КОМИТЕТ ПО ОБРАЗОВАНИЮ МИНГОРИСПОЛКОМА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МИНСКИЙ ГОСУДАРСТВЕННЫЙ КОЛЛЕДЖ ЦИФРОВЫХ ТЕХНОЛОГИЙ»

Специальность: 2-39 03 02

«Программируемые мобильные системы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Разработка мобильного приложения для работы поставщиков и магазинов

Автор проекта (Л.И.Рыжук )

Руководитель проекта (Д.С.Бровка )

Минск, 2025

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc213916228)

[**1 АНАЛИТИЧЕСКИЙ ОБЗОР И РАЗРАБОТКА ПРОЕКТА ПРИЛОЖЕНИЯ** 4](#_Toc213916229)

[**1.1 Функциональный анализ предметной области** 4](#_Toc213916230)

[**1.2 Анализ существующих решений и постановка задач** 6](#_Toc213916231)

[**2.** **РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ** 10](#_Toc213916232)

[**2.1.** **Описание инструментальных средств** 10](#_Toc213916233)

[**2.2 Организация данных** 13](#_Toc213916234)

[**2.3 Обоснование и разработка интерфейса** 16](#_Toc213916235)

[**2.4 Отладка и тестирование разрабатываемого приложения** 17](#_Toc213916236)

[**2.5 Системные требования** 19](#_Toc213916237)

[**3. ОПИСАНИЕ ПРИМЕНЕНИЯ ПРИЛОЖЕНИЯ** 21](#_Toc213916238)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 28](#_Toc213916239)

[**ПРИЛОЖЕНИЕ А Листинг кода** 29](#_Toc213916240)

[**ПРИЛОЖЕНИЕ Б Карта пользователя** 31](#_Toc213916241)

# **ВВЕДЕНИЕ**

В современных условиях розничной торговли эффективное взаимодействие между магазинами и поставщиками является критически важным для успешного функционирования всей товаропроводящей цепи. С развитием мобильных технологий все более актуальными становятся специализированные приложения, позволяющие автоматизировать ключевые бизнес-процессы в режиме реального времени как для розничных точек, так и для их поставщиков.

**Предметная область приложения** охватывает комплексную систему управления взаимоотношениями между магазинами и поставщиками. Разрабатываемая платформа включает два взаимосвязанных модуля: для магазинов розничной торговли и для поставщиков товаров.

Для **магазинов**система предоставляет функционал управления складским учетом с группировкой товаров, поиском и фильтрацией; оформления заказов у поставщиков с отслеживанием статусов; управления профилем магазина и генерации сопроводительной документации. Особенностью является возможность как заказа товаров у поставщиков, так и самостоятельного создания товаров на складе.

Для **поставщиков** система включает управление товарными партиями с возможностью массового добавления; обработку входящих заказов от магазинов; контроль остатков товаров; управление профилем поставщика и просмотр отзывов от магазинов.

**Актуальность разработки** обусловлена необходимостью создания единой платформы, которая объединяет розничные магазины и поставщиков, обеспечивая прозрачность цепочки поставок, сокращение времени обработки заказов и минимизацию ошибок при учете товародвижения.

**Целью курсового проекта** является разработка мобильного приложения «ПОМОЩНИК ПОСТАВЩИКАМ И МАГАЗИНАМ»., которое упростит магазинап поиск поставщиков и ведение склада, а поставщикам оборот товаров и контроль клиентов.

# **1 АНАЛИТИЧЕСКИЙ ОБЗОР И РАЗРАБОТКА ПРОЕКТА ПРИЛОЖЕНИЯ**

## **1.1 Функциональный анализ предметной области**

Предметная область проекта охватывает процессы взаимодействия между магазинами и поставщиками в контексте закупки, хранения и учета продукции. В условиях современной экономики торговые предприятия стремятся к оптимизации логистических процессов, снижению издержек и повышению оперативности принятия решений. Одним из ключевых факторов успешной работы магазина является наличие надежных каналов поставок, позволяющих своевременно пополнять ассортимент, контролировать остатки и реагировать на изменения спроса.

Взаимодействие между магазинами и поставщиками традиционно осуществляется через телефонные звонки, электронную почту, мессенджеры или специализированные платформы. Однако такие методы не всегда обеспечивают необходимую прозрачность, скорость и удобство. Особенно это актуально для малого и среднего бизнеса, где отсутствует возможность внедрения сложных ERP-систем и найма специалистов по логистике. В результате возникает потребность в доступных и простых инструментах, позволяющих автоматизировать ключевые процессы взаимодействия.

Предметная область включает в себя следующие аспекты:

- Поиск поставщиков и продукции. Магазины нуждаются в возможности оперативного поиска подходящих предложений, сравнения условий поставки, цен, сроков и характеристик продукции.

- Оформление заказов. После выбора товара необходимо оформить заказ, получитб товар и положить его на склад.

- Управление складом. Магазины должны вести учет поступившей продукции, отслеживать остатки, устанавливать цены, обновлять информацию о товарах.

- Документооборот. В процессе взаимодействия формируются накладные, которые необходимо хранить и обрабатывать.

- Представление продукции. Поставщики должны иметь возможность размещать информацию о своих товарах, включая описание, характеристики, фотографии, цены.

- Прием заявок. Поставщики должны получать уведомления о новых заказах, подтверждать их, планировать доставку и отслеживать выполнение.

Таким образом, предметная область характеризуется высокой степенью повторяемости операций, необходимостью точного учета и важностью своевременного обмена информацией. В ней присутствует множество типовых процессов, которые могут быть автоматизированы с помощью мобильного приложения. Это позволяет сократить время на выполнение рутинных задач, снизить вероятность ошибок и повысить прозрачность взаимодействия.

Нельзя не отметить, что при проведении всестороннего и детализированного анализа конкретной предметной области, существующей в условиях современного цифрового мира, крайне важным и заслуживающим самого пристального рассмотрения аспектом является повсеместное проникновение и фундаментальная роль мобильных технологий. В реалиях сегодняшнего дня, которые характеризуются исключительно высокой степенью конкурентной борьбы на рынке, а также острой необходимостью максимально оперативного и гибкого реагирования на любые, даже самые незначительные, изменения рыночной конъюнктуры, мобильные приложения трансформируются из простого удобного инструмента в стратегически важный, а зачастую и незаменимый, элемент системы управления бизнес-процессами. Данные программные решения предоставляют конечным пользователям, будь то сотрудники компаний или их руководители, уникальную возможность осуществлять свою профессиональную деятельность дистанционно, не будучи привязанными к стационарному рабочему месту в офисе, получать важные оповещения и уведомления в режиме реального времени, что, в свою очередь, создает основу для ускоренного принятия взвешенных и обоснованных управленческих решений, а также поддерживать непрерывное и продуктивное взаимодействие с контрагентами и деловыми партнерами.

Параллельно с этим, в рамках того же самого анализа, также в обязательном порядке необходимо принимать во внимание и тщательно учитывать всю специфику и многообразие существующего нормативно-правового поля и его регулирующего воздействия. Следует понимать, что любое деловое взаимодействие, например, между розничными магазинами и их поставщиками, неизбежно сопровождается необходимостью строгого оформления целого пакета официальных документов, имеющих полноценную юридическую значимость, а также неукоснительного соблюдения всех предписанных законодательством требований и стандартов, касающихся процессов хранения и последующей транспортировки различной продукции. Особенная сложность и повышенная ответственность возникают в тех ситуациях, когда речь заходит о товарных группах, имеющих строго ограниченный срок годности, или же требующих обеспечения специальных, подчас уникальных, условий их содержания, например, определенного температурного режима или уровня влажности. Все вышеперечисленные обстоятельства, безусловно, накладывают целый ряд дополнительных и весьма строгих требований на общий функционал разрабатываемого программного обеспечения, который в идеале должен включать в себя такие возможности, как автоматизированная генерация необходимых документов, ведение детализированной и структурированной истории всех совершаемых заказов, а также реализация комплексных мер, направленных на гарантированное обеспечение безопасности и конфиденциальности всех обрабатываемых данных.

Таким образом, подводя общий итог, можно с уверенностью заключить, что рассматриваемая предметная область представляет собой не простой набор разрозненных действий, а сложный, многогранный и комплексный механизм, состоящий из множества взаимосвязанных и взаимозависимых процессов, ключевая цель которых заключается в обеспечении бесперебойного, эффективного и юридически корректного функционирования всей системы торговли. Её тщательный и глубокий анализ является не просто полезной процедурой, а необходимой и фундаментальной операцией, которая позволяет системно вычленить и сформулировать ключевые задачи, подлежащие первоочередному решению в контексте процесса разработки приложения, а также дает возможность четко определить и детализировать весь спектр требований, предъявляемых к таким критически важным компонентам, как его общая архитектура, пользовательский интерфейс и внутренняя бизнес-логика.

## **1.2 Анализ существующих решений и постановка задач**

На современном рынке информационных технологий представлено огромное количество решений, направленных на автоматизацию процессов в сфере торговли и логистики. Это и мощные ERP-системы, и гибкие CRM-платформы, и разнообразные B2B-маркетплейсы, а также специализированные приложения, предназначенные для учета, планирования и управления поставками. Все эти инструменты призваны упростить и оптимизировать бизнес-процессы, повысить эффективность работы компаний и обеспечить прозрачность на всех этапах цепочки поставок.

Однако стоит отметить, что подавляющее большинство подобных решений ориентировано преимущественно на крупные предприятия. Их внедрение требует серьезных финансовых вложений, наличия квалифицированного персонала, готового к обучению и адаптации, а также соответствующей технической инфраструктуры. Это делает такие системы малодоступными для малого и среднего бизнеса, который зачастую не располагает необходимыми ресурсами для полноценной интеграции подобных платформ.

ERP-системы, такие как SAP, 1С:Предприятие (Рисунок 1.1), Microsoft Dynamics, действительно предлагают широкий спектр функциональных возможностей. Они охватывают управление закупками, складскими запасами, финансовыми потоками, персоналом и другими ключевыми аспектами деятельности компании. Тем не менее, их использование сопряжено с рядом сложностей: от необходимости настройки под конкретные бизнес-процессы до постоянной поддержки и обновлений. В результате, несмотря на очевидные преимущества, для небольших компаний такие решения остаются либо недоступными, либо чрезмерно сложными в реализации. CRM-платформы, например Bitrix24, Zoho CRM, Salesforce, позволяют управлять взаимодействием с клиентами и партнерами, вести историю контактов, планировать задачи и анализировать эффективность. Однако они не всегда адаптированы под специфику торговли, не включают полноценный складской учет и не обеспечивают удобного интерфейса для работы с поставками.

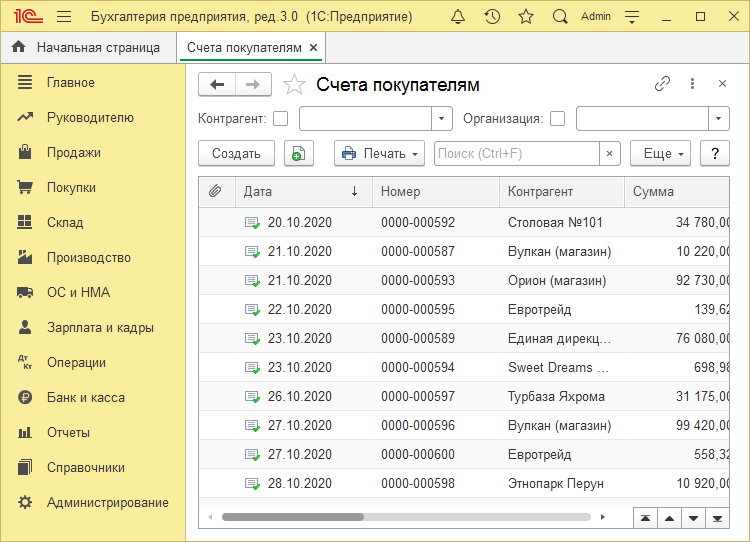


Рисунок 1.1 – 1С:Предприятие

B2B-маркетплейсы, такие как Alibaba, OptList, Allbiz, действительно предоставляют удобные инструменты для поиска поставщиков и размещения заказов. Они открывают доступ к широкому ассортименту продукции, позволяют сравнивать предложения и устанавливать деловые контакты. Однако основной акцент в их работе сделан на международную торговлю, что влечёт за собой ряд ограничений для локального бизнеса. Такие платформы зачастую не адаптированы под региональные особенности, не интегрируются с внутренними системами управления магазинов и не предоставляют возможности вести складской учёт или отслеживать движение товаров в рамках локальной логистики.

Параллельно с этим существуют мобильные приложения, ориентированные на учёт товаров — например, «Мой склад», «Складской учёт», «Товароучет». Они предлагают базовые функции для инвентаризации, фиксации остатков и контроля движения продукции. Тем не менее, их функциональность ограничена: отсутствует полноценное взаимодействие с поставщиками, нет гибкой системы фильтрации и сортировки предложений, не реализован документооборот, а работа с заказами зачастую сводится к простому фиксированию факта поступления или отгрузки. Это делает такие приложения полезными лишь в рамках узких задач, не охватывая весь спектр потребностей малого и среднего бизнеса в сфере торговли и логистики. Таким образом, можно выделить следующие недостатки существующих решений:

- Сложность внедрения и высокая стоимость;

- Отсутствие адаптации под мобильные устройства;

- Недостаточная гибкость и масштабируемость;

- Ограниченный функционал для взаимодействия между магазинами и поставщиками;

- Отсутствие автоматизации документооборота;

- Неудобный интерфейс для пользователей без технической подготовки.

На основании анализа существующих решений формулируются задачи, которые необходимо решить в рамках проекта:

1. Разработать мобильное Android-приложение, обеспечивающее полноценную работу как для магазинов, так и для поставщиков.
2. Реализовать простой и интуитивно понятный интерфейс, доступный для пользователей без специальной подготовки.
3. Обеспечить функционал регистрации и аутентификации с разделением ролей.
4. Реализовать поиск, фильтрацию и сортировку поставок по различным параметрам.
5. Обеспечить возможность оформления заказов и автоматической генерации накладных.
6. Реализовать модуль управления складом: добавление товаров, установка цен, контроль остатков, прикрепление фотографий.
7. Предусмотреть возможность добавления поставщиками новых партий продукции с указанием характеристик и изображений.
8. Обеспечить приём и обработку заявок поставщиками в режиме реального времени.
9. Обеспечить безопасность хранения данных и защиту персональной информации.
10. Предусмотреть возможность масштабирования приложения и интеграции с внешними сервисами.

Также наблюдается отсутствие единой экосистемы: CRM-системы, складской учёт и оформление заказов часто реализованы в виде отдельных решений, не связанных между собой. Это приводит к фрагментации данных, необходимости дублирования информации и повышенному риску ошибок при синхронизации.

Существенным недостатком является ограниченность в визуализации и аналитике. Большинство решений не предоставляют наглядных отчётов, графиков или дашбордов, особенно в мобильных версиях. Это затрудняет принятие управленческих решений, особенно для пользователей без технической подготовки.

Дополнительным ограничением является необходимость финансовых вложений в приобретение и продление лицензий для использования определённого программного обеспечения. Многие коммерческие платформы требуют регулярных платежей за доступ к базовому функционалу, обновлениям, технической поддержке или расширенным возможностям, что может стать существенным барьером для малого и среднего бизнеса, особенно на этапе запуска или масштабирования. Такие затраты не всегда оправданы, особенно если программное обеспечение не предоставляет гибкости в адаптации под специфические бизнес-процессы.

В контексте разработки автономного, адаптивного и масштабируемого решения такие ограничения становятся критичными. Использование открытых технологий и свободного ПО позволяет не только снизить затраты, но и обеспечить прозрачность, гибкость и устойчивость архитектуры. Возможность модификации исходного кода, подключения сторонних библиотек, настройки бизнес-логики и интерфейса под конкретные задачи делает систему более жизнеспособной и пригодной для долгосрочного развития. Именно поэтому выбор инструментов с открытым исходным кодом и свободной лицензией является стратегически важным решением при создании платформы, ориентированной на индивидуальные потребности и устойчивую автоматизацию.

Решение указанных задач упростит работу, позволит создать эффективный инструмент для автоматизации взаимодействия между магазинами и поставщиками, повысить прозрачность процессов, сократить издержки и улучшить качество обслуживания.

# **2. РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

## **2.1. Описание инструментальных средств**

Для разработки мобильного приложения была использована среда разработки Android Studio. Android Studio — это официальная IDE для создания Android-приложений на языках Java и Kotlin. Она предоставляет мощный инструментарий для написания, отладки и тестирования кода, а также встроенные средства работы с интерфейсом пользователя через XML-разметку. Для сборки и запуска проекта используется система Gradle, а для тестирования — фреймворки JUnit и Espresso. Android Studio поддерживает эмуляторы Android Virtual Device (AVD), что позволяет тестировать приложение без физического устройства. Также используется инструмент Instant Run, который ускоряет процесс обновления приложения на эмуляторе или устройстве.

Серверная часть проекта реализована с использованием платформы Node.js — асинхронной, событийно-ориентированной среды выполнения JavaScript, которая обеспечивает высокую производительность и масштабируемость. Node.js идеально подходит для обработки большого количества одновременных запросов, что важно при работе с заказами, регистрацией пользователей и обменом данными между магазинами и поставщиками. В качестве инструмента для взаимодействия с базой данных на сервере используется Prisma ORM (Рисунок 2.1) — современный объектно-реляционный маппер, предоставляющий удобный и типизированный интерфейс для работы с данными. Prisma позволяет описывать модели данных в декларативной форме, автоматически генерировать SQL-запросы, управлять миграциями схемы базы данных и обеспечивать целостность информации. Это значительно упрощает реализацию бизнес-логики, повышает надёжность и ускоряет разработку серверной части.

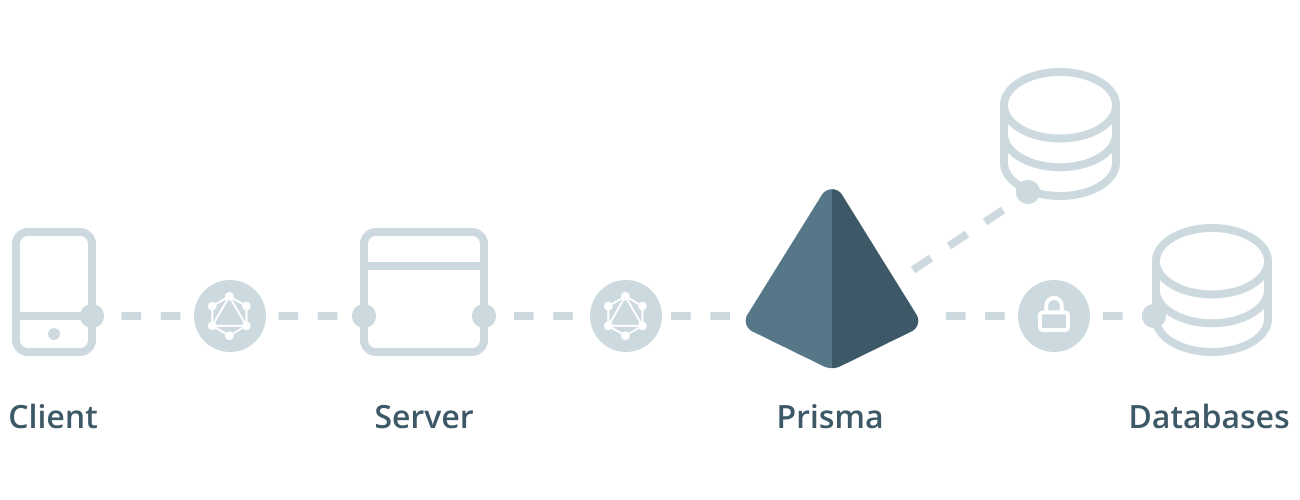


Рисунок 2.1 – принцип работы Prisma ORM

В качестве основной среды разработки используется Visual Studio Code — лёгкий, но при этом мощный и гибкий редактор кода, обладающий широкими возможностями расширения, настройки и интеграции с современными инструментами разработки. VS Code активно используется в профессиональной среде благодаря своей кроссплатформенности, высокой производительности и богатому набору функций, которые делают его идеальным выбором для реализации мобильных, веб- и серверных приложений. В контексте текущего проекта он обеспечивает удобную интеграцию с Android Studio SDK, позволяя быстро запускать сборку, выполнять горячую перезагрузку, отслеживать изменения в UI и проводить отладку в реальном времени. С помощью встроенного эмулятора и поддержки горячей перезагрузки (hot reload) можно мгновенно видеть изменения в UI при редактировании кода, что ускоряет итерации и позволяет оперативно тестировать гипотезы.

Visual Studio Code предоставляет расширенные возможности для организации удалённого доступа к локальным сервисам посредством функции проброса портов. Эта функция позволяет опубликовать локальный сервер, запущенный на произвольном порту (например, localhost:3000), через защищённый HTTPS-канал, предоставляя глобальную ссылку, доступную извне. Такая ссылка формируется через облачную инфраструктуру Microsoft и обеспечивает безопасный доступ к локальному окружению без необходимости ручного конфигурирования туннелей, прокси или внешнего хостинга.

Данная возможность особенно актуальна при разработке фронтенд-приложений на основе React, Vue или при тестировании API на Node.js, FastAPI, Express и других фреймворках. Проброс порта позволяет валидировать интерфейс и сетевую логику на реальных устройствах, включая мобильные браузеры, а также демонстрировать промежуточные версии заказчику или команде. Это существенно ускоряет цикл обратной связи, облегчает кросс-платформенное тестирование и позволяет выявлять ошибки, связанные с CORS, авторизацией, передачей токенов, обработкой заголовков и взаимодействием с внешними API.

Таким образом, функция проброса портов в Visual Studio Code выступает не просто как вспомогательный инструмент, а как полноценный механизм облачного доступа к локальному окружению, интегрированный в архитектуру разработки. Она способствует гибкой коллаборации, ускоряет тестирование, упрощает демонстрацию и делает локальную разработку доступной в глобальном контексте — без потери безопасности, производительности и контроля.

Для тестирования и отладки API-интерфейсов используется инструмент Postman — мощное средство для создания, отправки и анализа HTTP-запросов. С его помощью можно моделировать различные сценарии взаимодействия клиента и сервера, проверять корректность обработки запросов, анализировать структуру ответов, выявлять ошибки и документировать API (Рисунок 2.2). Postman позволяет создавать коллекции запросов, проводить автоматизированное тестирование, использовать переменные окружения и сценарии, что делает его незаменимым инструментом при разработке и проверке функционала регистрации, авторизации, оформления заказов, обработки заявок и генерации накладных.

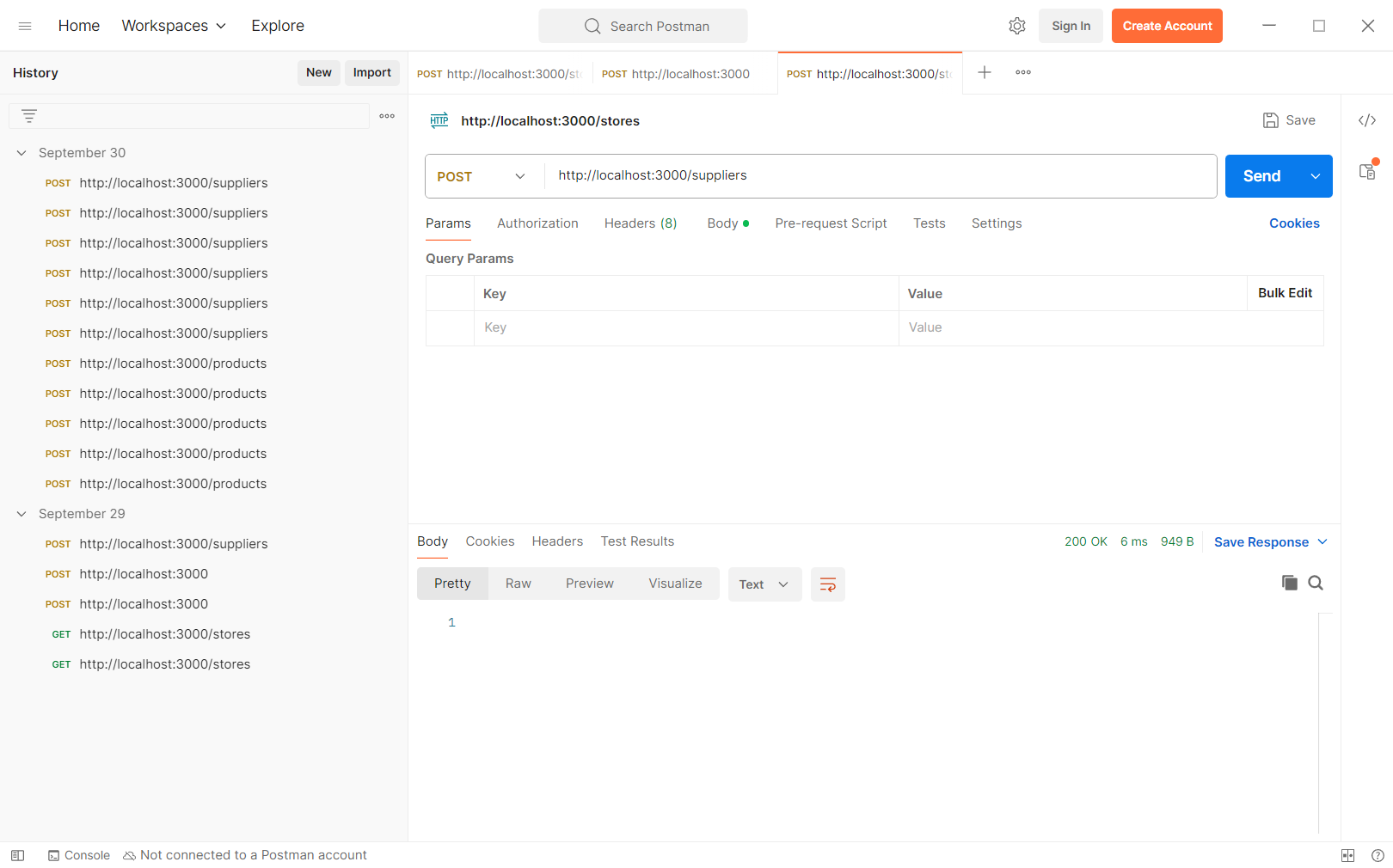


Рисунок 2.2 ­– окно программы postman

Для хранения данных на клиентской стороне используется встроенная реляционная база данных SQLite, которая зарекомендовала себя как надёжное, стабильное и широко применяемое решение в сфере разработки мобильных и десктопных приложений. Её архитектура позволяет сохранять информацию непосредственно на устройстве пользователя, что критически важно для обеспечения автономной работы приложения в условиях нестабильного или отсутствующего интернет-соединения. SQLite поддерживает полноценный набор функций, характерных для современных СУБД: транзакции, индексы, сложные запросы с фильтрацией, сортировкой и агрегацией, что делает её идеальной для реализации функционала складского учёта, фиксации остатков, хранения информации о товарах, заказах, пользователях, поставщиках, а также для ведения истории операций и аналитики. Благодаря своей лёгкости и отсутствию необходимости в отдельном сервере, SQLite обеспечивает высокую производительность и простоту интеграции, особенно в мобильных приложениях, где ресурсы ограничены.

Локальное хранение данных позволяет реализовать полноценный офлайн-режим, что критически важно для магазинов, работающих в удалённых регионах, на выездных точках продаж, в помещениях с ограниченным доступом к сети или в условиях нестабильной связи. Это обеспечивает непрерывность бизнес-процессов, возможность фиксации операций в реальном времени и последующую синхронизацию с сервером при восстановлении соединения. Такой подход повышает надёжность системы, снижает зависимость от внешних факторов и делает приложение более устойчивым к сбоям.

Система контроля версий построена на базе Git, а репозиторий проекта размещён на платформе GitHub, что обеспечивает прозрачность разработки, возможность командной работы, отслеживание изменений, управление ветками, проведение ревью и документирование процесса. GitHub предоставляет мощные инструменты для автоматизации сборки, тестирования и деплоя, включая интеграцию с CI/CD-пайплайнами, что позволяет ускорить выпуск новых версий, повысить качество кода и обеспечить стабильность релизов. Такой подход способствует масштабируемости проекта, упрощает сопровождение и открывает возможности для интеграции с внешними сервисами и платформами.

Выбор указанных инструментальных средств обусловлен их надёжностью, популярностью в профессиональной среде, активным сообществом, богатой документацией, регулярными обновлениями и высокой степенью совместимости между собой. Это позволяет реализовать все поставленные задачи проекта, обеспечить стабильную работу приложения, удобный и интуитивно понятный интерфейс, безопасность хранения данных, возможность масштабирования и адаптации под различные сценарии использования. Использование проверенных технологий снижает риски при разработке, упрощает сопровождение, ускоряет внедрение новых функций и обеспечивает долгосрочную устойчивость решения. Такой подход делает создаваемое приложение актуальным для малого и среднего бизнеса, нуждающегося в доступных, надёжных и эффективных инструментах автоматизации торговли, логистики, аналитики и управления клиентскими данными.

## **2.2 Организация данных**

Для работы приложения используется база данных SQLite вместе с Prisma ORM, что обеспечивает эффективное и типобезопасное взаимодействие с локальными данными. SQLite — это легковесная реляционная база данных, которая хранится в виде одного файла на устройстве и не требует отдельного сервера. Благодаря своей компактности и автономности, она идеально подходит для мобильных приложений, особенно на платформе Android. SQLite поддерживает стандартный SQL и широко применяется для хранения пользовательских данных, кэширования, офлайн-доступа и других задач.

Prisma ORM — это современный инструмент для работы с базами данных, ориентированный на разработчиков, использующих TypeScript и JavaScript. Он предоставляет удобный способ взаимодействия с базой данных через автогенерируемый клиент, который позволяет выполнять запросы к данным с помощью типизированного API. Prisma состоит из нескольких компонентов: Prisma Client, Prisma Migrate и Prisma Studio. Prisma Client — это основной интерфейс для работы с базой данных, Prisma Migrate — инструмент для управления миграциями, а Prisma Studio — визуальный редактор данных.

В контексте Android-приложения Prisma ORM может использоваться в серверной части или в приложениях, построенных с использованием кроссплатформенных технологий, таких как React Native или Node.js.. В классической Android-разработке на Java или Kotlin чаще применяются другие ORM-инструменты, такие как Room, однако Prisma становится актуальной при наличии серверного взаимодействия или при использовании JavaScript-стека.

Для подключения Prisma к базе данных SQLite необходимо корректно настроить файл schema.prisma, указав источник данных с провайдером sqlite и путь к файлу базы данных. Конфигурация источника данных задаётся в блоке datasource, где определяется тип провайдера и URL‑адрес, указывающий на физическое расположение файла базы.

Особое значение в работе с Prisma имеет механизм миграций. Миграции позволяют отслеживать изменения в структуре базы данных и обеспечивают согласованность между описанием моделей и фактическим состоянием таблиц. При внесении изменений в файл schema.prisma можно создать новую миграцию, которая будет применена к базе данных. Это особенно важно в условиях командной разработки, когда несколько участников проекта параллельно работают над одной системой, а также при необходимости поддерживать версионность базы данных и документировать все изменения.

Использование Prisma совместно с SQLite имеет ряд преимуществ. Во‑первых, SQLite является лёгкой и встроенной базой данных, которая не требует отдельного сервера и идеально подходит для небольших проектов, прототипов или учебных задач. Во‑вторых, Prisma обеспечивает удобный и современный способ работы с данными, предоставляя типизированный доступ к таблицам и позволяя использовать возможности автогенерации кода. В результате разработчик получает мощный инструмент, который сочетает простоту настройки и гибкость в управлении данными.Prisma Client генерируется автоматически на основе схемы и позволяет выполнять CRUD-операции (создание, чтение, обновление, удаление) с типовой безопасностью. Это снижает количество ошибок и повышает производительность разработки. Например, можно получить список пользователей с помощью client.user.findMany(), где user — это модель, описанная в схеме. Такой подход обеспечивает строгую типизацию и автодополнение в редакторе кода, что делает разработку более удобной и безопасной.

Структура базы данных (Рисунок 2.3) организована в виде взаимосвязанных таблиц, каждая из которых отвечает за хранение информации по ключевым сущностям бизнес-логики: поставщики, товары, магазины, партии товара и склады. Такая модель обеспечивает логическую целостность, упрощает выполнение запросов, позволяет реализовать фильтрацию, сортировку, агрегацию и аналитическую обработку данных.

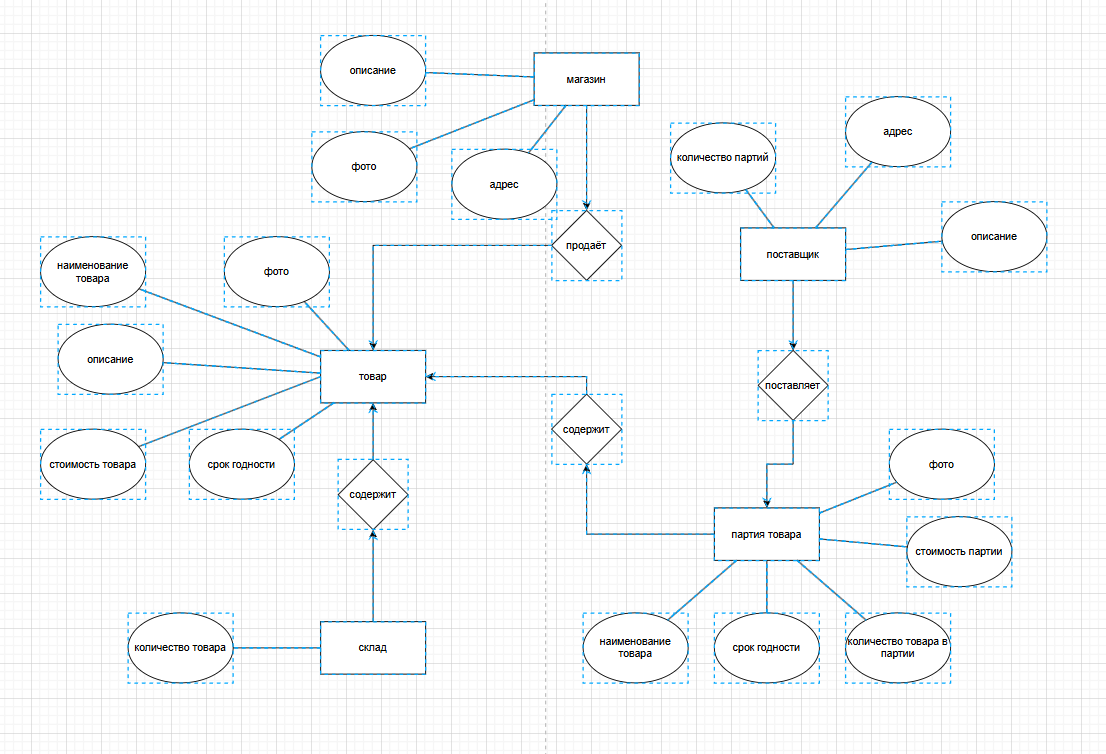


Рисунок 2.3 – схема базы данных

Таблица "поставщик" содержит сведения об организациях, поставляющих продукцию: адрес, описание, количество поставленных партий. Через связь "поставляет" она ассоциирована с таблицей "партия товара", отражающей конкретные поставки.

Таблица "партия товара" включает информацию о каждой партии: фото, стоимость, наименование товара, срок годности и количество. Каждая партия связана с конкретным товаром, что позволяет отслеживать происхождение и характеристики поставок.

Таблица "товар" представляет собой центральную сущность, содержащую наименование, стоимость, срок годности, количество, описание и фото. Товары связаны с магазинами (через отношение "продаёт") и складами (через отношение "содержит"), что позволяет отслеживать распределение и остатки.

Таблица "магазин" хранит адреса торговых точек и реализует связь с товарами, которые доступны для продажи в конкретном магазине.

Таблица "склад" отражает физические или виртуальные зоны хранения, где содержатся товары. Это позволяет реализовать учёт остатков, логистику и перемещение продукции.

Все связи между таблицами реализованы через внешние ключи, что обеспечивает согласованность данных и предотвращает логические ошибки. Такая структура позволяет легко масштабировать систему, добавлять новые сущности (например, категории товаров, заказы, возвраты), а также реализовать сложные бизнес-правила, включая фильтрацию по сроку годности, отслеживание партий, анализ поставок и управление запасами.

## **2.3 Обоснование и разработка интерфейса**

В процессе разработки пользовательского интерфейса приложения использовался фреймворк Flutter, предоставляющий широкие возможности для построения кроссплатформенных UI-компонентов. Flutter основан на декларативном подходе и использует язык Dart для описания визуальных элементов, что позволяет создавать гибкие и отзывчивые интерфейсы с минимальными усилиями.

В качестве основных компонентов интерфейса были задействованы следующие элементы: — Text — для отображения текстовой информации, включая заголовки, описания и вспомогательные надписи. Этот виджет позволяет задавать стиль текста, выравнивание и другие параметры визуализации. — Image — для отображения графических элементов, таких как иконки, иллюстрации или фотографии. Виджет Image поддерживает загрузку изображений из локальных ресурсов и сети, а также масштабирование и обрезку. — ElevatedButton — для реализации интерактивных элементов управления, таких как кнопки подтверждения, перехода или отправки данных. Кнопки оформлены с использованием стандартных стилей Flutter, обеспечивающих визуальную согласованность и доступность.

Для построения структуры экрана использовались стандартные контейнерные виджеты, такие как Column, Row и Container, позволяющие организовать элементы в вертикальные и горизонтальные блоки. Виджет Container также применялся для задания отступов, фона и границ, обеспечивая визуальное разделение и структурирование интерфейса.

В процессе верстки активно использовались следующие параметры: — width и height — для задания размеров компонентов, обеспечивая контроль над их визуальным представлением. — padding и margin — для управления внутренними и внешними отступами, что позволило добиться аккуратного расположения элементов и избежать визуального наложения. — alignment — для выравнивания содержимого внутри контейнеров, особенно в случаях, когда требовалось центрирование или привязка к краям.

Адаптация интерфейса под различные размеры экранов и ориентации не производилась, так как приложение ориентировано на использование в стандартных условиях. Все размеры и отступы задавались вручную, без применения адаптивных стратегий или медиазапросов. Это решение обусловлено спецификой проекта и ограниченным числом целевых устройств.

Таким образом, интерфейс приложения реализован с использованием базовых компонентов Flutter, обеспечивающих простоту, наглядность и функциональность. Выбор элементов и параметров был направлен на достижение минималистичного и понятного пользовательского опыта без избыточной визуальной нагрузки.

## **2.4 Отладка и тестирование разрабатываемого приложения**

Процесс тестирования и отладки является завершающим этапом создания программного продукта. Его основная цель — выявление и устранение ошибок, возникающих в ходе разработки, а также проверка корректности работы приложения в различных сценариях. Отладка и тестирование позволяют убедиться, что приложение функционирует стабильно, соответствует требованиям и обеспечивает ожидаемый пользовательский опыт.

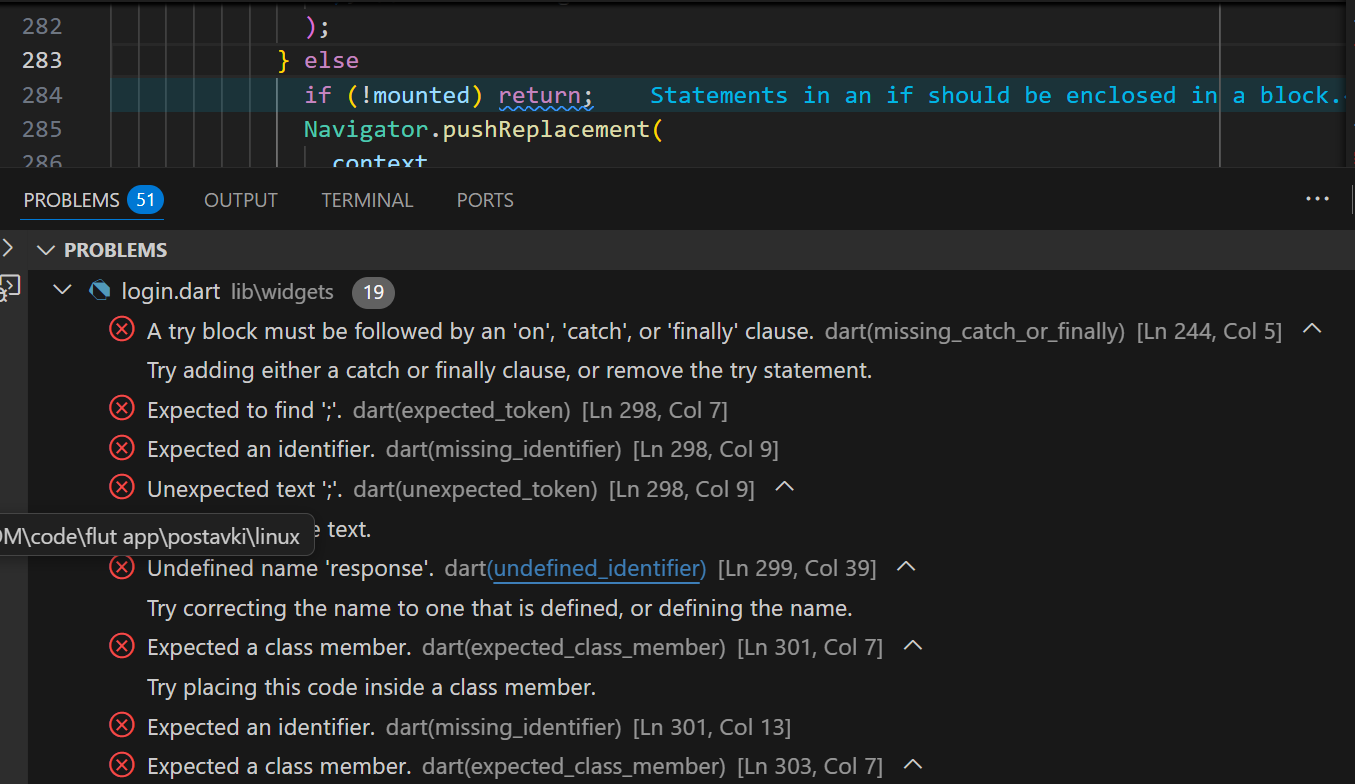
Отладка — это систематический процесс анализа и устранения ошибок в программном коде. Он включает в себя последовательные этапы: фиксацию и воспроизведение сбоя, детальное исследование его симптомов, установление причин возникновения, локализацию проблемных фрагментов кода и внесение целенаправленных исправлений. В ходе отладки разработчики проверяют гипотезы о поведении программы, отслеживают состояние переменных и поток выполнения, оценивают влияние правок на смежные участки и повторно тестируют систему, чтобы убедиться в устойчивости решения и отсутствии регрессий. В процессе разработки этого приложения отладка проводилась с использованием встроенного дебаггера в среде Visual Studio Code, что позволило выполнять пошаговое прохождение кода, просматривать актуальные значения переменных во время выполнения и анализировать логику работы отдельных компонентов для точной локализации ошибок.

Тестирование — это комплексный процесс проверки работы программы на заранее подготовленных данных, цель которого заключается в том, чтобы убедиться в корректности её поведения и соответствии ожидаемым результатам. Оно играет ключевую роль в обеспечении качества программного обеспечения, так как позволяет выявить ошибки и недочёты ещё до того, как продукт попадёт к конечному пользователю.

Процесс тестирования включает несколько последовательных этапов: подготовку сценариев, выбор входных параметров и их систематическое применение в ходе выполнения программы. Каждый сценарий моделирует определённую ситуацию использования, что позволяет проверить работу приложения в различных условиях. После выполнения действий результаты тщательно сопоставляются с ожидаемыми ответами, и при обнаружении несоответствий фиксируются ошибки, которые впоследствии устраняются разработчиками.

В процессе разработки были обнаружены и устранены следующие типы ошибок:

— Синтаксические ошибки — возникают при нарушении правил языка программирования. Примером может служить пропущенная запятая или неправильное закрытие скобки в структуре вид. Такие ошибки фиксировались средой разработки автоматически, и их устранение происходило на этапе компиляции. На рисунке 2.4 представлена ошибка, был пропущен знак «{» (фигурная скобка).

  
Рисунок 2.4 – синтаксическая ошибка

Семантические ошибки представляют собой ситуации, когда программа формально написана правильно, но её логика нарушена и она не выполняет задуманную функцию. В таких случаях код может компилироваться и запускаться без явных проблем, однако результат работы оказывается неверным. Примером может служить попытка отобразить текст до того, как было инициализировано соответствующее значение, что приводило к пустому экрану или некорректному выводу информации. Для устранения подобных ошибок приходилось пересматривать структуру кода, изменять порядок вызовов функций и уточнять условия отображения, чтобы гарантировать корректное выполнение задуманной логики и правильное взаимодействие компонентов приложения.

Ошибки выполнения проявляются непосредственно во время работы программы и часто приводят к её сбою или аварийному завершению. Одним из типичных примеров является обращение к переменной, значение которой оказалось равным null, что вызывало критическую ошибку. Для предотвращения подобных ситуаций применялись конструкции try/catch, позволяющие перехватывать исключения и обеспечивать продолжение работы программы без её краха. Дополнительно использовались проверки наличия значений перед выполнением операций, что позволяло заранее исключить возможность обращения к пустым данным. Такой подход обеспечивал более устойчивую работу приложения, снижал вероятность непредвиденных сбоев и повышал общую надежность системы.

Совместное использование отладки и ручного тестирования обеспечивало цикличный процесс улучшения качества: баги выявлялись во время тестовых прогонов, затем исследовались и устранялись через отладчик, после чего повторное тестирование подтверждало исправление и отсутствие побочных эффектов. Этот цикл способствовал стабилизации работы интерфейса и логики приложения и позволял своевременно реагировать на обнаруженные проблемы, повышая общую надежность и предсказуемость поведения системы.

Процесс отладки включал следующие этапы:

— выявление ошибок с помощью дебаггера и логов;

— диагностика и локализация проблемных участков;

— внесение изменений в структуру кода;

— повторное тестирование после исправлений.

Таким образом, проведённые процессы отладки и тестирования стали неотъемлемой частью жизненного цикла разработки и обеспечили высокий уровень качества программного продукта. Их систематическое применение позволило не только своевременно выявить и устранить критические ошибки, но и сформировать устойчивую архитектуру, способную предсказуемо реагировать на различные сценарии использования. Встроенные инструменты отладки в среде разработки предоставили возможность глубоко анализировать логику работы компонентов, отслеживать значения переменных и контролировать последовательность выполнения операций, что значительно повысило прозрачность процесса разработки.

Ручное тестирование, дополненное внимательной проверкой интерфейса и функционала, подтвердило корректность реализации ключевых модулей и обеспечило уверенность в стабильности работы приложения. В результате удалось минимизировать вероятность возникновения сбоев, повысить надёжность системы и создать прочную основу для её дальнейшего развития. Такой подход не только укрепил доверие к приложению со стороны пользователей, но и заложил фундамент для масштабирования проекта, интеграции новых функций и адаптации к изменяющимся требованиям бизнеса.

Итогом стало формирование зрелого и качественного программного решения, которое сочетает в себе устойчивость, предсказуемость и готовность к расширению, что делает его перспективным для долгосрочного использования и совершенствования.

## **2.5 Системные требования**

Для успешной установки и запуска приложения, устройство должно соответствовать следующим требованиям:

— Операционная система: Android 9 (Pie) Приложение разработано с учётом особенностей и API, доступных начиная с версии Android 9.0 (Pie). Данная версия операционной системы обеспечивает необходимую поддержку современных компонентов пользовательского интерфейса, а также гарантирует совместимость с актуальными библиотеками и механизмами безопасности. Android 9 включает улучшенное управление энергопотреблением, поддержку адаптивной яркости, а также расширенные возможности уведомлений, что положительно сказывается на пользовательском опыте.

— Оперативная память (RAM): не менее 2 ГБ Для обеспечения плавной работы приложения, особенно при одновременной загрузке изображений, обработке пользовательских действий и взаимодействии с сетью, требуется минимум 2 гигабайта оперативной памяти. Это позволяет избежать принудительного завершения процессов системой и обеспечивает стабильную работу даже при многозадачности.

— Постоянная память (ROM): не менее 50 МБ свободного пространства Приложение требует не менее 50 мегабайт свободного пространства во внутреннем хранилище устройства. Этот объём необходим для установки самого приложения, хранения временных файлов, кэшированных изображений и базовых данных. Указанный объём является минимальным и может увеличиваться при активной эксплуатации приложения.

— Процессор: не менее 1.5 ГГц Для обеспечения приемлемой скорости отклика интерфейса и выполнения фоновых операций (например, загрузки данных, обработки запросов к базе данных) требуется центральный процессор с тактовой частотой не ниже 1.5 гигагерц. Это позволяет избежать задержек при навигации по экранам и обеспечивает комфортную работу пользователя.

— Интернет-соединение: стабильный доступ к сети Приложение использует сетевые запросы для взаимодействия с удалённой базой данных и получения актуальной информации. Поэтому наличие стабильного интернет-соединения является обязательным условием для полноценного функционирования.

# **3. ОПИСАНИЕ ПРИМЕНЕНИЯ ПРИЛОЖЕНИЯ**

Приложение предназначено для упрощения взаимодействия между поставщиками и магазинами, обеспечивая удобный и структурированный процесс обмена товарными партиями. Основная функциональность заключается в том, что поставщики могут размещать партии товаров в системе, указывая их состав, количество и другие параметры, необходимые для последующей обработки. После размещения партии становятся доступными для магазинов, которые, в свою очередь, имеют возможность просматривать доступные предложения, оформлять заказы на нужные партии и получать товары для дальнейшей реализации.

Магазины заказывают партии с целью последующей розничной продажи, реализуя товары поштучно. Такой подход позволяет оптимизировать товарооборот, снизить издержки на логистику и упростить процесс закупки. Кроме того, приложение предоставляет магазинам возможность получать накладные на оформленные заказы, что способствует прозрачности документооборота и упрощает бухгалтерский учёт.

Работа с приложением начинается с окна входа и регистрации (Рисунок 3.1). При наличии аккаунта нужно в него войти, а при отсутствии – создать.

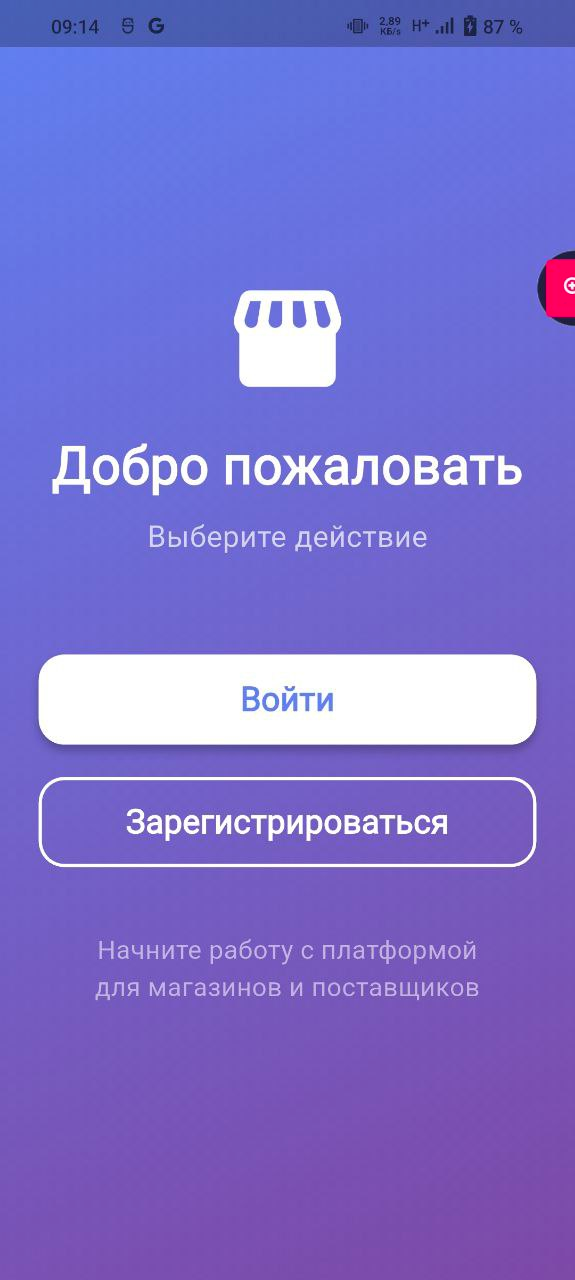
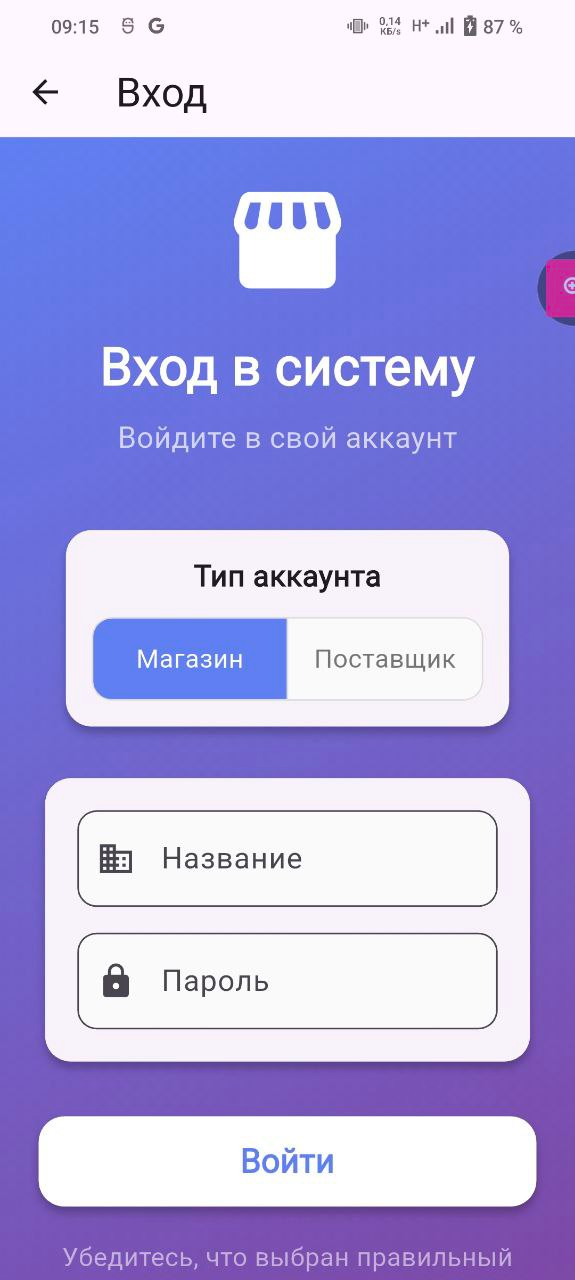
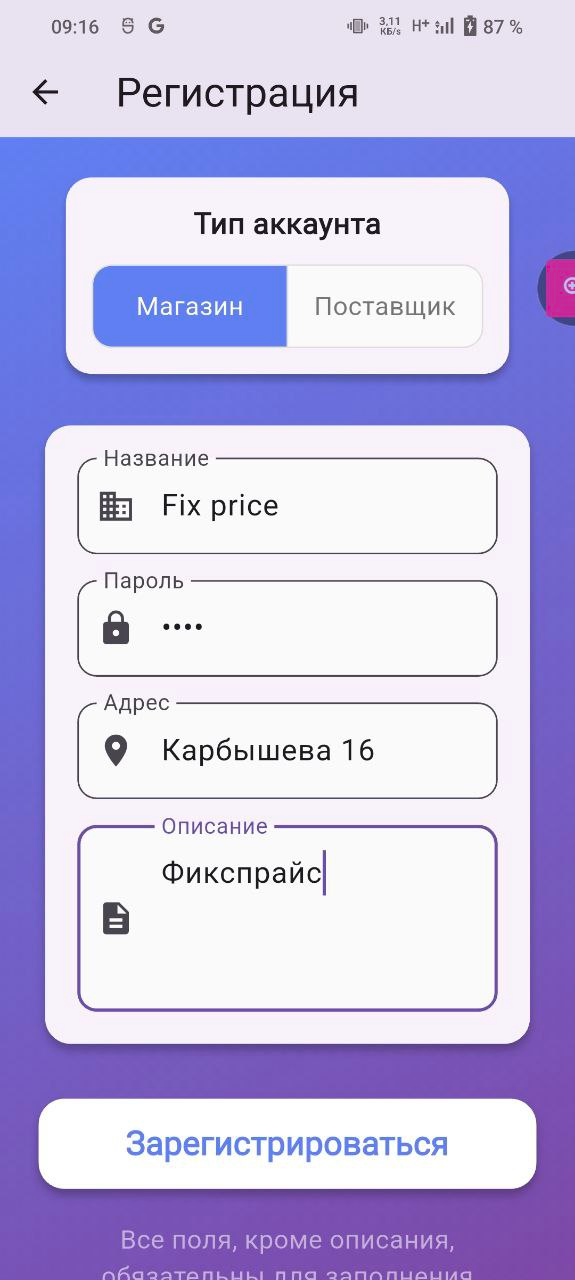
  

Рисунок 3.1 – вход и регистрация

После завершения процесса регистрации администратор магазина попадает в раздел склада. На данном этапе склад еще пустой, и для того чтобы наполнить его товарами, необходимо перейти враздел, посвященный поставкам, где можно оформить заказ на нужные позиции (Рисунок 3.2). В карточках каждого товара отображается его название, подробное описание, стоимость и фотография, что позволяет легко ориентироваться в ассортименте. Для удобства работы предусмотрены функции сортировки партий, возможность их фильтрации по различным параметрам, а также поиск, который помогает быстро находить нужные товары среди доступных вариантов.

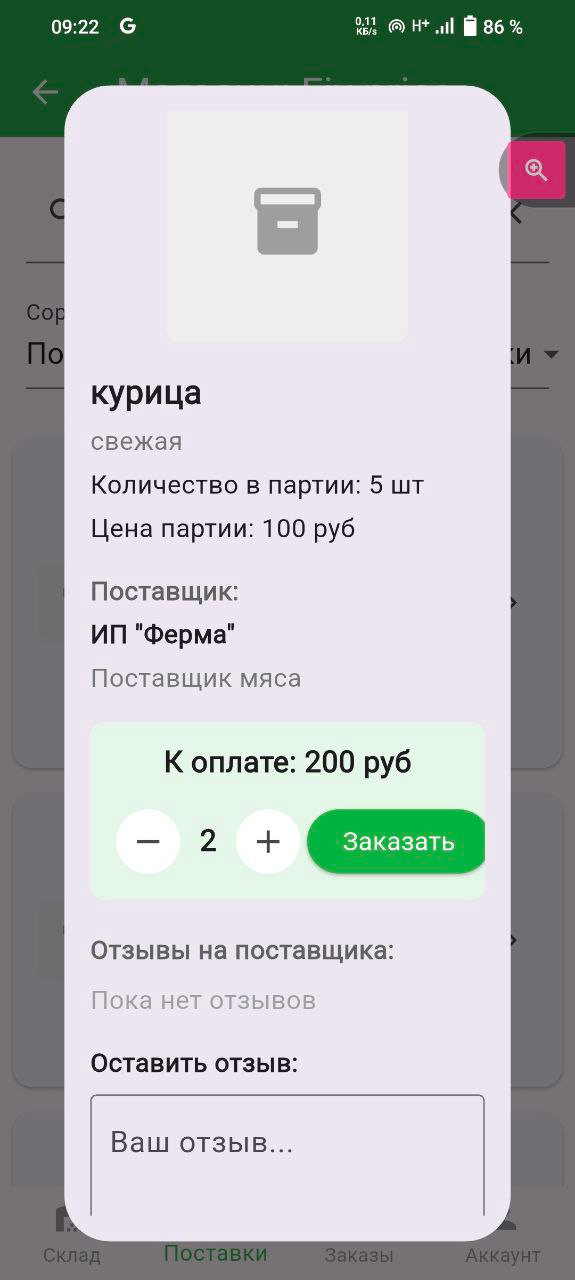
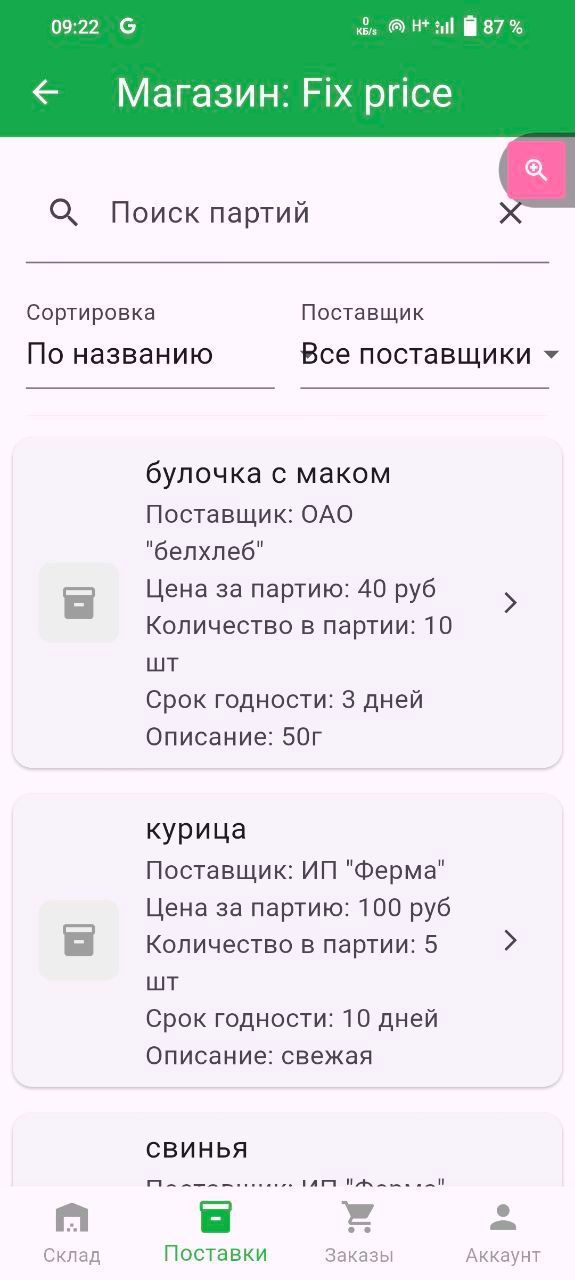


Рисунок 3.2 – заказ партии

Все заказы, включая оформленные, отправленные и полученные, отображаются в меню заказов (Рисунок 3.3). Для удобства работы предусмотрены функции сортировки и фильтрации, позволяющие быстро находить нужные позиции и управлять списком заказов. После того как заказ оформлен, магазину остается лишь ожидать момента, когда поставщик примет его и начнет дальнейшую обработку.

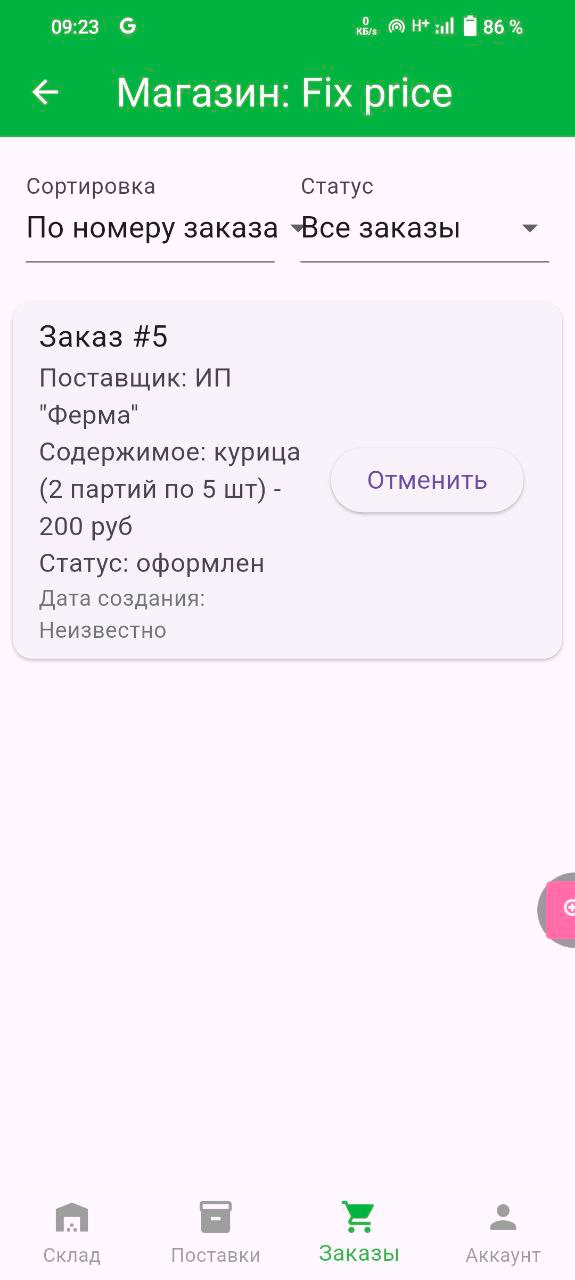


Рисунок 3.3 – меню заказов

Когда поставщик авторизуется и входит в приложение, на его экране автоматически отображаются все новые заказы, которые необходимо подготовить и отправить (Рисунок 3.4). Таким образом, он сразу видит актуальные заявки и может оперативно приступить к их обработке, не тратя время на поиск или дополнительные действия.

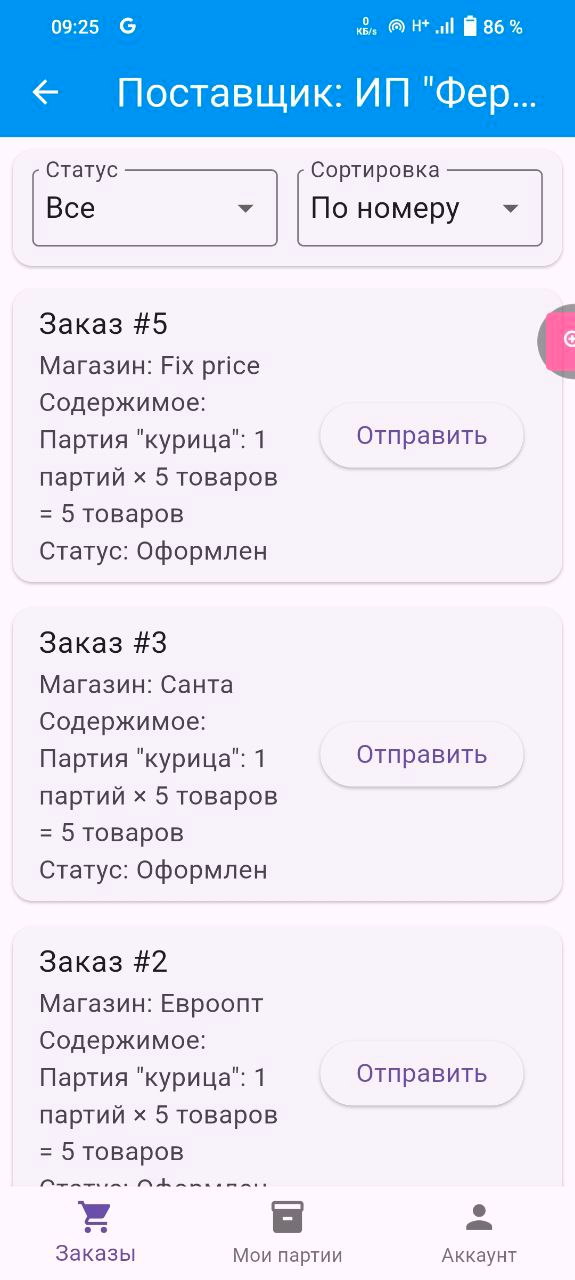
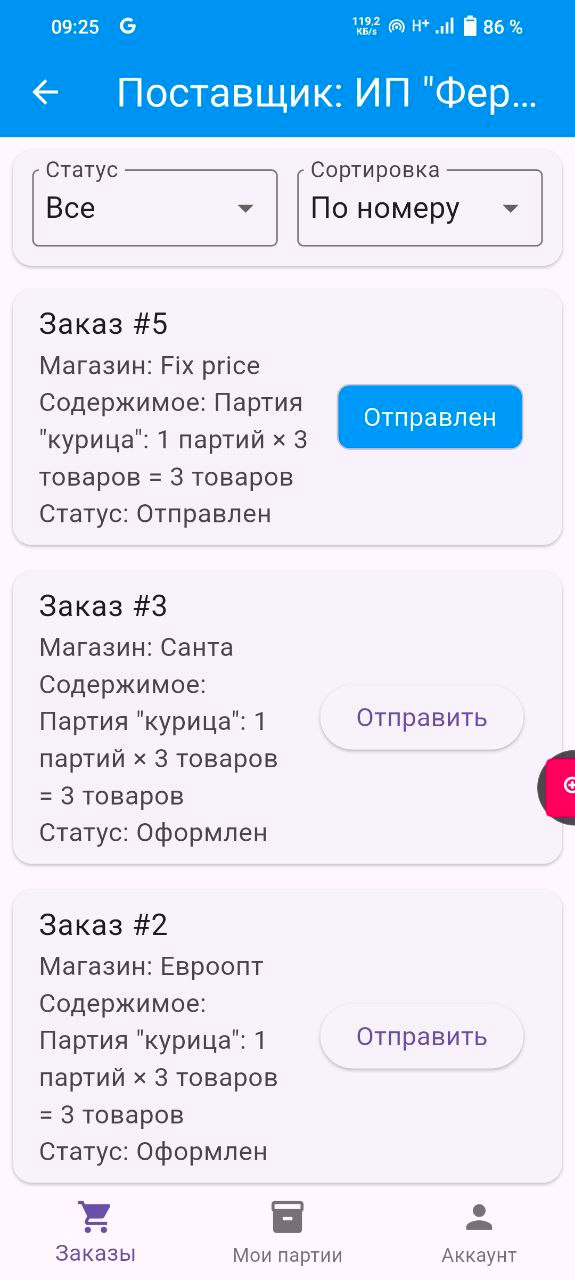
 

Рисунок 3.4 – отправка заказа

После того как заказ отправлен, поставщик получает возможность работать с партиями товаров. В приложении он может просматривать уже существующие партии, анализировать их состав и состояние, а также добавлять новые партии для расширения ассортимента (Рисунок 3.5). Такой функционал обеспечивает удобное управление товарными запасами и позволяет оперативно обновлять складскую информацию.

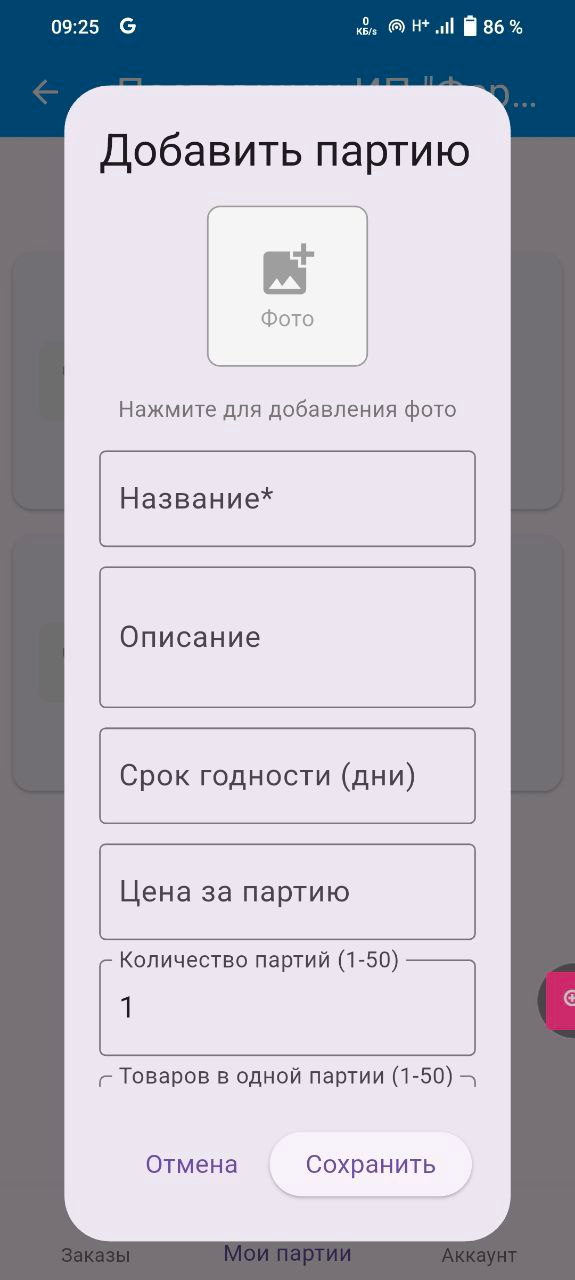
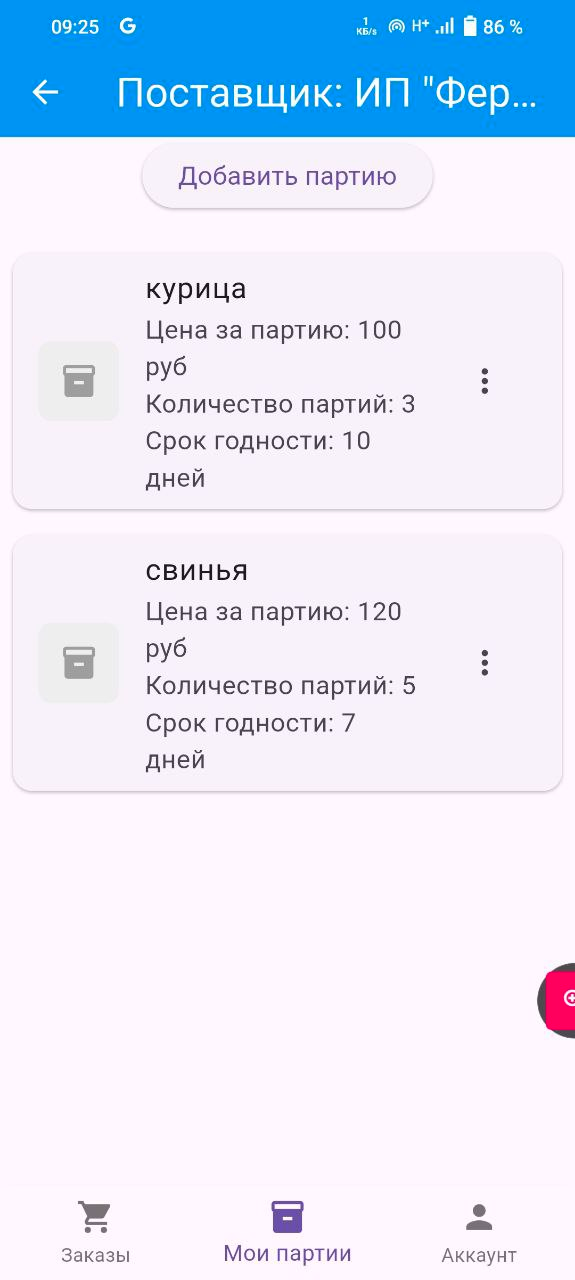


Рисунок 3.5 – просмотр и создание партий

Поставщик имеет возможность редактировать свой аккаунт: менять пароль, обновлять имя и корректировать адрес, что позволяет поддерживать актуальные учетные данные и оперативно вносить необходимые изменения в личную информацию.

После того как поставщик отправил заказ и доставка была выполнена, магазин получает возможность принять его. После подтверждения приема заказ автоматически добавляется на склад, где он становится доступным для дальнейшего использования и управления (Рисунок 3.6).

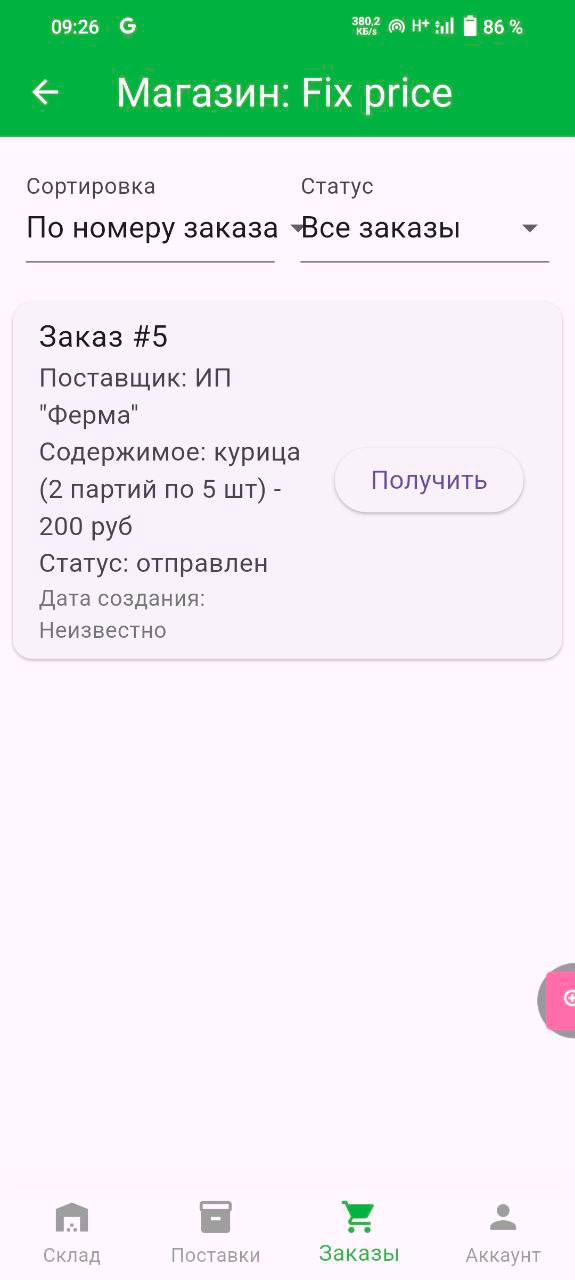


Рисунок 3.6 –получение заказа

После того как заказ был получен и принят магазином, становится доступна дополнительная функция — просмотр автоматически сформированной накладной. Этот документ создается системой без необходимости ручного ввода данных, что значительно упрощает процесс учета. Кроме того, магазин может распечатать накладную, используя встроенные в andriod инструменты, что обеспечивает удобство ведения документации и позволяет быстро оформить все необходимые бумаги (Рисунок 3.7).

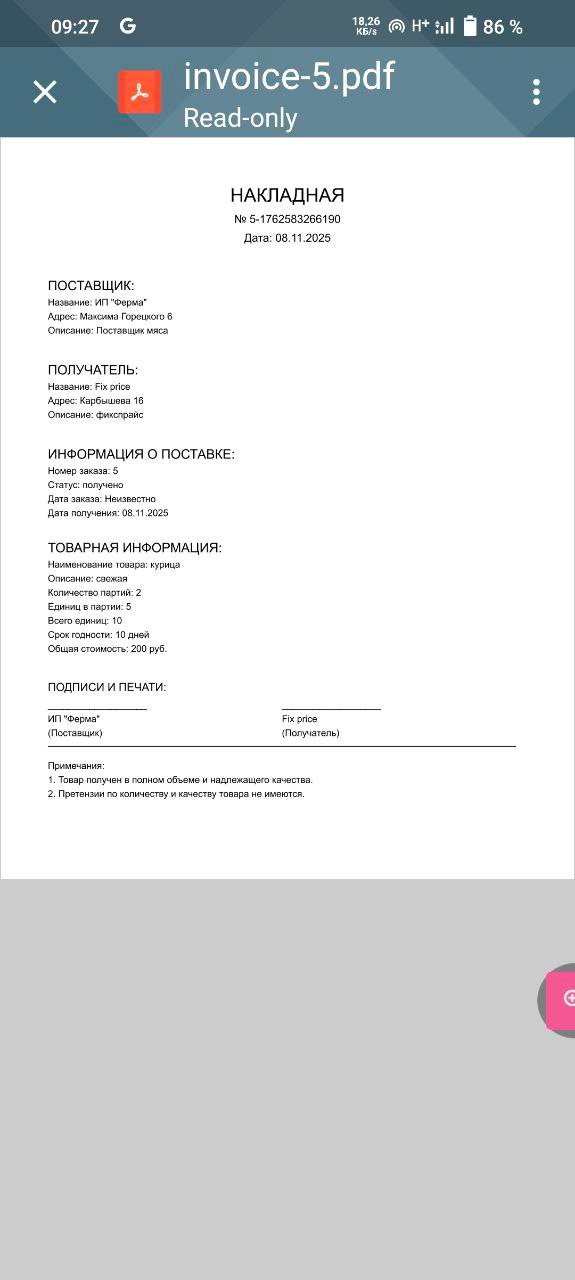
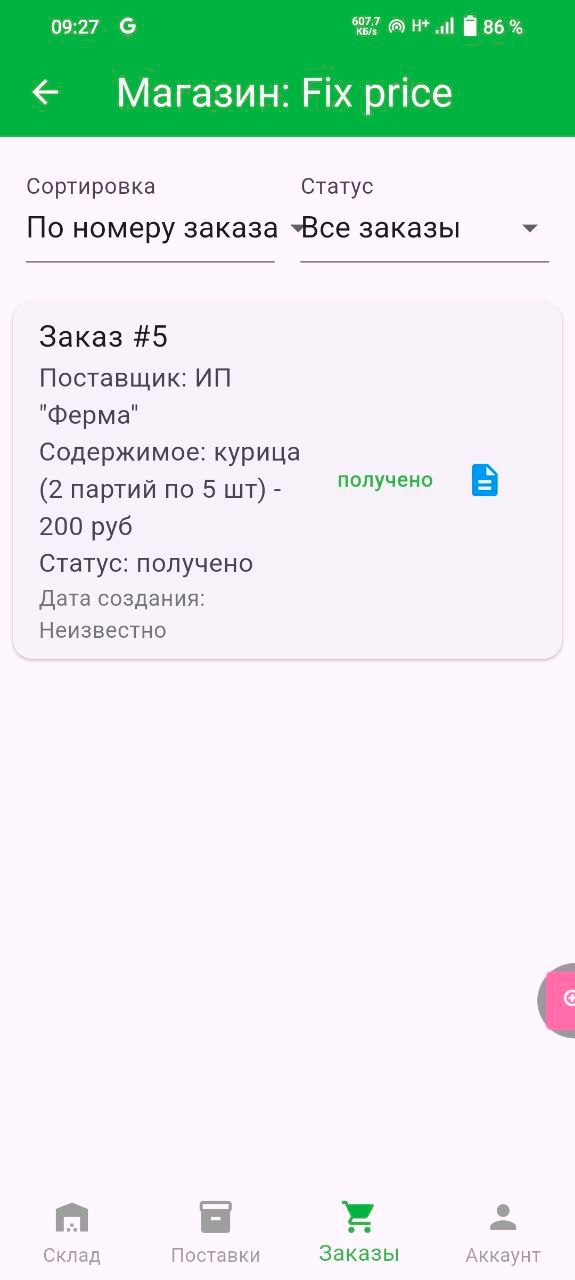


Рисунок 3.7 – накладная

В процессе розничной торговли магазин получает полный доступ к управлению складом. В этом разделе предусмотрены функции создания новых товаров, редактирования уже существующих позиций и их последующей продажи. Такой подход обеспечивает удобное ведение учета, позволяет оперативно обновлять ассортимент и контролировать движение товаров на складе, что делает процесс торговли более прозрачным и эффективным (Рисунок 3.8).

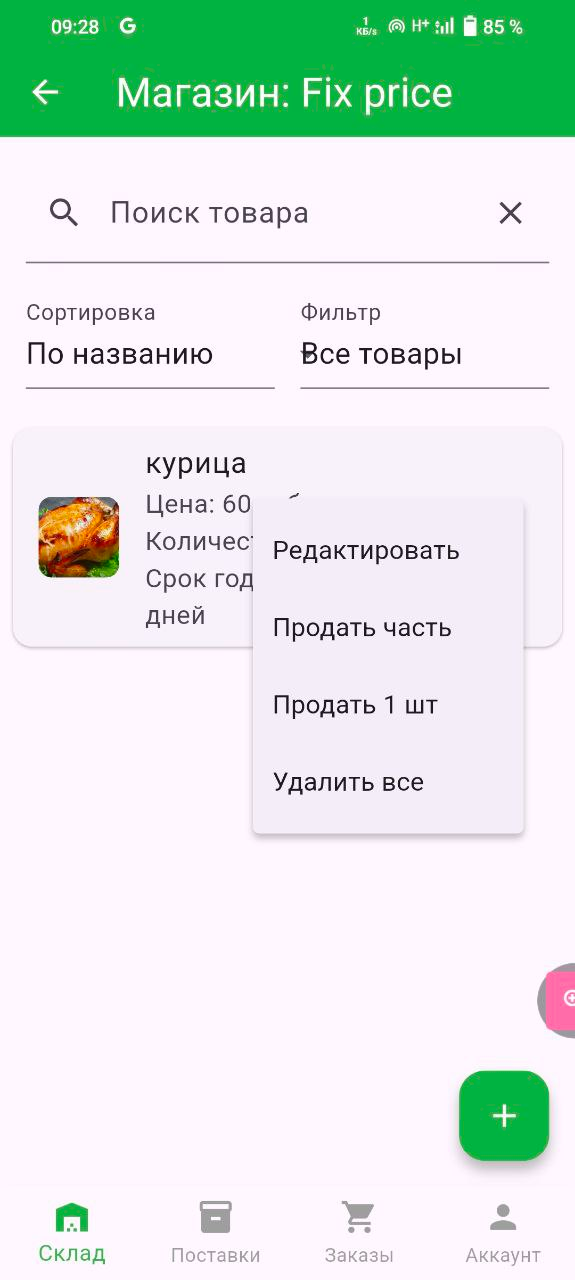
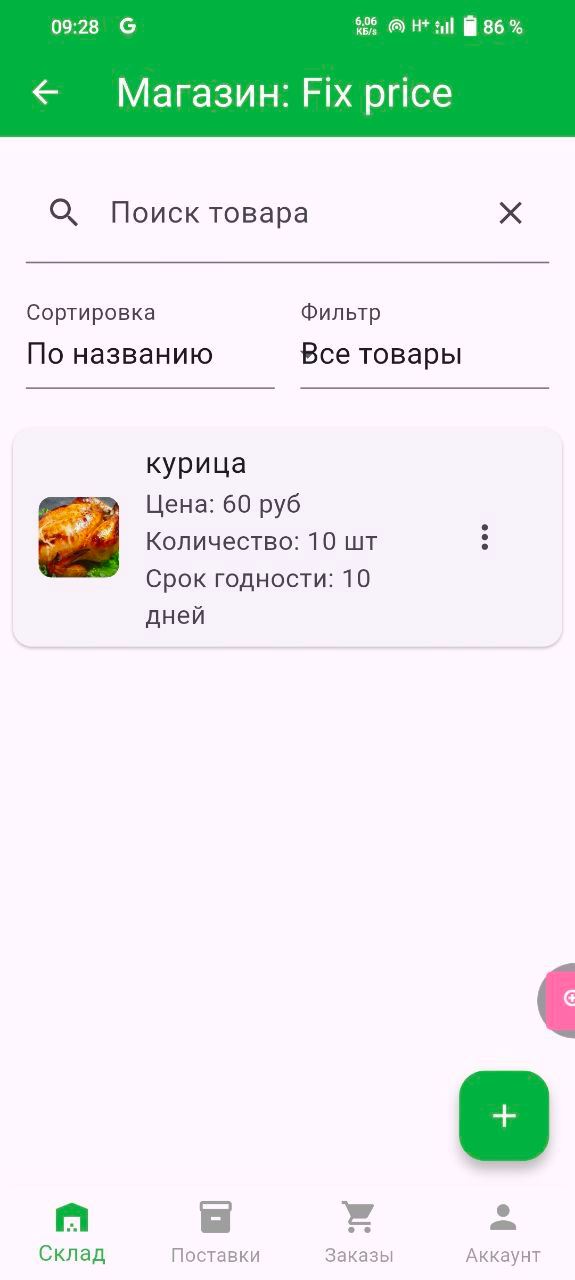


Рисунок 3.8 – управление складом.

Магазин имеет возможность оставить отзыв о поставщике, который затем будет отображаться в его интерфейсе. Таким образом, поставщик сможет увидеть полученный отзыв и учитывать его в дальнейшей работе, что способствует более прозрачному взаимодействию и повышает качество сотрудничества (Рисунок 3.9).

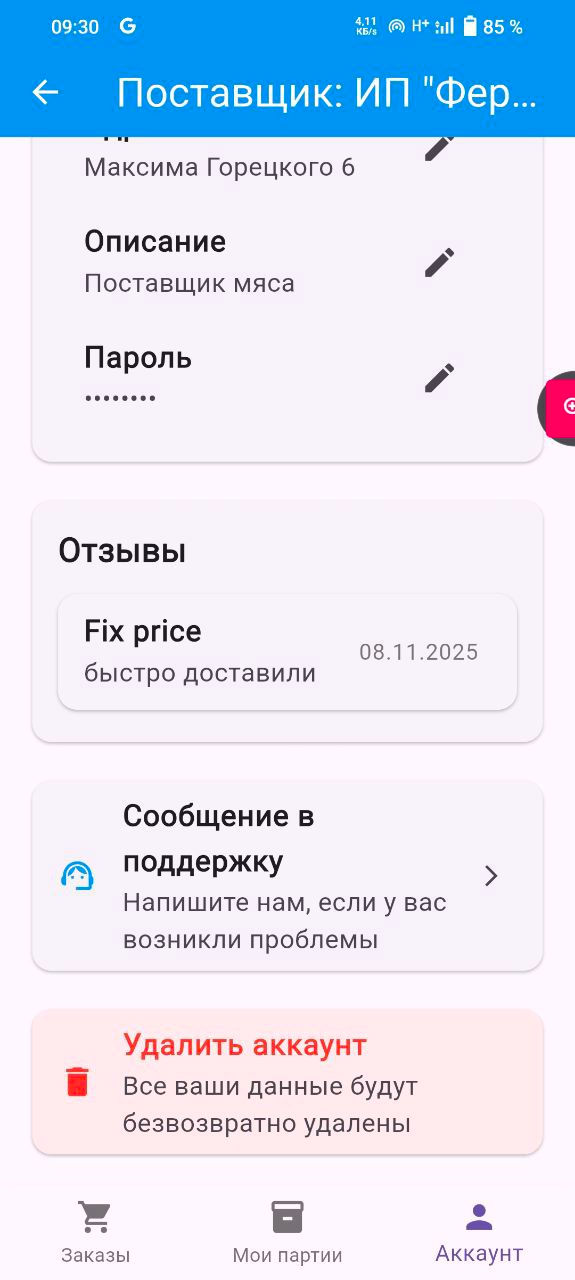
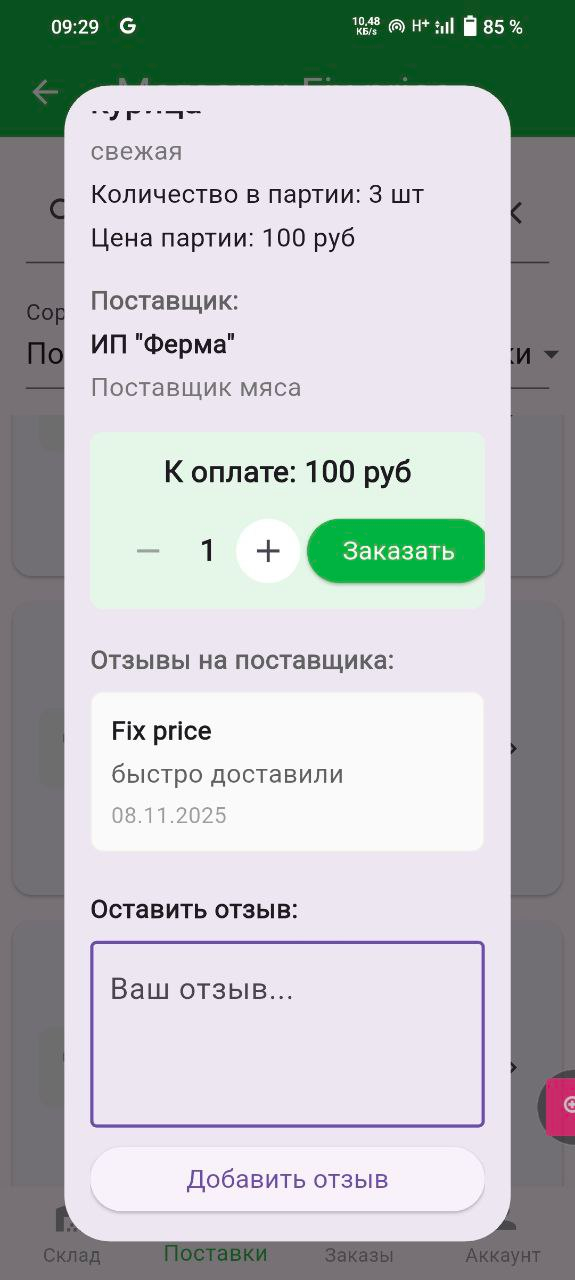


Рисунок 3.9 – оставление и просмотр отзыва

Приложение демонстрирует связный, логичный и завершённый контур работы, охватывающий все ключевые этапы взаимодействия между магазином и поставщиком. Магазин получает возможность оформлять и отслеживать заказы, принимать поставки и вести розничные операции в единой системе, а поставщик оперативно видит поступающие заявки, организует отгрузку и поддерживает актуальность товарных партий. Такой подход обеспечивает непрерывность процессов и исключает разрывы в информационном обмене.

Сквозные инструменты — сортировка, фильтры, поиск и автоматическая генерация накладных — значительно снижают объём ручных операций и вероятность ошибок. Это повышает скорость реакции на изменения, делает учёт прозрачным и управляемым. Простые карточки товара с ключевыми атрибутами обеспечивают наглядную навигацию по ассортименту, а единый раздел заказов устраняет разрозненность данных и ускоряет принятие решений.

Важным элементом является взаимная обратная связь через систему отзывов. Она укрепляет доверие и дисциплину процессов: поставщик получает сигнал о качестве взаимодействия, а магазин — дополнительный инструмент влияния на уровень сервиса. Конструкция ролей и чёткая стадийность жизненного цикла заказа создают предсказуемость и прозрачность: каждому участнику очевидно текущее состояние процесса и следующий шаг. В результате формируется устойчивая операционная среда, где каждое действие фиксируется и оставляет однозначный след в системе.

Итоговый эффект выражается в управляемости, прозрачности и снижении транзакционных издержек. Магазин быстрее формирует, принимает и реализует товар, а поставщик точнее планирует отгрузки и поддерживает актуальность ассортимента. Система масштабируется по объёму операций благодаря формализованным стадиям и унифицированным интерфейсам, оставаясь понятной и компактной в ежедневной работе.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была разработана мобильная платформа «ПОМОЩНИК ПОСТАВЩИКАМ И МАГАЗИНАМ», ориентированная на автоматизацию процессов взаимодействия между магазинами и поставщиками. Реализация данного решения позволила создать удобные и функциональные инструменты для управления складским учетом, оформления и отслеживания заказов, работы с товарными партиями и организации коммуникации между участниками цепочки поставок.

Проведённые этапы отладки и тестирования подтвердили корректность работы приложения, его устойчивость и надёжность. Использование встроенных инструментов отладки в среде Visual Studio Code обеспечило детальный контроль за логикой выполнения кода, а ручное тестирование позволило выявить и устранить критические ошибки, повысив стабильность интерфейса и функционала.

Внедрение платформы способствует снижению времени обработки заказов, минимизации ошибок и повышению прозрачности товародвижения. Гибкая архитектура приложения обеспечивает масштабируемость и адаптивность под различные бизнес‑процессы, что делает его актуальным для малого и среднего бизнеса в сфере розничной торговли.

Таким образом, проект не только продемонстрировал практическую ценность автоматизации взаимодействия между магазинами и поставщиками, но и заложил основу для дальнейшего развития. Перспективными направлениями совершенствования являются расширение функционала, интеграция аналитических инструментов и улучшение пользовательского интерфейса. Итогом работы стало создание надёжного и современного программного продукта, способного повысить эффективность торговых процессов и укрепить взаимодействие между ключевыми участниками рынка.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Операционные системы [Электронный ресурс]. – Режим доступа: https://ru.wikibooks.org/wiki/Операционные\_системы. – Дата доступа: 25.10.2022.
2. Операционная система Android [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Android. – Дата доступа: 25.10.2022.
3. Язык программирования Java [Электронный ресурс]. – Режим доступа: https://web-creator.ru/articles/java. – Дата доступа: 25.10.2022.
4. Основы создания интерфейса [Электронный ресурс]. – Режим доступа: https://metanit.com/java/android/3.1.php. – Дата доступа: 25.10.2022.
5. Тестирование и отладка программного средства [Электронный ресурс]. – Режим доступа: https://ru.wikibooks.org/wiki/Операционные\_системы. – Дата доступа: 25.10.2022.
6. Филлипс Билл, Стюарт Крис, Марсикано Кристин, Гарднер Брайан. Программирование для профессионалов. 4-е издание. – СПб.: Питер, 2021. – 28 с.
7. Соловьев С.В., Гринкруг Л.С., Цой Р.И. Технология разработки прикладного программного обеспечения. – Москва: Академия естествознания, 2011. – 123 с.
8. Голощапов А. Google Android: программирование для мобильных устройств. – СПб.: БХВ-Петербург, 2010. – 448 с.
9. Коматинэни С., Маклин Д., Хэшими С. Google Android: программирование для мобильных устройств = Pro Android 2. – СПб.: Питер, 2011. – 736 с.
10. Сатия Коматинени, Дэйв Маклин. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов = Pro Android 4. – М.: Вильямс, 2014. – 880 с.

**ПРИЛОЖЕНИЕ А Листинг кода**

Код активности «Registration»

package com.example.registrationapp;

import android.os.Bundle;

import android.text.TextUtils;

import android.util.Patterns;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class RegistrationActivity extends AppCompatActivity {

private EditText editUsername;

private EditText editPassword;

private EditText editConfirmPassword;

private EditText editEmail;

private Button btnRegister;

private Button btnClear;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_registration);

// Инициализация элементов интерфейса

initViews();

// Обработка нажатия кнопки "Зарегистрироваться"

btnRegister.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

handleRegistration();

}

});

// Обработка нажатия кнопки "Очистить"

btnClear.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

clearFields();

}

});

}

private void initViews() {

editUsername = findViewById(R.id.editUsername);

editPassword = findViewById(R.id.editPassword);

editConfirmPassword = findViewById(R.id.editConfirmPassword);

editEmail = findViewById(R.id.editEmail);

btnRegister = findViewById(R.id.btnRegister);

btnClear = findViewById(R.id.btnClear);

}

private void handleRegistration() {

String username = editUsername.getText().toString().trim();

String password = editPassword.getText().toString().trim();

String confirmPassword = editConfirmPassword.getText().toString().trim();

String email = editEmail.getText().toString().trim();

if (!validateInput(username, password, confirmPassword, email)) {

return;

}

// Передача данных во внешний обработчик

RegistrationService.sendData(username, password, email);

Toast.makeText(this,

"Регистрация прошла успешно!\n" +

"Имя: " + username + "\n" +

"Email: " + email,

Toast.LENGTH\_LONG).show();

clearFields();

}

private boolean validateInput(String username, String password,

String confirmPassword, String email) {

if (TextUtils.isEmpty(username)) {

showError("Введите имя пользователя");

return false;

}

if (TextUtils.isEmpty(password)) {

showError("Введите пароль");

return false;

}

if (password.length() < 6) {

showError("Пароль должен содержать минимум 6 символов");

return false;

}

if (!password.equals(confirmPassword)) {

showError("Пароли не совпадают");

return false;

}

if (TextUtils.isEmpty(email)) {

showError("Введите email");

return false;

}

return true;

}

private void clearFields() {

editUsername.setText("");

editPassword.setText("");

editConfirmPassword.setText("");

editEmail.setText("");

}

private void showError(String message) {

Toast.makeText(this, message, Toast.LENGTH\_SHORT).show();

}

}

# **ПРИЛОЖЕНИЕ Б Карта пользователя**

