd.from_user.id,
upd.text)
        await upd.answer(
            "Введите своё имя",
            reply_markup=back
        )
    else:
        await upd.message.edit_text(
            "Введите своё имя",
            reply_markup=back
        )

@dp.callback_query_handler(text="addr_enter", state="*")
@dp.message_handler(content_types=['text'], state=Lk.edit_name)
async def enter_address(upd: Message or CallbackQuery):
    await Lk.edit_addresses.set()
    back = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Назад",
callback_data=f"nm_enter")
            ]
        ]
    )
    if isinstance(upd, Message):
        await db_commands.change_client_name(upd.from_user.id,
upd.text)
        await upd.answer(
            "Укажите адрес (ул., дом, подъезд, этаж, кв)",
            reply_markup=back
        )

```

```

else:
    await db_commands.delete_order_address(
        upd.from_user.id
    )
    await upd.message.edit_text(
        "Укажите адрес (ул., дом, подъезд, этаж, кв)",
        reply_markup=back
    )

@dp.message_handler(content_types=['text'],
state=Lk.edit_addresses)
async def enter_address(message: Message):
    await Cart.menu.set()
    order = await
db_commands.select_no_compose_order(message.from_user.id)
    await db_commands.add_disposable_address(order, message.text)
    detail = await get_cart_detail(message, step=3)
    detail += "\n<i>Проверьте Ваш заказ.\n" \
        "Оплата картой/наличными в \n" \
        "момент получения</i>\n"
    detail += f'\n<code>Нажимая "Заказать", вы соглашаетесь с
\n</code>' \
        f'{{hlink("условиями работы сервиса",
url="https://www.google.com/"))}}'
    await message.answer(
        detail,
        reply_markup=pay_or_back,
        parse_mode="HTML",
        disable_web_page_preview=True
    )

```

Листинг Б.7 – Файл echo.py

```

from aiogram import types
from aiogram.dispatcher import FSMContext

from loader import dp

# Эхо хендлер, куда летят текстовые сообщения без указанного
состояния
@dp.message_handler(state=None)
async def bot_echo(message: types.Message):
    await message.answer("Извини, я тебя не понимаю, воспользуйся

```

```
пожалуйста кнопками или нажми /start")
```

```
# Эхо хендлер, куда летят ВСЕ сообщения с указанным состоянием
@dp.message_handler(state="*",
content_types=types.ContentTypes.ANY)
async def bot_echo_all(message: types.Message, state: FSMContext):
    await message.answer("Извини, я тебя не понимаю, воспользуйся
пожалуйста кнопками или нажми /start")
```

Листинг Б.8 – Файл help.py

```
from aiogram import types
from aiogram.dispatcher.filters.builtin import CommandHelp

from loader import dp

@dp.message_handler(CommandHelp())
async def bot_help(message: types.Message):
    text = ("Список команд: ",
            "/start - Начать диалог")

    await message.answer("\n".join(text))
```

Листинг Б.9 – Файл main_menu.py

```
from aiogram.dispatcher import FSMContext

from aiogram.types import Message

from handlers.users.buy_product import get_menu_keyboard
from handlers.users.cart import show_cart
from keyboards.inline.support_keyboards import
support_main_keyboard, connect_to_support_keyboard
from loader import dp
from states.states import Buy_product, Lk
from utils.db_api import db_commands

@dp.message_handler(text="Каталог🛒", state="*")
async def show_menu(message: Message, state: FSMContext):
    await state.reset_state()
    keyboard = await get_menu_keyboard()
    await Buy_product.wait_product.set()
    await message.answer("Выбор продуктов", reply_markup=keyboard)
```

```

@dp.message_handler(text="Корзина🛒", state="*")
async def show_my_cart(message: Message, state: FSMContext):
    await state.reset_state()
    order = await
db_commands.select_no_compose_order(message.from_user.id)
    if order:
        ordered_products = await
db_commands.select_ordered_products(order)
        if ordered_products:
            await show_cart(message)
            return
    await message.answer("Ваша корзина пуста")

@dp.message_handler(text="Help❓", state="*")
async def support(message: Message, state: FSMContext):
    await state.reset_state()
    await message.answer("Поддержка",
reply_markup=connect_to_support_keyboard)

```

Листинг Б.10 – Файл pay.py

```

from aiogram.dispatcher import FSMContext
from aiogram.types import CallbackQuery, InlineKeyboardMarkup,
InlineKeyboardButton
from aiogram.utils.markdown import hlink

from data.config import ADMINS
from handlers.couriers.delivery import get_order_detail
from handlers.users.cart import get_cart_detail
from keyboards.inline.delivery_keyboards import pay_or_back
from keyboards.inline.pay_keyboards import back_to_cart, to_pay,
to_choice_address, generate_paid_keyboard, \
    generate_to_chose_p_method_keyboard
from loader import dp, address_checker, bot
from states.states import Cart
from utils.db_api import db_commands
from utils.misc.qiwi import Payment, NoPaymentFound, NoEnoughMoney

async def complete(order_id):
    await db_commands.complete_order(order_id)
    order = await db_commands.select_order(order_id)
    await bot.send_message(

```

```

        order.courier_id.telegram_id,
        f"✅ Заказ №{order_id} закрыт"
    )
    await send_to_managers(order, step=2)
    await bot.send_message(
        order.client_id.telegram_id,
        "✅ Оплата прошла успешно"
    )

async def send_to_managers(order, step):
    if step == 1:
        detail = await get_order_detail(order.id)
        for admin in ADMINS:
            await bot.send_message(
                admin,
                detail
            )
    elif step == 2:
        for admin in ADMINS:
            await bot.send_message(
                admin,
                f"Клиент @{order.client_id.username} "
                f"оплатил заказ №{order.id} на сумму - "
                f"{order.amount} руб."
            )

async def check_address(call, address_id):
    keyboard = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Отправить адрес на
проверку", callback_data=f"address_send:{address_id}")
            ],
            [
                InlineKeyboardButton(text="Назад",
callback_data=f"to_pay")
            ]
        ]
    )
    await call.message.edit_text(
        "Адрес не прошел проверку. Что бы заказывать продукты на
этот адрес, он должен быть проверен",
        reply_markup=keyboard
    )

```

```

async def compose_order(user_id):
    order = await db_commands.select_no_compose_order(user_id)
    ordered_products = await
db_commands.select_ordered_products(order)
    for ordered_product in ordered_products:
        new_count = ordered_product.product_id.in_stock -
ordered_product.product_count
        if new_count < 0:
            return "✗ Заказ не был собран. Пожалуйста обратитесь
в поддержку."
        ordered_product.product_id.in_stock = new_count
        ordered_product.product_id.save()

    order.compose = True
    order.save()
    return "Спасибо! Ожидайте ваш заказ в течение 10 минут"

async def check_in_stock(order):
    not_in_stock = []
    ordered_products = await
db_commands.select_ordered_products(order)
    for ordered_product in ordered_products:
        if ordered_product.product_count >
ordered_product.product_id.in_stock:
            not_in_stock.append(ordered_product.product_id)

    if not_in_stock:
        answer = "✗ Ошибка при оформлении заказа. Нужного
количества продуктов нет в наличии" \
            "\n—————\n" \
            "<b>Не в наличии</b>\n"
        for product in not_in_stock:
            answer += f"{product.name}\n"
        return answer, False

    return None, True

@dp.callback_query_handler(text="compose", state=Cart.menu)
async def order_compose(call: CallbackQuery):
    order = await
db_commands.select_no_compose_order(call.from_user.id)

```

```

'''
Проверка адреса. Времено отключена

if not order.address.checked:
    await check_address(call, order.address.id)
    return
'''

answer, in_stock = await check_in_stock(order)
if not in_stock:
    await call.message.edit_text(
        answer,
        reply_markup=back_to_cart,
        parse_mode="HTML"
    )
    return

answer = await compose_order(call.from_user.id)
await call.message.edit_text(answer)
await send_to_managers(order, step=1)

@dp.callback_query_handler(text_contains="in_cash", state="*")
async def in_cash_paid(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    order = await db_commands.select_order(order_id)
    keyboard = await generate_to_chose_p_method_keyboard(order_id)
    await call.message.edit_text(
        f"Передайте {order.amount} руб. курьеру",
        reply_markup=keyboard
    )

@dp.callback_query_handler(text_contains="transfer", state="*")
async def transf(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    await call.message.delete()
    order = await db_commands.select_order(order_id)
    keyboard = InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Заказ оплачен",
callback_data=f"in_nal_p:{order_id}")
            ]
        ]
    )
    order_detail = await get_order_detail(int(order_id))

```

```

await bot.send_message(
    order.courier_id.telegram_id,
    order_detail,
    reply_markup=keyboard
)

@dp.callback_query_handler(text_contains="in_nal_p", state="*")
async def paid_handler(call: CallbackQuery):
    order_id = call.data.split(":")[-1]
    await call.message.delete()
    await complete(order_id)

@dp.callback_query_handler(text_contains="in_card", state="*")
async def init_pay(call: CallbackQuery, state: FSMContext):
    order_id = call.data.split(":")[-1]
    '''
    payment_window = await db_commands.select_paysys("QIWI")
    if not payment_window.open:
        answer = await db_commands.complete_order(order_id)
        await call.message.edit_text(answer)
        return
    '''
    order = await db_commands.select_order(order_id)
    amount = order.amount
    payment = Payment(amount=amount)
    payment.create()
    await state.update_data(pay=payment)

    paid_keyboard = await generate_paid_keyboard(order_id)
    await call.message.edit_text(
        "\n".join(
            [
                f"Оплата {amount:.2f} руб.",
                "",
                hlink("Ссылка для оплаты", url=payment.invoice)
            ]
        ),
        reply_markup=paid_keyboard
    )

@dp.callback_query_handler(text_contains="paid", state="*")
async def paid_handler(call: CallbackQuery, state: FSMContext):

```

```
data = await state.get_data()
payment: Payment = data.get("pay")
order_id = call.data.split(":")[-1]
try:
    payment.check_payment()
except NoPaymentFound:
    await call.answer("Транзакция не найдена")
    return
except NotEnoughMoney:
    await call.answer("Сумма оплаты меньше итога чека")
    return
else:
    await call.message.delete()
    await complete(order_id)

await state.finish()
```

@dp.callback_query_handler(text_contains="address_send",
state=Cart.menu)

async def send_addr(call: CallbackQuery, state: FSMContext):

```
    address_id = call.data.split(":")[-1]
    address = await db_commands.select_address(address_id)
    await
address_checker.send_address_to_check(str(call.from_user.id),
address, place=2)
    await call.message.edit_text(
        "Адрес проходит проверку. В скором времени вы "
```

```
        "получите уведомление с результатом"
```

```
)
```

Админ подтверждает адрес

@dp.callback_query_handler(text_contains="approve", state="*)

async def address_approved(call: CallbackQuery, state:

```
FSMContext):
    data = call.data.split(':')
    user_id = data[1]
    address_id = data[-1]

    address = call.message.text.split(' ')[0]
    await db_commands.approve_address(address_id)
    await address_checker.delete_messages(user_id)
    await bot.send_message(user_id, f"Адрес подтвержден!\n"
                                f"{address}"),
```

reply_markup=to_pay)

```

# Админ не подтверждает адрес
@dp.callback_query_handler(text_contains="not_confirm", state="*")
async def address_not_approved(call: CallbackQuery):
    user_id = call.data.split(':')[1]
    address = call.message.text.split(' ')[0]
    await address_checker.delete_messages(user_id)
    await bot.send_message(user_id, f"По этому адресу доставка не
    осуществляется, попробуйте ввести другой адрес\n"
                                f"адрес - {address}",
    reply_markup=to_choice_address)

@dp.callback_query_handler(text="to_pay", state="*")
async def to_pay_handler(call: CallbackQuery):
    detail = await get_cart_detail(call, step=3)
    await call.message.edit_text(
        detail,
        reply_markup=pay_or_back
    )

```

Листинг Б.11 – Файл start.py

```

from aiogram import types
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters.builtin import CommandStart

from keyboards.default.main_menu_keyboards import
main_menu_keyboard
from loader import dp, bot
from utils.db_api import db_commands

about_us = '''
Итак, экспресс-доставка продуктов к вашим услугам!

Заказывайте через каталог👉
Удобных и быстрых Вам покупок!
'''

@dp.message_handler(CommandStart(), state="*")
async def bot_start(message: types.Message, state: FSMContext):
    await state.reset_state()
    client = await db_commands.select_client(
        message.from_user.id
    )

```

```

if not client:
    await db_commands.add_client(
        message.from_user.id,
        message.from_user.full_name,
        message.from_user.username
    )
await message.answer(
    about_us,
    reply_markup=main_menu_keyboard,
)

```

Листинг Б.12 – Файл support.py

```

from aiogram.dispatcher import FSMContext
from aiogram.types import CallbackQuery, Message
from keyboards.inline.support_keyboards import back,
connect_to_support_keyboard
from loader import dp, message_sendler
from states.states import Support

@dp.callback_query_handler(text="to_sup_root", state="*")
async def support(call: CallbackQuery, state: FSMContext):
    await state.reset_state()
    await call.message.edit_text("Поддержка",
reply_markup=connect_to_support_keyboard)

@dp.callback_query_handler(text="send_to_managers", state="*")
async def send_message_to_managers(call: CallbackQuery):
    await Support.wait_question.set()
    await call.message.edit_text(
        "Введите свой вопрос, он будет отправлен менеджеру",
        reply_markup=back
    )

@dp.message_handler(content_types=["text"],
state=Support.wait_question)
async def send_message_to_manager(message: Message, state:
FSMContext):
    await state.reset_state()
    await message_sendler.send_message_to_managers(
        message.from_user.id,
        message.from_user.username,
        message.text,
    )

```

```

        await message.answer(
            "Ваш вопрос отправлен администратору. В скором времени вы
получите ответ."
        )

```

```

@dp.callback_query_handler(text_contains="answer", state="*")
async def send_message_to_manager(call: CallbackQuery, state:
FSMContext):
    user_id = call.data.split(':')[ -1]
    await Support.wait_answer.set()
    await state.update_data(wait_answer=user_id)
    await call.message.answer('Введите ответ на вопрос')
    await message_sendler.delete_messages(int(user_id))

```

```

@dp.message_handler(content_types=["text"],
state=Support.wait_answer)
async def send_message_to_clien(message: Message, state:
FSMContext):
    data = await state.get_data()
    user_id = data.get("wait_answer")
    await state.reset_state()
    await message_sendler.send_message_to_client(
        user_id,
        message.text
    )
    await message.answer(
        "Ответ отправлен"
    )

```

```

@dp.callback_query_handler(text="contacts", state="*")
async def show_contacts(call: CallbackQuery):
    await call.message.edit_text(
        "Контакты для связи\n"
        "+7 (903) 433-46-28 Администратор 1\n"
        "+7 (968) 767-19-29 Администратор 2",
        reply_markup=back
    )

```

Листинг Б.13 – Файл main_menu_keyboard.py

```

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

menu_button = KeyboardButton('Каталог 📁')

```

```

cart_button = KeyboardButton('Корзина🛒')
support_button = KeyboardButton('Help❓')

main_menu_keyboard = ReplyKeyboardMarkup(
    keyboard=[
        [
            menu_button,
            cart_button,
            support_button
        ]
    ],
    resize_keyboard=True,
    input_field_placeholder="Выберите ~"
)

```

Листинг Б.14 – Файл buy_product_keyboard.py

```

from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

async def generate_pre_cart_menu(product_id):
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="✚ Добавить ещё",
callback_data=f"change_count:{product_id}:1"),
                InlineKeyboardButton(text="✖ Убрать из корзины",
callback_data=f"change_count:{product_id}:0")
            ],
            [
                InlineKeyboardButton(text="🛒 Перейти в корзину",
callback_data=f"to_cart:{product_id}")
            ],
            [
                InlineKeyboardButton(text="🏠 Выбрать ещё",
callback_data=f"back_to_catalog")
            ]
        ]
    )

```

Листинг Б.15 – Файл cart_keyboards.py

```

from aiogram.dispatcher import FSMContext
from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

async def generate_cart_menu(product_id):
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="🚚 Заказать",
callback_data=f"delivery")
            ],
            [
                InlineKeyboardButton(text="📝 Указать
комментарий к заказу", callback_data=f"add_order_comment")
            ],
            [
                InlineKeyboardButton(text="✎ Редактировать
заказ", callback_data=f"edit_order")
            ],
            [
                InlineKeyboardButton(text="✖ Отменить заказ",
callback_data=f"remove_order")
            ]
        ]
    )

save_comment_button = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Готово",
callback_data=f"save_comment")
        ]
    ]
)

```

Листинг Б.16 – Файл cart_keyboards.py

```

from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

async def generate_choice_keyb(address_id):
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Сохранить адрес для

```

```

    дальнейших доставок", callback_data=f"save_a:1:{address_id}")
        ],
        [
            InlineKeyboardButton(text="Пропустить",
callback_data=f"save_a:0")
        ]
    ]
)

addresses_manage = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Ввести новый адрес",
callback_data="new_address")
        ],
        [
            InlineKeyboardButton(text="Выбрать из моих адресов",
callback_data="from_my_addresses")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="chosen_delivery:null")
        ]
    ]
)

pre_delivery_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Ввести адрес доставки",
callback_data="choice_address")
        ],
        [
            InlineKeyboardButton(text="Изменить номер телефона",
callback_data="edit_ph_number")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="delivery")
        ]
    ]
)

edit_phone = InlineKeyboardMarkup(
    inline_keyboard=[
        [

```

```

        InlineKeyboardButton(text="Изменить",
callback_data="edit_phone")
    ],
    [
        InlineKeyboardButton(text="Назад",
callback_data="chosen_delivery:null")
    ]
]
)

pay_or_back = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Заказать",
callback_data=f"compose")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="addr_enter")
        ]
    ]
)

compose_order_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Верно, поехали",
callback_data=f"pay")
        ],
        [
            InlineKeyboardButton(text="Изменить ,
callback_data="enter_address")
        ]
    ]
)

#

```

Листинг Б.17 – Файл delivery_keyboards.py

```

from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

async def generate_choice_keyb(address_id):

```

```

return InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Сохранить адрес для
дальнейших доставок", callback_data=f"save_a:1:{address_id}")
        ],
        [
            InlineKeyboardButton(text="Пропустить",
callback_data=f"save_a:0")
        ]
    ]
)

addresses_manage = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Ввести новый адрес",
callback_data="new_address")
        ],
        [
            InlineKeyboardButton(text="Выбрать из моих адресов",
callback_data="from_my_addresses")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="chosen_delivery:null")
        ]
    ]
)

pre_delivery_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Ввести адрес доставки",
callback_data="choice_address")
        ],
        [
            InlineKeyboardButton(text="Изменить номер телефона",
callback_data="edit_ph_number")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="delivery")
        ]
    ]
)

```

```

edit_phone = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Изменить",
callback_data="edit_phone")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="chosen_delivery:null")
        ]
    ]
)

pay_or_back = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Заказать",
callback_data=f"compose")
        ],
        [
            InlineKeyboardButton(text="Назад",
callback_data="addr_enter")
        ]
    ]
)

compose_order_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Верно, поехали",
callback_data=f"pay")
        ],
        [
            InlineKeyboardButton(text="Изменить ,
callback_data="enter_address")
        ]
    ]
)

```

#

Листинг Б.18 – Файл pay_keyboards.py

```

from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

async def generate_to_chose_p_method_keyboard(order_id):
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Передал",
callback_data=f"transferred:{order_id}")
            ],
            [
                InlineKeyboardButton(text="Назад",
callback_data=f"put_order:{order_id}")
            ]
        ]
    )

async def generate_paid_keyboard(order_id):
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="Оплатил",
callback_data=f"paid:{order_id}")
            ],
            [
                InlineKeyboardButton(text="Назад",
callback_data=f"put_order:{order_id}")
            ]
        ]
    )

back_to_cart = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="В корзину",
callback_data="back_to_cart")
        ]
    ]
)

to_pay = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Вернуться к оплате",
callback_data="to_pay")
        ]
    ]
)

```

```

    ]
)

to_choice_address = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Вернуться к выбору адреса",
callback_data="enter_address")
        ]
    ]
)

```

Листинг Б.19 – Файл support_keyboards.py

```

from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

support_main_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="FAQ", callback_data="faq")
        ],
        [
            InlineKeyboardButton(text="Связаться с поддержкой",
callback_data="connect_to_support")
        ]
    ]
)

connect_to_support_keyboard = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Написать менеджеру",
callback_data="send_to_managers")
        ],
        [
            InlineKeyboardButton(text="Позвонить",
callback_data="contacts")
        ]
    ]
)

back = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Назад",

```

```

callback_data="to_sup_root")
    ]
]
)

```

Листинг Б.20 – Файл states.py

```

from aiogram.dispatcher.filters.state import StatesGroup, State

class Registration(StatesGroup):
    wait_phone = State()

    wait_address = State()
    check_address = State()
    save_address = State()

    wait_email = State()

    wait_convention = State()

class Buy_product(StatesGroup):
    wait_category = State()
    wait_product = State()
    wait_pre_cart = State()
    wait_cart = State()

class Cart(StatesGroup):
    menu = State()

    add_comment = State()
    save_comment = State()

    choice_delivery = State()

    add_new_address = State()
    edit_phone = State()

    pay = State()

class Lk(StatesGroup):
    edit_phone = State()
    edit_name = State()
    edit_addresses = State()
    add_new_address = State()

```

```
class Support(StatesGroup):
    wait_question = State()
    wait_answer = State()

class DeliveryTerms(StatesGroup):
    delivery_terms_menu = State()
```

```
class Couriers(StatesGroup):
    init_comand = State()
```

Листинг Б.21 – Файл db_comands.py

```
logging

from asgiref.sync import sync_to_async

from django_project.telegrambot.delivery_bot_manage.models import
Client, Address, Product, Category, Order, \
    OrderedProduct, Courier, PaymentSystem

# Clients
@sync_to_async
def add_client(telegram_id, full_name, username, phone=None,
email=None):
    try:
        Client(
            telegram_id=int(telegram_id),
            fullname=full_name,
            username=username,
            phone=phone,
            email=email
        ).save()
        return
    except Exception:
        return
    Client.objects.filter(telegram_id=int(telegram_id)).first()

@sync_to_async
def select_client(telegram_id):
    client =
    Client.objects.filter(telegram_id=int(telegram_id)).first()
    return client
```

```

@sync_to_async
def change_client_phone(telegram_id, phone):
    client =
Client.objects.filter(telegram_id=int(telegram_id)).first()
    client.phone = phone
    client.save()
    return client

@sync_to_async
def change_client_name(telegram_id, name):
    client =
Client.objects.filter(telegram_id=int(telegram_id)).first()
    client.fullname = name
    client.save()
    return client

# Addresses
@sync_to_async
def add_address(client_id, address, checked=False):
    try:
        return Address(
            client_id=client_id,
            address=address,
            checked=checked
        ).save()
    except Exception:
        return Address.objects.filter(client_id=client_id).first()

@sync_to_async
def set_client_to_address(client_id, address_id):
    try:
        address = Address.objects.filter(
            id=int(address_id)
        ).first()
        address.client_id = client_id
        return address.save()
    except Exception:
        return Address.objects.filter(client_id=client_id).first()

@sync_to_async
def add_disposable_address(order, address):

```

```

if order.address:
    order.address.delete()

```

```

address = Address(
    address=address
)
order.address = address
address.save()
return order.save()

```

```

@sync_to_async
def delete_address(address_id):
    try:
        return Address(
            id=address_id
        ).delete()
    except Exception:
        logging.info(f"address {address_id} delete error")

```

```

@sync_to_async
def delete_order_address(telegram_id):
    order = Order.objects.filter(
        client_id__telegram_id=int(telegram_id),
        completed=False,
        compose=False
    ).first()
    if not order.address:
        order.address.delete()
    return order.save()

```

```

@sync_to_async
def select_client_addresses(client_id):
    try:
        return Address.objects.filter(
            client_id__telegram_id=int(client_id),
        ).all()
    except Exception:
        return False

```

```

@sync_to_async
def approve_address(address_id):
    try:
        address = Address.objects.filter(
            id = int(address_id)
        ).first()
        address.checked = True
        return address.save()
    except Exception:
        logging.info("approve_address error")
        return False

@sync_to_async
def select_address(address_id):
    address = Address.objects.filter(id=int(address_id)).first()
    return address

# Categorys
@sync_to_async
def select_all_categorys():
    return Category.objects.all()

# Products
@sync_to_async
def select_products_by_category(category, part_of_name):
    if category == '':
        products = Product.objects.all().filter(
            name__icontains=part_of_name.lstrip(),
            in_stock__gt = 0
        )
        return list(products)

    products = Product.objects.all().filter(
        category=category,
        name__icontains=part_of_name.lstrip(),
        in_stock__gt=0
    ).order_by("position_category")
    return list(products)

@sync_to_async
def select_products_by_name(name):
    logging.info(f"NAME {name}")

```

```

    try:
        return Product.objects.filter(
            name=name
        ).first()
    except:
        logging.info(f"{name} select products by name error")
        return False

@sync_to_async
def select_products_by_id(product_id):
    logging.info(f"NAME {product_id}")
    try:
        return Product.objects.filter(
            id=product_id
        ).first()
    except:
        logging.info(f"{product_id} select products by id error")
        return False

# Orders
@sync_to_async
def select_no_compose_order(telegram_id):
    try:
        return Order.objects.filter(
            client_id__telegram_id=telegram_id,
            completed=False,
            compose=False
        ).first()
    except:
        logging.info(f"{telegram_id} select no compose order
error")
        return False

@sync_to_async
def get_user_id_by_order(order_id):
    try:
        order = Order.objects.filter(
            id=int(order_id)
        ).first()
        return int(order.client_id.telegram_id)
    except:
        logging.info(f"{order_id} get_user_id_by_order")
        return False

```

```

@sync_to_async
def get_courier_by_order(order_id):
    try:
        order = Order.objects.filter(
            id=int(order_id)
        ).first()
        return int(order.courier_id)
    except:
        logging.info(f"{order_id} get_user_id_by_order")
        return False

```

```

@sync_to_async
def select_compose_orders():
    try:
        return Order.objects.filter(
            completed=False,
            compose=True,
            courier_id=None
        ).all()
    except:
        logging.info(f"select compose orders error")
        return False

```

```

@sync_to_async
def select_order(order_id):
    try:
        return Order.objects.filter(
            id=int(order_id)
        ).first()
    except:
        logging.info(f"select order error")
        return False

```

```

@sync_to_async
def complete_order(order_id):
    try:
        order = Order.objects.filter(
            id=int(order_id)
        ).first()
        order.completed = True
        return order.save()

```

```

except:
    logging.info(f"select order error")
    return False

@sync_to_async
def set_courier(order_id, telegram_id):
    try:
        courier = Courier.objects.filter(
            telegram_id=int(telegram_id)
        ).first()
        order = Order.objects.filter(
            id=order_id
        ).first()
        order.courier_id = courier
        return order.save()
    except:
        logging.info(f"set_courier error")
        return False

@sync_to_async
def create_new_order(client):
    Order(
        client_id=client,
    ).save()
    order = Order.objects.filter(
        client_id=client,
        completed=False,
        compose=False
    ).first()
    return order

@sync_to_async
def compose_order(telegram_id):
    order = Order.objects.filter(
        client_id__telegram_id=int(telegram_id),
        completed=False,
        compose=False
    ).first()
    order.compose = True
    order.save()

```

```

@sync_to_async
def remove_product_from_order(order, product):
    try:
        all_product = OrderedProduct.objects.filter(
            order_id=order
        ).all()
        if len(all_product) == 1:
            return Order.objects.filter(
                id=order.id,
            ).delete()

        return OrderedProduct.objects.filter(
            order_id=order,
            product_id=product
        ).delete()
    except Exception:
        logging.info(f"{product} delete from order error")

@sync_to_async
def change_count(order, product, arg):
    if arg == 'none':
        return OrderedProduct.objects.filter(
            order_id=order,
            product_id=product
        ).first(), None

    try:
        ordered_product = OrderedProduct.objects.filter(
            order_id=order,
            product_id=product
        ).first()
        if not ordered_product:
            OrderedProduct(
                order_id=order,
                product_id=product
            ).save()
            ordered_product = OrderedProduct.objects.filter(
                order_id=order,
                product_id=product
            ).first()
        if arg == "1":
            if ordered_product.product_count >=
ordered_product.product_id.in_stock:
                order.save()
                return ordered_product, "Этого продукта больше нет

```

в наличии"

```

        ordered_product.product_count += 1
        order.amount += product.price
    else:
        if ordered_product.product_count == 1:
            order.amount -= product.price
            order.save()
            return False, None
        ordered_product.product_count -= 1
        order.amount -= product.price
    ordered_product.save()
    order.save()
    return ordered_product, None
except Exception:
    logging.info(f"change product_count {product} to_order
{product} error")

@sync_to_async
def select_ordered_products(order_id):
    try:
        return OrderedProduct.objects.filter(
            order_id=order_id,
        ).all()
    except Exception:
        logging.info(f"{order_id} select_ordered_products error")

@sync_to_async
def delete_order(client):

    order = Order.objects.filter(
        client_id=client,
        compose=False,
        completed=False
    ).first()
    if order.address:
        if not order.address.client_id:
            order.address.delete()
    return order.delete()

@sync_to_async
def add_order_comment(client, comment):
    order = Order.objects.filter(

```

```

        client_id=client,
        compose=False,
        completed=False
    ).first()
    order.comment = comment
    return order.save()

@sync_to_async
def add_order_address(telegram_id, address):
    order = Order.objects.filter(
        client_id__telegram_id=int(telegram_id),
        compose=False,
        completed=False
    ).first()
    order.address = address
    return order.save()

@sync_to_async
def add_order_delivery(telegram_id, delivery):
    order = Order.objects.filter(
        client_id__telegram_id=int(telegram_id),
        compose=False,
        completed=False
    ).first()
    order.delivery_id = delivery
    return order.save()

#Couriers
@sync_to_async
def init_courier(data):
    Courier(
        telegram_id=int(data.id),
        fullname=data.full_name,
        username=data.username
    ).save()

@sync_to_async
def select_courier(telegram_id):
    try:
        return Courier.objects.filter(
            telegram_id=telegram_id
        ).first()

```

```

except:
    logging.info(f"{telegram_id} select_courier error")
    return False

#PaymentSystems
@sync_to_async
def select_paysys(name):
    try:
        return PaymentSystem.objects.filter(
            name=name
        ).first()
    except:
        logging.info(f"{name} select pay_sys error")
        return False

```

Листинг Б.22 – Файл address_checker.py

```

import logging

from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton

from data.config import ADMINS

class AddressChecker():
    def __init__(self, bot):
        self.bot = bot
        self.admins = ADMINS
        self.user_addresses = {}
        self.messages = {}

    async def save_address(self, user_id, address):
        self.user_addresses[user_id] = address

    async def get_address(self, user_id):
        address = self.user_addresses[user_id]
        self.user_addresses.pop(user_id)
        return address

    async def delete_messages(self, user_id):
        for message in self.messages[user_id]:
            try:
                await message.delete()
            except:
                pass

```

```

        self.messages.pop(user_id)

    async def send_address_to_check(self, user_id, address,
place=1):
        if place == 1:
            clear_address = address.replace(' ', '')
            approve_address = InlineKeyboardMarkup(
                inline_keyboard=[
                    [
                        InlineKeyboardButton(text="Да",
callback_data=f"yes:{user_id}"),
                        InlineKeyboardButton(text="Нет",
callback_data=f"no:{user_id}")
                    ]
                ]
            )
        else:
            clear_address = address.address.replace(' ', '')
            approve_address = InlineKeyboardMarkup(
                inline_keyboard=[
                    [
                        InlineKeyboardButton(text="Да",
callback_data=f"approve:{user_id}:{address.id}"),
                        InlineKeyboardButton(text="Нет",
callback_data=f"not_confirm:{user_id}:{address.id}")
                    ]
                ]
            )
        self.messages[user_id] = []
        for user in self.admins:
            msg = await self.bot.send_message(
                user, f"{clear_address} \n\n"
                    f"Обслуживаем этот адрес?",
                    reply_markup=approve_address
            )
            self.messages[user_id].append(msg)

```

Листинг Б.23 – Файл address_checker.py

```

import logging

from aiogram.types import InlineKeyboardButton,
InlineKeyboardMarkup

from data.config import ADMINS

```

```

class MessageSendler():
    def __init__(self, bot):
        self.bot = bot
        self.admins = ADMINS
        self.messages = {}

    async def delete_messages(self, user_id):
        for message in self.messages[user_id]:
            try:
                await message.delete()
            except:
                pass
        self.messages.pop(user_id)

    async def send_message_to_managers(self, user_id, username,
message):
        answer = InlineKeyboardMarkup(
            inline_keyboard=[
                [
                    InlineKeyboardButton(text="Ответить",
callback_data=f"answer:{user_id}")
                ]
            ]
        )
        self.messages[user_id] = []
        for user in self.admins:
            logging.info(user)
            logging.info(message)
            msg = await self.bot.send_message(
                user,
                f"Вопрос от клиента @{username}\n\n"
                f"{message}",
                reply_markup=answer
            )
            self.messages[user_id].append(msg)

    async def send_message_to_client(self, user_id, message):
        await self.bot.send_message(
            user_id,
            "Ответ от менеджера\n\n" +
            message
        )

```

Листинг Б.24 – Файл app.py

```
import os

import django
from aiogram import executor

import middlewares, filters, handlers
from loader import bot
from utils.notify_admins import on_startup_notify
from utils.set_bot_commands import set_default_commands


async def on_startup(dispatcher):
    # Устанавливаем дефолтные команды
    await set_default_commands(dispatcher)
    # Уведомляет про запуск
    await on_startup_notify(dispatcher)


if __name__ == '__main__':
    from handlers import dp
    executor.start_polling(dp, on_startup=on_startup)
```

Приложение В
ИСХОДНЫЙ КОД ЮНИТ-ТЕСТОВ
(рекомендуемое)

Листинг В.1 – файл tests.py

```

import asyncio
import logging
from functools import wraps
from unittest import TestCase
import pickle
from utils.db_api import db_commands

def async_test(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        return asyncio.run(f(*args, **kwargs))

    return wrapper

class TestDbCommands(TestCase):
    def setUp(self):
        asyncio.run(self.add_client())
        asyncio.run(self.select_client())
        asyncio.run(self.change_client_phone())
        asyncio.run(self.change_client_name())
        asyncio.run(self.create_new_order())
        asyncio.run(self.add_disposable_address())
        asyncio.run(self.select_address())
        asyncio.run(self.approve_address())
        asyncio.run(self.delete_address())
        asyncio.run(self.select_all_categories())
        asyncio.run(self.select_products_by_category())
        asyncio.run(self.select_products_by_name())
        asyncio.run(self.select_products_by_id())
        asyncio.run(self.select_no_compose_order())
        asyncio.run(self.get_user_id_by_order())
        asyncio.run(self.select_order())
        asyncio.run(self.add_product_unit_to_order())
        asyncio.run(self.remove_product_unit_from_order())
        asyncio.run(self.add_order_comment())
        asyncio.run(self.compose_order())
        asyncio.run(self.select_compose_orders())
        asyncio.run(self.set_courier())
        asyncio.run(self.select_courier())
        asyncio.run(self.select_paysys())

```

```

        asyncio.run(self.complete_order())
        asyncio.run(self.remove_product_from_order())

    async def add_client(self):
        logging.info("Добавление пользователя")
        await db_commands.add_client(
            "123456789",
            "test_user",
            "test_username",
            "79278002784",
            "gipard446@gmail.com"
        )
        self.assertIsNotNone(await
db_commands.select_client("123456789"))

    async def select_client(self):
        logging.info("Получение пользователя")
        client = await db_commands.select_client("123456789")
        self.assertIsNotNone(client)

    async def change_client_phone(self):
        logging.info("Изменение номера телефона пользователя")
        await db_commands.change_client_phone("123456789",
"+79278000000")
        client = await db_commands.select_client("123456789")
        self.assertEqual(client.phone, "+79278000000")

    async def change_client_name(self):
        logging.info("Изменение имени пользователя")
        await db_commands.change_client_name("123456789", "new
name")
        client = await db_commands.select_client("123456789")
        self.assertEqual(client.fullname, "new name")

    async def create_new_order(self):
        logging.info("Создание заказа")
        client = await db_commands.select_client("123456789")
        new_order = await db_commands.create_new_order(client)
        with open('data.pickle', 'wb') as f:
            pickle.dump(new_order.id, f)
        self.assertIsNotNone(new_order)

    async def add_disposable_address(self):
        logging.info("Указание адреса к заказу")
        with open('data.pickle', 'rb') as f:
            order_id = pickle.load(f)

```

```

order = await db_commands.select_order(order_id)
await db_commands.add_disposable_address(
    order,
    "test address 2"
)
updated_order = await db_commands.select_order(order_id)
with open('data.pickle', 'wb') as f:
    pickle.dump([order_id, updated_order.address.id], f)
self.assertEqual(updated_order.address.address, "test
address 2")

async def select_address(self):
    logging.info("Получение адреса")
    with open('data.pickle', 'rb') as f:
        data = pickle.load(f)
    address = await db_commands.select_address(data[1])
    self.assertIsNotNone(address)

async def approve_address(self):
    logging.info("Подтверждение адреса")
    with open('data.pickle', 'rb') as f:
        data = pickle.load(f)
    await db_commands.approve_address(data[1])
    address = await db_commands.select_address(data[1])
    self.assertTrue(address.checked)

async def delete_address(self):
    logging.info("Удаление адреса")
    with open('data.pickle', 'rb') as f:
        data = pickle.load(f)
    await db_commands.delete_address(data[1])
    self.assertIsNone(await
db_commands.select_address(data[1]))

async def select_all_categories(self):
    logging.info("Получение всех категорий продуктов")
    categories = await db_commands.select_all_categorys()
    self.assertIsNotNone(categories)

async def select_products_by_category(self):
    logging.info("Получение продукта по категории")
    products = await
db_commands.select_products_by_category("Молочная продукция",
"Сыр")
    self.assertIsNotNone(products)

```

```

    async def select_products_by_name(self):
        product = await db_commands.select_products_by_name('Сыр
"Российский"')
        self.assertIsNotNone(product)

    async def select_products_by_id(self):
        product = await db_commands.select_products_by_id(1)
        self.assertIsNotNone(product)

    async def select_no_compose_order(self):
        order = await
db_commands.select_no_compose_order(123456789)
        self.assertIsNotNone(order)

    async def get_user_id_by_order(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
        user_id = await db_commands.get_user_id_by_order(data[0])
        self.assertEqual(user_id, 123456789)

    async def select_order(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
        order = await db_commands.select_order(data[0])
        self.assertIsNotNone(order)
    async def add_order_comment(self):
        client = await db_commands.select_client("123456789")
        await db_commands.add_order_comment(client, "test
comment")
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
        order = await db_commands.select_order(data[0])
        self.assertEqual(order.comment, "test comment")

    async def add_product_unit_to_order(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
        order = await db_commands.select_order(data[0])
        product = await db_commands.select_products_by_id(1)
        await db_commands.change_count(order, product, "1")
        await db_commands.change_count(order, product, "1")
        await db_commands.change_count(order, product, "1")

```

```

        ordered_product = await
db_commands.select_ordered_products(order.id)
        self.assertEqual(ordered_product[0].product_count, 3)

    async def remove_product_unit_from_order(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
            order = await db_commands.select_order(data[0])
            product = await db_commands.select_products_by_id(1)
            await db_commands.change_count(order, product, "0")
            ordered_product = await
db_commands.select_ordered_products(order.id)
            self.assertEqual(ordered_product[0].product_count, 2)

    async def compose_order(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
            await db_commands.compose_order("123456789")
            order = await db_commands.select_order(data[0])
            self.assertTrue(order.compose)

    async def select_compose_orders(self):
        compose_orders = await db_commands.select_compose_orders()
        self.assertIsNotNone(compose_orders)

    async def set_courier(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
            await db_commands.set_courier(data[0], 894508156)
            order = await db_commands.select_order(data[0])
            self.assertIsNotNone(order.courier_id)

    async def select_courier(self):
        courier = await db_commands.select_courier(894508156)
        self.assertIsNotNone(courier)

    async def select_paysys(self):
        paysys = await db_commands.select_paysys("QIWI")
        self.assertIsNotNone(paysys)

    async def complete_order(self):

```

```

        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
        await db_commands.complete_order(data[0])
        order = await db_commands.select_order(data[0])
        self.assertTrue(order.completed)

    async def remove_product_from_order(self):
        with open('data.pickle', 'rb') as f:
            data = pickle.load(f)
        order = await db_commands.select_order(data[0])
        product = await db_commands.select_products_by_id(1)
        await db_commands.remove_product_from_order(order,
product)
        ordered_product = await
db_commands.select_ordered_products(order.id)
        self.assertEqual(list(ordered_product), [])

    def test_start(self):
        logging.info("-----")

```