

ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 «Информационные системы и программирование»

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ТОРГОВОЙ ФИРМЫ
(ТОРГОВАЯ ФИРМА)

Пояснительная записка
к учебной практике
УП.23.09.02.07.606.01.ПЗ

Студент

«__» _____ 20__ г.

Михайлов. Д.В.

Руководитель

«__» _____ 20__ г.

Маюнова. А.Ю.

Томск 2023

СОДЕРЖАНИЕ

| | |
|---|----|
| Введение | 2 |
| 1 ОБЩАЯ ЧАСТЬ | 3 |
| 1.1 Цель разработки (Описание предметной области) | 3 |
| 1.2 Средства и среда разработки | 5 |
| 1.3 Описание языка программирования | 12 |
| 1.4 Теоретический раздел | 13 |
| 2 СПЕЦИАЛЬНАЯ ЧАСТЬ | 16 |
| 2.1 Постановка задачи | 16 |
| 2.2 Эскизный проект | 17 |
| 2.3 Технический проект | 20 |
| 2.4 Инструкция пользователя | 29 |
| ЗАКЛЮЧЕНИЕ | 43 |
| Перечень используемых источников | 44 |
| Приложение А. Листинг программы | 45 |
| Приложение Б. Результат работы программы | 72 |
| Приложение В. Тесты | 78 |

1. ОБЩАЯ ЧАСТЬ

1.1 Описание предметной области

Информационная система «Торговая фирма» специализируется на управлении процессами, связанными с покупками товаров для магазинов. В целом система представляет полный комплекс средств для эффективного управления процессами, связанными с покупками товаров для торговой фирмы. Система предлагает широкий спектр функциональности, включая управление закупками, управление товарной номенклатурой, мониторинг и отчётность, предоставление информации для оптимизации эффективности работы торговой фирмы. Все вводные данные вводятся в систему, а далее она автоматизирует процессы и генерирует аналитические отчёты.

В Информационной системе присутствует несколько ролей. Основные роли в этой системе — это администратор, менеджер по снабжению, аудитор, сотрудник продаж и менеджер для каждого магазина. Администратор отвечает за общую установку системы и её обновления то есть у него есть все права в системе, менеджер по снабжению занимается приобретением товаров для магазина и заносит их в систему, аудитор проверяет эффективность работы ИС и корректность данных, сотрудники продаж занимаются продвижением товаров, а менеджеры для каждого магазина анализируют актуальность условий поставки и наличие товаров.

В Информационной системе присутствует несколько функций,

- 1) Добавление товара в базу данных системы.
- 2) Добавить сотрудника.
- 3) Добавить магазин.
- 4) Обновить данные сотрудника.
- 5) Обновить данные магазина.
- 6) Добавление товара на склад магазина.
- 7) Удаление товара со склада магазина.
- 8) Купить товар.
- 9) Генерация формы возврата.

У Администратора есть доступ ко всем функциям панели администратора. А так же к функциям «Купить товар» и «Генерация формы возврата»

Менеджер по снабжению имеет доступ к функциям «Добавление товара на склад магазина», «Удаление товара со склада магазина», «Купить товар», «Удаление товара со склада магазина», «Купить товар», «Генерация формы возврата»

Аудитор имеет право просмотра всей информации о магазине где он работает.

Сотрудники продаж имеет доступ к функциям «Добавление товара на склад магазина», «Удаление товара со склада магазина», «Обновить данные магазина», «Купить товар», «Генерация формы возврата»

Менеджер имеет доступ к функциям «Добавление товара на склад магазина», «Удаление товара со склада магазина», «Купить товар», «Генерация формы возврата»

1.2 Средства и среда разработки.

Выделены и охарактеризованы основные этапы разработки программного обеспечения. Для каждого этапа приведены и описаны средства, которые могут быть применены для достижения целей этапа.

1. Основные средства, используемые на разных этапах разработки программ

В зависимости от предметной области и задач, поставленных перед разработчиками, разработка программ может представлять собой достаточно сложный, поэтапный процесс, в котором задействовано большое количество участников и разнообразных средств. Для того, чтобы определить, когда и в каких случаях какие средства применяются, выделим основные этапы разработки программного обеспечения. Наибольший интерес для проблематики рассматриваемого вопроса представляют следующие этапы разработки:

- 1) Проектирование приложения.
- 2) Реализация программного кода приложения.
- 3) Тестирование приложения.

Здесь сознательно опущены этапы, связанные с написанием технического задания, планирования сроков, бюджета и т.д. Причина этого заключается в том, что на данных этапах, за редким исключением, практически не используются специфические средства разработки.

1.1 Средства проектирования приложений

На этапе проектирования приложения в зависимости от сложности разрабатываемого программного продукта, напрямую зависящего от предъявляемых требований, выполняются следующие задачи проектирования:

- 1) Анализ требований.
- 2) Разработка пользовательских интерфейсов.

Блок-схемы (DrawIO).

Draw.io - это бесплатное приложение для создания диаграмм, блок-схем, графиков и других типов визуальных элементов. Некоторые из его преимуществ и недостатков:

Преимущества:

- 1) Бесплатность. Draw.io является бесплатным приложением и не требует регистрации.

2) Простота использования. Draw.io имеет интуитивно понятный интерфейс, который позволяет быстро создавать диаграммы и другие визуальные элементы.

3) Широкий спектр функций. Draw.io поддерживает множество различных типов диаграмм и графиков, а также имеет различные опции форматирования и стилизации элементов.

4) Интеграция с другими сервисами. Draw.io может быть интегрирован с другими сервисами, такими как Google Drive, Dropbox и Github.

5) Кроссплатформенность. Draw.io доступен для Windows, Mac, Linux, а также в виде онлайн-версии.

Недостатки:

1) Ограниченные возможности коллаборации. Draw.io не имеет полноценных функций коллаборации, что может затруднить работу в команде.

2) Недостаточная безопасность. Draw.io является онлайн-сервисом и не всегда обеспечивает должный уровень безопасности данных.

3) Ограниченные возможности экспорта. Draw.io имеет ограниченные возможности экспорта диаграмм в другие форматы, что может быть проблемой для некоторых пользователей.

Средства реализации программного кода

На этапе реализации программного кода выполняется кодирование отдельных компонент программы в соответствии с разработанным техническим проектом. Средства, которые могут быть применены, в значительной степени зависят от того, какие подходы были использованы во время проектирования и, кроме этого, от степени проработанности технического проекта. Тем не менее, среди средств разработки программного кода необходимо выделить следующие основные виды средств (в скобках приведены примеры средств): • методы и методики алгоритмирования.

Средство создания пользовательского интерфейса Avalonia UI

Avalonia UI - это кроссплатформенный фреймворк для создания графических пользовательских интерфейсов на .NET. Вот несколько плюсов и минусов этого фреймворка:

Плюсы:

1) Кроссплатформенность: Avalonia UI позволяет разрабатывать приложения, которые работают на нескольких операционных системах, включая Windows, MacOS, Linux и другие.

2) Мощный механизм разметки: Avalonia UI предлагает XAML-подобный язык разметки, который облегчает создание и настройку пользовательских интерфейсов.

3) Гибкость: Avalonia UI поддерживает различные стили и темы оформления, позволяя разработчикам легко настроить внешний вид своих приложений.

Минусы:

1) Небольшое сообщество: в настоящее время сообщество Avalonia UI не так велико, как у некоторых других фреймворков для разработки пользовательских интерфейсов. Это может означать, что поддержка и обновления могут быть менее частыми и медленнее.

2) Ограниченная документация: так как Avalonia UI является новым фреймворком, он имеет ограниченную документацию по сравнению с более популярными фреймворками.

3) Относительно медленная скорость компиляции: приложения, созданные с помощью Avalonia UI, могут иметь длительное время компиляции в сравнении с некоторыми другими фреймворками для создания пользовательских интерфейсов.

Средство получения исполняемого кода Rider.

Плюсы JetBrains Rider IDE:

1) Мощная и интуитивно понятная интегрированная среда разработки (IDE), которая поддерживает большое количество языков программирования, включая C#, F#, VB.NET, JavaScript, TypeScript и другие.

2) Обширный набор инструментов, которые помогают разработчикам быстро находить и исправлять ошибки, а также улучшать производительность и качество своего кода.

3) Интеграция с другими инструментами JetBrains, такими как ReSharper, dotTrace и dotMemory, что обеспечивает более эффективную работу с проектами.

Минусы JetBrains Rider IDE:

1) Высокая стоимость подписки на продукт, которая может быть недоступна для небольших команд или отдельных разработчиков.

2) Некоторые пользователи могут считать интерфейс среды разработки менее удобным, чем интерфейсы других IDE, таких как Visual Studio.

3) Некоторые функции IDE могут работать нестабильно или медленно в зависимости от настроек системы и используемого оборудования.

Средство управления базами PostgreSQL

PostgreSQL - это объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом. Она предоставляет широкий набор возможностей и функций, включая поддержку транзакций, многоверсионность, репликацию и многое другое.

Три плюса PostgreSQL:

- 1) Надежность: PostgreSQL считается одной из самых надежных СУБД на рынке. Она имеет встроенные механизмы защиты от сбоев и ошибок, в том числе транзакционную систему, которая гарантирует целостность данных.
- 2) Расширяемость: PostgreSQL имеет широкие возможности для расширения и настройки, позволяющие адаптировать ее под различные потребности бизнеса. В ней есть множество встроенных функций и возможностей для расширения функциональности, таких как написание хранимых процедур на языке программирования PL/Python.
- 3) Свободный и открытый исходный код: PostgreSQL является свободной и открытой СУБД, что означает, что она доступна для использования бесплатно и все ее исходные коды доступны для просмотра и изменения.

Три минуса PostgreSQL:

- 1) Требовательность к ресурсам: PostgreSQL потребляет большое количество ресурсов, включая оперативную память и процессорное время, особенно при работе с большими объемами данных. Это может привести к снижению производительности, если СУБД не настроена правильно.
- 2) Сложность настройки и управления: PostgreSQL может быть сложным в установке, настройке и управлении, особенно для новичков. Это может потребовать дополнительных затрат на обучение и поддержку.
- 3) Не очень популярна: PostgreSQL не так широко распространена, как, например, MySQL. Это может создавать проблемы с наймом разработчиков или настройкой инфраструктуры для работы с PostgreSQL.

отладчики .NET

Который позволяет отслеживать выполнение кода, устанавливая точки останова и анализировать стек вызовов.

Отладчик ".NET" в JetBrains Rider IDE - это инструмент для отладки приложений, написанных на платформе .NET. Рассмотрим три плюса и три минуса этого отладчика.

Плюсы:

- 1) **Расширенная отладка:** Отладчик ".NET" в JetBrains Rider IDE предоставляет широкий набор инструментов для отладки приложений, включая подробную информацию о переменных и стеке вызовов.
- 2) **Многопоточность:** Отладчик ".NET" в JetBrains Rider IDE обеспечивает поддержку отладки многопоточных приложений, что позволяет быстро находить и устранять ошибки в многопоточных приложениях.
- 3) **Интеграция с другими инструментами:** Отладчик ".NET" в JetBrains Rider IDE интегрируется с другими инструментами IDE, такими как Refactorings и Code Inspections, что облегчает работу разработчиков.

Минусы:

- 1) **Требовательность к ресурсам:** Отладчик ".NET" в JetBrains Rider IDE требует большое количество ресурсов, что может привести к замедлению работы компьютера.
- 2) **Сложность:** Отладка может быть сложной задачей, особенно для начинающих разработчиков. Использование отладчика ".NET" в JetBrains Rider IDE может требовать определенного уровня знаний и навыков.
- 3) **Ограничения:** Отладчик ".NET" в JetBrains Rider IDE имеет некоторые ограничения, например, он может работать только с приложениями, написанными на платформе .NET.

1.4 Средства тестирования программ

Тестирование информационных систем – процесс проверки соответствия между фактическим результатом работы функций системы и ожидаемым. Тестирование ИС является важным этапом производства ПО, направленным на детальное исследование программного кода и выявление ошибок в работе систем. В рамках данного проекта необходимо провести функциональное тестирование, которое проверяет программную систему на соответствие функциональным требованиям. Цель функционального тестирования состоит в том, чтобы проверить каждую функцию программного приложения, предоставляя соответствующий ввод, проверяя выход в соответствии с функциональными требованиями. Преимущества функционального тестирования

- 1) Функциональное тестирование приложений полностью имитирует фактическое использование системы;
- 2) Своевременно выявленные системные ошибки ПО помогают избежать множества проблем при работе с ним в будущем;
- 3) Значительная экономия и снижение рисков за счет исправления ошибок на более раннем этапе жизненного цикла ПО;

Тестирование будет проходить с использованием метода чёрного ящика.

Метод чёрного ящика - это функциональное и нефункциональное тестирование без доступа к внутренней структуре компонентов системы. Метод тестирования «чёрного ящика» – процедура получения и выбора тестовых случаев на основе анализа спецификации (функциональной или нефункциональной), компонентов или системы без ссылки на их внутреннее устройство.

Достоинства метода

1) Тестирование методом «чёрного ящика» позволяет найти ошибки, которые невозможно обнаружить методом «белого ящика». Простейший пример: разработчик забыл добавить какую-то функциональность. С точки зрения кода все работает идеально, но с точки зрения спецификации это – сверхкритичный баг.

2) «Чёрный ящик» позволяет быстро выявить ошибки в функциональных спецификациях (в них описаны не только входные значения, но и то, что мы должны в итоге получить). Если полученный при тестировании результат отличается от заявленного в спецификации, то у нас появляется повод для общения с аналитиком для уточнения конечного результата.

3) Тестировщику не нужна дополнительная квалификация. Часто мы пользуемся различными сервисами и приложениями, не очень в них разбираясь. Для того, чтобы открыть инстаграм и обработать свою фотографию, нам совсем не нужно знать способ реализации фильтров. Мы хотим открыть фотографию, выбрать фильтр и получить красивую картинку на выходе. Задача тестировщика, который тестирует эту функцию в инстаграм, – убедиться, что пользователь получит эту самую красивую картинку в соответствии с выбранным фильтром. При этом нам совсем не обязательно иметь какую-либо специализацию – нужны лишь телефон и инстаграм.

4) Тестирование проходит «с позиции пользователя». Пользователь всегда прав, он конечный потребитель практически любого ПО, а значит, ему должно быть удобно, комфортно и понятно.

5) Составлять тест-кейсы можно сразу после подготовки спецификации. Это значительно сокращает время на тестирование: к тому моменту, как продукт готов к тестированию, тест-кейсы уже разработаны, и тестировщик может сразу приступить к проверке.

Недостатки метода

1) Основным недостатком метода «чёрного ящика» является возможность пропуска границ и переходов, которые не очевидны из спецификации, но есть в реализации кода (собственно, это и заставляет тестировщиков использовать метод «белого ящика»). Вспоминается случай, когда система получала котировки валют с биржи Forex и округляла до 3 знаков после запятой. Система успешно прошла тестирование методом

«черного ящика» (так как ни одна валюта не выходила за соответствующие границы) и хорошо работала до тех пор, пока курс доллара к биткоин не вышел за границы 1000 долларов. Тестирование «белым ящиком» выявило бы ошибку: специалист увидел бы, что коэффициент конверсии валюты ограничен 3 знаками.

2) Можно протестировать только небольшое количество возможных входных (входящих) значений; многие варианты остаются без проверки.

3) Тесты могут быть избыточными, если разработчик уже проверил данную функциональность (например, Unit-тестом).

4) При отсутствии четкой и полной спецификации проектировать тесты и тест-сценарии оказывается затруднительно.

1.3 Описание языка программирования

C# является объектно-ориентированным языком программирования, используется мною в моих проектах для создания приложений на платформе .NET. Он имеет синтаксис, который легко читать и понимать.

Одним из его преимуществ является мощная система типов, что позволяет разработчикам создавать более надежные и безопасные приложения. Также он имеет большую библиотеку классов, что упрощает работу с различными задачами, такими как работа с файлами, сетевое взаимодействие и многое другое.

Еще одним плюсом C# является его интеграция с другими технологиями, такими как WPF, ASP.NET и Xamarin, что позволяет разрабатывать различные типы приложений на разных платформах.

Среди минусов можно отметить то, что некоторые задачи могут потребовать большого количества кода, что увеличивает время разработки. Также некоторые разработчики могут считать его менее гибким, чем некоторые другие языки программирования. И, наконец, из-за его популярности он может стать мишенью для злоумышленников, что может увеличить риск взлома приложений.

1.6 Теоретический раздел

Этот раздел должен включать:

Список терминов и их определения.

- 1) Пользователи - люди, которые зарегистрировались в системе магазина и имеют возможность покупать товары, просматривать информацию о товарах и возвращать товары.
- 2) Магазины - места, где продаются товары, зарегистрированные в системе, и где пользователи могут осуществлять покупки.
- 3) Товары в определенном магазине - продукты, которые доступны для покупки в определенном магазине.
- 4) Проданные товары в определенном магазине - продукты, которые были проданы в определенном магазине.
- 5) Продажи - количество проданных продуктов в определенный период времени.
- 6) Возвраты - процесс, при котором пользователи возвращают товары, которые им не подходят, или которые имели дефекты.
- 7) Возвраты в определённом магазине – процесс, при котором происходит возврата товара в магазин, где ранее он бы куплен.
- 8) Продукты - товары, которые могут быть куплены в магазине.
- 9) Поставки в определенном магазине - процесс поставки продуктов в определенный магазин.
- 10) Поставки - процесс доставки продуктов в магазины из внешнего источника.
- 11) Фирма — это обособленная специализированная организация, основанием которой является профессионально организованный трудовой коллектив, способный с помощью имеющихся в его распоряжении средств производства изготавливать нужную потребителям продукцию (выполнять работы, оказывать услуги) соответствующего назначения, профиля и ассортимента.
- 12) Торговая фирма - организация, обеспечивающая продажу и доставку материальных ценностей на основе юридически оформленного соглашения с указанием коммерческой ответственности за отклонение в сроках поставки и количестве единиц заказа

Требования.

Справочно-информационная система Торговой фирмы должна удовлетворять следующим требованиям:

Добавление товара на склад магазина:

- 1) Система должна проверять, есть ли товар в базе данных системы.
- 2) Система должна проверять, существует ли магазин в базе данных системы.
- 3) Система должна проверять, есть ли данный товар уже на складе магазина.
- 4) Система должна запрашивать количество товара, которое необходимо добавить на склад магазина.
- 5) Система должна обновлять данные о количестве товара на складе магазина в базе данных системы.

Удаление товара со склада магазина:

- 1) Система должна проверять, есть ли товар в базе данных системы.
- 2) Система должна проверять, существует ли магазин в базе данных системы.
- 3) Система должна проверять, есть ли данный товар на складе магазина.

Купить товар:

- 1) Система должна проверять, есть ли товар в базе данных системы.
- 2) Система должна проверять, существует ли магазин в базе данных системы.
- 3) Система должна проверять, есть ли данный товар на складе магазина в достаточном количестве для продажи.
- 4) Система должна запрашивать количество товара, которое необходимо купить.
- 5) Система должна обновлять данные о количестве товара на складе магазина в базе данных системы.
- 6) Система должна создавать новую запись о продаже товара в базе данных системы.

Генерация формы возврата:

- 1) Система должна проверять, есть ли товар в базе данных системы.
- 2) Система должна проверять, был ли данный товар продан в определенном магазине.
- 3) Система должна запрашивать информацию о покупке данного товара (дата покупки, название магазина, количество купленного товара).

- 4) Система должна создавать новую запись о возврате товара в базе

2 СПЕЦИАЛЬНАЯ ЧАСТЬ

2.1 Постановка задачи

Исходя из поставленной цели, необходимо выполнить следующие задачи:

- 1) Разработать функцию «Добавления товара в базу данных системы»
- 2) Разработать функцию «Добавить сотрудника»
- 3) Разработать функцию «Добавить магазин»
- 4) Разработать функцию «Обновить данные сотрудника»
- 5) Разработать функцию «Обновить данные магазина»
- 6) Разработать функцию «Добавление товара на склад магазина»
- 7) Разработать функцию «Удаление товара со склада магазина»
- 8) Разработать функцию «Купить товар»
- 9) Разработать функцию «Генерация формы возврата»

2.2 Эскизный проект

Функция «Добавление товара на склад магазина».

Входные данные: данные о магазине, продукте, количестве и наконец о пользователе.

Выходные данные: добавляется новая запись в таблицу товаров в определённом магазине.

Описание функции: Функция на вход принимает данные магазина, товара, пользователя, и количество для добавления. Далее функция проверяет есть ли на складе магазина этот товар, если нет, то добавляет новый товар, если есть, то добавит к уже существующему количеству.

Функция "Удаление товара со склада магазина".

Входные данные: данные о магазине и товаре

Выходные данные: Нет.

Описание функции: Функция на вход принимает данные магазина и товара, который нужно удалить со склада. Далее функция проверяет, есть ли такой товар на складе данного магазина. Если товар найден, то он удаляется из базы данных, и пользователь получает уведомление об успешном удалении товара. Также функция обновляет информацию о складе магазина, чтобы отобразить текущее количество товаров после удаления. Если товар не найден на складе, то пользователь получает уведомление о неудачной попытке удаления.

Функция "Купить товар".

Входные данные: данные о продукте, магазине, пользователе и количестве

Выходные данные: обновляется количество товара в магазине.

Описание функции: Функция сначала проверяет существование магазина и продукта. Затем она проверяет, что в магазине достаточно товара для продажи. Если проверки пройдены успешно, то функция создает объект продажи и сохраняет его в базе данных. Затем функция создает объект проданного товара и сохраняет его в базе данных. После этого функция уменьшает количество товара в магазине на величину продажи. Функция не возвращает никаких значений.

Функция "Генерация формы возврата".

Входные данные: данные о продукте, магазине, пользователе

Выходные данные: добавятся данные в таблицы возвращённого товара и возврата.

Описание функции: на вход функции передаются соответствующие данные затем функция ищет магазин и продукт в базе данных. Если они не найдены, то выбрасывается исключение с сообщением об ошибке. Затем

функция ищет продажи этого пользователя в данном магазине. Если они не найдены, то выбрасывается исключение с сообщением об ошибке.

Далее функция ищет информацию о купленном товаре. Если этот товар не был найден, то выбрасывается исключение с сообщением об ошибке. Если количество возвращаемого товара больше, чем было куплено, то выбрасывается исключение с сообщением об ошибке.

Затем функция создает объект возврата и добавляет его в базу данных. Затем создается объект возвращенного товара и добавляется в базу данных. Далее функция ищет информацию о товаре в конкретном магазине. Если информация не найдена, то она создается и добавляется в базу данных.

Количество возвращаемого товара добавляется к общему количеству товара в магазине, а количество проданного товара уменьшается на количество возвращаемого товара. Наконец, функция сохраняет изменения в базе данных.

Функция "Добавление товара в базу данных системы".

Входные данные: данные о магазине, продукте, пользователе и количестве.

Выходные данные: Добавление новой записи в таблицу Товаров или обновление существующей.

Описание функции: на вход функции передаются соответствующие данные затем функция проверяет наличие записи в таблице продуктов в магазине для данного магазина и продукта. Если запись существует, то функция увеличивает количество продуктов, если записи не существует, то функция создает новую запись в таблице продуктов в магазине.

Затем функция создает записи в таблицах поставок для отслеживания поставок товаров. В таблице поставки создается новая запись с указанием идентификатора магазина, идентификатора пользователя, который добавил продукт, и даты добавления. В таблице поставленные продукты создается новая запись с указанием идентификатора поставки (полученного из таблицы поставок).

В конце функция сохраняет все изменения в базе данных.

Функция "Добавить сотрудника".

Входные данные: логин, пароль и роль.

Выходные данные: Сотрудник добавлен в базу данных.

Описание функции: Функция получает список всех пользователей в базе данных. Затем она находит объект роли в таблице ролей по переданным данным.

Далее создается новый объект пользователя с переданным именем, паролем и найденным объектом роли. Этот новый объект добавляется в список пользователей, а затем в базу данных.

Функция "Добавить магазин".

Входные данные: название магазина, местоположение магазина и данные пользователя, который является менеджером магазина.

Выходные данные: Магазин добавлен в базу данных.

Описание функции: Функция сначала создает новый экземпляр контекста базы данных и загружает список всех магазинов в переменную.

Затем она находит пользователя в таблице пользователей.

Если все проверки пройдены успешно, функция создает новый экземпляр магазина с помощью переданных параметров и менеджера, затем добавляет этот магазин в список и сохраняет его в базе данных.

Если же один или несколько параметров являются неправильными, функция выводит сообщение об ошибке в консоль.

Функция "Обновить данные сотрудника".

Входные данные: данные пользователя

Выходные данные: данные были изменены если пользователь был найден, и выводит сообщение пользователю.

Описание функции: сначала на вход принимаются данные пользователя, и функция создаёт новый объект базы данных и получает список всех пользователей. Затем она находит пользователя в базе данных.

Если пользователь найден, функция обновляет информацию о пользователе, заменяя его логин, пароль и роль на новые. В противном случае, она выдает сообщение об ошибке, указывая на то, что что-то не так с заполненными полями.

Наконец, функция сохраняет изменения в базе данных и выводит сообщение о том, что данные пользователя были изменены, если все прошло успешно.

Функция "Обновить данные магазина".

Входные данные: данные магазина, менеджера, которого нужно назначить для магазина, новое имя магазина и новое местоположение магазина.

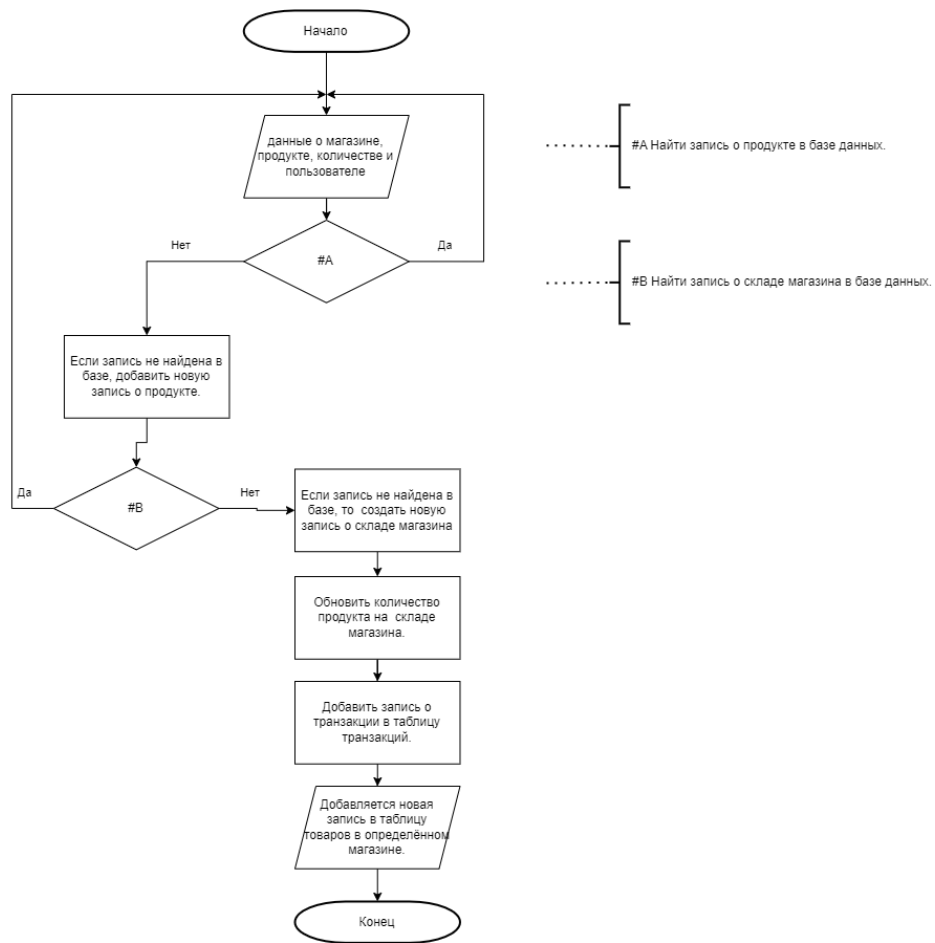
Выходные данные: Данные магазина были изменены.

Описание функции: внутри функции создается экземпляр контекста данных, после чего производится поиск менеджера магазина и самого магазина в базе данных. Если менеджер и магазин найдены, то данные обновляются с помощью присваивания новых значений переданных параметров. В конце функции происходит сохранение изменений в базу данных.

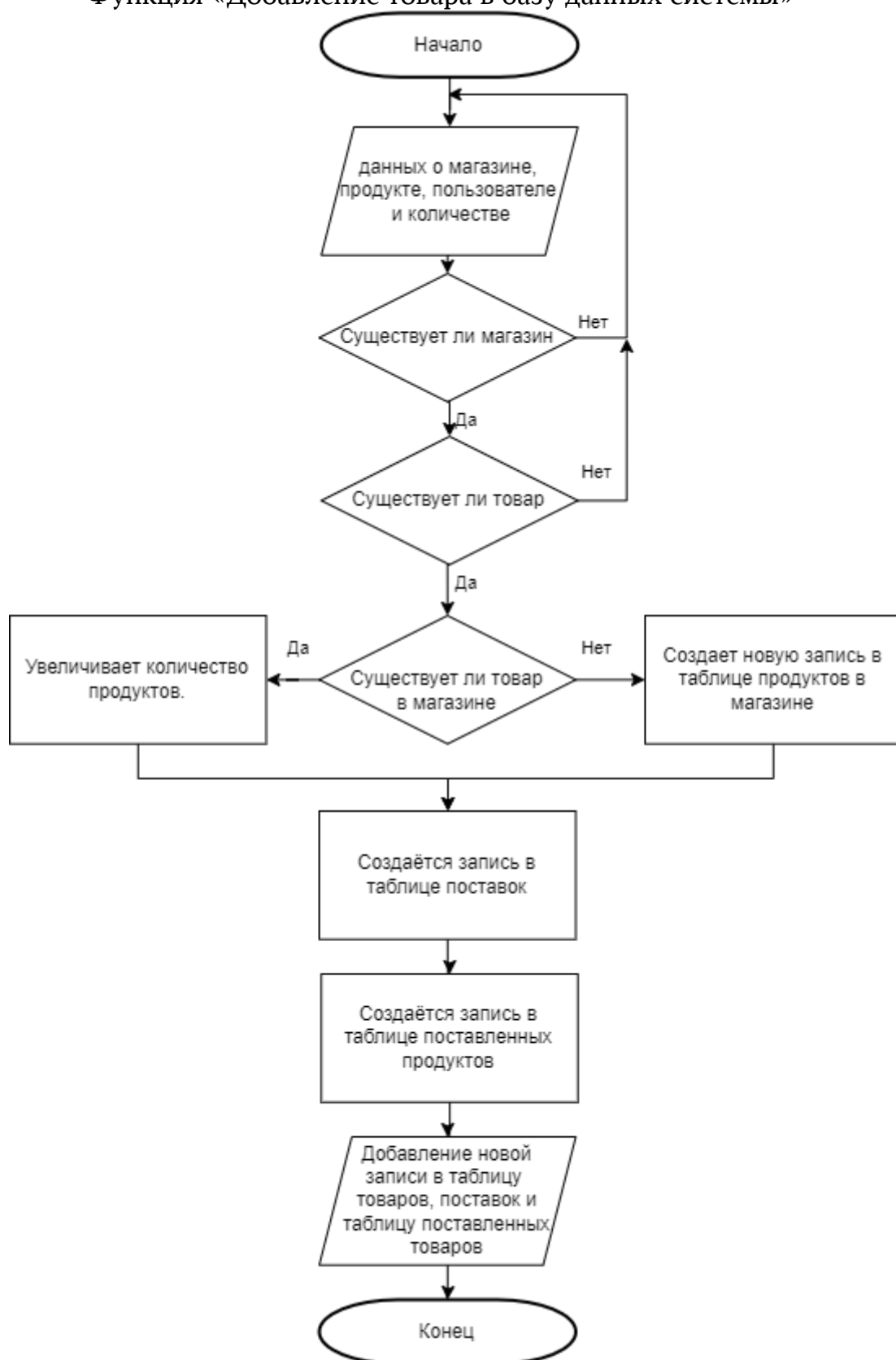
Функция позволяет обновлять информацию о магазинах в базе данных, изменяя имя, местоположение и назначая нового менеджера.

2.3 Технический проект

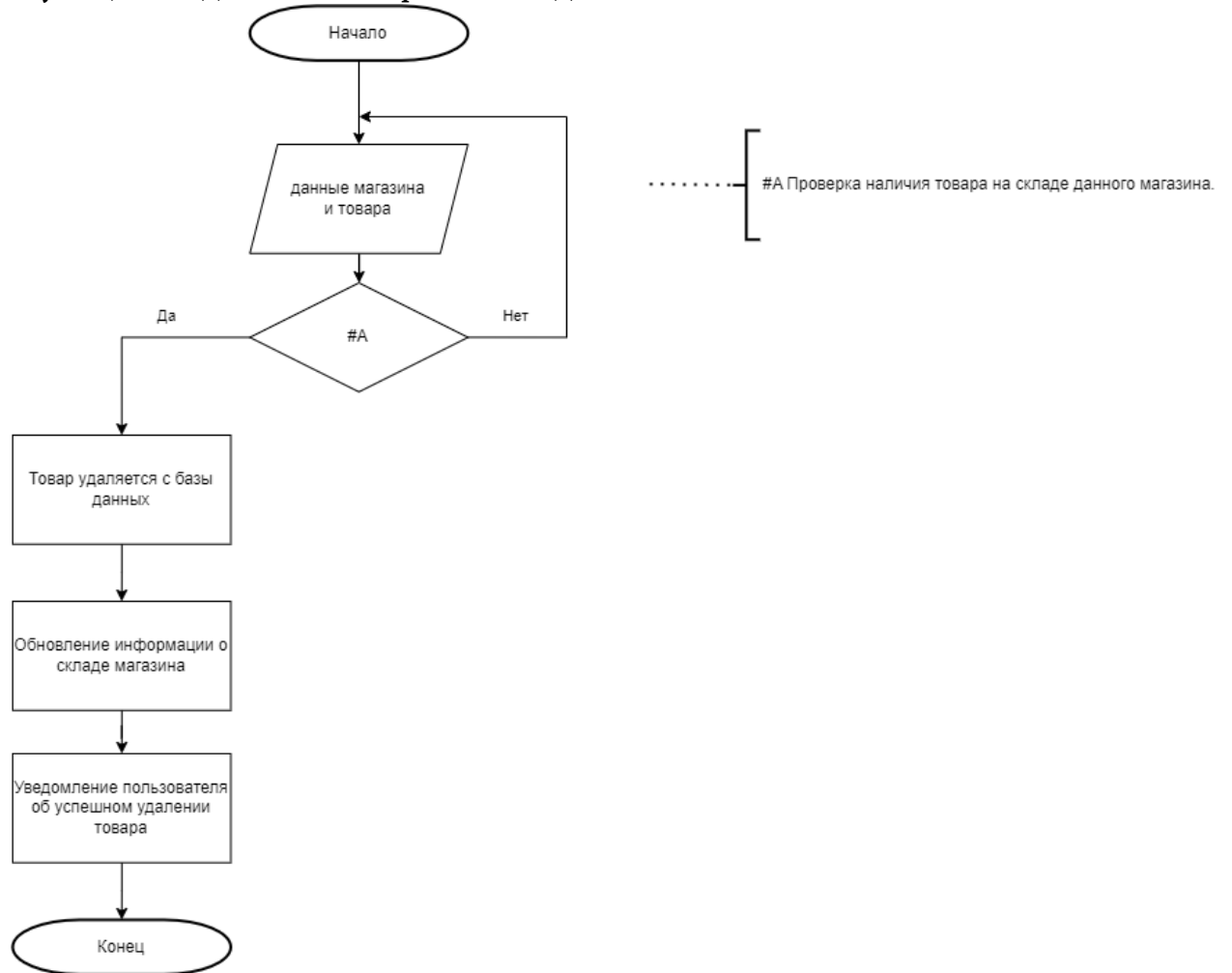
Функция «Добавление товара на склад магазина»



Функция «Добавление товара в базу данных системы»



Функция «Удаление товара со склада магазина»



Функция «Купить товар»

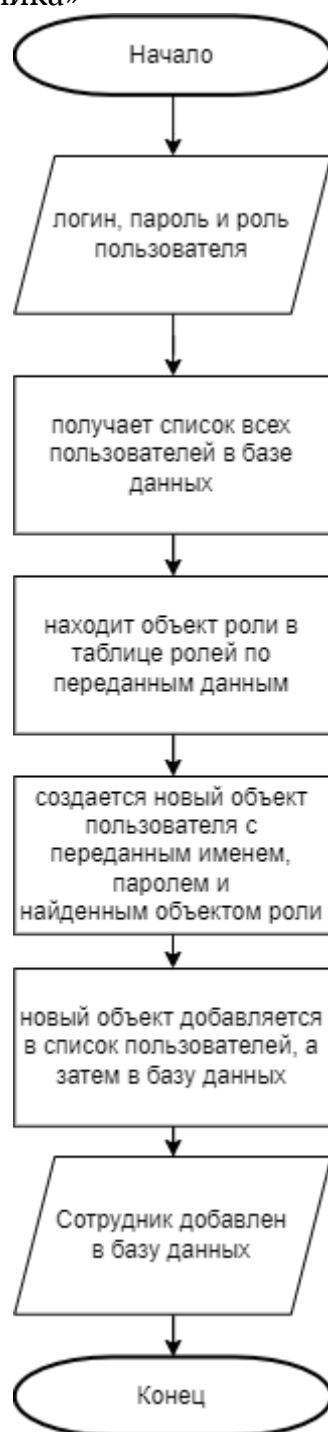


..... { #А Проверить наличие достаточного количества товара в магазине

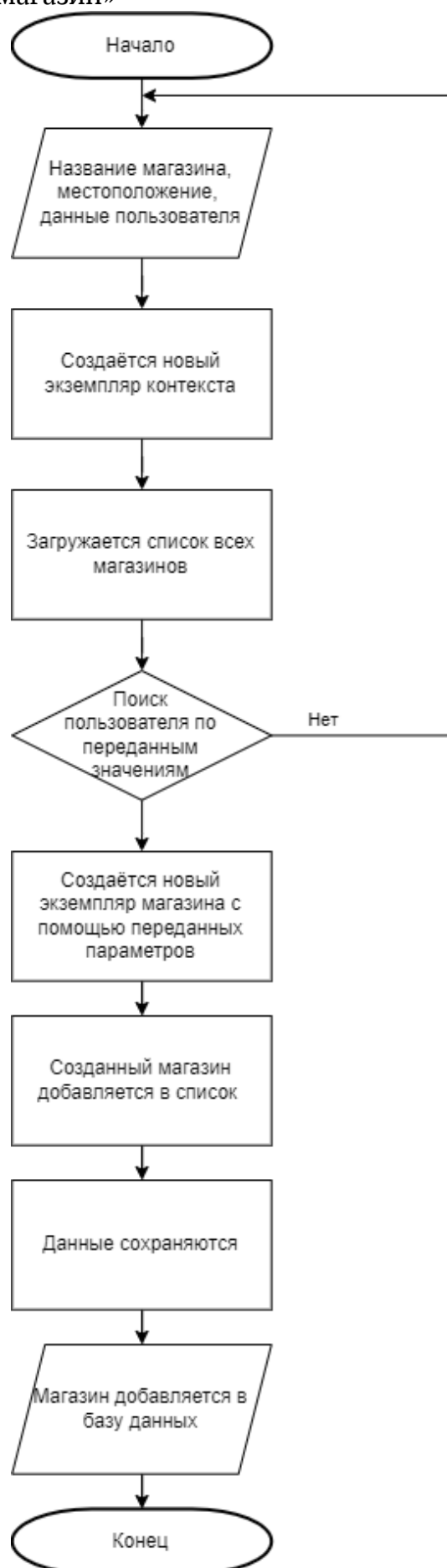
Функция «Генерация формы возврата»



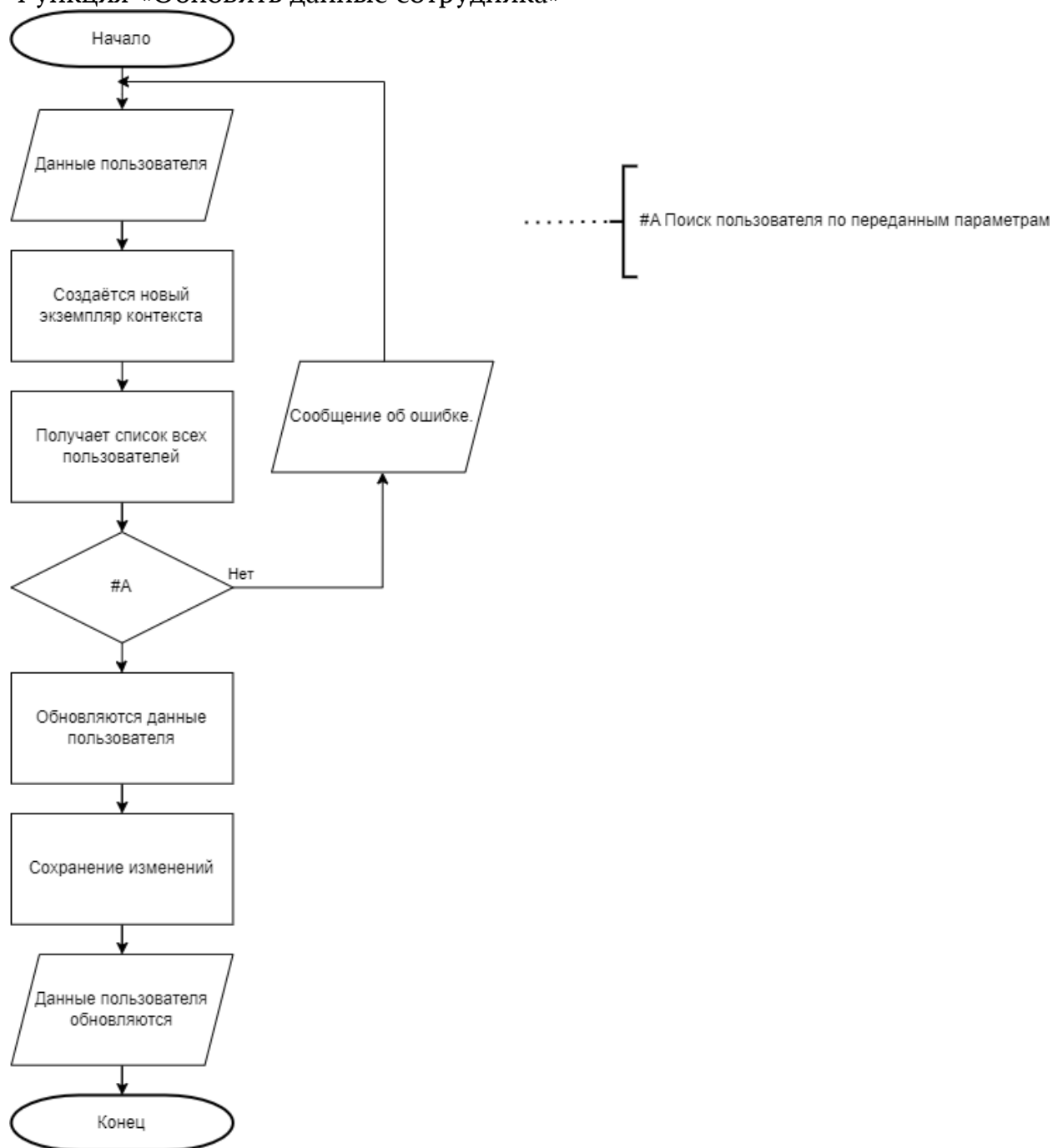
Функция «Добавить сотрудника»



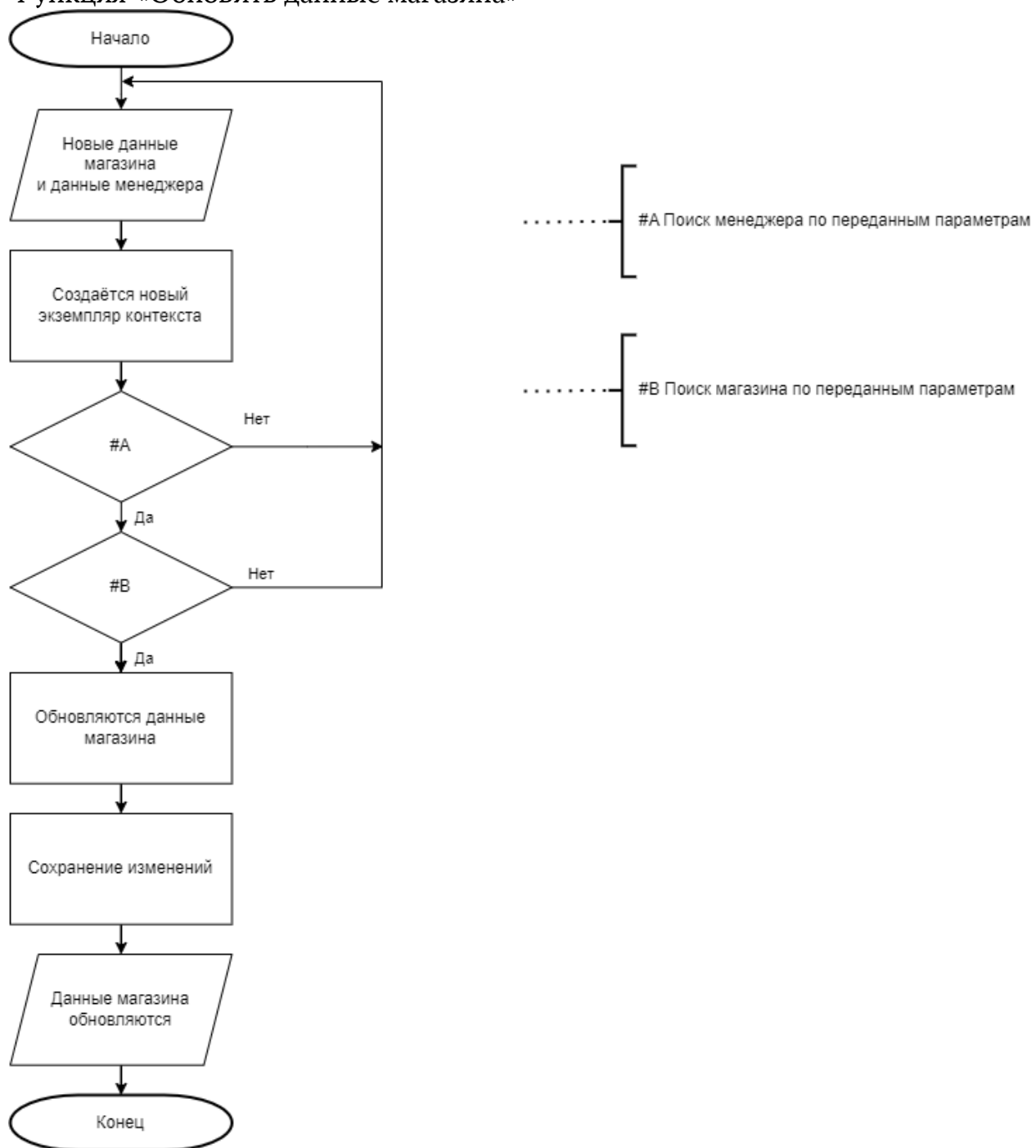
Функция «Добавить магазин»



Функция «Обновить данные сотрудника»



Функция «Обновить данные магазина»



2.4 Инструкция пользователя

В данном разделе приводится описание всех операций, существующих в Торговую Фирму.

Таблица 1 - Роли и права доступа к данным и операциям

| Роль | Доступные пункты меню | Доступные операции |
|-----------------------|---------------------------------|---|
| Администратор | Панель администратора | Добавление товара в базу данных системы Добавить сотрудника Добавить магазин Обновить данные сотрудника Обновить данные магазина |
| | Меню | Купить товар Генерация формы возврата |
| Менеджер по снабжению | Меню | Добавление товара на склад магазина Удаление товара со склада магазина Купить товар Генерация формы возврата |
| Аудитор | Создание отчёта склада магазина | Создание отчёта склада магазина |
| | Меню | Купить товар Генерация формы возврата |
| Сотрудники продаж | Меню | Добавление товара на склад магазина Удаление товара со склада магазина Обновить данные магазина Купить товар Генерация формы возврата |
| Менеджер | Меню | Добавление товара на склад магазина |

Продолжение таблицы 1

| | | |
|--|--|--|
| | | Удаление товара со склада магазина Купить товар Генерация формы возврата |
|--|--|--|

Вход в приложение Торговой фирмы

Для входа в информационную систему Торговой фирмы необходимо запустить программу, после запуска откроется окно Авторизация (Рисунок 2.4.1).

Рисунок 2.4.1 - Авторизация в Торговой фирме

Для получения прав доступа к информационной системе Торговой фирме необходимо предварительно обратиться к администратору Торговой фирмы.

Для доступа к данным необходимо выполнить авторизацию для этого необходимо заполнить поля логин и пароль после нажать кнопку «Вход».

Регистрация пользователей в системе.

Регистрация пользователей системы доступна пользователю с ролью Администратор. Администратор имеет право создать аккаунт пользователя, и указать роль этому пользователю (администратор, менеджер по снабжению, аудитор, сотрудник продаж и менеджер для каждого магазина).

Для создания аккаунта необходимо открыть Панель администратора и выбрать пункт «Добавить сотрудника» (Рисунок 2.4.2)

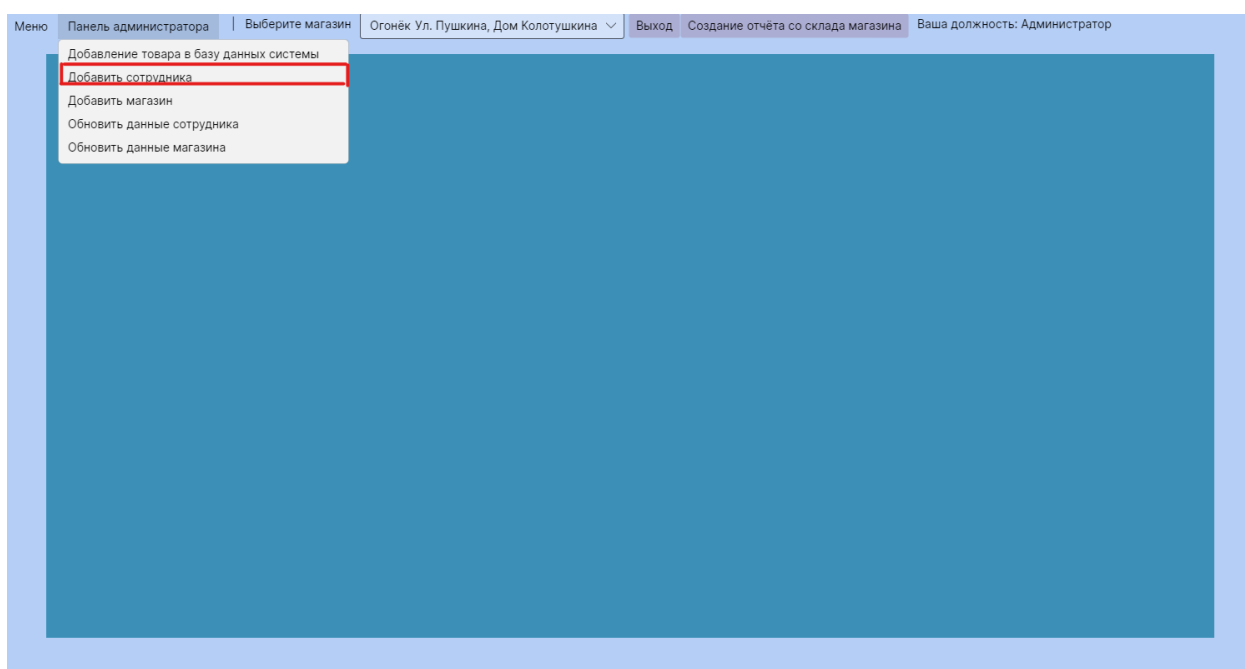


Рисунок 2.4.2 – Добавление сотрудника.

В открывшемся окне (Рисунок 2.4.3) необходимо заполнить поля (Логин; Пароль; Ид роли).

Создания аккаунта

Логин

Пароль

Ид Роли.

Создать аккаунт Отмена

// Роли:
// Администратор 1,
// Менеджер по снабжению 2,
// Аудитор 3,
// Сотрудник продаж 4,
// Менеджер для каждого магазина 5

Рисунок 2.4.3 – Добавить сотрудника.

Далее остаётся нажать кнопку «Создать аккаунт». Если создание ещё одного аккаунта не требуется, или создание аккаунта уже не актуально, то можно нажать кнопку «Отмена».

Добавление товара в базу данных системы

Для добавления товара в базу данных системы необходимо на вкладке Панели администратора выбрать пункт «Добавление товара в базу данных системы» (Рисунок 2.4.4)

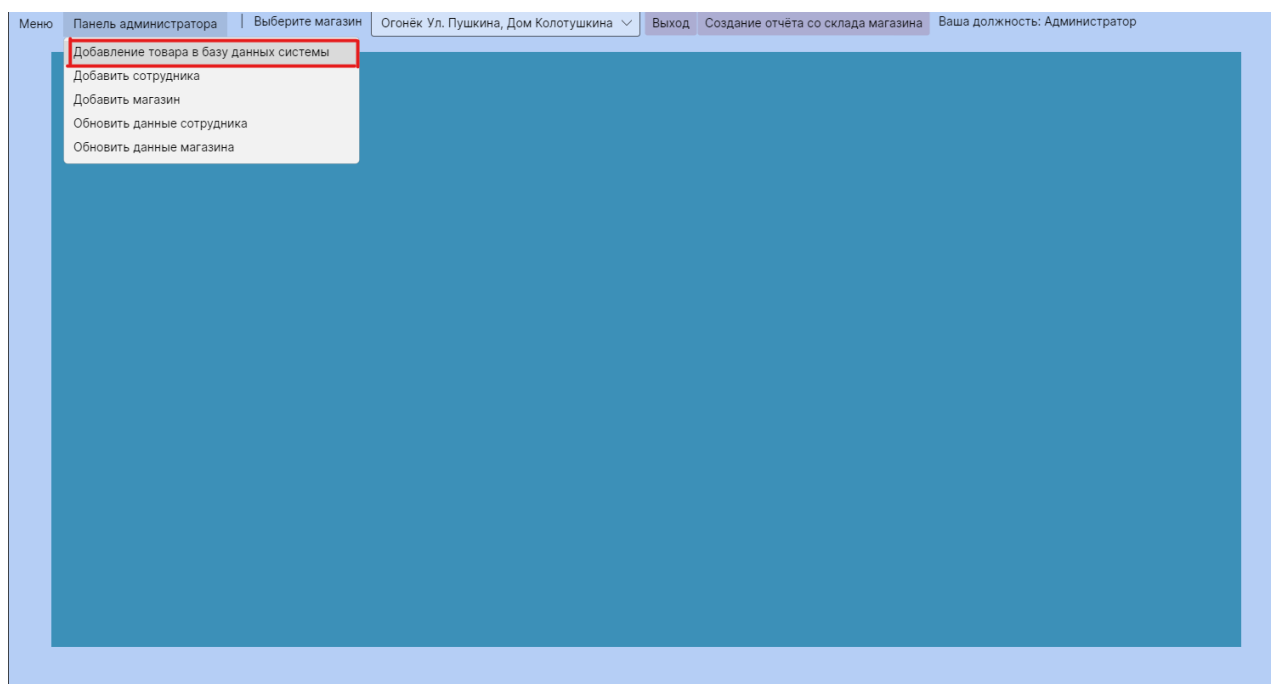


Рисунок 2.4.4 - Добавление товара в базу данных системы

В открывшемся окне (Рисунок 2.4.5) необходимо заполнить поля (Название продукта; Описание продукта; Стоимость продукта; Количество продукта)

The screenshot displays a form titled 'Добавление продукта' (Add product) within a blue-themed interface. The form consists of four input fields, each with a label on the left: 'Название продукта' (Product name), 'Описание продукта' (Product description), 'Стоимость продукта' (Product cost), and 'Количество продукта' (Product quantity). Below these fields, there are two buttons: 'Создать продукт' (Create product) and 'Отмена' (Cancel). The form is set against a solid blue background.

Рисунок 2.4.5 - Добавление товара в базу данных системы

Далее необходимо нажать кнопку «Создать продукт». Если создание ещё одного продукта не требуется, или создание продукта уже не актуально, то можно нажать кнопку «Отмена».

Добавить магазин

Для добавления магазина в базу данных системы необходимо открыть Панель администратора и выбрать пункт «Добавить магазин» (Рисунок 2.6)

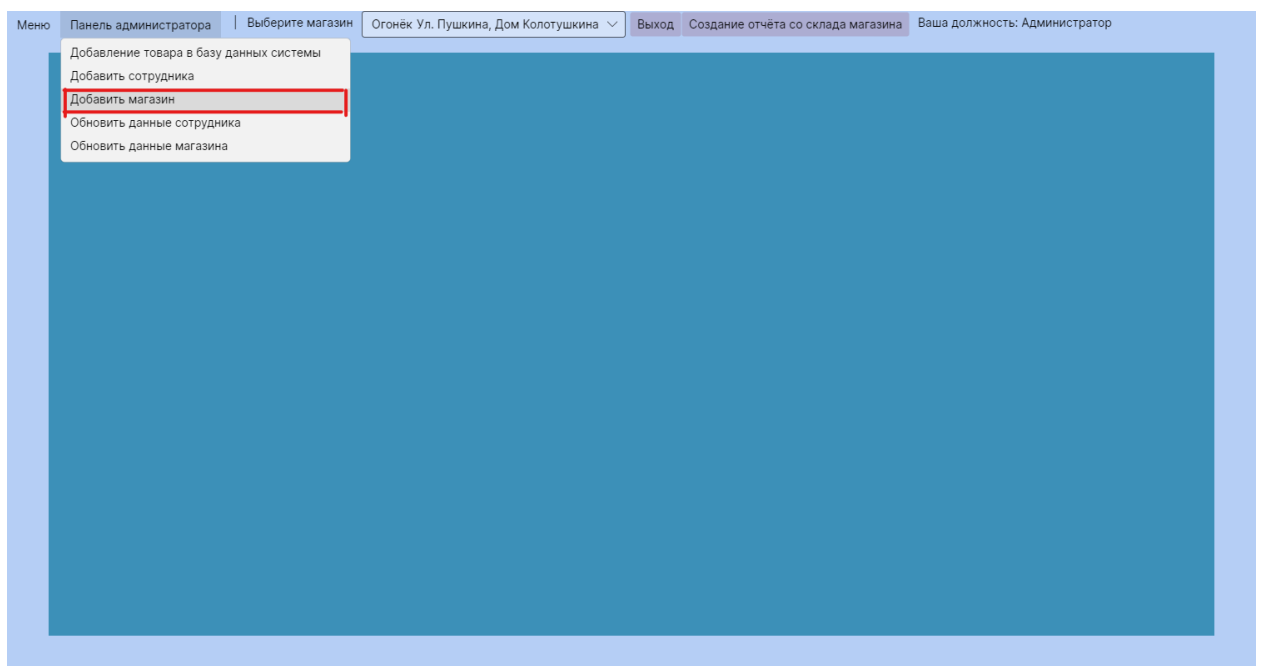


Рисунок 2.4.6 – Добавление магазина

В открывшемся окне (Рисунок 2.4.7) необходимо заполнить поля (Название магазина; Адрес магазина; Менеджер магазина)

Добавление продукта

Название магазина

Адрес магазина

Менеджер магазина

Создать магазин

Отмена

Рисунок 2.4.7 - Добавление магазина

Далее остаётся нажать кнопку «Создать магазин». Если создание ещё одного магазина не требуется, или создание магазина уже не актуально, то можно нажать кнопку «Отмена».

Обновить данные сотрудника

Для обновления данных сотрудника системы необходимо открыть Панель администратора и выбрать пункт «Обновить данные сотрудника» (Рисунок 2.4.8)

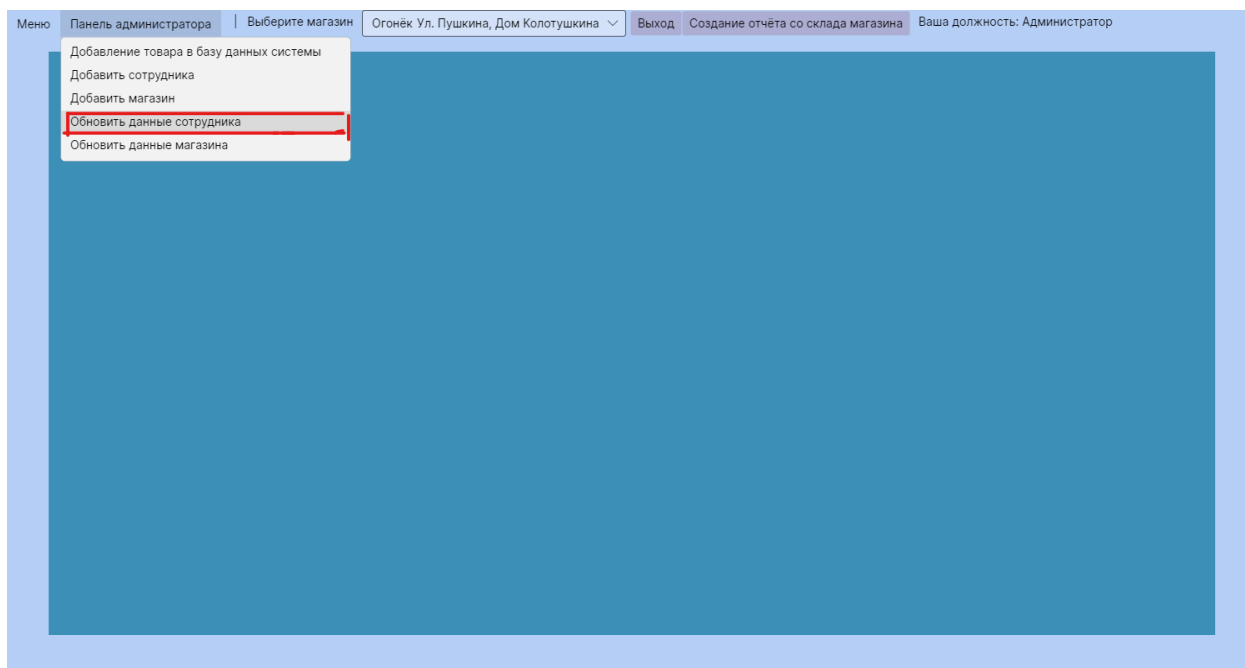


Рисунок 2.4.8 - Обновить данные сотрудника

В открывшемся окне (Рисунок 2.4.9) необходимо заполнить поля (Пользователь; Логин; Пароль; Роль)

The screenshot displays a form titled 'Обновление аккаунта' (Update account). On the left side, there are four labels with corresponding input fields: 'Пользователь' (User) with a dropdown arrow, 'Логин' (Login) with a text input field, 'Пароль' (Password) with a text input field, and 'Роль' (Role) with a dropdown arrow. On the right side of the form, there are two buttons: 'Обновить данные' (Update data) and 'Отмена' (Cancel).

Рисунок 2.4.9- Обновить данные сотрудника.

Далее остаётся нажать кнопку «Обновить данные». Если обновления данных не требуется, или обновления данных уже не актуально, то можно нажать кнопку «Отмена».

Обновить данные сотрудника

Для обновления данных магазина в системе необходимо открыть Панель администратора и выбрать пункт «Обновить данные магазина» (Рисунок 2.4.10)

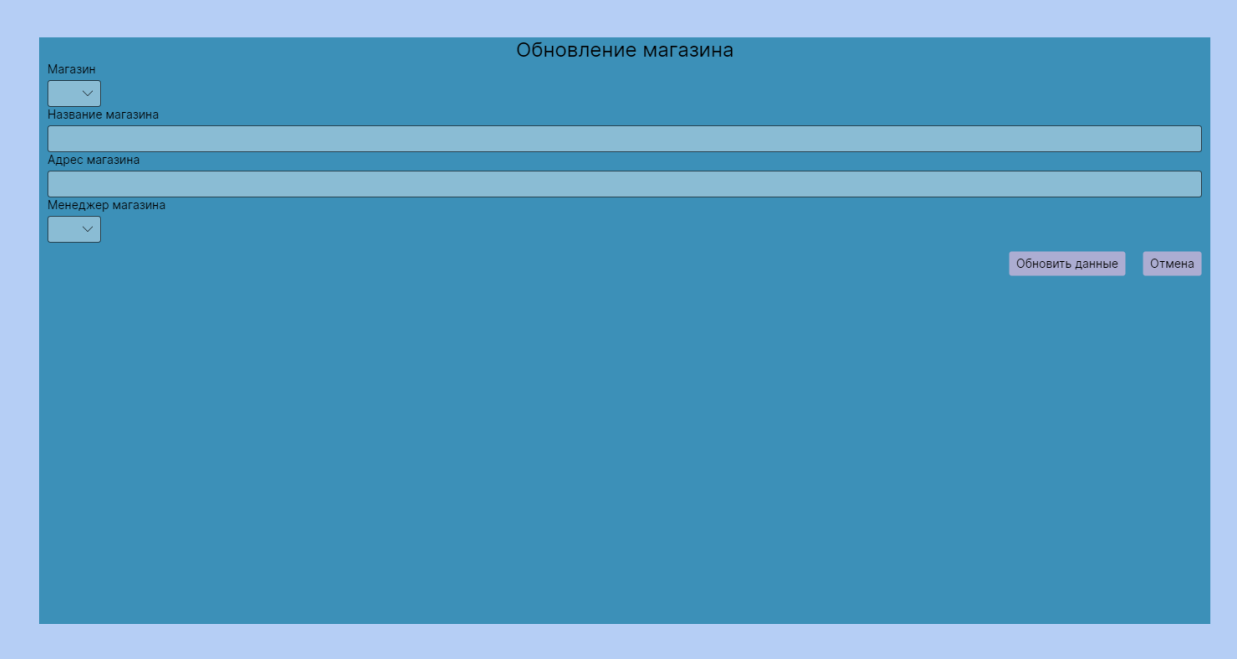


Рисунок 2.4.10 - Обновить данные сотрудника.

Далее остаётся нажать кнопку «Обновить данные». Если обновления данных не требуется, или обновления данных уже не актуально, то можно нажать кнопку «Отмена».

Добавление товара на склад магазина

Для добавления товара на склад магазина в системе необходимо открыть Меню и выбрать пункт «Добавление товара на склад магазина» (Рисунок 2.4.11)

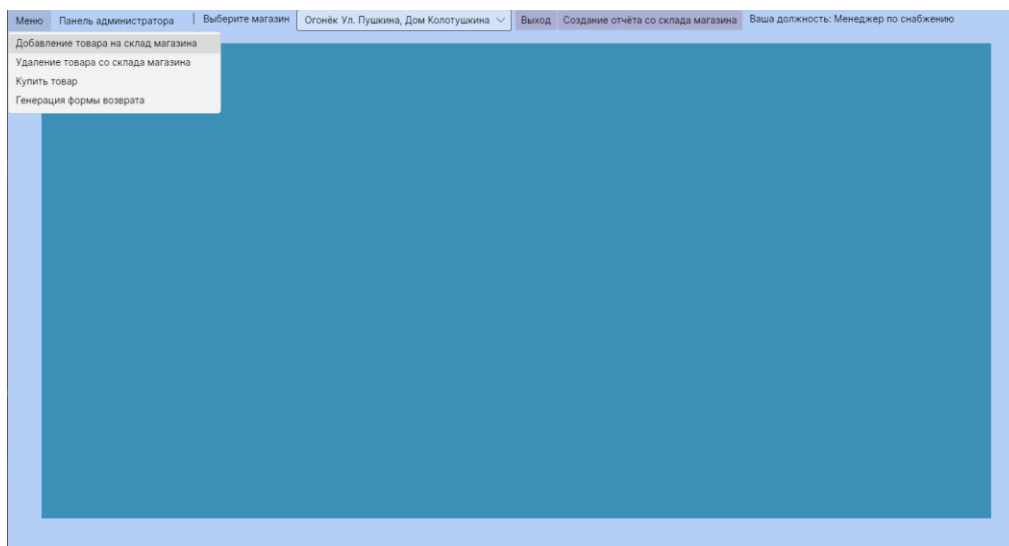


Рисунок 2.4.11 - Добавление товара на склад магазина
В открывшемся окне (Рисунок 2.4.12) необходимо заполнить поля (Продукт; Количество)

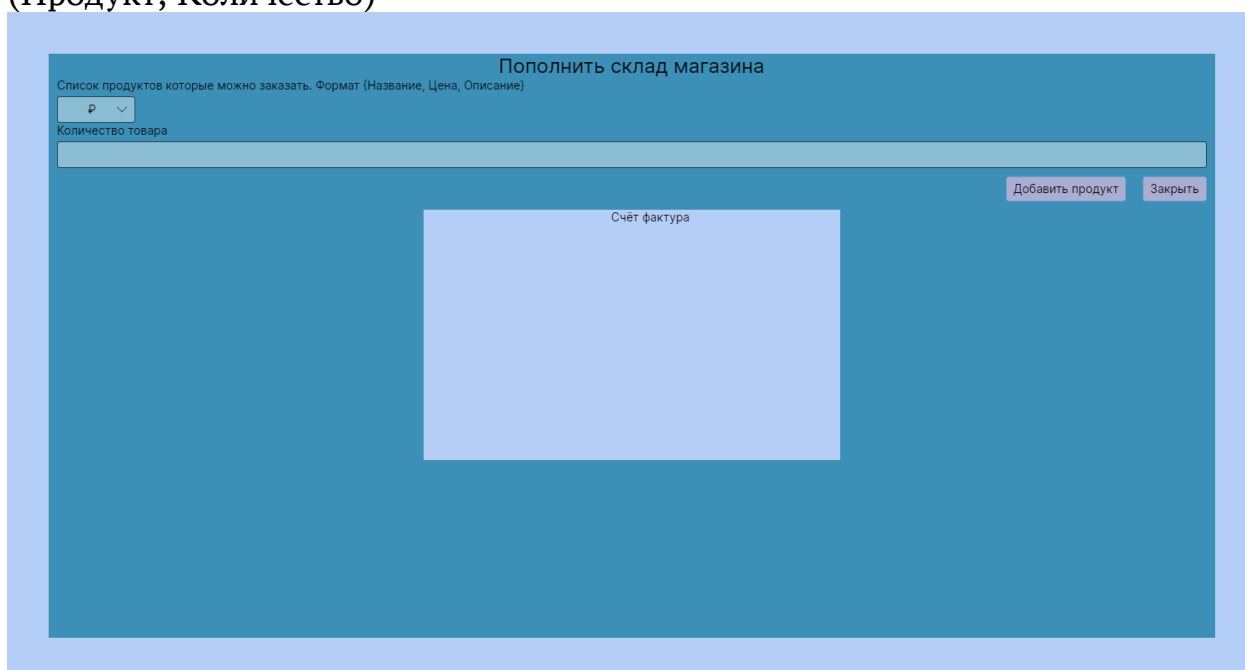


Рисунок 2.4.12 - Добавление товара на склад магазина

Далее остаётся нажать кнопку «Добавить продукт». Если добавление продукта не требуется, или добавление продукта уже не актуально, то можно нажать кнопку «Заккрыть».

Удаление товара со склада магазина
Для того что бы удалить товар необходимо открыть Меню и выбрать пункт «Удаление товара со склада магазина» (Рисунок 2.4.13)

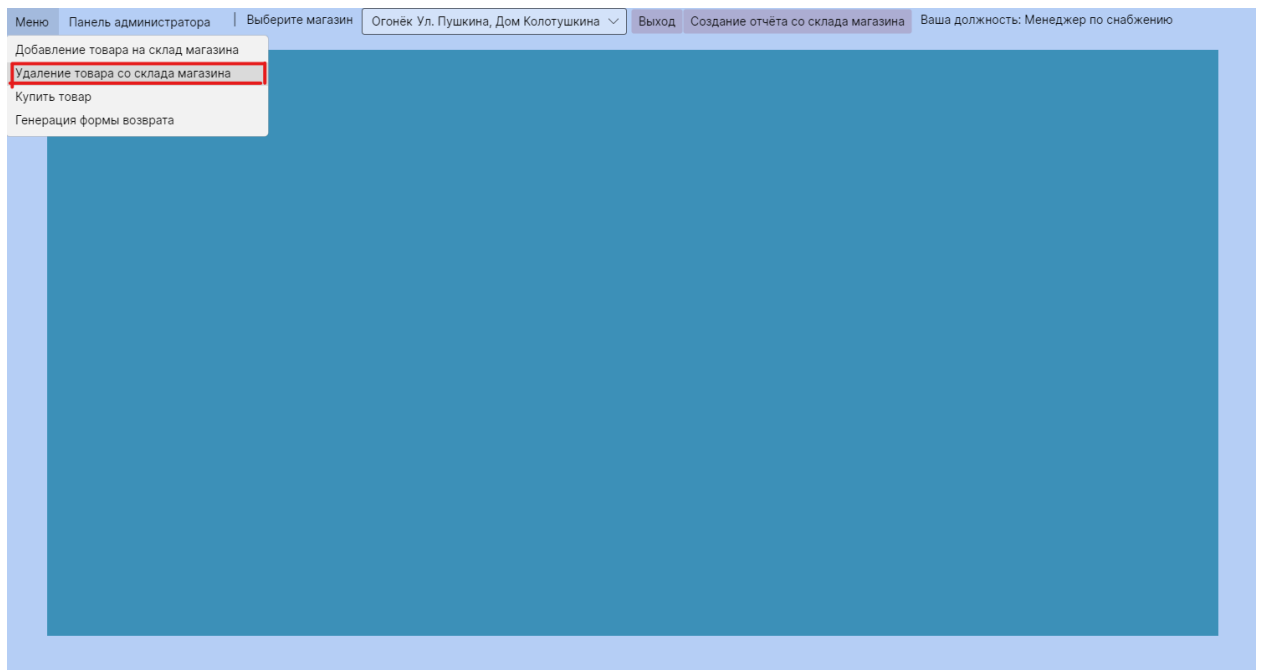


Рисунок 2.4.13 - Удаление товара со склада магазина

В открывшемся окне (Рисунок 2.4.14 необходимо заполнить поле (Товар)).

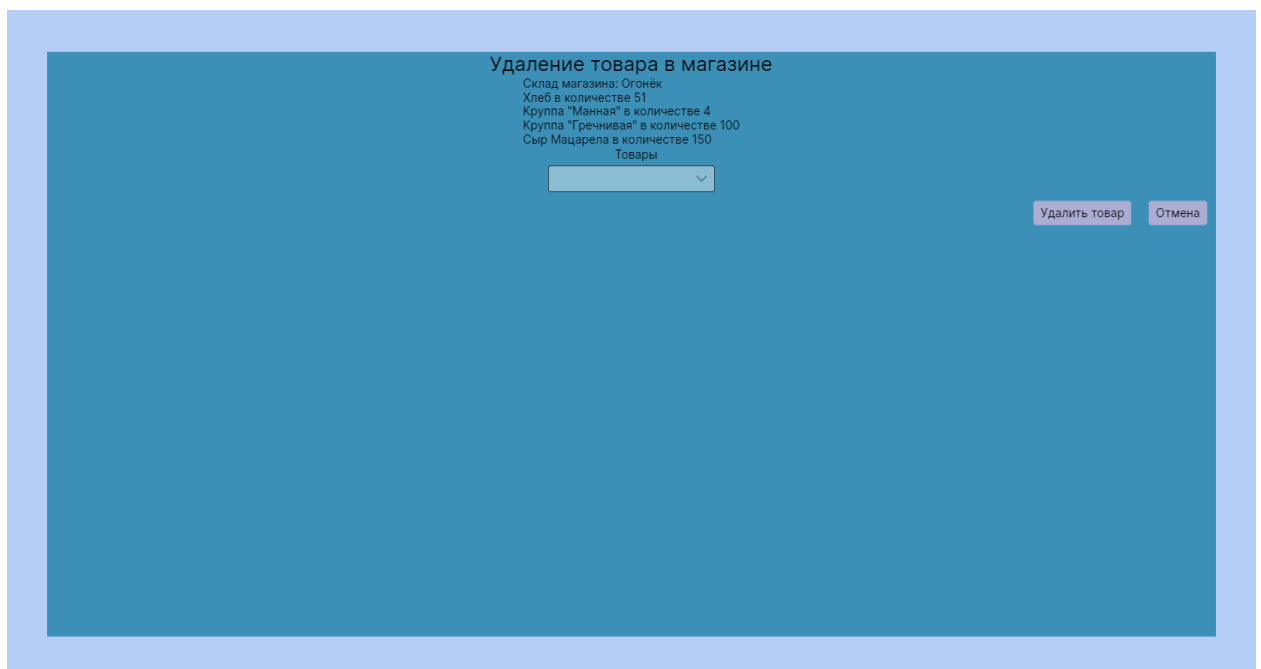


Рисунок 2.4.14 - Удаление товара со склада магазина

Далее остаётся нажать кнопку «Удалить товар». Если удаление товара не требуется, или удаление товара уже не актуальна, то можно нажать кнопку «Отмена».

Купить товар
Для того что бы купить товар необходимо открыть Меню и выбрать пункт «Купить товар» (Рисунок 2.4.15)



Рисунок 2.4.15 – Купить товар

В открывшемся окне (Рисунок 2.4.16) необходимо заполнить поля (Товар; Количество).

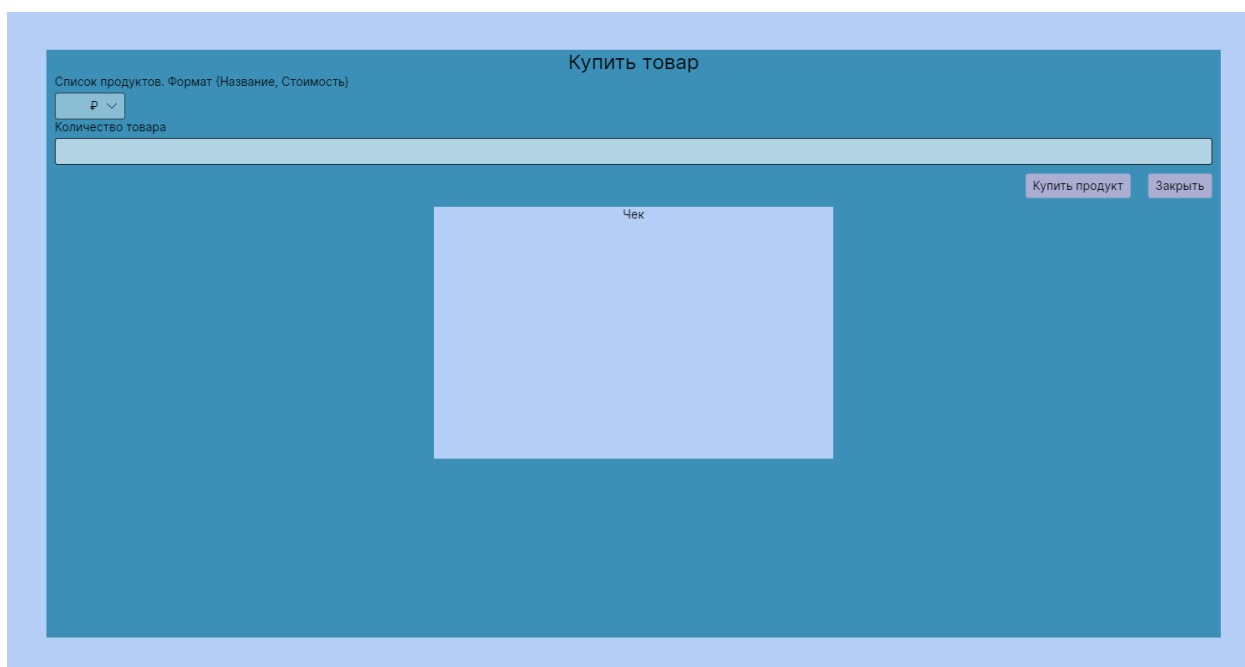


Рисунок 2.4.16– Купить товар

Далее остаётся нажать кнопку «Купить продукт». Если покупка продукта не требуется, или покупка продукта уже не актуальна, то можно нажать кнопку «Отмена».

Генерация формы возврата
Для того что бы купить товар необходимо открыть Меню и выбрать пункт «Генерация формы возврата» (Рисунок 2.4.17)

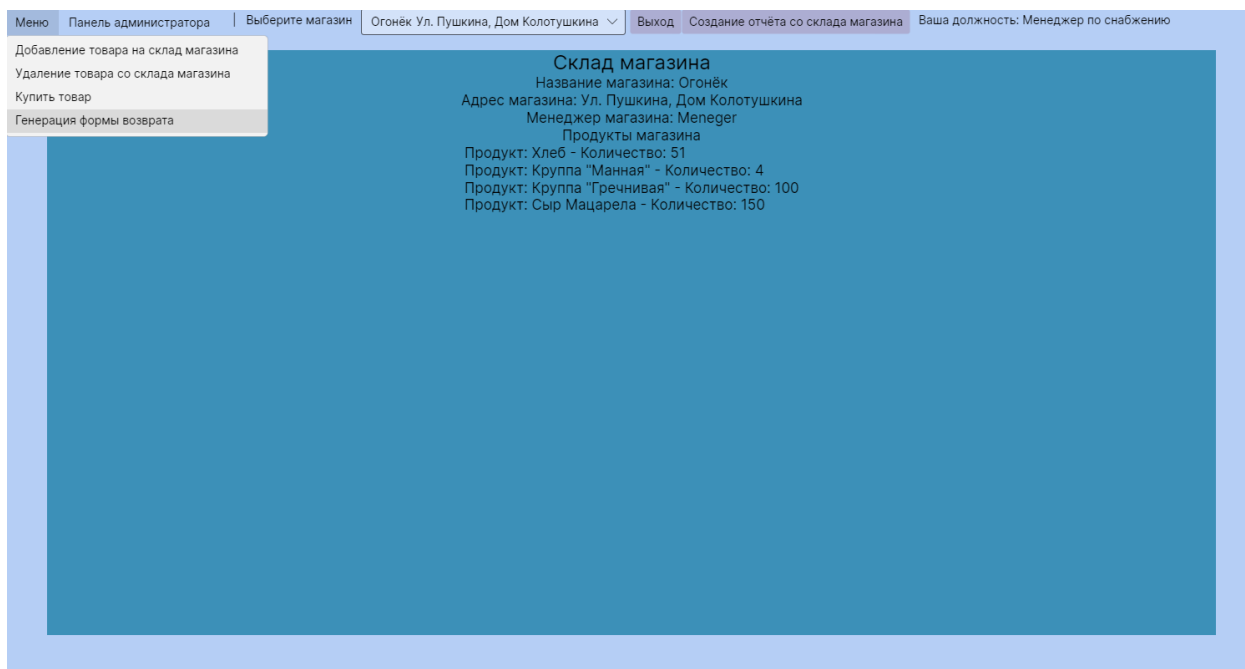


Рисунок 2.4.18 – Генерация формы возврата
В открывшемся окне (Рисунок 2.4.19) необходимо заполнить поле (Товар).

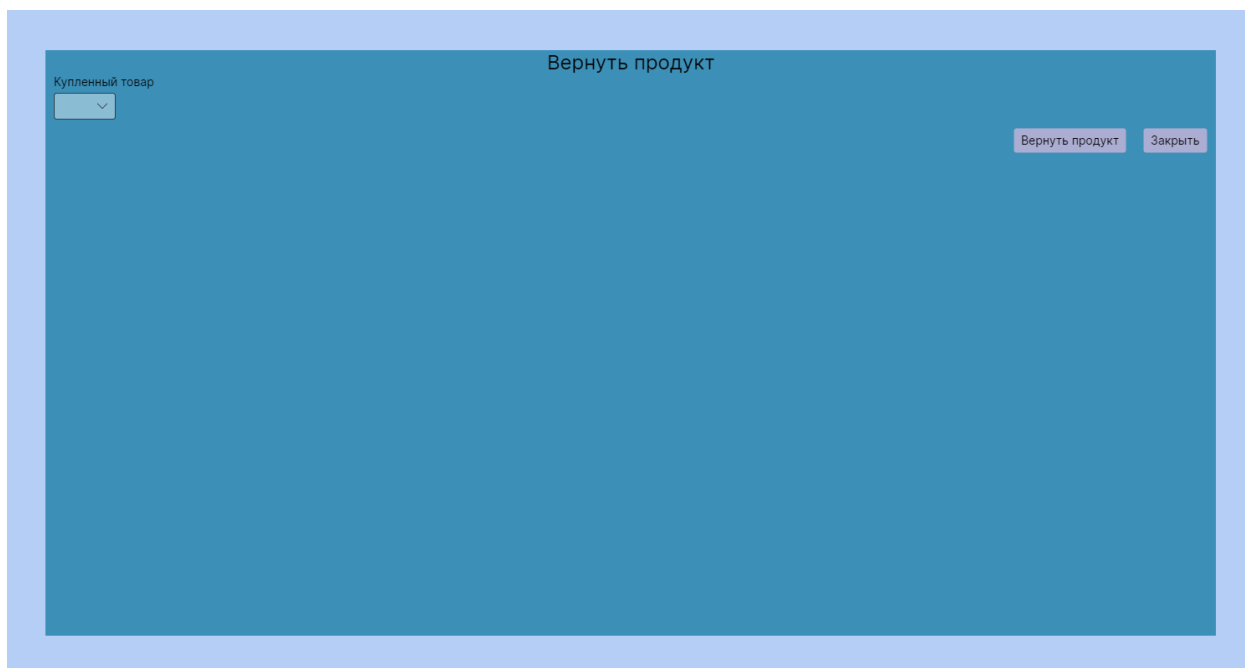


Рисунок 2.4.19 – Генерация формы возврата

Далее остаётся нажать кнопку «Вернуть продукт». Если возврат продукта не требуется, или возврат продукта уже не актуален, то можно нажать кнопку «Заккрыть».

ЗАКЛЮЧЕНИЕ

В ходе работы была проанализирована предметная область, разработан интерфейс и разработаны следующие функции:

- 1) Добавление товара в базу данных системы.
- 2) Добавить сотрудника.
- 3) Добавить магазин.
- 4) Обновить данные сотрудника.
- 5) Обновить данные магазина.
- 6) Добавление товара на склад магазина.
- 7) Удаление товара со склада магазина.
- 8) Купить товар.
- 9) Генерация формы возврата.

Перечень используемых источников.

- 1) Avalonia UI Документация. — Текст : электронный // Avalonia : [сайт] — URL: <https://docs.avaloniaui.net/> (дата обращения: 22.04.2023).
- 2) Руководство по Entity Framework – Текст: электронный // Code First: [сайт] — URL: <https://metanit.com/sharp/entityframework/1.2.php> (дата обращения: 22.04.2023).
- 3) Linq Tutorial – Текст: электронный // Learn Tutorials : [сайт] - URL: <https://learntutorials.net/ru/csharp/topic/68/запросы-linq> (дата обращения: 22.04.2023).
- 4) Использование Code-First. Текст: электронный // Professor Web : [сайт] - URL: https://professorweb.ru/my/entity-framework/6/level1/1_4.php (дата обращения: 22.04.2023).
- 5) XAML - Краткое руководство. Текст: электронный // Tutorials point: [сайт] – URL: https://translated.turbopages.org/proxy_u/en-ru.ru.c03f151b-644348f2-b77490e3-74722d776562/https/www.tutorialspoint.com/xaml/xaml_quick_guide.html (дата обращения: 22.04.2023).

Листинг программы

```

public partial class MainWindow : Window
{
    public static bool isAuth = false; // false - Сотрудник не авторизирован, true -
    сотрудник авторизирован
    public static int number_attempts = 0; // Хранит количество не удачных
    входов в аккаунт
    public static int idUser = 0;
    decimal totalCost = 0; // Хранит сумму чека при добавлении товара на склад
    магазина

    public void updateListStores()
    {
        Context _context = new Context();
        var stores = _context.Stores.ToList();
        StoreMenu.Items = stores;
    }

    public MainWindow()
    {
        InitializeComponent();
        updateListStores();
        #if DEBUG
        this.AttachDevTools();
        #endif
        this.WindowState = WindowState.Maximized;
    }

    // 1 Функция авторизации сотрудника
    private void LoginButton_Click(object? sender, RoutedEventArgs e)
    {
        Context _context = new Context();
        var user = _context.Users.Include(x => x.Role).SingleOrDefault(u => u.Name
        == LoginTextBox.Text && u.Password == PasswordBox.Text);

        if (user != null)
        {
            StatusBar.Text = $"Добро пожаловать {user.Name}!";
            InfoRole.Text = $"Ваша должность: {user.Role.Title}";
            idUser = user.Id;
            number_attempts = 0; // Обнуление попыток входа.
            isAuth = true;
        }
    }
}

```

```

    Auth.IsVisible = false; // Скрыть окно авторизации
    Menu.IsVisible = true; // Показать меню
    StoreInfo.IsVisible = true;
    LoginTextBox.Text = "";
    PasswordBox.Text = "";
}
else
{
    if (number_attempts >= 3)
    {
        StatusBar.Text = $"Вы не авторизированный уже {number_attempts}
паза,\n" +
        $"Обратитесь к Администратору для восстановления
доступа к аккаунту!";
    }
    else
    {
        StatusBar.Text = $"Не авторизирован!";
    }

    number_attempts++;
    PasswordBox.Text = "";
}
}

// 3 Функция для того что бы скрыть окно авторизации
private void HideBorder_Click(object? sender, RoutedEventArgs e)
{
    if (isAuth == true)
    {
        Auth.IsVisible = false; // Скрыть окно авторизации
        Menu.IsVisible = true; // Показать меню
        StoreInfo.IsVisible = true;
        LoginTextBox.Text = "";
        PasswordBox.Text = "";
    }
    else
    {
        StatusBar.Text = "Для выхода нужно Авторизироваться!";
    }
}
}

```

```

// 4 Кнопка в меню для открытия окна создания сотрудника
private void AP_AddUser(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var checkUser = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var role = _context.Roles.SingleOrDefault(x => x.IdRoles == 1); // Роль
администратора

    if (checkUser.Role == role) // Создавать аккаунты может только
администратор
    {
        Menu.IsVisible = false;
        StoreInfo.IsVisible = false;
        ApWinAddUser.IsVisible = true;
    }
    else
    {
        InfoBoxMenu.Text = $"Отказано в доступе!";
    }
}

// 5 Кнопка для закрытия окна создания сотрудника
private void AP_AddUser_Cancel(object? sender, RoutedEventArgs e)
{
    ApWinAddUser.IsVisible = false;
    Menu.IsVisible = true;
    StoreInfo.IsVisible = true;
}

// 6 Кнопка что бы создать аккаунт сотрудника
private void AP_AddUser_Create(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var checkUser = _context.Users.SingleOrDefault(x => x.Name ==
UserLogin.Text && x.Password == UserPass.Text);
    if (checkUser == null)
    {
        AddFunctions.AddUser(UserLogin.Text, UserPass.Text,
Convert.ToInt32(UserRole.Text));
        InfoBox.Text =
        $"Аккаунт создан\nLogin: {UserLogin.Text}\nPassword:

```

```

{UserPass.Text}\nДля выхода нажмите 'Отмена'\n\n";

    UserLogin.Text = "";
    UserPass.Text = "";
    UserRole.Text = "";
}
else
{
    InfoBox.Text = $"Создать аккаунт не удалось!\nВозможно логин
({UserLogin.Text}) уже занят.\n\n";
}

}

// 6 Кнопка что бы выйти из аккаунта
private void LeaveAcc(object? sender, RoutedEventArgs e)
{
    isAuth = false;
    Menu.IsVisible = false;
    Auth.IsVisible = true;
    StoreInfo.IsVisible = false;
    InfoBoxMenu.Text = "";
    StoreInfo.IsVisible = false;
}

// 7 Кнопка в меню для открытия окна создания продукта
private void Add_Product(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var checkUser = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var role = _context.Roles.SingleOrDefault(x => x.IdRoles == 1); // Роль
администратора

    if (checkUser.Role == role) // Создавать аккаунты может только
администратор
    {
        AddProductWin.IsVisible = true;
        StoreInfo.IsVisible = false;
        Menu.IsVisible = false;
    }
    else

```



```

    {
        InfoBoxMenu.Text = $"Отказано в доступе!";
    }
}

// 10 Кнопка закрыть окно создания продукта
private void AddProductWin_Cancel(object? sender, RoutedEventArgs e)
{
    AddProductWin.IsVisible = false;
    Menu.IsVisible = true;
    StoreInfo.IsVisible = true;
}

// 11 Кнопка для создания продукта
private void AddProductWin_Create(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var checkUser = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var role = _context.Roles.SingleOrDefault(x => x.IdRoles == 1); // Роль
администратора

    if (checkUser.Role == role) // Создавать аккаунты может только
администратор
    {
        AddFunctions.AddProduct(ProductName.Text,
Convert.ToDecimal(ProductPrice.Text), ProductDiscription.Text,
(StoreMenu.SelectedItem as stores).StoreId, idUser,
Convert.ToInt32(ProductQuantity.Text) );
        ProductInfoBox.Text =
            $"Продукт: {ProductName.Text} Стоимостью
{ProductPrice.Text}\nДля выхода нажмите 'Отмена'\n\n";

        ProductName.Text = "";
        ProductDiscription.Text = "";
        ProductPrice.Text = "";
    }
    else
    {
        InfoBoxMenu.Text = $"К этой функции есть доступ только у
Администрации!";
    }
}

```

```

    }

    // 12 Открыть окно для создания магазина
    private void AP_AddStore(object? sender, RoutedEventArgs e)
    {
        Context _context = new Context();
        var checkUser = _context.Users.SingleOrDefault(x => x.Id == idUser);
        var role = _context.Roles.SingleOrDefault(x => x.IdRoles == 1); // Роль администратора

        if (checkUser.Role == role) // Создавать аккаунты может только администратор
        {
            Menu.IsVisible = false;
            AddStoreWin.IsVisible = true;
            StoreInfo.IsVisible = false;
            var users = _context.Users.Where(x => x.Role.IdRoles == 2 || x.Role.IdRoles == 5).ToList();
            StoreMeneger.Items = users.ToList();
        }
        else
        {
            InfoBoxMenu.Text = $"Отказано в доступе!";
        }
    }

    // 13 Кнопка для закрытия окна Создания магазина
    private void AddStoreWin_Cancel(object? sender, RoutedEventArgs e)
    {
        Menu.IsVisible = true;
        AddStoreWin.IsVisible = false;
        StoreInfo.IsVisible = true;
    }

    // 14 Кнопка для создания магазина
    private void AddStoreWin_Create(object? sender, RoutedEventArgs e)
    {
        try
        {
            var selectedUser = (StoreMeneger.SelectedItem as users).Id;
            if (selectedUser != null)

```

```

        {
            AddFunctions.AddStore(StoreName.Text, StoreLocation.Text,
selectedUser);
            updateListStores();
            AddStoreInfoBox.Text = "Магазин создан!";
            StoreName.Text = "";
            StoreLocation.Text = "";
        }
        else
        {
            AddStoreInfoBox.Text = "Ошибка!";
            StoreName.Text = "";
            StoreLocation.Text = "";
        }
    }
    catch (Exception exception)
    {
        AddStoreInfoBox.Text = "Критическая ошибка!";
        StoreName.Text = "";
        StoreLocation.Text = "";
    }
}

```

```

private void AP_UpUser(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var checkUser = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var role = _context.Roles.SingleOrDefault(x => x.IdRoles == 1); // Роль
администратора

    if (checkUser.Role == role) // Создавать аккаунты может только
администратор
    {
        Menu.IsVisible = false;
        UpdateWinAddUser.IsVisible = true;
        var users = _context.Users.ToList();
        var roles = _context.Roles.ToList();

        UpdateUser.Items = users;
        UpUserRole.Items = roles;
    }
}

```

```

        StoreInfo.IsVisible = false;
    }
    else
    {
        InfoBoxMenu.Text = $"К этой функции есть доступ только у
Администрации!";
    }
}

private void AP_UpUser_Cancel(object? sender, RoutedEventArgs e)
{
    Menu.IsVisible = true;
    UpdateWinAddUser.IsVisible = false;
    StoreInfo.IsVisible = true;
}

private void AP_UpUser_Create(object? sender, RoutedEventArgs e)
{
    try
    {
        Context _context = new Context();
        var user = _context.Users.ToList();

        var selectedUser = (UpdateUser.SelectedItem as users).Id;
        var selectUser = _context.Users.SingleOrDefault(x => x.Id ==
selectedUser);

        var selectedRole = (UpUserRole.SelectedItem as roles).IdRoles;
        var selectRole = _context.Roles.SingleOrDefault(x => x.IdRoles ==
selectedRole);

        if (selectedUser != null && selectedRole != null && UserLogin.Text != ""
&& UserPass.Text != "")
        {
            AddFunctions.UpdateUser(selectedUser, selectedRole, UserLogin.Text,
UserPass.Text);

            UserLogin.Text = "";
            UserPass.Text = "";
            UpInfoBox.Text = "Успешно!";
        }
    }
}

```

```

else
{
    UserLogin.Text = "";
    UserPass.Text = "";
    UpInfoBox.Text = "Ошибка!";
}
}
catch (Exception exception)
{
    UpInfoBox.Text = "Критическа ошибка!";
}
}

private void SelectStore(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();

    if (StoreMenu.SelectedItem != null)
    {
        var store = (StoreMenu.SelectedItem as stores).StoreId;
        var storeP = StoreMenu.SelectedItem as stores;

        var storeInfo = _context.Stores.Include(x =>
x.ManagerId).SingleOrDefault(x => x.StoreId == store);

        TitleBlock.Text = "Склад магазина\n";
        StoreInfo_NameStore.Text = $"Название магазина:
{storeInfo.StoreName}";
        StoreInfo_StoreLocation.Text = $"Адрес магазина:
{storeInfo.StoreLocation}";
        StoreInfo_Meneger.Text = $"Менеджер магазина:
{storeInfo.ManagerId.Name}";

        var products = _context.StoreProducts.Include(sp =>
sp.ProductId).Where(x => x.StoreId == storeP).ToList();
        ProductsListStore.Text = "";
        NameProductList.Text = "Продукты магазина";
        foreach (var item in products)
        {
            ProductsListStore.Text += $"Продукт: {item.ProductId.ProductName} -
Количество: {item.Quantity}\n";
        }
    }
}

```

```

    }
}
else
{
    InfoBoxMenu.Text = $"Сначала нужно выбрать магазин.";
}

}

private void AP_UpStore(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var checkUser = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var role = _context.Roles.SingleOrDefault(x => x.IdRoles == 1); // Роль
администратора

    if (checkUser.Role == role) // Создавать аккаунты может только
администратор
    {
        UpdateStore.IsVisible = true;
        Menu.IsVisible = false;
        StoreInfo.IsVisible = false;
        var store = _context.Stores.ToList();
        var menegers = _context.Users.Where(x => x.Role.IdRoles == 2).ToList();

        UpStoreId.Items = store;
        UpStoreMeneger.Items = menegers;
    }
    else
    {
        InfoBoxMenu.Text = $"Отказано в доступе!";
    }
}

private void AP_UpStore_Cancel(object? sender, RoutedEventArgs e)
{
    Menu.IsVisible = true;
    StoreInfo.IsVisible = true;
    UpdateStore.IsVisible = false;
}

```

```

private void AP_UpStore_Update(object? sender, RoutedEventArgs e)
{
    try
    {
        var storeId = (UpStoreId.SelectedItem as stores).StoreId;
        var storeMeneger = (UpStoreMeneger.SelectedItem as users).Id;

        var newStoreName = UpStoreName.Text;
        var newStoreLocation = UpStoreLocation.Text;

        if (storeId != null && storeMeneger != null && newStoreName != "" &&
newStoreLocation != "")
        {
            AddFunctions.UpdateStore(storeId, storeMeneger, newStoreName,
newStoreLocation);
            updateListStores();
            UpStoreInfoBox.Text = $"Данные обновлены!";
            UpStoreName.Text = "";
            UpStoreLocation.Text = "";
        }
        else
        {
            UpStoreInfoBox.Text = "Ошибка! Проверьте все ли поля заолнены.";
            UpStoreName.Text = "";
            UpStoreLocation.Text = "";
        }
    }
    catch (Exception exception)
    {
        UpStoreInfoBox.Text = "Критическая ошибка! Проверьте все ли поля
заолнены.";
        UpStoreName.Text = "";
    }
}

private void Add_Product_Store(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    if (StoreMenu.SelectedItem != null)
    {

```

```

        var selectedStore = (StoreMenu.SelectedItem as stores).StoreId;
        var menegerStore = _context.Users.SingleOrDefault(x => x.Id == idUser);
        var storeInfo = _context.Stores.SingleOrDefault(x => x.StoreId ==
selectedStore && x.ManagerId == menegerStore);
        if (storeInfo == null)
        {
            InfoBoxMenu.Text = "Отказано в доступе!";
        }
        else
        {
            var products = _context.Products.ToList();
            AddProductStore_Product.Items = products;
            AddProductStore.IsVisible = true;
            Menu.IsVisible = false;
            StoreInfo.IsVisible = false;
        }
    }
    else
    {
        InfoBoxMenu.Text = "Нужно выбрать магазин!";
    }
}

private void Menu_AddProductStore_Cancel(object? sender, RoutedEventArgs
e)
{
    AddProductStore.IsVisible = false;
    Menu.IsVisible = true;
    StoreInfo.IsVisible = true;
    AddProductStore_InfoBox.Text = "";
    AddProductStore_TotalCost.Text = "";
}

private void Menu_AddProductStore_Add(object? sender, RoutedEventArgs e)
{
    int idProduct = (AddProductStore_Product.SelectedItem as
products).ProductId;
    Context _context = new Context();
    var selectedProduct = _context.Products.SingleOrDefault(x => x.ProductId
== idProduct);
    var selectedStore = _context.Stores.SingleOrDefault(x => x.StoreId ==

```



```

(StoreMenu.SelectedItem as stores).StoreId);

    AddProductStore_InfoBox.Text += $"Продукт:
{selectedProduct.ProductName}, Количество:
{AddProductStore_CuounProduct.Text}\n";

    int count = _context.StoreProducts.Count();
    Debug.WriteLine($"Количество записей StoreProducts: {count}");

    if (selectedStore != null && selectedProduct != null
    && !string.IsNullOrEmpty(AddProductStore_CuounProduct.Text))
    {
        AddFunctions.AddProductStore(selectedStore.StoreId,
selectedProduct.ProductId,
Convert.ToInt32(AddProductStore_CuounProduct.Text), idUser);
    }

    totalCost += Convert.ToDecimal(AddProductStore_CuounProduct.Text) *
selectedProduct.Price;
    AddProductStore_TotalCost.Text = $"Итого: {totalCost}Р";
}

private void Delete_Product_Store(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();
    var menegerStore = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var selectedStoreCheck = (StoreMenu.SelectedItem as stores).StoreId;
    var storeInfoCheck = _context.Stores.SingleOrDefault(x => x.StoreId ==
selectedStoreCheck && x.ManagerId == menegerStore);

    if (storeInfoCheck == null)
    {
        InfoBoxMenu.Text = "Отказано в доступе!";
    }
    else if (StoreMenu.SelectedItem != null && StoreMenu.SelectedItem != null)
    {
        var storeInfo = _context.Stores.SingleOrDefault(x => x.StoreId ==
(StoreMenu.SelectedItem as stores).StoreId && x.ManagerId == menegerStore);
        var selectedStore = _context.Stores.SingleOrDefault(x => x.StoreId ==
(StoreMenu.SelectedItem as stores).StoreId);
        var products = _context.StoreProducts.Where(x => x.StoreId ==
selectedStore).Select(x => x.ProductId).ToList();

```

```

        var list = _context.StoreProducts.Include(x => x.ProductId).Where(x =>
x.StoreId == selectedStore).ToList();
        Delete_Product_Store_Product.Items = products;

```

```

        Delete_Product_Store_InfoSklad.Text = $"Склад магазина:
{selectedStore.StoreName}\n";
        foreach (var item in list)
        {
            Delete_Product_Store_InfoSklad.Text +=
$" {item.ProductId.ProductName} в количестве {item.Quantity}\n";
        }

```

```

        Delete_Product_Store_StakPanel.IsVisible = true;
        Menu.IsVisible = false;
        InfoBox.IsVisible = false;
    }
    else
    {
        InfoBoxMenu.Text = "Выберите магазин!";
    }
}

```

```

private void Delete_Product_Store_StakPanel_Cancel(object? sender,
RoutedEventArgs e)
{
    Delete_Product_Store_StakPanel.IsVisible = false;
    Menu.IsVisible = true;
    InfoBox.IsVisible = true;
}

```

```

private void Delete_Product_Store_StakPanel_Delete(object? sender,
RoutedEventArgs e)
{
    Context _context = new Context();
    var del = _context.StoreProducts.Include(x =>
x.ProductId).SingleOrDefault(x => x.StoreId == (StoreMenu.SelectedItem as
stores) && x.ProductId == (Delete_Product_Store_Product.SelectedItem as
products));

```

```

    if (del != null)
    {

```

```

        _context.StoreProducts.Remove(del);
        _context.SaveChanges();
        Delete_Product_Store_InfoBox.Text = $"{del.ProductId.ProductName}
была удалено со склада";

        var selectedStore = _context.Stores.SingleOrDefault(x => x.StoreId ==
        (StoreMenu.SelectedItem as stores).StoreId);
        var list = _context.StoreProducts.Include(x => x.ProductId).Where(x =>
        x.StoreId == selectedStore).ToList();
        Delete_Product_Store_InfoSklad.Text = $"Склад магазина:
{selectedStore.StoreName}\n";
        foreach (var item in list)
        {
            Delete_Product_Store_InfoSklad.Text +=
            $"{item.ProductId.ProductName} в количестве {item.Quantity}\n";
        }
    }
    else
    {
        Delete_Product_Store_InfoBox.Text = $"{del.ProductId.ProductName} не
удалось удалить.";
    }
}
private async void Sold_Product_Store(object? sender, RoutedEventArgs e)
{
    Context _context = new Context();

    var store = (StoreMenu.SelectedItem as stores);

    if (store == null)
    {
        InfoBoxMenu.Text = "Нужно выбрать магазин!";
        return;
    }

    var products = await _context.StoreProducts
        .Include(x => x.ProductId)
        .Include(x => x.StoreId)
        .Where(x => x.StoreId == store)
        .Select(x => new { x.ProductId, x.ProductId.ProductName,
        x.ProductId.Price, x.Quantity, x.StoreId, x.IdStoreProducts })

```

```

        .ToListAsync();

        Sold_Product_Store_StakPanel_List.DataContext = products;
        Sold_Product_Store_StakPanel.IsVisible = true;
        Menu.IsVisible = false;
        InfoBox.IsVisible = false;
    }

    private void Sold_Product_Store_StakPanel_List_Add(object? sender,
        RoutedEventArgs e)
    {
        try
        {
            Context _context = new Context();
            var selectedProduct = Sold_Product_Store_StakPanel_List.SelectedItem as
dynamic;
            int productId = selectedProduct.ProductId.ProductId;

            var selectProdict2 = _context.Products.SingleOrDefault(x =>
                x.ProductId == productId);

            var storeSelect = StoreMenu.SelectedItem as stores;

            var product = _context.Products.SingleOrDefault(x =>
                x.ProductName == selectProdict2.ProductName).ProductId;

            var store = _context.Stores.SingleOrDefault(x => x.StoreName ==
storeSelect.StoreName).StoreId;

            AddFunctions.SellProduct(
                product,
                Convert.ToInt32(Sold_Product_Store_StakPanel_CuounProduct.Text),
                store,
                idUser);

            Sold_Product_Store_StakPanel_InfoBox.Text +=
$"{{selectProdict2.ProductName}} -
{{Sold_Product_Store_StakPanel_CuounProduct.Text}} *
{{selectProdict2.Price}}\n";

```

```

        totalCost +=
Convert.ToDecimal(Sold_Product_Store_StakPanel_CuounProduct.Text) *
selectProduct2.Price;
        Sold_Product_Store_StakPanel_TotalCost.Text = $"Итого: {totalCost}Р";
    }
    catch (Exception exception)
    {
        Sold_Product_Store_StakPanel_InfoBox.Text = $"{exception.Message}";
    }
}

private void Sold_Product_Store_StakPanel_Cancel(object? sender,
RoutedEventArgs e)
{
    Sold_Product_Store_StakPanel.IsVisible = false;
    Menu.IsVisible = true;
    InfoBox.IsVisible = true;

    Context _context = new Context();
    var store = (StoreMenu.SelectedItem as stores).StoreId;
    var storeP = StoreMenu.SelectedItem as stores;

    var storeInfo = _context.Stores.Include(x => x.ManagerId).SingleOrDefault(x
=> x.StoreId == store);

    TitleBlock.Text = "Склад магазина\n";
    StoreInfo_NameStore.Text = $"Название магазина:
{storeInfo.StoreName}";
    StoreInfo_StoreLocation.Text = $"Адрес магазина:
{storeInfo.StoreLocation}";
    StoreInfo_Meneger.Text = $"Менеджер магазина:
{storeInfo.ManagerId.Name}";

    var products = _context.StoreProducts.Include(sp => sp.ProductId).Where(x
=> x.StoreId == storeP).ToList();
    ProductsListStore.Text = "";
    NameProductList.Text = "Продукты магазина";
    foreach (var item in products)
    {
        ProductsListStore.Text += $"Продукт: {item.ProductId.ProductName} -
Количество: {item.Quantity}\n";
    }
}

```

```

    }
}

private void Return_Products_To_Store(object? sender, RoutedEventArgs e)
{
    // Пока что не работает, в будущем это должна быть кнопка для
открытия формы возврата
    Context _context = new Context();
    var storeP = StoreMenu.SelectedItem as stores;
    var user = _context.Users.SingleOrDefault(x => x.Id == idUser);
    var soldproduct = _context.SoldProducts.Include(x =>
x.ProductId).SingleOrDefault(x => x.IdSoldProducts == 1);
    var sale = _context.Sales.SingleOrDefault(x => x.SaleId == 1);

    // AddFunctions.ReturnProduct(soldproduct.ProductId.ProductId, 1,
storeP.StoreId, idUser);
    ReturnProduct.IsVisible = true;
    Menu.IsVisible = false;
    InfoBox.IsVisible = false;

    var listSales = _context.Sales
        .Where(x => x.UserId == _context.Users.SingleOrDefault(u => u.Id ==
idUser) && x.StoreId == storeP)
        .ToList();

    //var listProducts = _context.SoldProducts
    //    .Include(x => x.SaleId)
    //    .ThenInclude(s => s.StoreId)
    //    .Include(x => x.ProductId)
    //    .Where(x => listSales.Any(s => s.SaleId == x.SaleId.SaleId))
    //    .ToList();

    var products = _context.SoldProducts
        .Include(x => x.ProductId)
        .Where(x => x.SaleId.UserId == _context.Users.SingleOrDefault(x => x.Id
== idUser) &&
            DateTime.Compare(x.SaleId.SaleDate.AddDays(14),
DateTime.Now.ToUniversalTime()) >= 0 &&
            x.Quantity > 0)
        .ToList();

```

```

ReturnProduct_StakPanel_List.Items = products;

}

private void ReturnProduct_Cancel(object? sender, RoutedEventArgs e)
{
    ReturnProduct.IsVisible = false;
    Menu.IsVisible = true;
    InfoBox.IsVisible = true;

    SelectStore(null, null);
}

private void ReturnProduct_Add(object? sender, RoutedEventArgs e)
{
    var productId = (ReturnProduct_StakPanel_List.SelectedItem as dynamic);
    var storeP = StoreMenu.SelectedItem as stores;
    try
    {
        AddFunctions.ReturnProduct(productId.ProductId.ProductId,
productId.Quantity, storeP.StoreId, idUser);
        ReturnProduct_InfoBox.Text = "Товар возвращён.";
        ReturnProduct_StakPanel_List.SelectedItem = null;
    }
    catch (Exception exception)
    {
        ReturnProduct_InfoBox.Text = $"{exception.Message}";
    }
}

}

public class AddFunctions
{
    // Создание аккаунта
    public static void AddUser(string UserName, string Passord, int Role)
    {
        Context _context = new Context();

        var _users = _context.Users.ToList();

```

```

var roleId = _context.Roles.SingleOrDefault(x => x.IdRoles == Role);

var newUser = new users
{
    Name = UserName,
    Password = Passord,
    Role = roleId
};
_users.Add(newUser);

_context.Add(newUser);
_context.SaveChanges();
}

// Создание магазина
public static void AddStore(string StoreName, string StoreLocation, int userId)
{
    Context _context = new Context();
    var storesList = _context.Stores.ToList();

    var meneger = _context.Users.SingleOrDefault(x => x.Id == userId);

    if (userId != null && StoreName != "" && StoreLocation != "")
    {
        var newStore = new stores
        {
            StoreName = StoreName,
            StoreLocation = StoreLocation,
            ManagerId = meneger
        };

        storesList.Add(newStore);
        _context.Add(newStore);
        _context.SaveChanges();
    }
    else
    {
        Debug.WriteLine($"Ошибка! Проверь правильность заполнения
полей.");
    }
}

```



```

    }

    // Обновление данных пользователя
    public static void UpdateUser(int userId, int roleId, string userLogin, string
userPass)
    {
        Context _context = new Context();
        var user = _context.Users.ToList();

        var selectUser = _context.Users.SingleOrDefault(x => x.Id == userId);

        var selectRole = _context.Roles.SingleOrDefault(x => x.IdRoles == roleId);

        if (selectUser != null && selectRole != null)
        {
            selectUser.Name = userLogin;
            selectUser.Password = userPass;
            selectUser.Role = selectRole;
            Debug.WriteLine($"Данные пользователя: {selectUser.Name} были
изменены.");
        }
        else
        {
            Debug.WriteLine($"Ошибка! Что то не так, проверьте заполненны ли
все поля!");
        }
        _context.SaveChanges();
    }

    // Добовление продукта в общий склад
    public static void AddProduct(string Title, decimal Price, string Discription, int
storeId, int userId, int Quantity)
    {
        Context _context = new Context();

        var productsList = _context.Products.ToList();
        var newProduct = new products
        {
            ProductName = Title,
            Description = Discription,
            Price = Price

```

```

    };
    productsList.Add(newProduct);
    _context.Add(newProduct);
    _context.SaveChanges();
}

// Обновление информации о магазине
public static void UpdateStore(int storeId, int storeMeneger, string
newStoreName, string newStoreLocation)
{
    Context _context = new Context();
    var newStoreMeneger = _context.Users.SingleOrDefault(x => x.Id ==
storeMeneger);
    var updateStore = _context.Stores.SingleOrDefault(x => x.StoreId ==
storeId);

    updateStore.StoreName = newStoreName;
    updateStore.StoreLocation = newStoreLocation;
    updateStore.ManagerId = newStoreMeneger;

    _context.SaveChanges();
}

// Метод для добавления продукта в магазин
public static void AddProductStore(int selectedStore, int selectedProduct, int
count, int userId)
{
    using (var context = new Context())
    {
        var storeIdCheck = context.Stores.SingleOrDefault(x => x.StoreId ==
selectedStore);
        var productIdCheck = context.Products.SingleOrDefault(x => x.ProductId
== selectedProduct);

        // Проверяем наличие записи в таблице store_products для данного
магазина и продукта
        var existingProduct = context.StoreProducts.FirstOrDefault(sp =>
sp.StoreId == storeIdCheck && sp.ProductId == productIdCheck);

        if (existingProduct != null)
        {

```

```

        // Если запись существует, то увеличиваем значение поля Quantity
        existingProduct.Quantity += count;
    }
    else
    {
        // Иначе создаем новую запись
        var storeId = context.Stores.SingleOrDefault(x => x.StoreId ==
selectedStore);
        var productId = context.Products.SingleOrDefault(x => x.ProductId ==
selectedProduct);

        var newProduct = new store_products
        {
            StoreId = storeId,
            ProductId = productId,
            Quantity = count
        };
        context.StoreProducts.Add(newProduct);

        var deliveriesList = context.Deliveries.ToList();
        var newDeliveries = new deliveries
        {
            StoreId = storeId,
            UserId = context.Users.SingleOrDefault(x => x.Id == userId),
            DeliveryDate = DateTime.Now.ToUniversalTime()
        };
        deliveriesList.Add(newDeliveries);
        context.Add(newDeliveries);
        context.SaveChanges();

        var deliveriesProductsList = context.DeliveredProducts.ToList();
        var newdeliveriesProducts = new delivered_products
        {
            DeliveryId = newDeliveries,
            ProductId = productId,
            Quantity = count
        };
        deliveriesProductsList.Add(newdeliveriesProducts);
        context.Add(newdeliveriesProducts);
        context.SaveChanges();
    }
}

```

```

        context.SaveChanges();
    }
}

// Метод для продажи продукта
public static void SellProduct(int productId, int quantity, int storeId, int userId)
{
    using (var context = new Context())
    {
        // Проверяем, что магазин с заданным идентификатором существует
        var store = context.Stores.SingleOrDefault(s => s.StoreId == storeId);
        if (store == null)
        {
            throw new ArgumentException("Магазин с таким идентификатором
не найден.");
        }

        // Проверяем, что продукт с заданным идентификатором существует
        var product = context.Products.SingleOrDefault(p => p.ProductId ==
productId);
        if (product == null)
        {
            throw new ArgumentException("Продукт с таким идентификатором
не найден.");
        }

        // Проверяем, что в магазине есть достаточное количество товара
        var storeProduct = context.StoreProducts
            .SingleOrDefault(sp => sp.StoreId == store && sp.ProductId ==
product);
        if (storeProduct == null || storeProduct.Quantity < quantity)
        {
            throw new ArgumentException("В магазине нет достаточного
количества товара.");
        }

        // Создаем объект продажи и сохраняем его в базу данных
        var sale = new sales
        {
            StoreId = store,

```

```

        UserId = context.Users.Find(userId),
        SaleDate = DateTime.Now.ToUniversalTime()
    };
    context.Sales.Add(sale);
    context.SaveChanges();

    // Создаем объекты проданных товаров и сохраняем их в базу данных
    var soldProduct = new sold_products
    {
        SaleId = sale,
        ProductId = product,
        Quantity = quantity,
        Price = product.Price
    };
    context.SoldProducts.Add(soldProduct);
    context.SaveChanges();

    // Уменьшаем количество товара в магазине
    storeProduct.Quantity -= quantity;
    context.SaveChanges();
    }
}

// Метод для возврата продукта на склад

/*
 * Метод должен создавать запись в таблицах returned_products, returns.
 * Так-же он должен возвращать продукт в таблицу store_products
 * Продукт можно вернуть в течении 14 дней
 */
public static void ReturnProduct(int productId, int quantity, int storeId, int
userId)
{
    using (var context = new Context())
    {
        var store = context.Stores.SingleOrDefault(s => s.StoreId == storeId);
        if (store == null)
        {
            throw new ArgumentException("Магазин с таким идентификатором
не найден.");
        }
    }
}

```

```

        var product = context.Products.SingleOrDefault(p => p.ProductId ==
productId);
        if (product == null)
        {
            throw new ArgumentException("Продукт с таким идентификатором
не найден.");
        }

        var sale = context.Sales.Where(s => s.StoreId.StoreId == storeId &&
s.UserId.Id == userId);
        if (sale == null)
        {
            throw new ArgumentException("Этот пользователь не покупал товар
в этом магазине.");
        }

        var soldProduct = context.SoldProducts
            .SingleOrDefault(sp => sp.ProductId.ProductId == product.ProductId
&&
                sp.Quantity == quantity);
        if (soldProduct == null)
        {
            throw new ArgumentException("Этот пользователь не купил этот
товар.");
        }

        if (quantity > soldProduct.Quantity)
        {
            throw new ArgumentException("Количество возвращаемого товара
больше, чем было куплено.");
        }

        var returnObj = new returns
        {
            StoreId = store,
            UserId = context.Users.Find(userId),
            ReturnDate = DateTime.Now.ToUniversalTime()
        };
        context>Returns.Add(returnObj);

```

```

var returnedProduct = new returned_products
{
    ReturnId = returnObj,
    ProductId = product,
    Quantity = quantity,
    Price = product.Price
};
context>ReturnsProducts.Add(returnedProduct);

var storeProduct = context.StoreProducts.SingleOrDefault(sp =>
sp.StoreId.StoreId == store.StoreId && sp.ProductId.ProductId ==
product.ProductId);
if (storeProduct == null)
{
    storeProduct = new store_products
    {
        StoreId = store,
        ProductId = product,
        Quantity = 0
    };
    context.StoreProducts.Add(storeProduct);
}
storeProduct.Quantity += quantity;
soldProduct.Quantity -= quantity;
context.SaveChanges();
}
}
}

```

Результат работы программы

Окно авторизации пользователя в системе. (Рисунок Б.1)

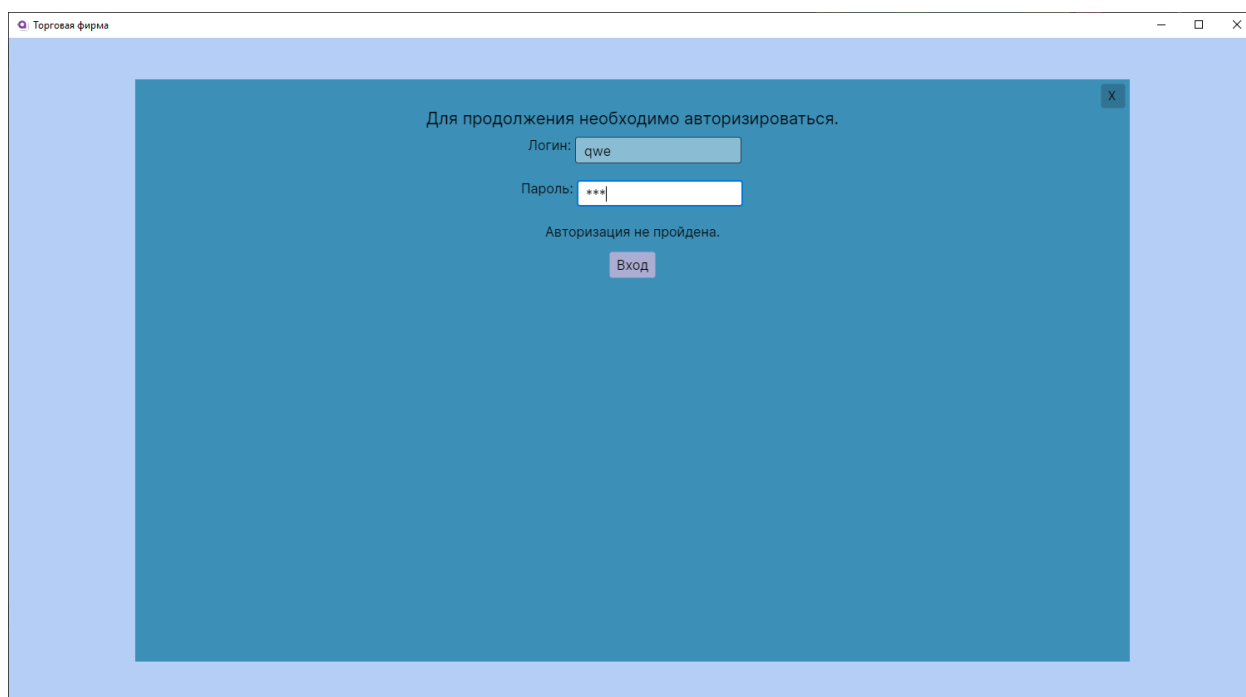


Рисунок Б.1 – Оно авторизации в системе.

Окно, которое откроется при нажатии кнопки «Вход» и, если пользователь с такими данными существует (Рисунок Б.2), Если данные были не верны (Рисунок Б.3).



Рисунок Б.2 – Основное окно Информационной системы

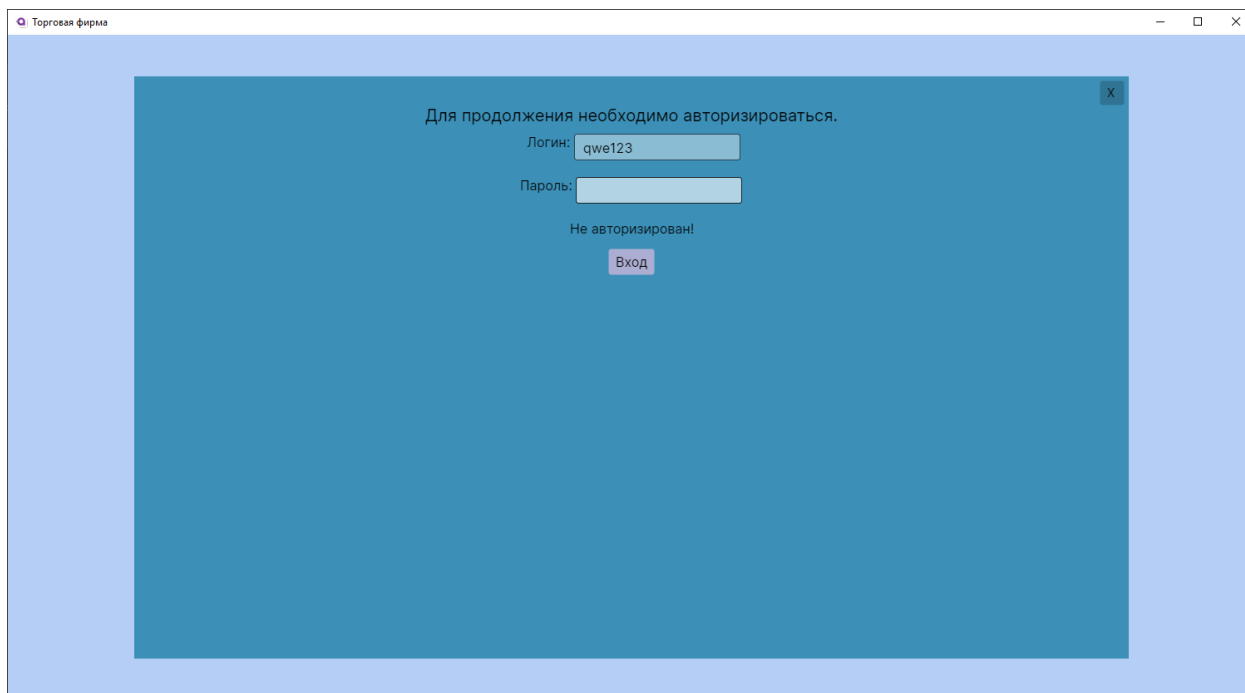


Рисунок Б.3 – Сообщение о том, что пользователь не смог авторизоваться.

Окно создания продукта, и добавления его на общий склад. (Рисунок Б.4)

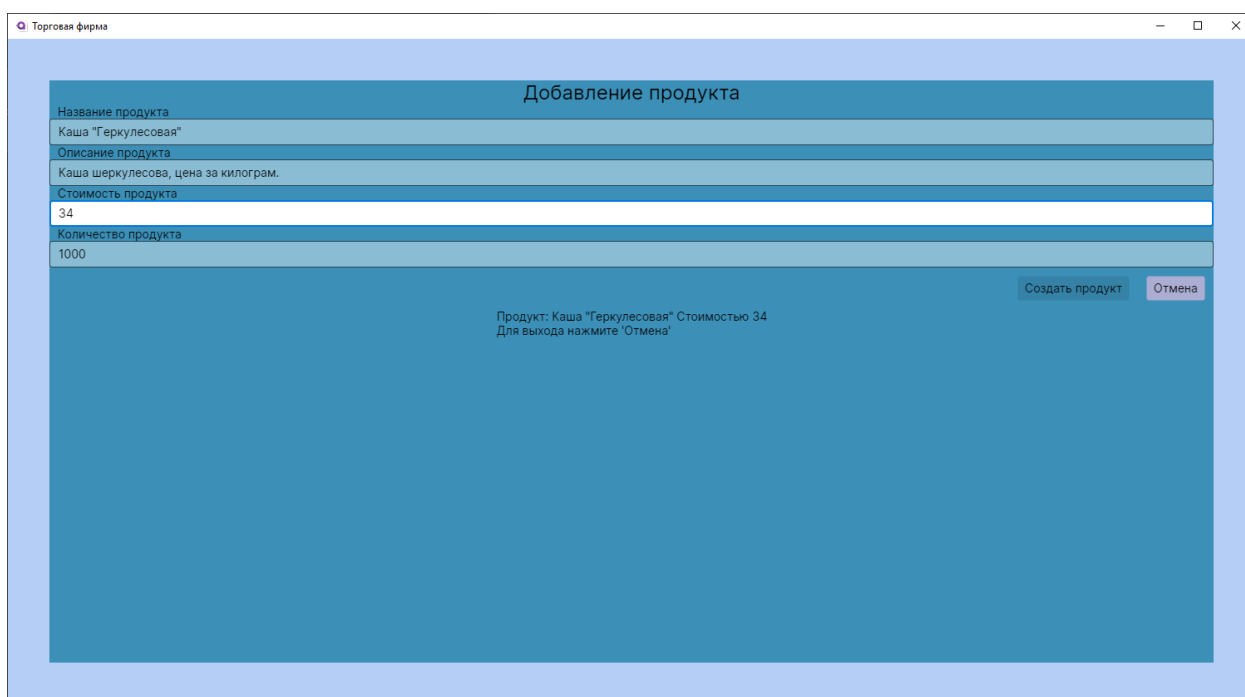


Рисунок Б.4 – Создание продукта

Окно создания нового пользователя. (Рисунок Б.5)

Торговая фирма

Создания аккаунта

Логин
Dmitry

Пароль
123Dm2Q

Ид Роли.
3

Создать аккаунт Отмена

Аккаунт создан
Login: Dmitry
Password: 123Dm2Q
Для выхода нажмите 'Отмена'

// Роли:
// Администратор 1,
// Менеджер по снабжению 2,
// Аудитор 3,
// Сотрудник продаж 4,
// Менеджер для каждого магазина 5

Рисунок Б.5 – Создания нового пользователя.

Окно создания нового магазина в системе Торговой Фирмы. (Рисунок Б.6)

Торговая фирма

Добавление продукта

Название магазина
Мирия Ра

Адрес магазина
Герцина 23

Менеджер магазина
2 Meneger

Создать магазин Отмена

Магазин создан!

Рисунок Б.6 – Создание нового магазина/

Окно обновления данных пользователя. (Рисунок Б.7)

Рисунок Б.7 – Обновление данные пользователя.

Окно обновления данных магазина. (Рисунок Б.8)

Рисунок Б.8 – Обновление данных магазина.

Окно для пополнения склада определённого магазина. (Рисунок Б.9)

Торговая фирма

Пополнить склад магазина

Список продуктов которые можно заказать. Формат (Название, Цена, Описание)

Каша "Геркулесовая" 34Р Каша из геркулеса стоимость за килограмм

Количество товара

20

Добавить продукт

Заккрыть

Счёт фактура

Продукт: Каша "Геркулесовая", Количество: 20

Итого: 680Р

Рисунок Б.9 – Пополнение склада.

Окно удаления товара со склада определённого магазина. (Рисунок Б.10)

Торговая фирма

Удаление товара в магазине

Склад магазина: Мария Ра

Товары

Каша "Геркулесовая"

Удалить товар

Отмена

Каша "Геркулесовая" была удалено со склада

Рисунок Б.10 – Удаления товара со склада.

Окно для покупки товара в определённом магазине. (Рисунок Б.11)

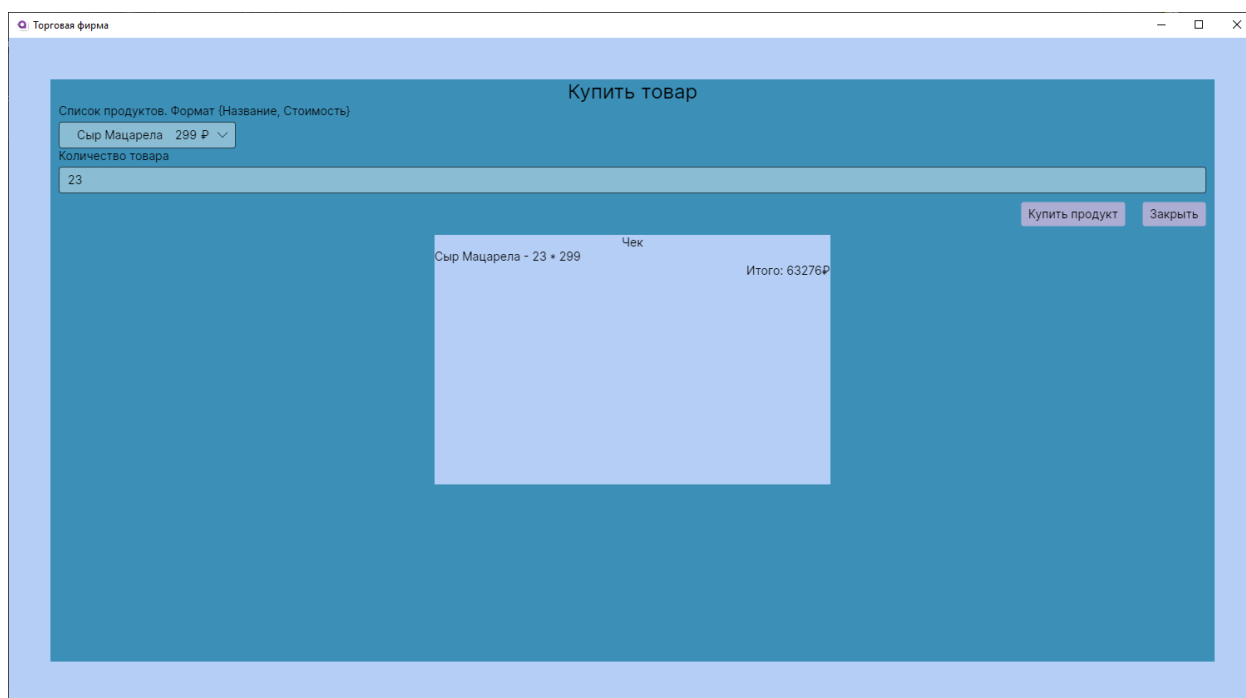


Рисунок Б.11 – Окно покупок товара.

Окно возвратов купленного товара в определённом магазине. (Рисунок Б.12)

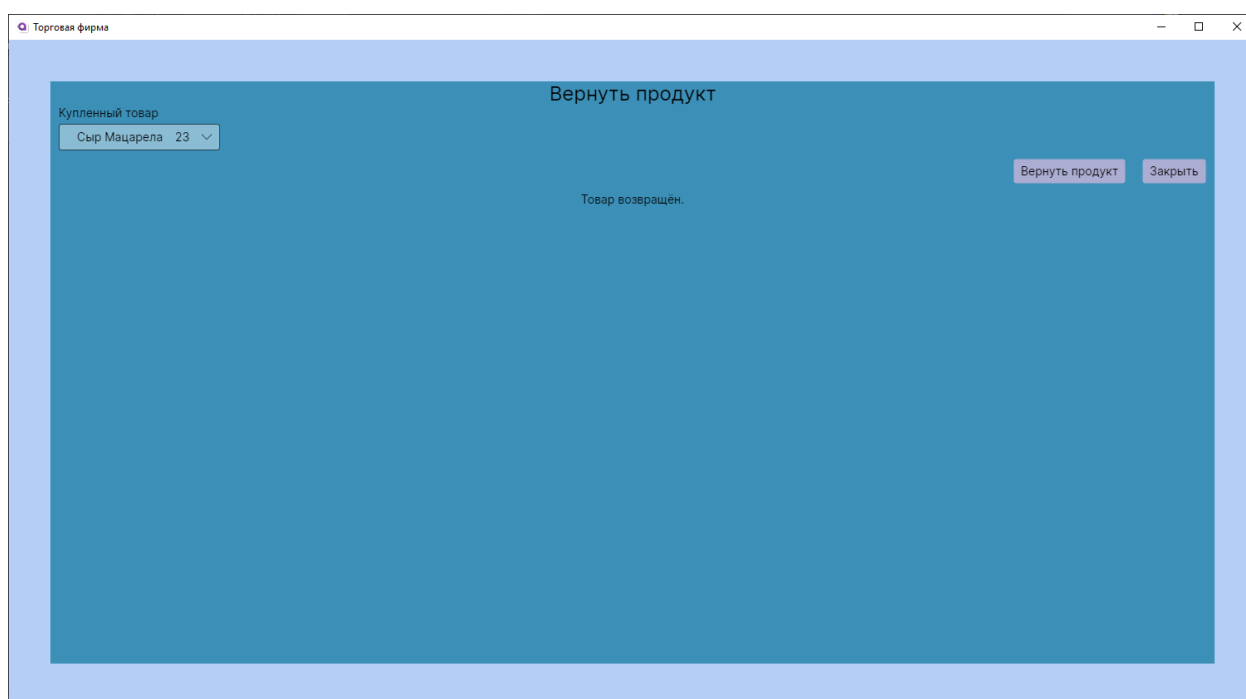


Рисунок Б.12 – Окно возвратов товаров.

Тесты

Таблица 2 – Тест покупки продукта.

| | |
|-------------------------|---|
| Тестовый пример | ТП_П_1 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование покупки продукта в магазине |
| Краткое изложение теста | Проверка функции покупки продуктов. |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Меню” -> “Купить товар” 2. Выбрать товар 3. Указать количество 4. Нажать кнопку “Купить продукт” |
| Тестовые данные | Хлеб 5, Хлеб 12, Хлеб 54 |
| Ожидаемый результат | Товар будет приобретён |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Покупка товара в магазине |
| Комментарии | |

Таблица 3 – Тест добавления продукта.

| | |
|-------------------------|--|
| Тестовый пример | ТП_П_2 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование добавления продукта на склад магазина |
| Краткое изложение теста | Проверка функции пополнения склада. |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Меню” -> “Добавление товара на склад магазина” 2. Выбрать товар 3. Указать количество 4. Нажать кнопку “Добавить продукт” |
| Тестовые данные | Хлеб 5, Крупка «Манная» 12, Сыр «Моцарелла» 54 |
| Ожидаемый результат | Товар будет добавлен |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Добавление товара на склад магазина. |
| Комментарии | |

Таблица 4 – Тест удаления продукта.

| | |
|-------------------------|--|
| Тестовый пример | ТП_П_3 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование удаления продукта со склада магазина |
| Краткое изложение теста | Проверка функции удаления продукта со склада магазина. |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Меню” -> “Удаление продукта со склада магазина” 2. Выбрать товар 3. Нажать кнопку “Удалить товар” |
| Тестовые данные | Хлеб, Крупка «Манная», Сыр «Моцарелла» |
| Ожидаемый результат | Товар будет удалён |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Удаления товара со склада магазина. |
| Комментарии | |

Таблица 5 – Тест возврата продукта.

| | |
|-------------------------|---|
| Тестовый пример | ТП_П_4 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование возврата продукта в магазин |
| Краткое изложение теста | Проверка функции возврата продукта в магазин. |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Меню” -> “Генерация формы возврата” 2. Выбрать товар 3. Нажать кнопку “Вернуть продукт” |
| Тестовые данные | Хлеб, Крупка «Манная», Сыр «Моцарелла» |
| Ожидаемый результат | Товар будет возвращён |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Возврат ранее купленного товара, в тот магазин где он был куплен. |
| Комментарии | |

Тест 6 – Тест добавления продукта.

| | |
|-------------------------|--|
| Тестовый пример | ТП_П_5 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование добавления товара в базу системы |
| Краткое изложение теста | Проверка функции добавления товара в базу данных системы |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Панель администратора” -> “Добавление товара в базу данных системы” 2. Заполнить все поля тестовыми данными 3. Нажать кнопку “Создать продукт” |
| Тестовые данные | Хлеб / Хлебобулочное изделие / 24 / 12, Крупка «Манная» / Каша / 36 / 18, Сыр «Моцарелла» / Молочный продукт / 28 / 14 |
| Ожидаемый результат | Товар будет создан |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Создать товар, и добавить его в базу данных системы. |
| Комментарии | |

Тест 7 – Тест добавления сотрудника.

| | |
|-------------------------|--|
| Тестовый пример | ТП_П_6 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование добавления сотрудника |
| Краткое изложение теста | Проверка функции добавления сотрудника в систему |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Панель администратора” -> “Добавить сотрудника” 2. Заполнить все поля тестовыми данными 3. Нажать кнопку “Создать аккаунт” |
| Тестовые данные | Test1 / pass1 / 3, Test2 / pass2 / 4, Test3 / pass3 / 5 |
| Ожидаемый результат | Аккаунт будет создан |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Создать аккаунт в с данными для входа. |
| Комментарии | |

Таблица 8 – Тест функции добавления продукта.

| | |
|-------------------------|---|
| Тестовый пример | ТП_П_7 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование добавления магазина |
| Краткое изложение теста | Проверка функции добавления магазина в систему |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Панель администратора” -> “Добавить магазин” 2. Заполнить все поля тестовыми данными 3. Нажать кнопку “Создать магазин” |
| Тестовые данные | Name1 / location1 / 2, Name2 / location2 / 4, Name3 / location3 / 2 |
| Ожидаемый результат | Магазин будет создан |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Создать новый магазин в системе. |
| Комментарии | |

Таблица 9 – Тест обновления данных сотрудника.

| | |
|-------------------------|---|
| Тестовый пример | ТП_П_8 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование обновления данных сотрудника |
| Краткое изложение теста | Проверка функции обновления данных сотрудника |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Панель администратора” -> “Обновить данные сотрудника” 2. Заполнить все поля тестовыми данными 3. Нажать кнопку “обновить данные” |
| Тестовые данные | 1 / Name1 / pass1 / 1, 2 / Name2 / pass2 / 2, 3/ Name3 / pass3 / 3 |
| Ожидаемый результат | Данные будут обновлены |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Обновить данные о каком либо пользователе. |
| Комментарии | |

Таблица 10 – Тест обновления данных магазина.

| | |
|-------------------------|---|
| Тестовый пример | ТП_П_9 |
| Приоритет тестирования | Высокий |
| Название теста | Тестирование обновления данных магазина |
| Краткое изложение теста | Проверка функции обновления данных магазина |
| Этапы теста | 1. Авторизироваться 1. Перейти в раздел “Панель администратора” -> “Обновить данные магазина” 2. Заполнить все поля тестовыми данными 3. Нажать кнопку “обновить данные” |
| Тестовые данные | 1 / Name1 / location1 / 4, 2 / Name2 / location2 / 2, 1 / Name3 / location3 / 4 |
| Ожидаемый результат | Данные будут обновлены |
| Фактический результат | Соответствует ожидаемому |
| Статус | Зачет |
| Предварительное условие | Запущенное приложение |
| Постусловие | Обновить данные о каком-либо магазине. |
| Комментарии | |