

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛО-
ГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Программной инженерии
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема «Система Управления Складом»

Исполнитель
студент 2 курса группы 4 Кравченко Н.В.
(Ф.И.О.)

Руководитель работы ассистент Северинчик Н.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____
Председатель Пацей Н.В.
(подпись)

Утверждаю
Заведующий кафедрой ПИ
_____ Н.В. Пацей
подпись инициалы и фамилия
“ ” 2021г.

к курсовому проектированию

по дисциплине «Объектно-ориентированное программирование»

Специальность: 1-40 01 01 Программное обеспечение Информационных технологий Группа: 4

Студент: Кравченко Н.В.

Тема: Программное средство «Система Управления Складом»

1. Срок сдачи студентом законченной работы: "20 мая 2021 г."

2. Исходные данные к проекту:

2.1. Функционально ПС поддерживает:

- **Функции администратора сервиса:**
 - Поддерживать работу с базой данных;
 - Регистрировать пользователей приложения;
- **Функции клиента:**
 - Выполнять регистрацию и авторизацию (с помощью адреса электронной почты и пароля);
 - Отслеживание количество товаров на складе, их названия, категорий, стоимости, описания, фотографий, заказов;
 - Добавление нового товара, клиента, заказа товара на вход/выход;
 - Обновление товара, клиента
 - Удаление товара, клиента, заказа
 - Поиск товара, клиента

2.2. При выполнении курсового проекта необходимо использовать принципы проектирования ООП. Приложение разрабатывается под ОС Windows и представляет собой настольное приложение (desktop). Отображение, бизнес-логика должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, взаимодействия разработать на основе UML. Язык разработки проекта – C#. Управление программой должно быть интуитивно понятным и удобным. При разработке использовать несколько наиболее подходящих шаблонов проектирования ПО.

3. Содержание расчетно-пояснительной записки

(перечень вопросов, подлежащих разработке)

- Введение

- Постановка задачи и обзор литературы (алгоритмы решения, обзор прототипов, актуальность задачи)
- Проектирование архитектуры проекта (структура модулей, классов).
- Разработка функциональной модели и модели данных ПС (выполняемые функции)
- Тестирование
- Заключение
- Список используемых источников
- Приложения

4. Форма представления выполненной курсовой работы:

- Теоретическая часть курсового проекта должны быть представлены в формате docx. Оформление записки должно быть согласно выданным правилам.
- Листинги программы представляются частично в приложении.
- Пояснительную записку, листинги, проект (инсталляцию проекта) необходимо загрузить диск, указанный преподавателем.

1.1.1.1 Календарный план

п/п	Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание
1	Введение	20.02.2021	
2	Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования	12.03.2021	
3	Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных)	26.03.2021	
4	Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса	02.04.2021	
5	Кодирование программного средства	23.04.2021	
6	Тестирования и отладка программного средства	30.04.2021	
7	Оформление пояснительной записки	07.05.2021	
	Сдача проекта	20.05.2021	

5. Дата выдачи задания 20.02.2021

Руководитель _____
(подпись)

Н.А. Северинчик

Задание принял к исполнению _____
(дата и подпись студента)

20.02.2021

Содержание

Введение	3
1. Постановка задачи и анализ прототипов.....	4
1.2 Средства разработки.....	4
1.3 Обзор прототипов и аналогов.....	5
2. Разработка функциональных требований проекта.....	10
1.4 Обобщенная структура.....	10
1.5 Диаграмма UML.....	11
1.6 Диаграмма классов	12
1.7 Проектирование базы данных	16
3. Проектирование программного средства	19
3.1 Описание функций приложения и MVVM	19
3.2 Основные окна приложения	21
3.2.1 Главное окно приложения	21
3.3 Основные страницы приложения.....	21
3.3.1 Страница авторизации	21
3.3.2 Страница регистрации	22
3.3.3 Страница клиентов	23
3.3.4 Страница товаров.....	23
3.3.5 Страница заказов	24
3.3.6 Страница дополнительной информации о заказе.....	25
4. Создание(реализация) программного средства	26
5. Тестирование, проверка работоспособности и анализ полученных результатов	27
6. Руководство по установке и использования	36
Заключение	38
Список используемых источников.....	39
Приложение А.....	41
Приложение Б.....	41
Приложение В.....	43
Приложение Г.....	46
Приложение Д.....	48

Введение

С каждым годом различные отрасли промышленности растут в достаточно быстрой прогрессии. В целом система управления товарами используется в различных отраслях промышленности, от производства до коммунальных услуг, здравоохранения, образования, государственного управления и т.д.

Моё приложение инвентаризации будет процессом организации и хранения достаточного количества продуктов на складе для удовлетворения спроса, а также контроля потока и поддержания запасов, чтобы гарантировать, что нужное количество инвентаря доступно в нужное время и нужного качества.

Это приложение инвентаризации позволяет пользователю управлять подсчетом товаров и их движением, заказами на их покупку и продажу, импортом и экспортом, управлять поставщиками и клиентами в любом бизнесе.

Это поможет пользователю сделать многие из этих процессов автоматическими без необходимости запоминать детали, связанные с заказом, обработкой, комплектованием и упаковкой.

Целью курсового проекта является: развитие практических навыков вёрстки приложений на базе WPF с использованием языка разметки — XAML, и языка программирования — C#, показать навыки разработки дизайна, макета для приложения и умение грамотно разделить информацию для пользователя с использованием интересных цветовых сочетаний.

1. Постановка задачи и анализ прототипов

Основной задачей курсового проекта является разработка приложения с возможностью управлять подсчетом товаров и их движением, заказами на их покупку и продажу, импортом и экспортом, управлять поставщиками и клиентами, что поможет пользователям сделать многие из этих процессов автоматическими без необходимости запоминать детали, связанные с заказом.

Функционально данное программное средство должно выполнять следующие задачи:

- выполнять авторизацию;
- поддерживать работу с базой данных;
- управлять клиентами, товарами, заказами, пользователями
- отслеживать количество товаров на складе, их названия, описание, фотографии, категорий, стоимости;
- добавлять новые товары, клиентов, пользователей
- обновлять товары и их данные
- осуществлять удаление товаров
- осуществлять поиск товаров и клиентов

1.2 Средства разработки

При разработке приложения были использованы:

- среда разработки Microsoft Visual Studio 2019;
- технология WPF;
- объектно-ориентированная технология Entity Framework на базе фреймворка .NET
- система управления Microsoft SQL Server.

Microsoft Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки.

Windows Presentation Foundation (WPF) — это система следующего поколения для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем. С WPF можно создавать широкий спектр как автономных приложений, так и приложений, размещенных в веб-обозревателе. в основе WPF лежит векторная система визуализации, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования. WPF расширяет базовую систему полным набором функций разработки приложений, элементами управления, привязкой данных, макетом, графикой, анимацией, стилями, шаблонами, документами, мультимедиа,

текстом и оформлением. Именно использование WPF позволило гибко управлять дизайном интерфейса, также стало возможным подключение различных сторонних пакетов и использование паттернов.

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Microsoft SQL Server — система управления реляционными БД. SQL Server Management Studio – утилита из Microsoft SQL Server, необходимая для управления и администрирования всех компонентов Microsoft SQL Server. Она в себя включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера. SQL является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных.

1.3 Обзор прототипов и аналогов

В интернете есть десятки программ, которые позволяют предпринимателям вести учет в рознице, но большинство из них являются «сырыми» или мало функциональными.

Покупая случайную учетную программу для склада, есть риск, что она:

- неудобна для освоения;
- не поддерживает работу с имеющимся на складе оборудованием;
- не содержит критичного для работы функционала;
- не имеет круглосуточной службы поддержки
- раскрывает достаточные возможности только на максимальном тарифе.

Ниже перечислены критерии выбора программ учёта, которые нужно учесть предпринимателю:

1. Перечень поддерживаемых операций. Кому-то нужно знать просто приход/расход, а для кого-то важен дополнительно ценовой учет и аналитика продаж.

2. Стоимость внедрения и сопровождения. Нет смысла подробно вникать в обзор программы складского учета, если предприниматель не готов платить минимальный ежемесячный платеж.

3. Круглосуточная техническая поддержка.

4. Наличие дополнительных опциональных модулей (CMS, бухгалтерских, логистических). При планировании дисконтной системы CMS просто необходима.

5. Сетевые возможности. Например, для распределенных в пространстве складов будет актуальна только облачная программа складского учета.

6. Простота освоения. Новый сотрудник должен за несколько часов освоить основные возможности программы.

7. Стабильность работы. Программа не должна подвисать и перезагружаться, потому что это может приводить к потере последних введенных данных.

8. Наличие полнофункциональной демо-версии. Гораздо проще выбрать программу, загрузив ее полнофункциональную версию и испытав возможности

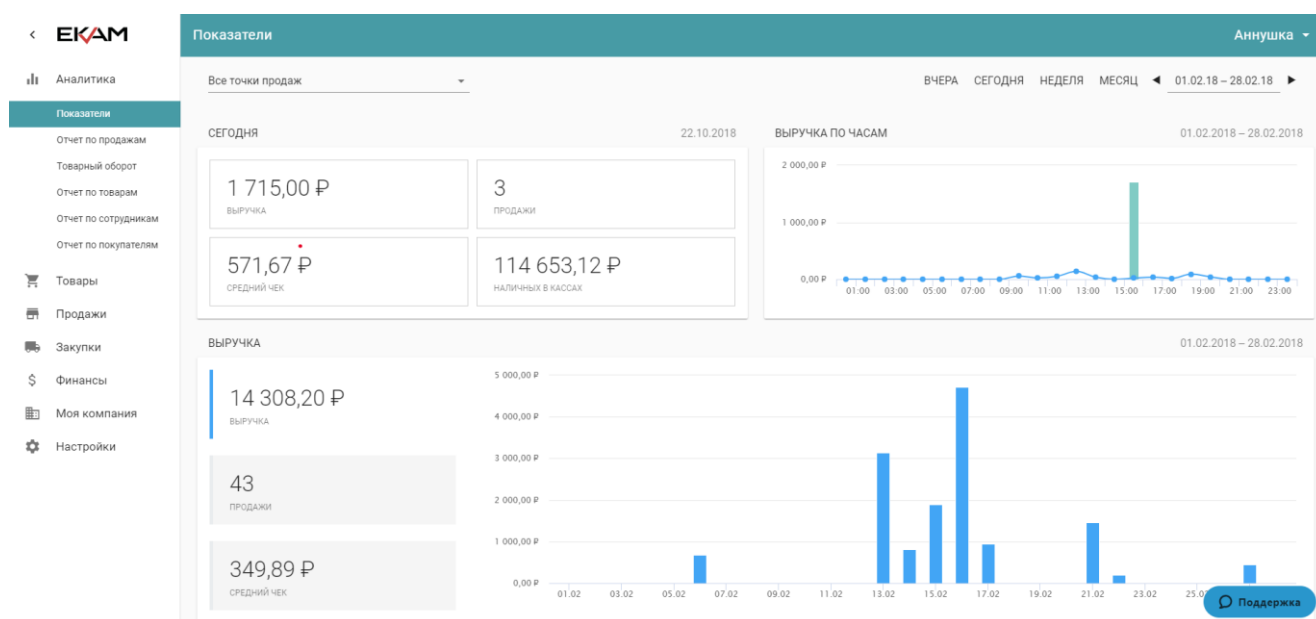
9. Открытый API, позволяющий дорабатывать программу под индивидуальные потребности клиента.

10. Удобный интерфейс. Переключение между меню во время работы должно занимать у персонала минимум времени.

Далее я приведу примеры программ для розничной торговли в разрезе ответственности их перечисленным критериям.

«ЕКАМ»

Товароучетная система «ЕКАМ» является одним из лидеров на СНГ рынке в своей нише и занимает высокие позиции в рейтингах.



Интерфейс программы складского учета «ЕКАМ» (рис. 1.1)

Программа работает через «облако» и предназначена для комплексной автоматизации малого и среднего бизнеса в сфере торговли и оказания услуг. «ЕКАМ» позволяет автоматизировать следующие операции:

- складской учет;
- продажи;
- закупки;
- управленческая отчетность;

- создание программ лояльности и дисконтных программ;
- прием и выполнение заказов в кафе и ресторанах;
- движение денег.
-

Также программа складского учета «ЕКАМ» интегрируется с бухгалтерскими приложениями, упрощая составление налоговой отчетности.

Плюсы программы складского учета «ЕКАМ»:

1. Оперативный и точный контроль за складскими остатками.
2. Автоматическая установка продажной цены, исходя из заданной наценки.
3. Удобное и быстрое проведение инвентаризации.
4. Интеллектуальная система закупок.
5. Удаленная работа с программой.
6. Большое разнообразие настраиваемых аналитических, финансовых, товарных и управленческих отчетов.
7. Возможность одновременного учета товаров в нескольких структурных подразделениях.
8. Загрузка прайсов поставщиков в программу, облегчающее создание номенклатуры.
9. Мультиплатформенность: программой можно пользоваться на компьютерах через браузер и на мобильных гаджетах через специальное приложение.
10. Приемлемые для малого бизнеса тарифные планы.
11. Подключение интернет-магазина.
12. Дружелюбный, понятный интерфейс, позволяющий самостоятельно проводить настройку программы и быстро обучать работе с программой новых сотрудников..
13. Наличие практичного модуля клиентской базы с возможностью привязки дисконтных карт.
14. Наличие полнофункциональной демо-версии, позволяющей 14 дней пользоваться всеми возможностями программы.
15. Наличие телефона круглосуточной технической поддержки.
16. Автоматическая передача данных о продажах в 1С.
17. Составление технологических карт для общепита.
18. Высокая стабильность работы
19. Открытый API для индивидуальной настройки под требования клиента.

Минусы «ЕКАМ»:

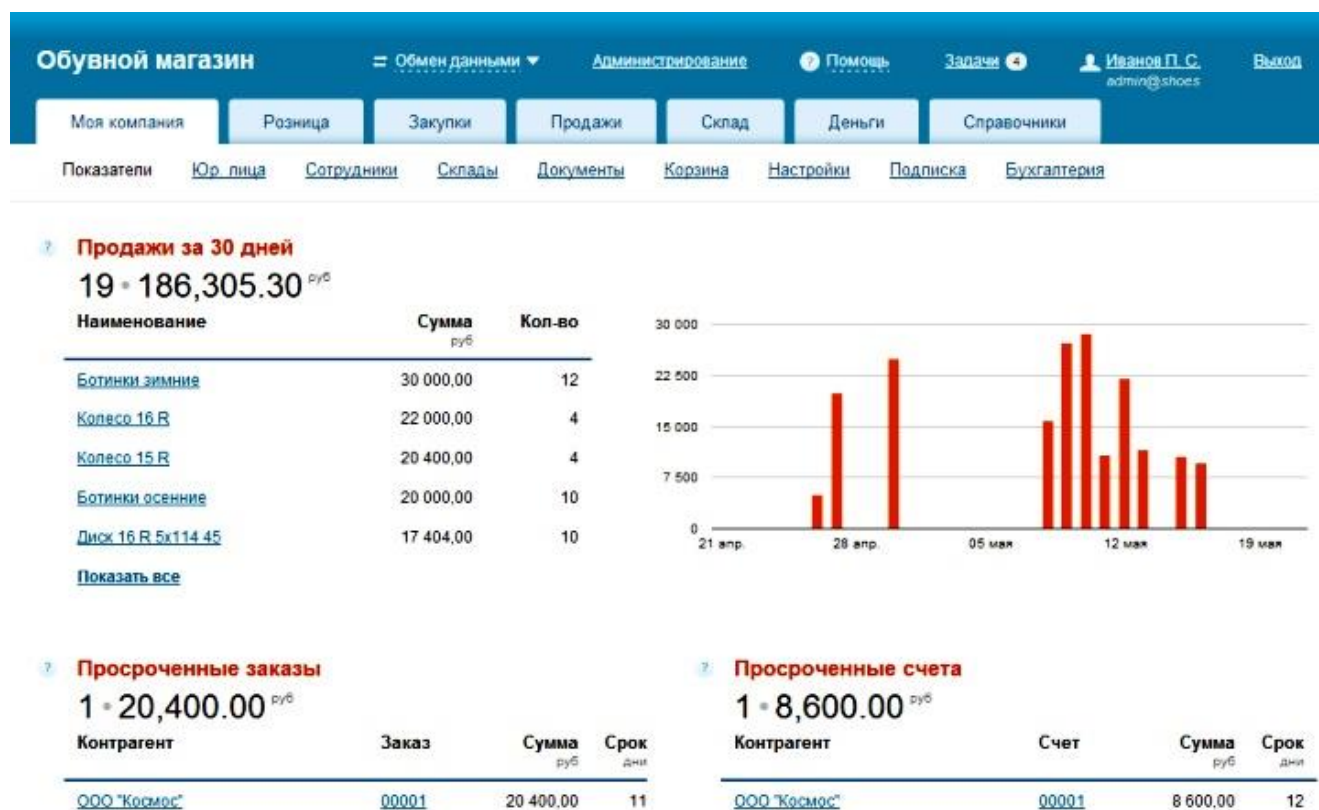
1. Отсутствует функционал по управлению доставкой.
2. Нет блока календаря и задач.

Одним из достоинств «ЕКАМ» является оперативная и профессиональная техническая поддержка. Её специалисты решают проблему как подсказками, так и при помощи удаленного подключения к компьютеру пользователя.

Настройка «ЕКАМ» производится сотрудниками компании. После этого предприниматели могут пользоваться всеми преимуществами приложения.

«МойСклад»

Рейтинг облачной программы складского учета «МойСклад» среди бизнесменов довольно высокий. Это обусловлено стабильностью работы приложения и достаточностью его функций для мелких предпринимателей.



Интерфейс программы складского учета «МойСклад» (рис. 1.2)

Кроме того, разработчики приложения не стали ограничиваться поддержкой торговли и добавили в функционал производственные операции.

Плюсы программы «МойСклад»:

1. Широкий функционал, подходящий для розничной, оптовой торговли, сферы общепита и небольших производств.
2. Поддержка дисконтных карт, создание клиентской базы, формирование воронки продаж.
3. Стабильная работа.
4. Дружелюбный интерфейс и легкость освоения программы новыми сотрудниками.
5. Наличие демо-версии с полным функционалом.
6. Мультиплатформенность: программой можно пользоваться на ОС Windows, macOS, Android, Linux, iOS.

7. Открытый API.

Минусы «МойСклад»:

1. Стоимость ежемесячной абонплаты выше среднерыночной.
2. Отсутствие шаблонов для продаж.
3. Тарифные планы программы складского учета МойСклад
4. Тарифные планы программы складского учета «МойСклад»

Можно сказать, что программа «МойСклад» создавалась для небольших магазинов и компаний, а приобретают ее и представители более крупного бизнеса. Но учитывать пожелания и замечания клиентов разработчики не спешат, что приводит к недовольству технической поддержкой и программой в целом.

Проанализировав плюсы и минусы данных товароучетных систем, которые являются одними из лидеров на СНГ рынке в своей нише и занимает высокие позиции в рейтингах я попытался реализовать плюсы и избежать минусы этих приложений.

2. Разработка функциональных требований проекта

1.4 Обобщенная структура

Решение представлено проектом «WarehouseManagementSystem» с имею-щей структурой (Рисунок 2.1.).

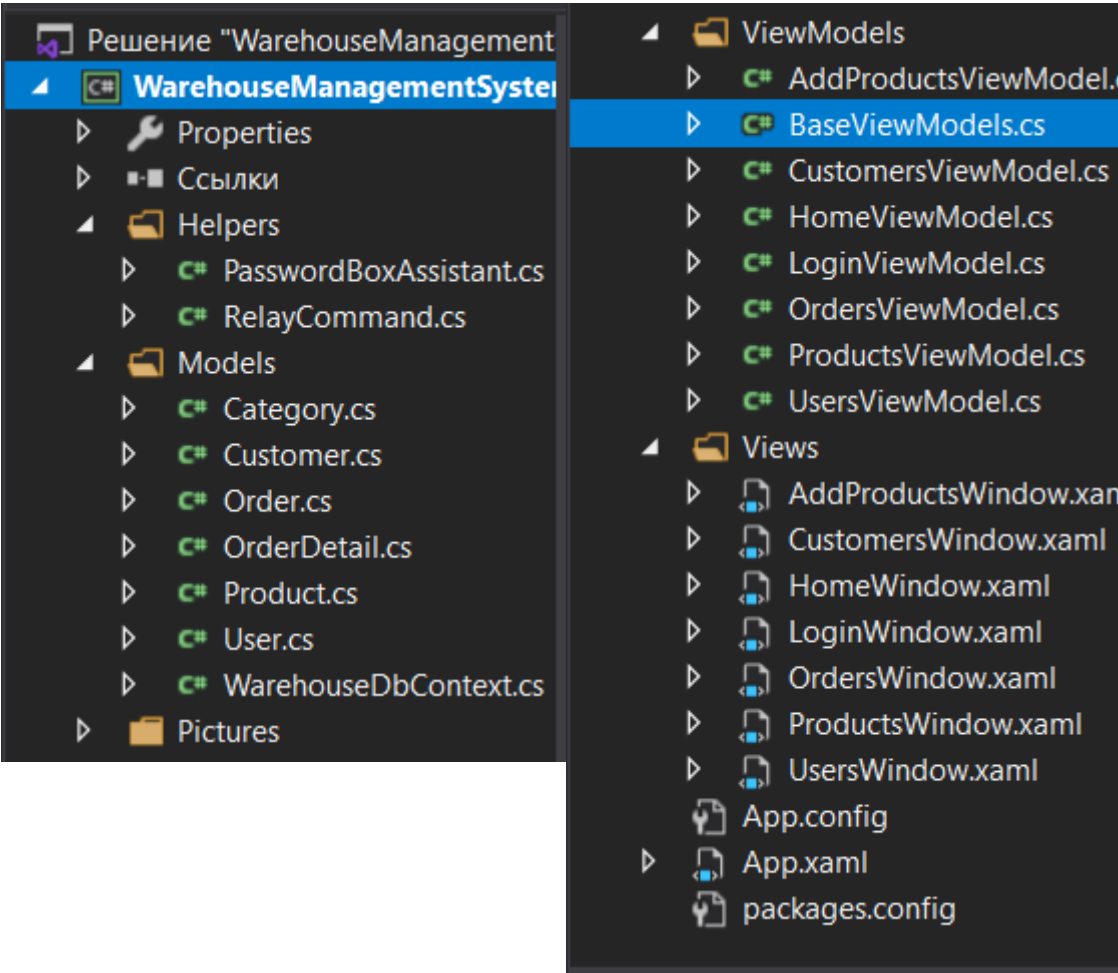


Рисунок 2.1 – Обобщенная структура проекта

Описание обобщенной структуры проекта «WarehouseManagementSystem» представлено в таблице 2.1.

Таблица 2.1 – Обобщенная структура проекта

Объект	Описание
Properties	свойства проекта, содержит информацию о сборке, используемых ресурсах и настройках

Ссылки	перечень сборок, используемых в проекте
Папка Helpers	привязка(соединение) команды с имеющей к ней прикладной логикой, отвечающей за обслуживание интерфейса
Папка Models	описывает используемые в приложении данные
Папка Pictures	содержит основные картинки проекта
Папка ViewModels	Связывает модель и представление, определяет логику по обновлению данных в модели
Папка Views	определяет визуальный интерфейс, через который пользователь взаимодействует с приложением
App.config	файл с параметрами проекта
App.xaml	класс Application
packages.config	файл для поддержки списка пакетов, на которые ссылается проект

1.5 Диаграмма UML

UML — унифицированный язык моделирования (Unified Modeling Language) — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования системного проектирования и отображения организационных структур.

В данной программе существует две разновидности ролей пользователя. На диаграмме вариантов использования, представлены роли администратора и пользователя (Рисунок 2.2.).

Пользователю предоставляется выполнять авторизацию (с помощью адреса электронной почты и пароля), управление клиентами, товарами, заказами, отслеживание количества товаров на складе, их названия, категорий, стоимости, описания, фотографий, добавление нового товара, категории, клиента, обновление товара, удаление товара, поиск товара.

Администратору предоставляется поддерживать работу с базой данных и регистрировать пользователей приложения.

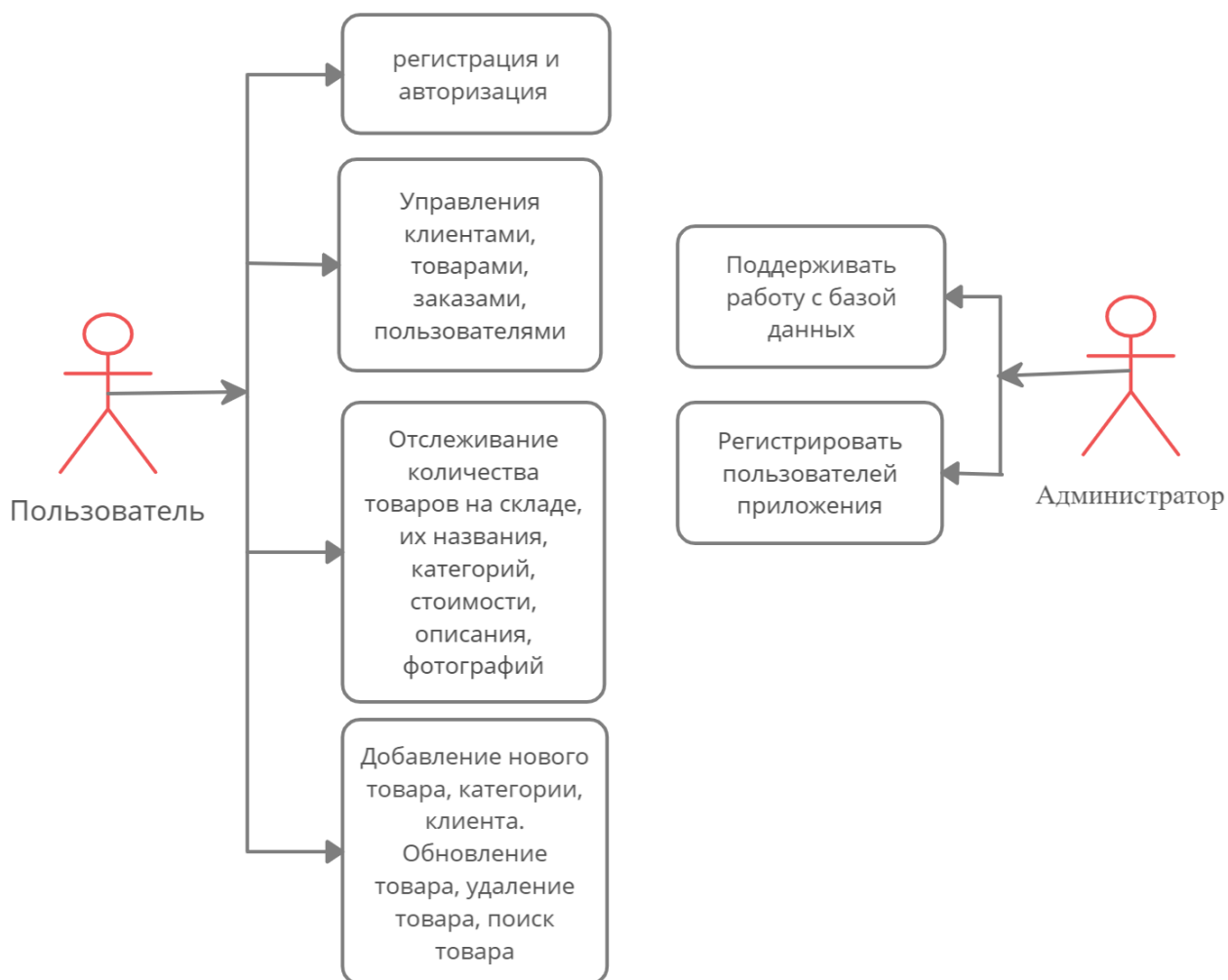


Рисунок 2.2 – Диаграмма вариантов использования

1.6 Диаграмма классов

Структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов, методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

На данной диаграмме классов изображены классы программы, методы, поля и свойства классов. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы) и типы отношений (наследование, реализация интерфейсов).

Диаграмма классов моделей представления (связывает модель и представление через механизм привязки данных) представлена на рисунке 2.3.

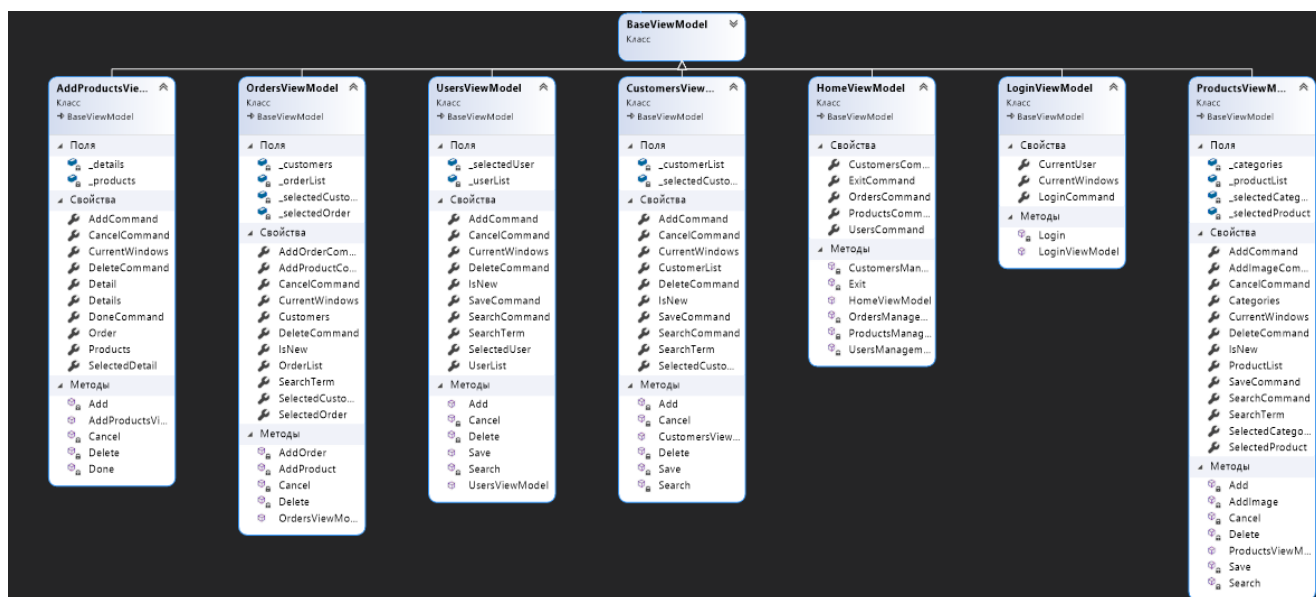


Рисунок 2.3 – Диаграмма классов модели представления.

Диаграмма классов представления (определяют визуальный интерфейс) представлена на рисунке 2.4.

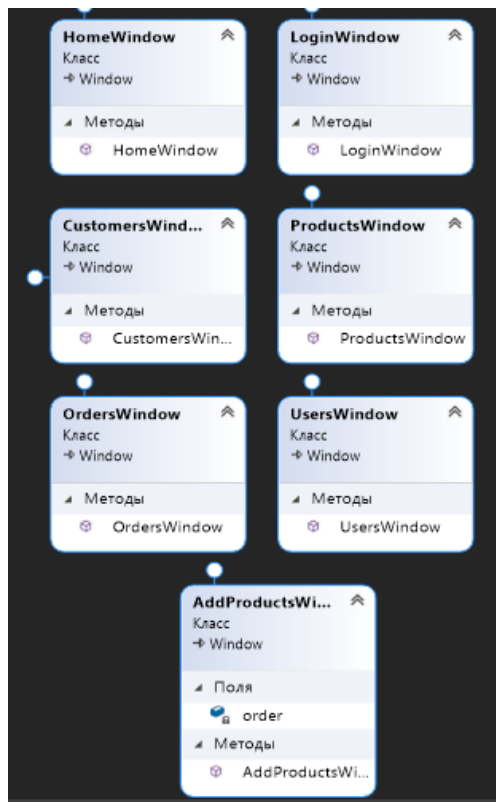


Рисунок 2.4 – Диаграмма классов представления.

Диаграмма классов моделей (описывает используемые в приложении данные) представлена на рисунке 2.5.

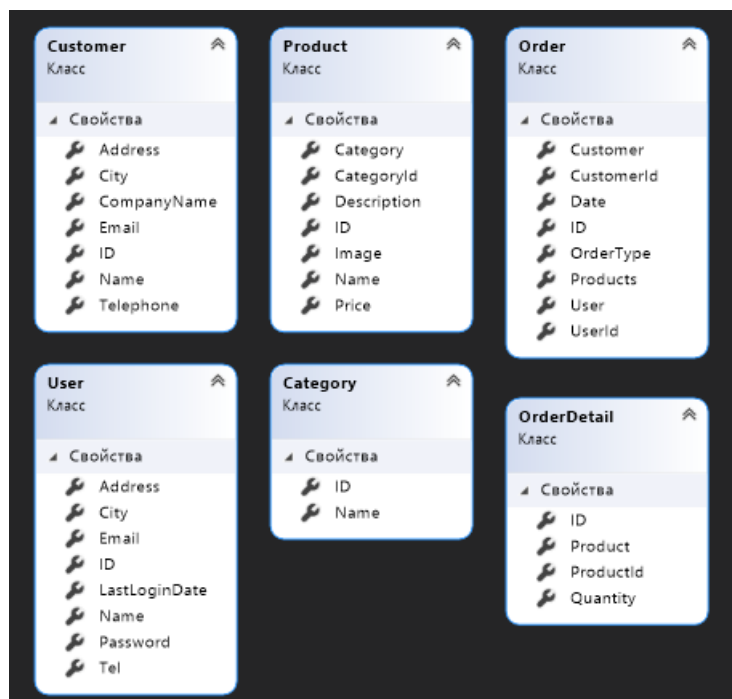


Рисунок 2.5 – Диаграмма классов моделей.

Диаграмма классов ассистирования пароля и привязки команды с имеющей к ней прикладной логикой, отвечающей за обслуживание интерфейса представлена на рисунке 2.6.

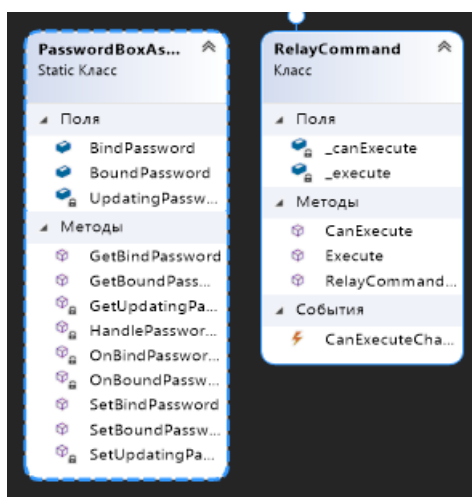


Рисунок 2.6 – Диаграмма классов ассистирования пароля и привязки команды с прикладной логикой.

Диаграмма контекста базы данных 2.7.

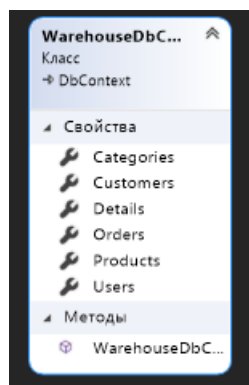


Рисунок 2.7 – Диаграмма контекста базы данных.

Диаграмма классов ресурсов и настроек рисунке 2.8.

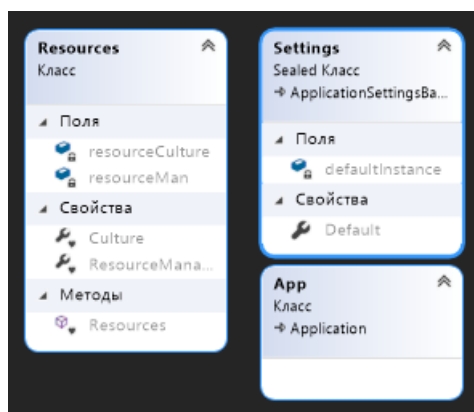


Рисунок 2.8 – Диаграмма классов ресурсов и настрое.

1.7 Проектирование базы данных

Данные для работы будут храниться в базе данных. Её создание происходит с помощью Microsoft SQL Server — это система управления реляционными базами данных, которая подключается к приложению.

При разработке приложения сначала разрабатывалась база данных, которая должна содержать всю нужную информацию для приложения, после программное обеспечение. База данных состоит из 6 таблиц, между которыми установлены связи один ко многим. (Рисунок 2.9.)

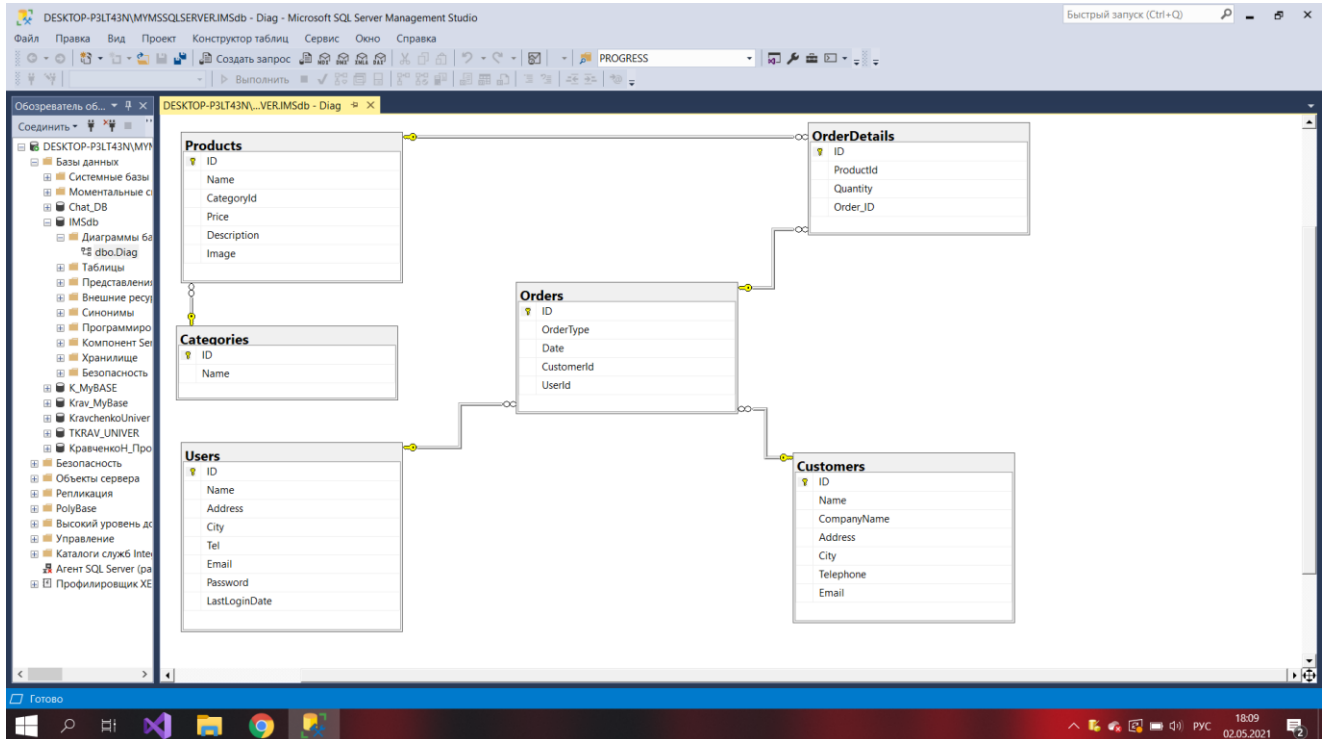


Рисунок 2.9 – Диаграмма базы данных.

Таблица Products предназначена для хранения информации о добавленных на склад продуктов, состоит из 6 столбцов:

- ID - уникальный идентификатор продукта;
- Name - название продукта;
- Category - категория продукта;
- Price - цена за единицу продукта;
- Description - описание продукта;
- Image - изображение продукта;

Таблица Category предназначена для хранения информации о категориях добавленных на склад продуктов, состоит из 2 столбцов:

- ID - уникальный идентификатор категории продукта;
- Name - название категории продукта;

Таблица Orders предназначена для хранения информации о добавленных заказах продуктов на (вход/выход) со склада, состоит из 5 столбцов:

- ID - уникальный идентификатор заказа продукта;
- OrderType - тип заказа (вход/выход продукта);
- Date - дата заказа на вход/выход продукта;
- CustomerId - уникальный идентификатор заказчика;
- UserId - уникальный идентификатор пользователя;

Таблица OrderDetails предназначена для хранения дополнительной информации заказов продуктов на (вход/выход) со склада, состоит из 4 столбцов:

- ID - уникальный идентификатор дополнительной информации о заказах продуктов на (вход/выход) со склада
- ProductId - уникальный идентификатор продукта
- Quantity - количество продуктов для заказов на (вход/выход) со склада;
- Order_ID - уникальный идентификатор заказа;

Таблица Customers предназначена для хранения информации о заказчиках, состоит из 7 столбцов:

- ID - уникальный идентификатор заказчика;
- Name - имя заказчика;
- CompanyName - название компании заказчика;
- Address - адрес заказчика или офиса его компании;
- City - город заказчика;
- Telephone - мобильный телефон заказчика;
- Email - почтовый адрес заказчика (имэйл);

Таблица Users предназначена для хранения информации о добавленных на склад продуктов, состоит из 8 столбцов:

- ID - уникальный идентификатор пользователя;
- Name – Имя пользователя;
- Address – адрес пользователя;
- City – город пользователя;
- Tel – мобильный телефон пользователя;
- Email – почтовый адрес пользователя(имэйл) для входа в приложение;
- Password – пароль пользователя для входа в приложение;
- LastLoginDate – дата последнего входа в приложение;

Также в каждой таблице для поля ID прописан атрибут Identity, который позволяет сделать столбец идентификатором, значение которого будет автоматически увеличиваться на единицу для каждого следующего значения(строки).

Имя	Тип данных	Допустимы значения NULL	По умолчанию
ID	int	<input type="checkbox"/>	
Name	nvarchar(50)	<input checked="" type="checkbox"/>	
CategoryId	int	<input checked="" type="checkbox"/>	
Price	decimal(18,0)	<input checked="" type="checkbox"/>	
Description	nvarchar(50)	<input checked="" type="checkbox"/>	
Image	image	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	

Проектирование

T-SQL

```

CREATE TABLE [dbo].[Products] (
    [ID] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (50) NULL,
    [CategoryId] INT NULL,
    [Price] DECIMAL (18) NULL,
    [Description] NVARCHAR (50) NULL,
    [Image] IMAGE NULL,
    CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_Products_Categories] FOREIGN KEY ([CategoryId]) REFERENCES [dbo].[Categories] ([ID])
);

```

	Имя	Тип данных	Допустимы значения NULL	По умолчанию
🔑	ID	int	<input type="checkbox"/>	
	Name	nvarchar(50)	<input checked="" type="checkbox"/>	
	Address	nvarchar(50)	<input checked="" type="checkbox"/>	
	City	nvarchar(50)	<input checked="" type="checkbox"/>	
	Tel	nvarchar(50)	<input checked="" type="checkbox"/>	
	Email	nvarchar(50)	<input checked="" type="checkbox"/>	
	Password	nvarchar(50)	<input checked="" type="checkbox"/>	
	LastLoginDate	datetime	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

Проектирование

T-SQL

```
CREATE TABLE [dbo].[Users] (  
    [ID] INT IDENTITY (1, 1) NOT NULL,  
    [Name] NVARCHAR (50) NULL,  
    [Address] NVARCHAR (50) NULL,  
    [City] NVARCHAR (50) NULL,  
    [Tel] NVARCHAR (50) NULL,  
    [Email] NVARCHAR (50) NULL,  
    [Password] NVARCHAR (50) NULL,  
    [LastLoginDate] DATETIME NULL,  
    CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED ([ID] ASC)  
);
```

	Имя	Тип данных	Допустимы значения NULL	По умолчанию
🔑	ID	int	<input type="checkbox"/>	
	Name	nvarchar(50)	<input checked="" type="checkbox"/>	
	CompanyName	nvarchar(50)	<input checked="" type="checkbox"/>	
	Address	nvarchar(50)	<input checked="" type="checkbox"/>	
	City	nvarchar(50)	<input checked="" type="checkbox"/>	
	Telephone	nvarchar(50)	<input checked="" type="checkbox"/>	
	Email	nvarchar(50)	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

Проектирование

T-SQL

```
CREATE TABLE [dbo].[Customers] (  
    [ID] INT IDENTITY (1, 1) NOT NULL,  
    [Name] NVARCHAR (50) NULL,  
    [CompanyName] NVARCHAR (50) NULL,  
    [Address] NVARCHAR (50) NULL,  
    [City] NVARCHAR (50) NULL,  
    [Telephone] NVARCHAR (50) NULL,  
    [Email] NVARCHAR (50) NULL,  
    CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED ([ID] ASC)  
);
```

	Имя	Тип данных	Допустимы значения NULL	По умолчанию
🔑	ID	int	<input type="checkbox"/>	
	ProductId	int	<input checked="" type="checkbox"/>	
	Quantity	int	<input checked="" type="checkbox"/>	
	Order_ID	int	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

Проектирование

T-SQL

```
CREATE TABLE [dbo].[OrderDetails] (  
    [ID] INT IDENTITY (1, 1) NOT NULL,  
    [ProductId] INT NULL,  
    [Quantity] INT NULL,  
    [Order_ID] INT NULL,  
    CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED ([ID] ASC),  
    CONSTRAINT [FK_OrderDetails_Orders] FOREIGN KEY ([Order_ID]) REFERENCES [dbo].[Orders] ([ID]),  
    CONSTRAINT [FK_OrderDetails_Products] FOREIGN KEY ([ProductId]) REFERENCES [dbo].[Products] ([ID])  
);
```

	Имя	Тип данных	Допустимы значения NULL	По умолчанию
🔑	ID	int	<input type="checkbox"/>	
	Name	nvarchar(50)	<input checked="" type="checkbox"/>	
			<input type="checkbox"/>	

Проектирование

T-SQL

```
CREATE TABLE [dbo].[Categories] (  
    [ID] INT IDENTITY (1, 1) NOT NULL,  
    [Name] NVARCHAR (50) NULL,  
    CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED ([ID] ASC)  
);
```

3. Проектирование программного средства

3.1 Описание функций приложения и MVVM

- Выполнять авторизацию (регистрация пользователя происходит под аккаунтом админа уже в самом приложении);
- отслеживать количество товаров на складе, их названия, описание, фотографий, категорий, стоимости;
- добавлять новые товары, клиентов, пользователей, ордера(заказы) на вход/выход продукта со склада и его количество
- осуществлять обновление данных о товаре, заказчике, ордере(заказе), пользователе
- осуществлять поиск товаров, клиентов и пользователей
- осуществлять удаление товаров, клиентов и пользователей

В процессе создания приложения был реализован паттерн MVVM. Паттерн MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения.

MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

Паттерн MVVM в WPF:

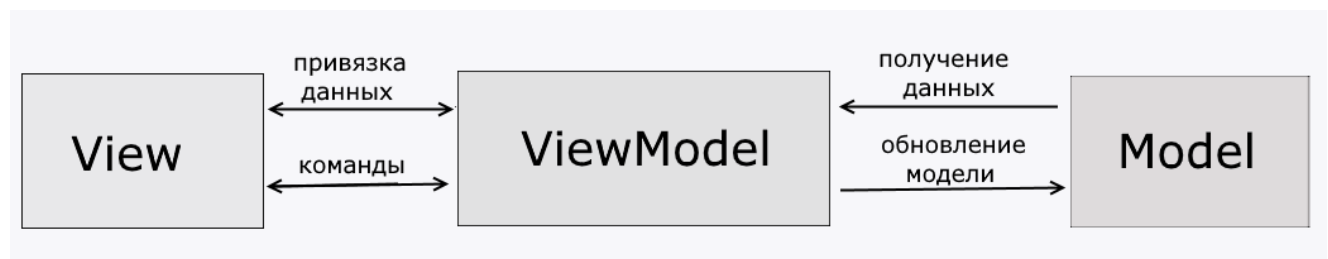


Рисунок 3.1 – схема паттерна MVVM

Model

Модель описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

View

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых

полей и прочих визуальных элементов.

Хотя окно (класс `Window`) в WPF может содержать как интерфейс в xaml, так и привязанный к нему код C#, однако в идеале код C# не должен содержать какой-то логики, кроме разве что конструктора, который вызывает метод `InitializeComponent` и выполняет начальную инициализацию окна. Вся же основная логика приложения выносится в компонент `ViewModel`.

Однако иногда в файле связанного кода все может находиться некоторая логика, которую трудно реализовать в рамках паттерна MVVM во `ViewModel`.

Представление не обрабатывает события за редким исключением, а выполняет действия в основном посредством команд.

ViewModel

`ViewModel` или модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса `INotifyPropertyChanged` автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

`ViewModel` также содержит логику по получению данных из модели, которые потом передаются в представление. И также `ViewModel` определяет логику по обновлению данных в модели.

Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с `ViewModel` посредством команд.

Например, пользователь хочет сохранить введенные в текстовое поле данные. Он нажимает на кнопку и тем самым отправляет команду во `ViewModel`. А `ViewModel` уже получает переданные данные и в соответствии с ними обновляет модель.

Итогом применения паттерна MVVM является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать.

3.2 Основные окна приложения

3.2.1 Главное окно приложения

HomeWindow — основное окно игры. Оно содержит кнопки навигации, а также объект отображения страниц (Рисунок 3.3).

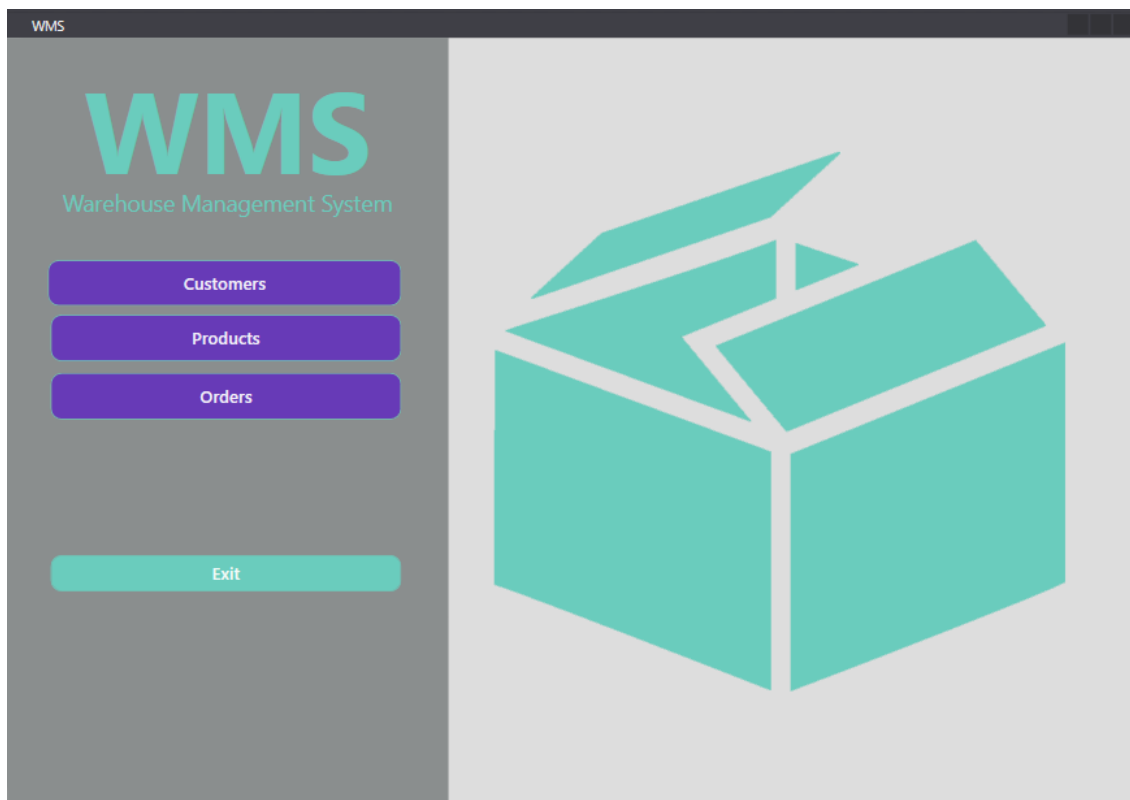


Рисунок 3.3 – главное окно приложения.

3.3 Основные страницы приложения

Навигация по приложению осуществляется с помощью окна HomeWindow. При использовании определённой кнопки меняется страница, реализованная с помощью команд (RelayCommand).

3.3.1 Страница авторизации

Страница авторизации содержит поля Email и Password. Кнопка авторизации не работает до тех пор, пока не будут введены корректные данные в поля. Если при авторизации введены данные админа, то мы попадаем на страницу UserWindow, где далее админ может добавить различных пользователей, которые имеют данные (имя, телефон, адрес, город, email, пароль), используя которые пользователь может потом зайти в приложение.

Рисунок 3.4 – Окно авторизации LoginWindow.

3.3.2 Страница регистрации

Как таковая видимая регистрация в приложении отсутствует. Для регистрации пользователя мы заходим под учётной записью админа, регистрируем пользователя, и далее передаём ему данные для входа в его аккаунт. Данные пользователя могут содержать имя, телефон, адрес, почту, город, пароль. Админ может добавлять и удалять пользователей, а также изменять их данные.

Рисунок 3.5 – Окно пользователей UsersWindow.

3.3.3 Страница клиентов

Страница клиентов позволяет пользователю управлять клиентами. Клиент имеет такие данные: уникальный идентификатор, имя, название компании, адрес, телефон, город, почта. Пользователь может осуществлять поиск клиентов по имени, добавлять клиентов, обновлять данные клиентов и так же удалять клиентов. Для удаления клиента изначально нужно удалить все его заказанные товары на вход/выход со склада и сам заказ.

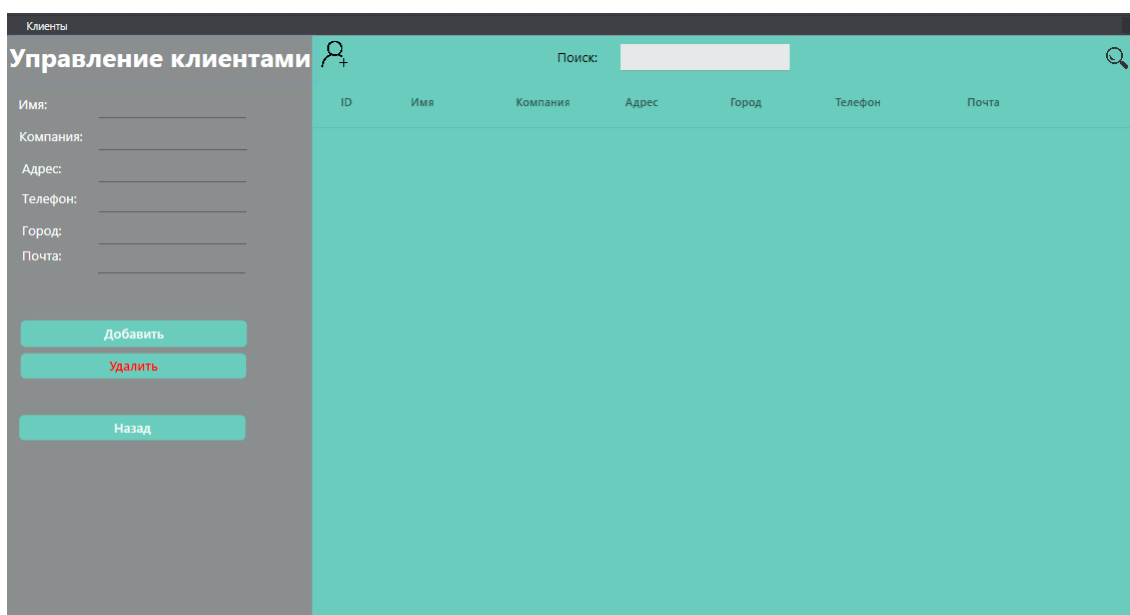


Рисунок 3.6 – Окно клиентов CustomersWindow.

3.3.4 Страница товаров

Страница товаров позволяет пользователю управлять товарами. Продукт имеет такие данные: уникальный идентификатор, название, описание, категория товара, цена, а также его фотография. Пользователь может осуществлять поиск товаров по названию и описанию, добавлять товары, обновлять данные о товарах и так же удалять товары. Для удаления товара изначально нужно удалить все заказы товара на вход/выход со склада и сам заказ.

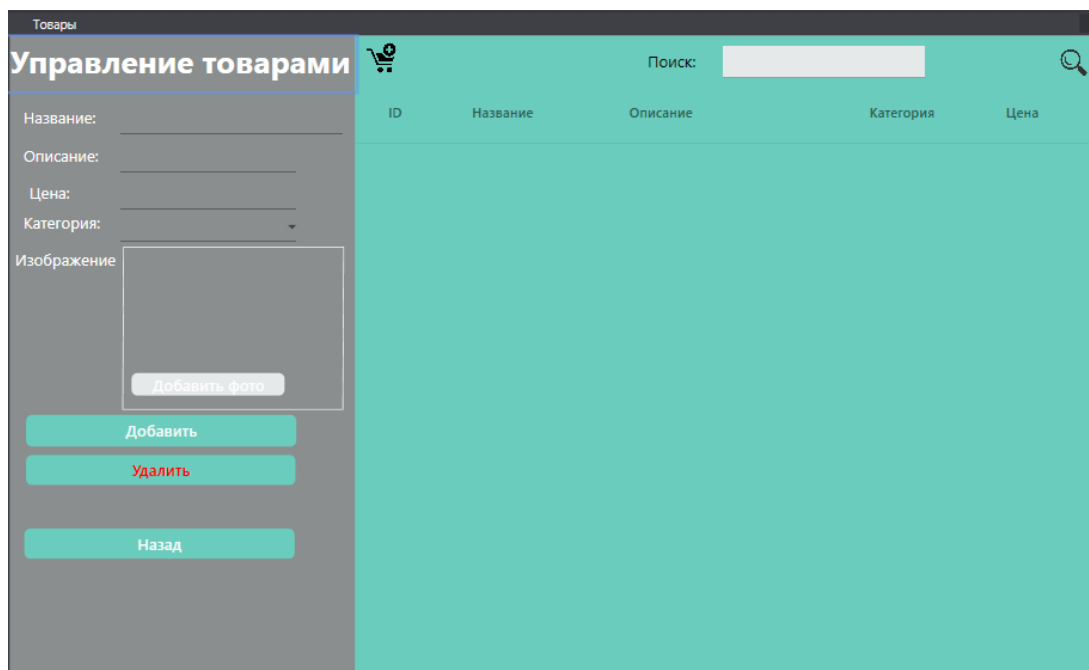


Рисунок 3.7 – Окно товаров ProductsWindow.

3.3.5 Страница заказов

Страница заказов позволяет пользователю добавлять заказы на определённые товары и выбирать их количество. Заказ имеет такие данные: уникальный идентификатор, дата заказа, имя клиента, сделавшего заказ. Пользователь может добавлять заказы и также удалять заказы. Для удаления заказа изначально нужно удалить все заказанные товары на вход/выход со склада этого заказа.

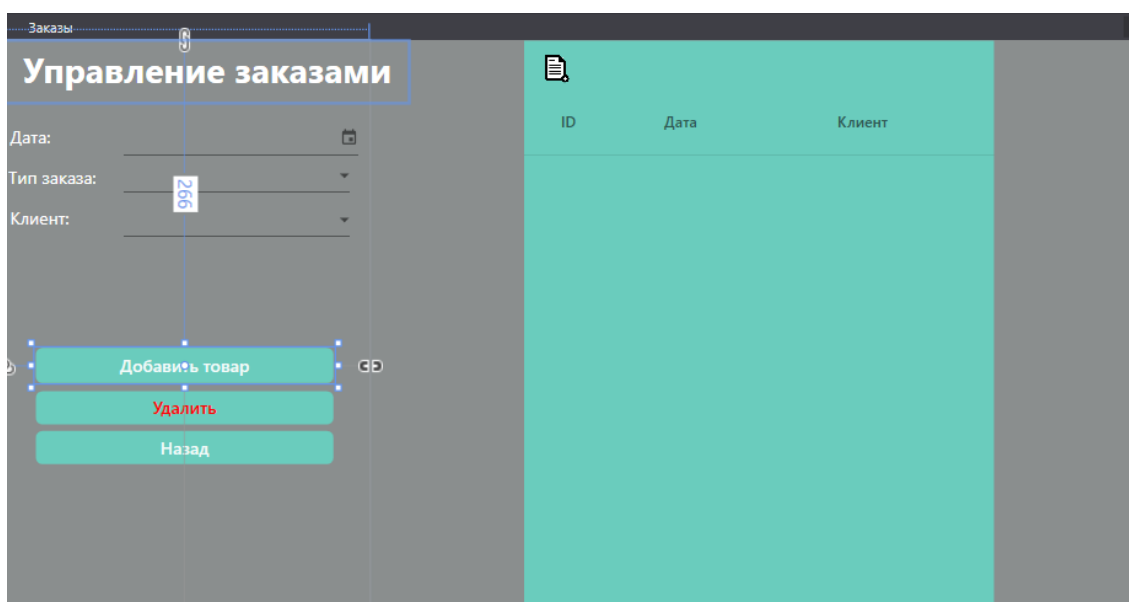


Рисунок 3.8 – Окно заказов OrdersWindow.

3.3.6 Страница дополнительной информации о заказе

Страница дополнительной информации о заказе, где указывается товар и его количество для заказа определённым заказчиком. В окне отображаются такие данные о заказе: название товара, его количество. После добавления товаров и их количества на заказ, при последующей попытке добавления товаров в заказ они добавляться не будут. Для этого нужно создать уже новый заказ.

The screenshot shows a window titled "Добавить товар" (Add Product). On the left, there is a form with a label "Товар:" (Product:) and a dropdown menu, and a label "Количество" (Quantity) with a text input field. Below the form are four buttons: "Добавить" (Add), "Удалить" (Delete), "Выполнить" (Execute), and "Назад" (Back). On the right, there is a table with two columns: "Товар" (Product) and "Количество" (Quantity). The table is currently empty.

Товар	Количество
-------	------------

Рисунок 3.9 – Окно дополнительной информации о заказе AddProductsWindow (выбор продукта и его количество)

4. Создание(реализация) программного средства

Главное окно `HomeWindow` наследуется от `BaseViewModels`, которое в свою очередь реализует интерфейс *`INotifyPropertyChanged`*, для того чтобы объект мог полноценно реализовывать механизм привязки. Когда объект класса изменяет значение свойства, то он через событие `PropertyChanged` извещает систему об изменениях свойства и система обновляет все привязанные объекты.

Код класса `BaseViewModels` представлен в **приложении А**.

В главном окне `HomeWindow` для взаимодействия пользователя и приложения используются команды. В WPF команды представлены интерфейсом `ICommand`.

Для использования команд добавим в наше приложение класс, который назовём `RelayCommand`. Класс реализует два метода:

- `CanExecute` – определяет, может ли команда выполняться
- `Execute` – собственно выполняет логику команды

Код классов `RelayCommand` и `HomeWindow` представлен в **приложении Б**.

Классы `Category`, `Customers`, `Orders`, `OrderDetails`, `Product`, `User` являются классами моделями(описывают используемые в приложении данные), а класс `WarehouseDbContext` представляет собой контекст базы данных

Код классов модели данных и контекста бд представлены в **приложении В**.

Класс `LoginViewModel` представляет собой авторизацию пользователя. В зависимости от введенных данных вход приложения происходит в роли админа или пользователя.

Код класса `LoginViewModel` представлен в **приложении Г**.

Классы `CustomersViewModel`, `OrdersViewModel`, `ProductsViewModel`, `UsersViewModel` и `AddProductsViewModel` управление клиентами, товарами, пользователями и заказами товаров(их добавление, изменение, удаление, поиск). Классы имеют похожую реализацию, поэтому в приложении будет показан только код одного из данных классов.

Код класса `ProductsViewModel` представлен в **приложении Д**.

5. Тестирование, проверка работоспособности и анализ полученных результатов

Программное средство проходило тестирование. Сначала происходила отладка программы в процессе написания кода, после завершения основного кода, во время тестирования на работоспособность, и при наличии ошибок, исправлялся. Таким образом добивалась максимальная работоспособность программы. Пользователю нужно указать на неправильно введенные данные или недочеты, чтобы он мог их исправить.

Результат обработки исключения при авторизации, если не введена почта или пароль, или данные являются неверными (Рисунок 5.1).

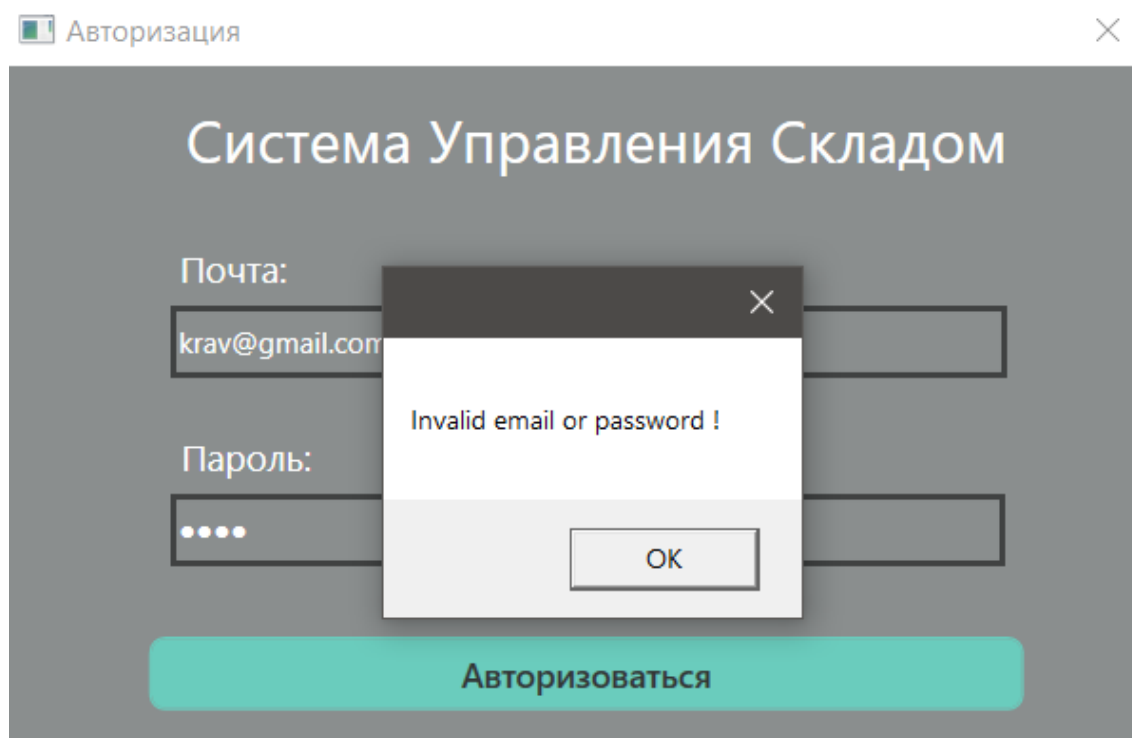


Рисунок 5.1 – Пример проверки авторизации.

Ошибка при регистрации, если введена почта пользователя, уже зарегистрированного в системе ранее исключена, т.к регистрация осуществляется администратором, который имеет доступ к базе данных всех пользователей и проверяет данные на соответствие нормам.

Результат при добавлении клиента (Рисунок 5.2)

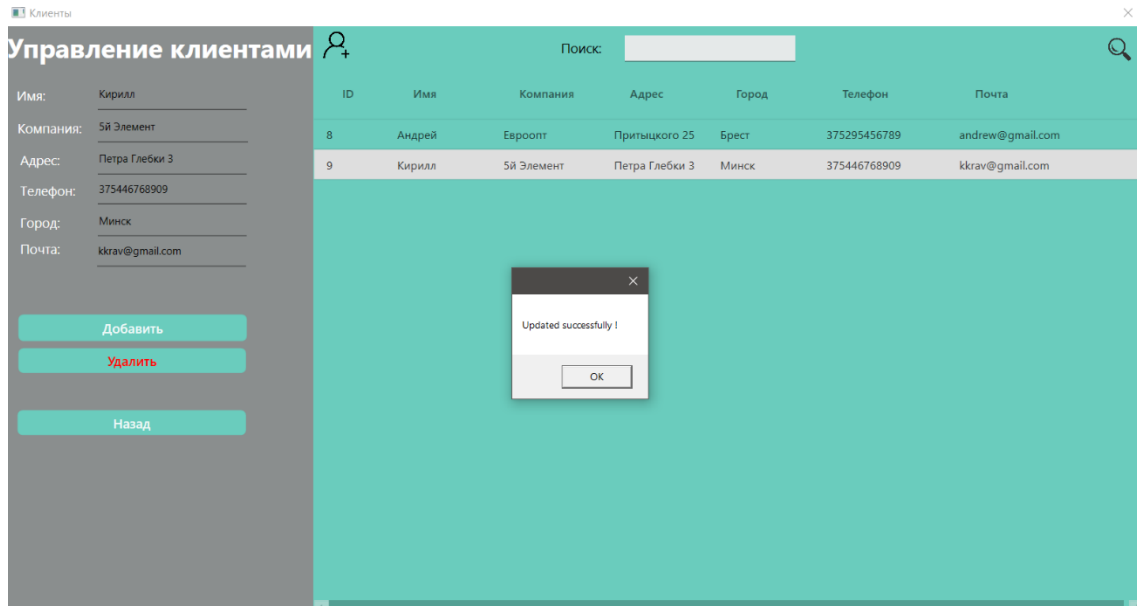


Рисунок 5.2 – Пример теста добавления клиента.

Результат обработки исключения при попытке удалить клиента, если клиент не выбран (Рисунок 5.3).

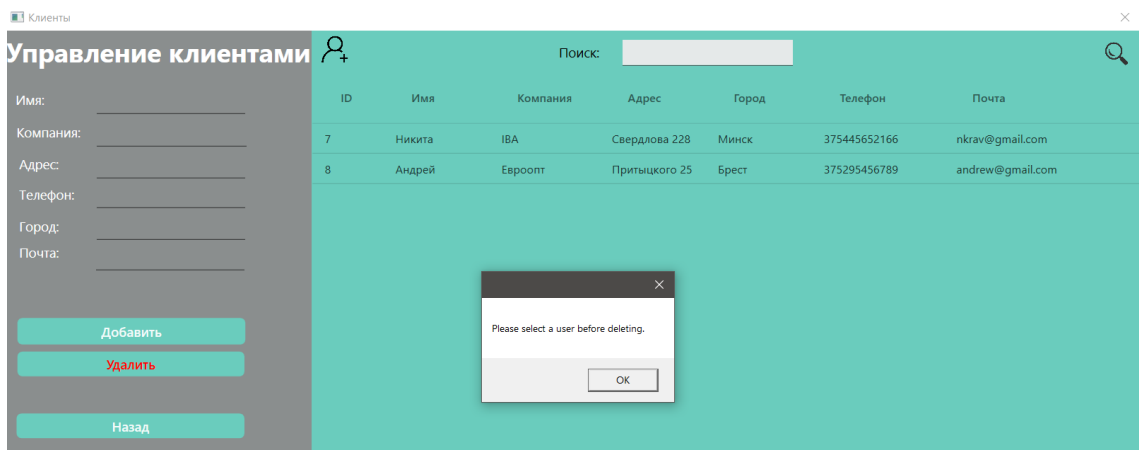


Рисунок 5.3 – Пример теста удаления невыбранного клиента.

Пример теста при попытке удаления клиента (Рисунок 5.4).

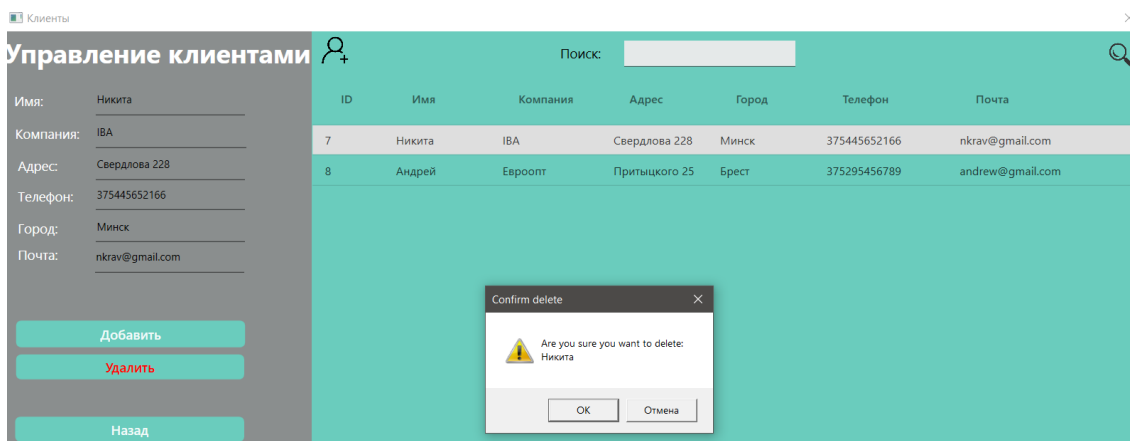


Рисунок 5.4 – Пример теста при удалении клиента.

Результат при добавлении товара (Рисунок 5.5)

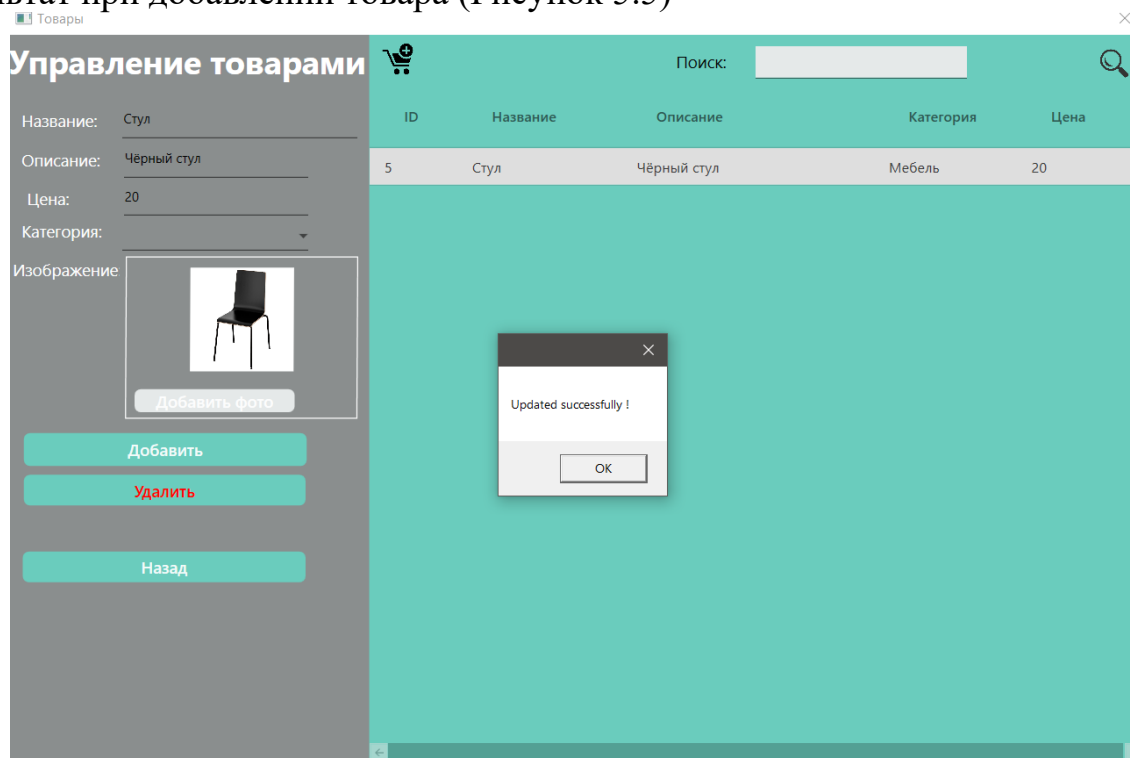


Рисунок 5.5 – Пример теста добавления товара.

Результат обработки исключения при попытке удалить товар, если товар не выбран (Рисунок 5.6).

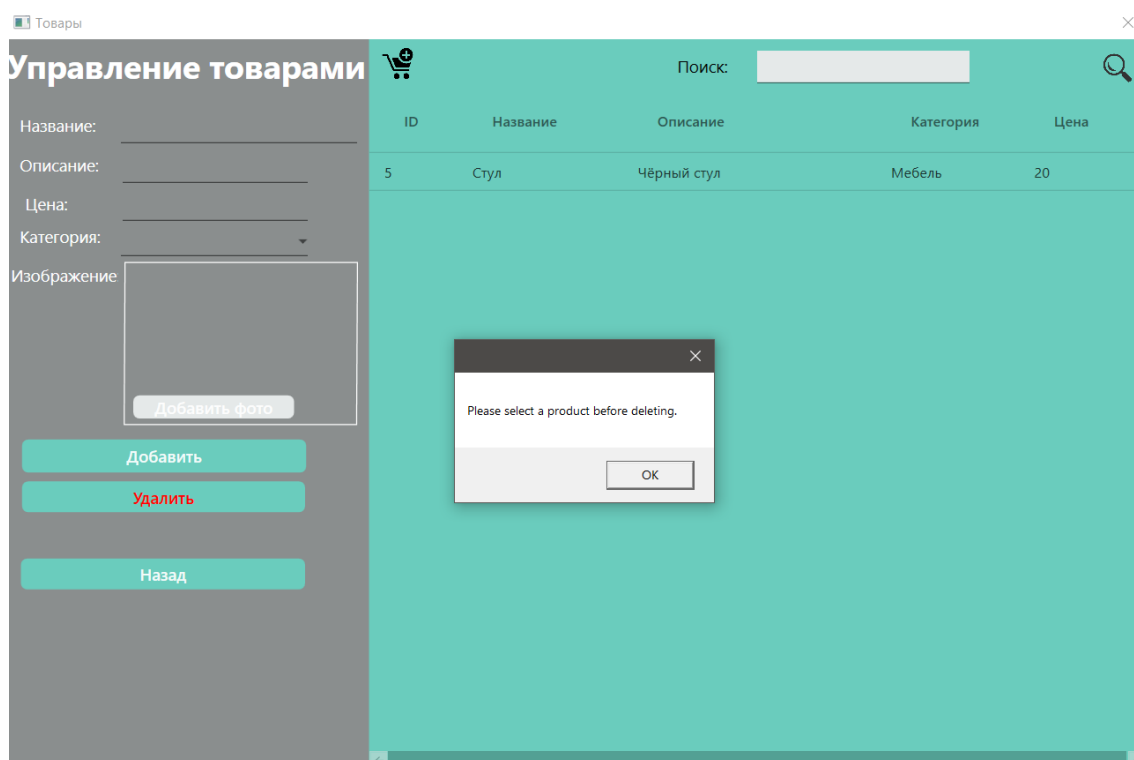


Рисунок 5.6 – Пример теста удаления невыбранного товара.

Пример теста при попытке удаления товара (Рисунок 5.7).

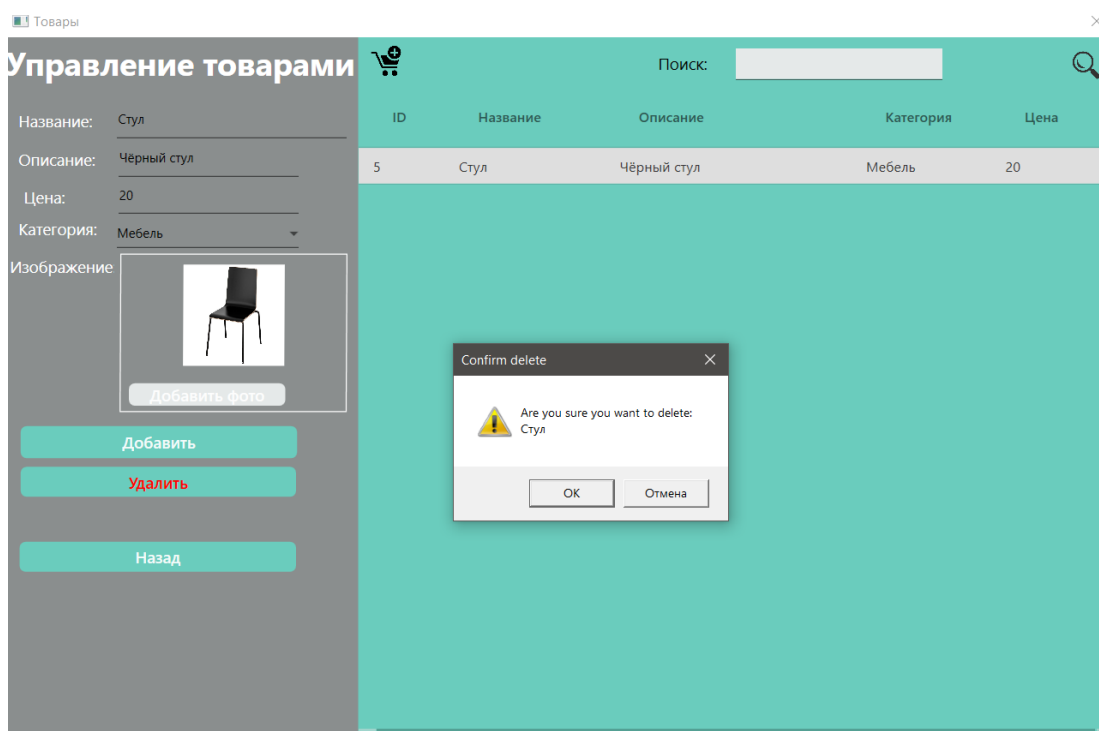


Рисунок 5.7 – Пример теста при удалении товара.

Результат при добавлении заказа (Рисунок 5.8)

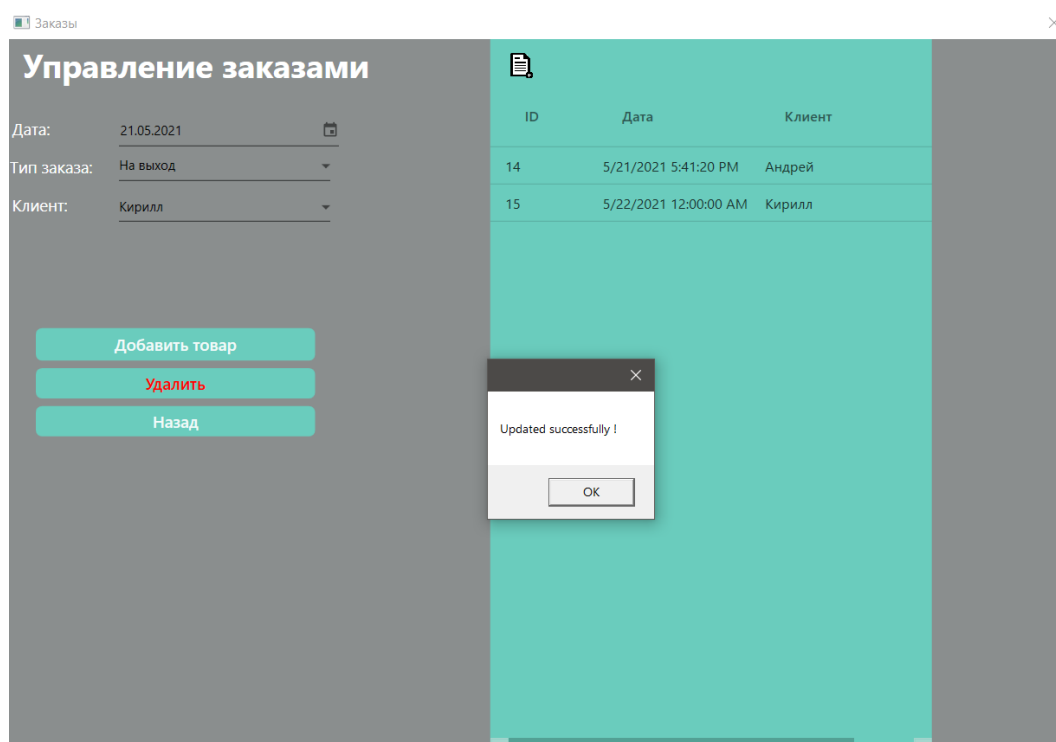


Рисунок 5.8 – Пример теста добавления заказа.

Результат обработки исключения при попытке удалить заказ, если заказ не выбран (Рисунок 5.9).

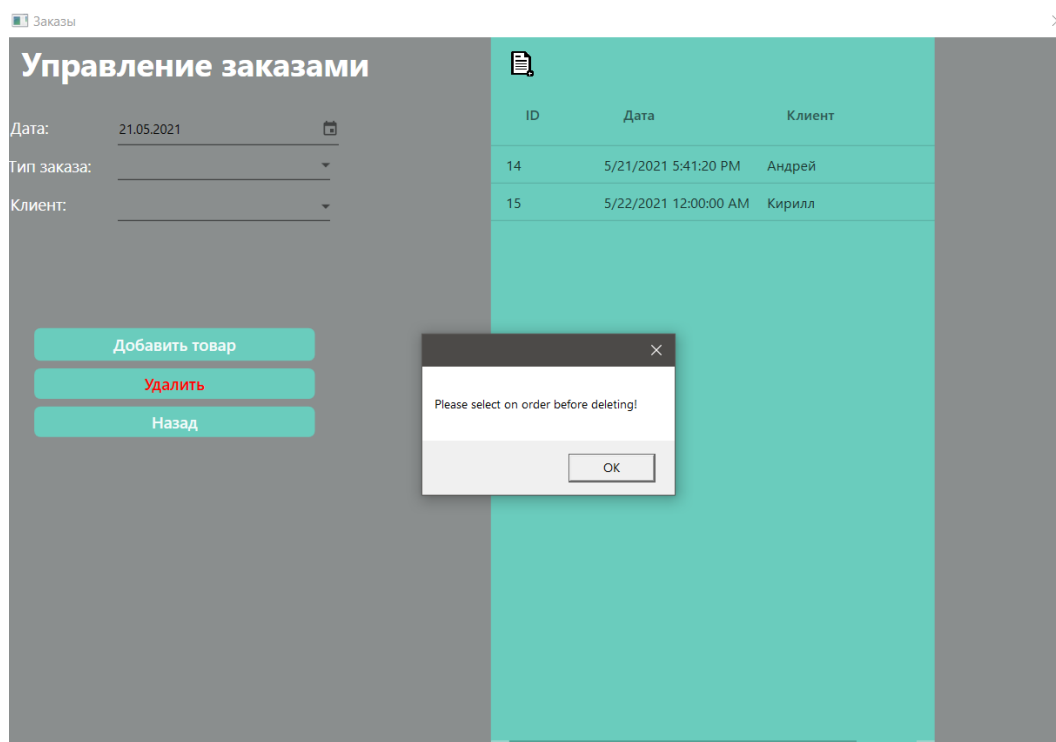


Рисунок 5.9 – Пример теста удаления невыбранного заказа.

Пример теста при попытке удаления заказа (Рисунок 5.10).

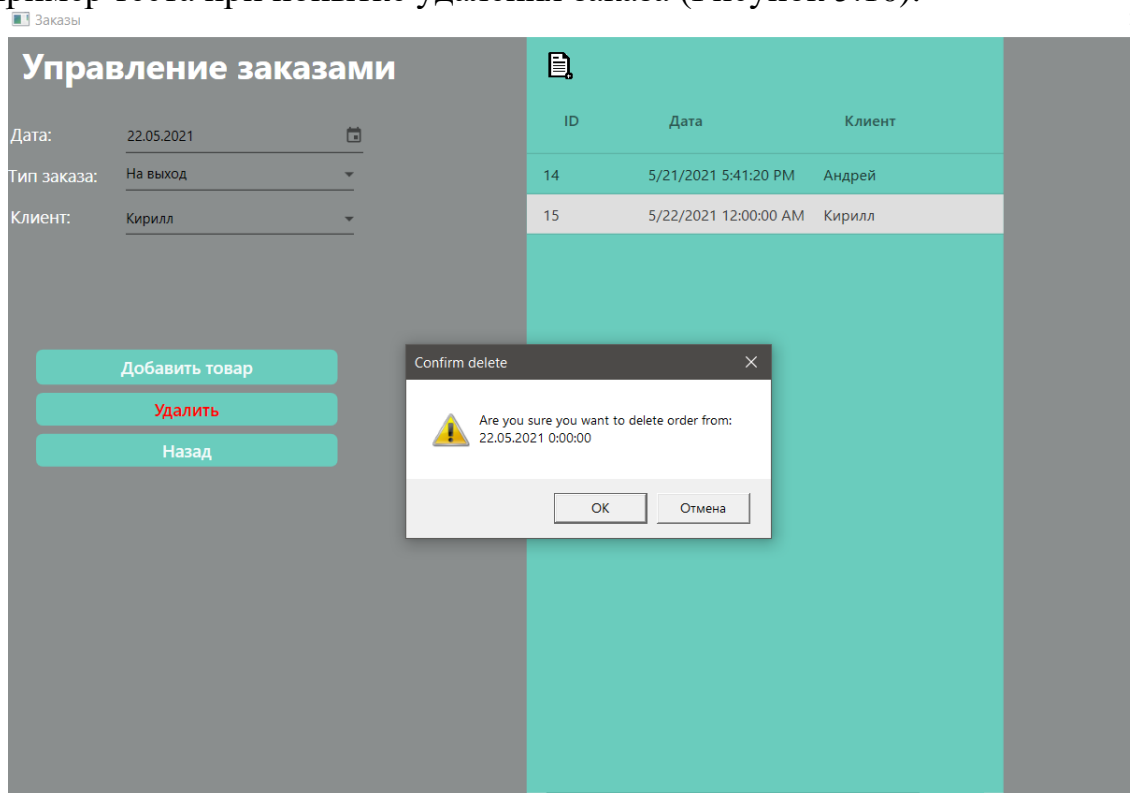


Рисунок 5.10 – Пример теста при удалении заказа.

Результат обработки исключения при попытке добавить заказ, если клиент не выбран (Рисунок 5.11).

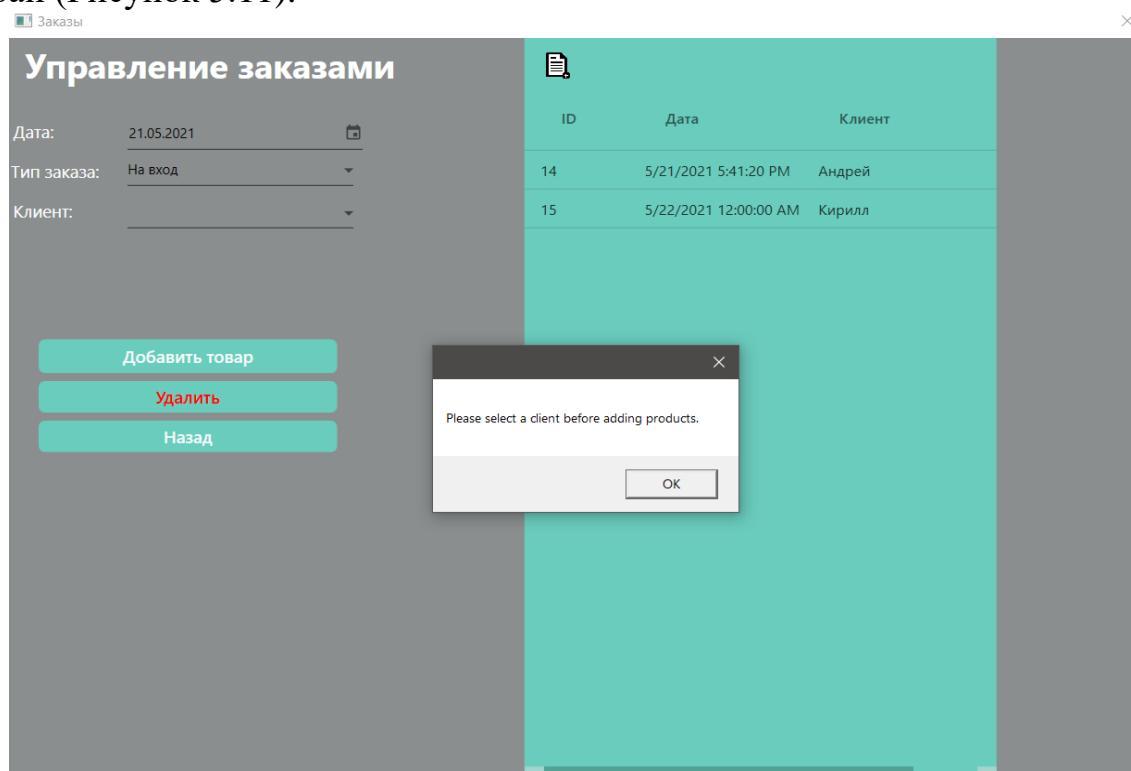


Рисунок 5.11 – Пример теста добавления заказа при невыбранном клиенте. Результат при добавлении товара и его количества на заказа не выбрав товар или количество товара (Рисунок 5.12)

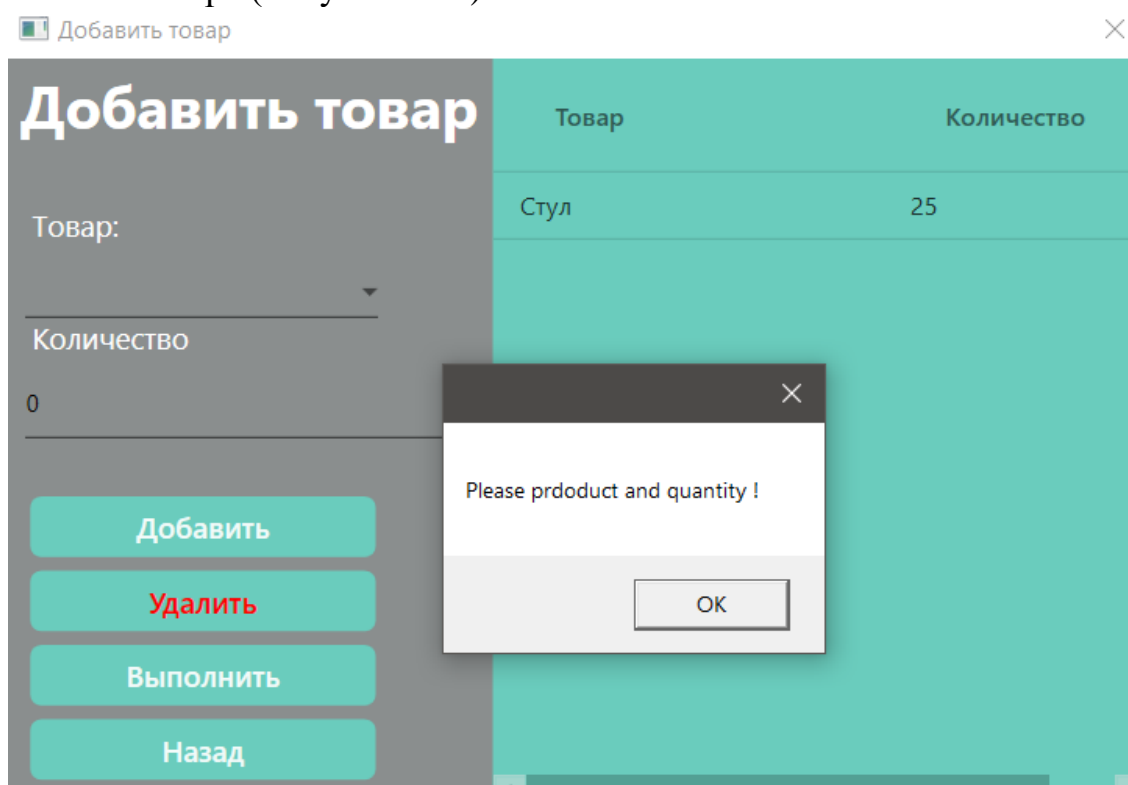


Рисунок 5.12 – Пример теста добавления товара на заказ не выбрав товар и к-во.

Результат при удалении товара и его количества на заказа (Рисунок 5.13)

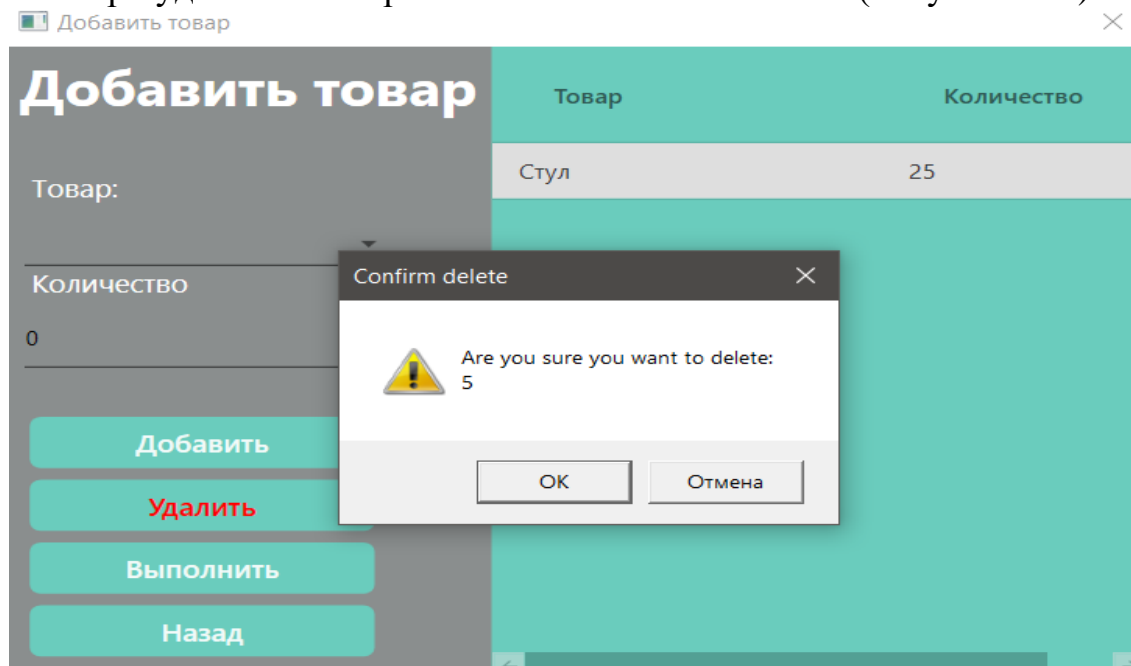


Рисунок 5.13 – Пример теста удаления товара на заказ.

Результат обработки исключения при попытке удалить заказ, если не удалены товары на заказ (Рисунок 5.14).

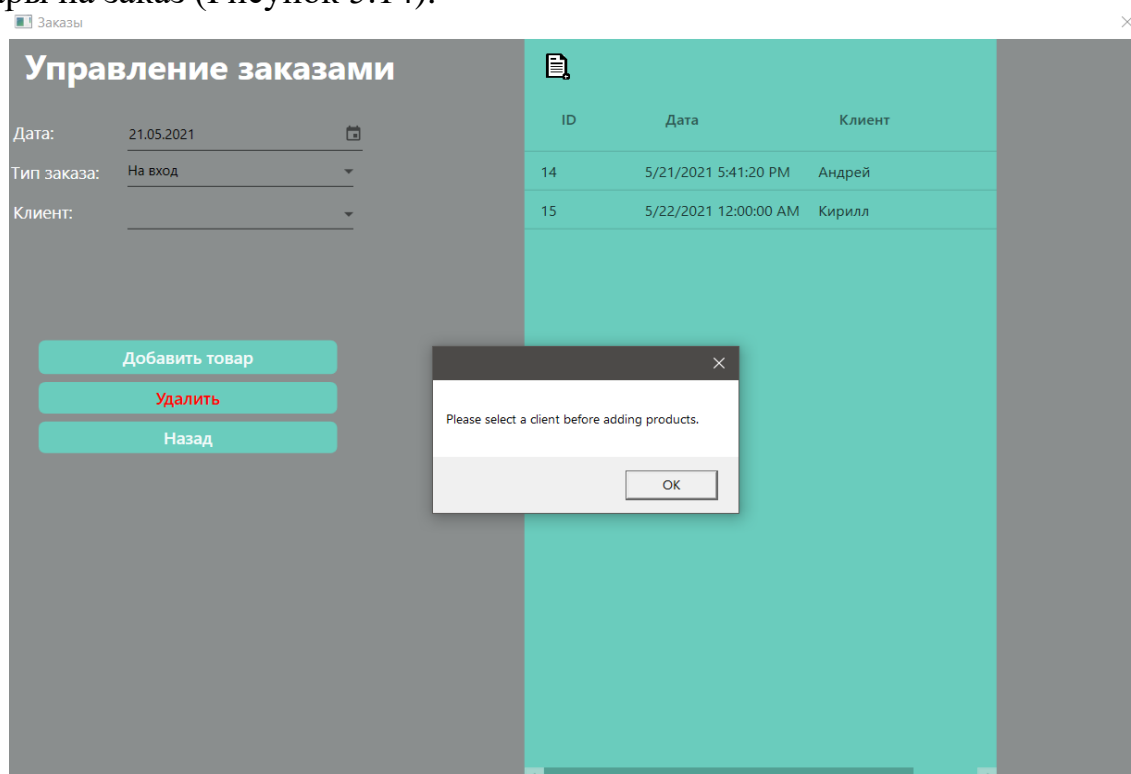


Рисунок 5.14 – Пример теста удаления заказа, не удалив заказанные товары.

Результат при попытке удалить пользователя при входе в приложение со стороны админа (Рисунок 5.15).

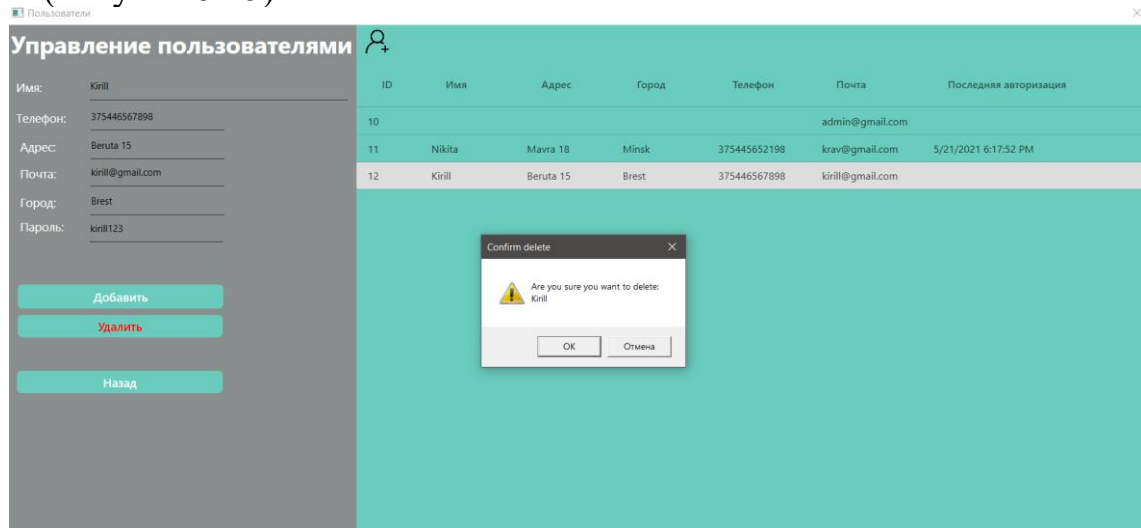


Рисунок 5.15 – Пример теста удаления пользователя админом.

Результат обработки исключения при попытке удалить пользователя, если пользователь не выбран (Рисунок 5.16).

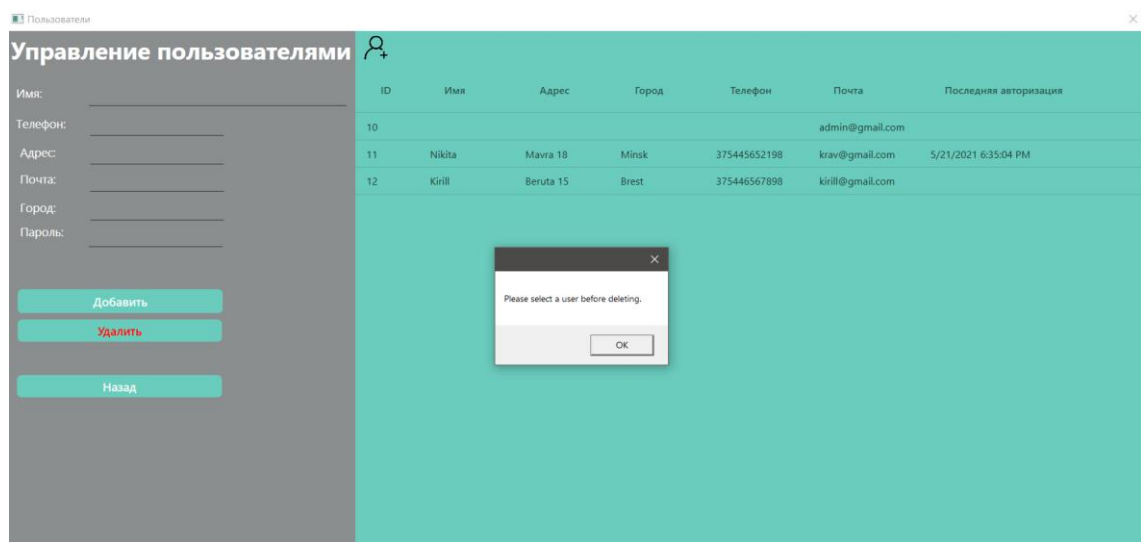
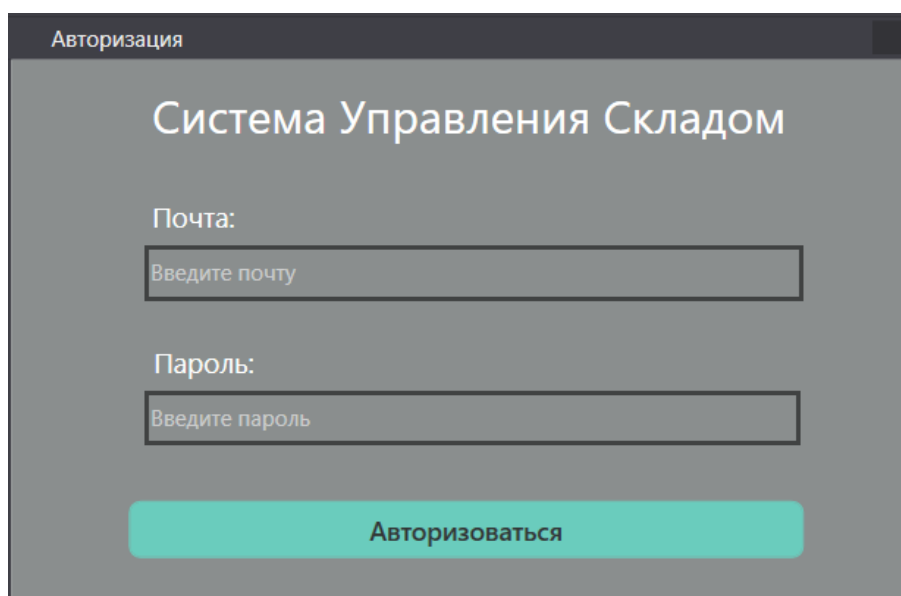


Рисунок 5.16 – Пример теста удаления невыбранного пользователя

6. Руководство по установке и использования

На начальной странице необходимо ввести учетные данные, для дальнейшего входа в программу. В зависимости от введенных данных открывается либо окно администрирования пользователями, где администратор может добавлять, изменять или удалять пользователей, либо главное окно программы, откуда можно перейти в раздел с клиентами, товарами, заказами.

На странице авторизации необходимо ввести данные для входа в программу (Рисунок 6.1) и дальнейшего её использования.



The image shows a web application window titled "Авторизация" (Authorization). The main heading is "Система Управления Складом" (Warehouse Management System). Below the heading, there are two input fields: "Почта:" (Email) with a placeholder "Введите почту" (Enter email) and "Пароль:" (Password) with a placeholder "Введите пароль" (Enter password). At the bottom, there is a large teal button labeled "Авторизоваться" (Authorize).

Рисунок 6.1 – Страница авторизации.

Авторизация выполнена в роли администратора (Рисунок 6.2)

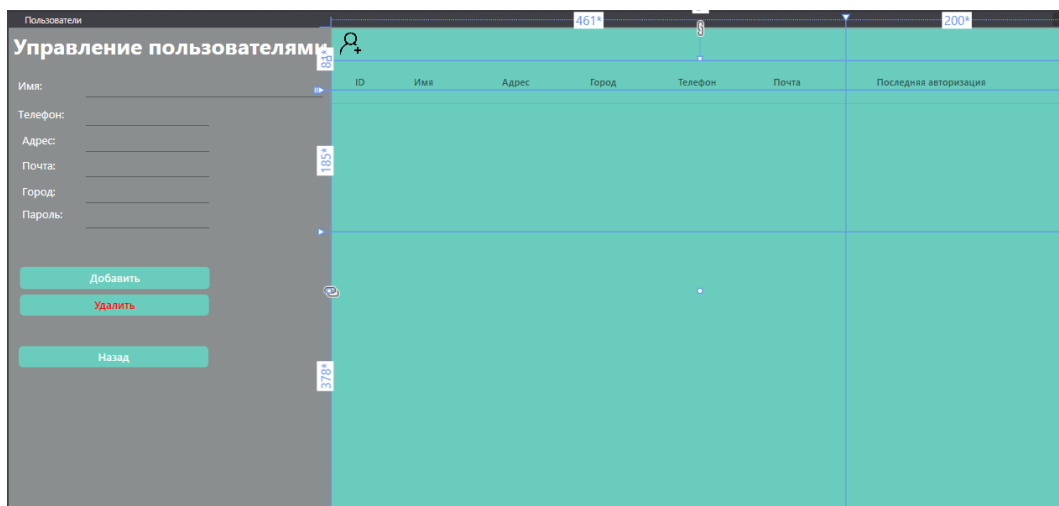


Рисунок 6.2 – Главное окно администратора.

Авторизация выполнена в роли пользователя (Рисунок 6.3)

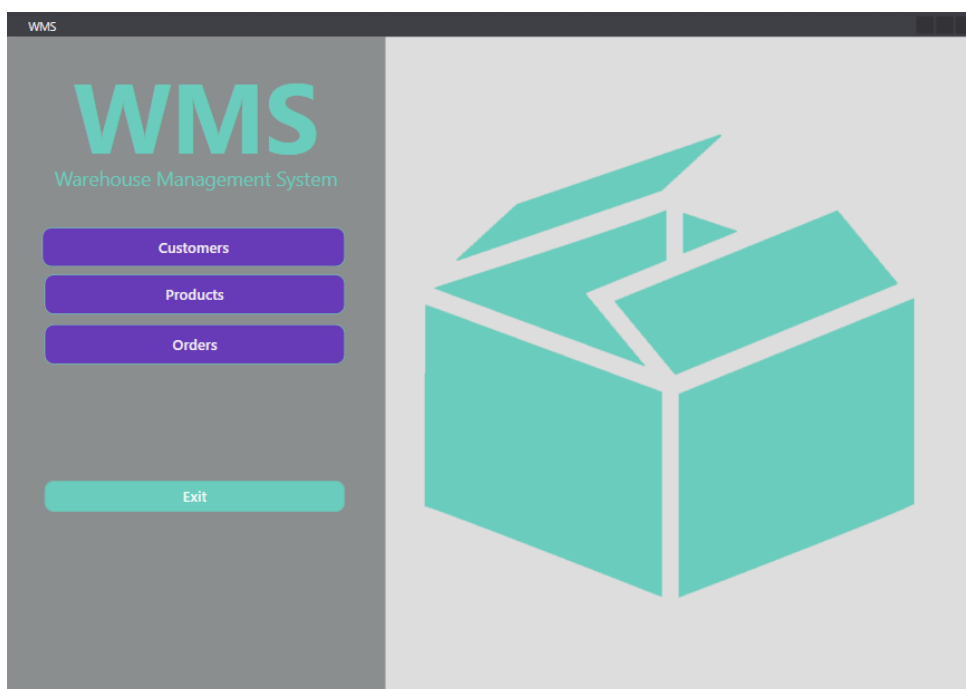


Рисунок 6.3 – Главное окно пользователя.

Заключение

Итогом курсового проекта является рабочее приложение типа «Система Управления Складом» с возможностью управлять подсчетом товаров и их движением, заказами на их покупку и продажу, импортом и экспортом, управлять поставщиками и клиентами, что поможет пользователям сделать многие из этих процессов автоматическими без необходимости запоминать детали, связанные с заказом.

В ходе выполнения курсового проекта было выполнено следующее:

- создание базы данных для хранения всей информации;
- разработка архитектуры приложения;
- разработка функциональной части приложения;
- разработка пользовательского интерфейса;
- написание исходного кода приложения;
- тестирование приложения.

В соответствии с полученным результатом работы программы можно сделать вывод, что требования технического задания выполнены, интерфейс получился достаточно простой и понятный, в нём сможет разобраться любой, даже далёкий от информационных технологий пользователь.

За время выполнения курсового проекта по созданию приложения были изучены способы работы с «SQL Server», принципы создания приложения с использованием .NET и WPF, изучен новый язык верстки XAML для построения интерфейса и работы с ним. Использовался Entity Framework для работы с базой данных, а само приложение написано на языке C#.

Список используемых источников

1. Общие сведения о Microsoft Visual Studio [Электронный ресурс]. / Режим доступа <https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019/>
2. Windows Presentation Foundation [Электронный ресурс]. / Режим доступа: https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation
3. Учебник по языку SQL [Электронный ресурс]. / Режим доступа: <https://habr.com/ru/post/255361/>

Приложение А

BaseViewModels.cs

```
using System;
...
namespace WarehouseManagementSystem.ViewModels
{
    public class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        // Create the OnPropertyChanged method to raise the event
        protected void OnPropertyChanged(string name)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null)
            {
                handler(this, new PropertyChangedEventArgs(name));
            }
        }
    }
}
```

Приложение Б

RelayCommand.cs

```
using System;
...
namespace WarehouseManagementSystem.Helpers
{
    public class RelayCommand : ICommand
    {
        #region Fields

        readonly Action<object> _execute;
        readonly Predicate<object> _canExecute;

        #endregion // Fields

        #region Constructors

        /// <summary>
        /// Creates a new command that can always execute.
        /// </summary>
        /// <param name="execute">The execution logic.</param>
        public RelayCommand(Action<object> execute)
            : this(execute, null)
        {
        }

        /// <summary>
        /// Creates a new command.
        /// </summary>
        /// <param name="execute">The execution logic.</param>
        /// <param name="canExecute">The execution status logic.</param>
        public RelayCommand(Action<object> execute, Predicate<object> canExecute)
        {
            if (execute == null)
                throw new ArgumentNullException("execute");
        }
    }
}
```

```

        _execute = execute;
        _canExecute = canExecute;
    }

    #endregion // Constructors

    #region ICommand Members

    [DebuggerStepThrough]
    public bool CanExecute(object parameters)
    {
        return _canExecute == null ? true : _canExecute(parameters);
    }

    public event EventHandler CanExecuteChanged
    {
        add { CommandManager.RequerySuggested += value; }
        remove { CommandManager.RequerySuggested -= value; }
    }

    public void Execute(object parameters)
    {
        _execute(parameters);
    }

    #endregion // ICommand Members
    }
}

```

HomeViewModel.cs

```

using System;
...
namespace WarehouseManagementSystem.ViewModels
{
    public class HomeViewModel : BaseViewModel
    {
        public ICommand UsersCommand { get; set; }
        public ICommand CustomersCommand { get; set; }
        public ICommand ProductsCommand { get; set; }
        public ICommand OrdersCommand { get; set; }
        public ICommand ExitCommand { get; set; }
        public HomeViewModel()
        {
            this.CustomersCommand = new RelayCommand(param => this.CustomersManagement(), param =>
true);
            this.ProductsCommand = new RelayCommand(param => this.ProductsManagement(), param =>
true);
            this.OrdersCommand = new RelayCommand(param => this.OrdersManagement(), param => true);
            // this.UsersCommand = new RelayCommand(param => this.UsersManagement(), param => true);
            this.ExitCommand = new RelayCommand(param => this.Exit(), param => true);
        }

        private void Exit()
        {
            MessageBoxResult rsltMessageBox = MessageBox.Show("Do you want to exit?", "Exit", Mes-
sageBoxButton.YesNo, MessageBoxImage.Warning);

            switch (rsltMessageBox)
            {
                case MessageBoxResult.Yes:

```

```

        Application.Current.Shutdown();
        break;

    case MessageBoxResult.No:
        break;
    }
}

private void CustomersManagement()
{
    CustomersWindow myWindow = new CustomersWindow();
    myWindow.Show();
}

private void ProductsManagement()
{
    ProductsWindow myWindow = new ProductsWindow();
    myWindow.Show();
}

private void OrdersManagement()
{
    OrdersWindow myWindow = new OrdersWindow();
    myWindow.Show();
}

private void UsersManagement()
{
    UsersWindow myWindow = new UsersWindow();
    myWindow.Show();
}
}
}

```

Приложение В

WarehouseDbContext.cs

```

using System;
...
namespace WarehouseManagementSystem.Models
{
    //DbContext: определяет контекст данных, используемый для взаимодействия с базой данных
    public class WarehouseDbContext : DbContext
    {
        public WarehouseDbContext() : base("name=WarehouseConnection")
        {
        }

        // представляет набор сущностей, хранящихся в базе данных
        public DbSet<Customer> Customers { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<OrderDetail> Details { get; set; }
    }
}

```

Category.cs

```
using System;
...
namespace WarehouseManagementSystem.Models
{
    public class Category
    {
        [Key]
        public int ID { get; set; }

        [Required, StringLength(50)]
        public string Name { get; set; }
    }
}
```

Customer.cs

```
using System;
...
namespace WarehouseManagementSystem.Models
{
    public class Customer
    {
        [Key]
        public int ID { get; set; }

        [Required, StringLength(50)]
        public string Name { get; set; }

        [StringLength(50)]
        public string CompanyName { get; set; }

        [StringLength(50)]
        public string Address { get; set; }

        [StringLength(50)]
        public string City { get; set; }

        [StringLength(50)]
        public string Telephone { get; set; }

        [StringLength(50)]
        public string Email { get; set; }
    }
}
```

Order.cs

```
using System;
...
namespace WarehouseManagementSystem.Models
{
    public class Order
    {
        [Key]
        public int ID { get; set; }

        public string OrderType { get; set; }

        public DateTime Date { get; set; }

        //Relations
    }
}
```

```

        [Required, ForeignKey("Customer")]
        public int CustomerId { get; set; }
        public virtual Customer Customer { get; set; }

        [Required, ForeignKey("User")]
        public int UserId { get; set; }
        public virtual User User { get; set; }

        public virtual ICollection<OrderDetail> Products { get; set; }
    }
}

```

OrderDetail.cs

```

using System;
...
namespace WarehouseManagementSystem.Models
{
    public class OrderDetail
    {
        [Key]
        public int ID { get; set; }

        //Relations
        [Required, ForeignKey("Product")]
        public int ProductId { get; set; }
        public virtual Product Product { get; set; }

        public int Quantity { get; set; }
    }
}

```

Product.cs

```

using System;
...
namespace WarehouseManagementSystem.Models
{
    public class Product
    {
        [Key]
        public int ID { get; set; }

        [Required, StringLength(50)]
        public string Name { get; set; }

        //Relations
        [Required, ForeignKey("Category")]
        public int CategoryId { get; set; }
        public virtual Category Category { get; set; }

        public decimal? Price { get; set; }

        [StringLength(50)]
        public string Description { get; set; }

        public byte[] Image { get; set; }    } }
}

```

User.cs

```

using System;
...
namespace WarehouseManagementSystem.Models

```

```

{
    public class User
    {
        [Key]
        public int ID { get; set; }

        [Required, StringLength(50)]
        public string Name { get; set; }

        [StringLength(50)]
        public string Address { get; set; }

        [StringLength(50)]
        public string City { get; set; }

        [StringLength(50)]
        public string Tel { get; set; }

        [Required, StringLength(50)]
        public string Email { get; set; }

        [Required]
        public string Password { get; set; }

        public DateTime? LastLoginDate { get; set; }

        public User() { }

        public User(string Email, string Password)
        {
            this.Email = Email;
            this.Password = Password;
        }
    }
}

```

Приложение Г

LoginViewModel.cs

```

using System;
...
namespace WarehouseManagementSystem.ViewModels
{
    public class LoginViewModel : BaseViewModel
    {
        public LoginWindow CurrentWindows { get; set; }
        public ICommand LoginCommand { get; set; }

        public User CurrentUser { get; set; }

        public LoginViewModel(LoginWindow windows)
        {
            CurrentWindows = windows;
            this.CurrentUser = new User();

            this.LoginCommand = new RelayCommand(param => this.Login(), param => true);
        }

        private void Login()
        {
            WarehouseDbContext ctx = new WarehouseDbContext();

            var user = ctx.Users.SingleOrDefault(u => u.Email == CurrentUser.Email);

```

```

        if (user != null && CurrentUser.Email!="admin@gmail.com" && CurrentUser.Password ==
user.Password)
        {
            user.LastLoginDate = DateTime.Now;

            try
            {
                ctx.SaveChanges();
            }
            catch (DbEntityValidationException ex)
            {
                foreach (var entityValidationErrors in ex.EntityValidationErrors)
                {
                    foreach (var validationError in entityValidationErrors.ValidationErrors)
                    {
                        string property = ("Property:" + validationError.PropertyName + "Er-
ror:" + validationError.ErrorMessage);
                    }
                }
            }
            HomeWindow myWindow = new HomeWindow();
            myWindow.Show();

            this.CurrentWindows.Close();

            //keep current user id in the application
            App.Current.Properties["CurrentUserID"] = user.ID;
        }
        else if (user != null && CurrentUser.Email == "admin@gmail.com" && Curren-
tUser.Password == "admin")
        {
            user.LastLoginDate = DateTime.Now;

            try
            {
                ctx.SaveChanges();
            }
            catch (DbEntityValidationException ex)
            {
                foreach (var entityValidationErrors in ex.EntityValidationErrors)
                {
                    foreach (var validationError in entityValidationErrors.ValidationErrors)
                    {
                        string property = ("Property:" + validationError.PropertyName + "Er-
ror:" + validationError.ErrorMessage);
                    }
                }
            }
            UsersWindow myWindow = new UsersWindow();
            myWindow.Show();

            this.CurrentWindows.Close();

            //keep current user id in the application
            App.Current.Properties["CurrentUserID"] = user.ID;
        }
        else
        {
            MessageBox.Show(CurrentWindows, "Invalid email or password !");
            return;
        }
    }
}
}

```


Приложение Д

ProductsViewModel.cs

```
using System;
...
namespace WarehouseManagementSystem.ViewModels
{
    public class ProductsViewModel : BaseViewModel
    {
        public ProductsWindow CurrentWindows { get; set; }
        public ICommand CancelCommand { get; set; }
        public ICommand AddCommand { get; set; }
        public ICommand SaveCommand { get; set; }
        public ICommand DeleteCommand { get; set; }
        public ICommand SearchCommand { get; set; }
        public ICommand AddImageCommand { get; set; }

        private List<Product> _productList;
        public List<Product> ProductList
        {
            get { return _productList; }
            set
            {
                _productList = value;
                OnPropertyChanged("ProductList");
            }
        }

        private Product _selectedProduct;
        public Product SelectedProduct
        {
            get { return _selectedProduct; }
            set
            {
                _selectedProduct = value;
                if (_selectedProduct != null && _selectedProduct.Category != null)
                {
                    SelectedCategory = _selectedProduct.Category.Name;
                }
                OnPropertyChanged("SelectedProduct");
                IsNew = false;
            }
        }

        public bool IsNew { get; set; }
        public string SearchTerm { get; set; }

        private List<Category> _categories;
        public List<Category> Categories
        {
            get { return _categories; }
            set
            {
                _categories = value;
                OnPropertyChanged("Categories");
            }
        }

        private string _selectedCategory;
        public string SelectedCategory
        {
            get { return _selectedCategory; }
        }
    }
}
```

```

        set
        {
            _selectedCategory = value;
            OnPropertyChanged("SelectedCategory");
        }
    }

    //Constructor
    public ProductsViewModel(ProductsWindow window)
    {
        this.CurrentWindows = window;
        this.Add();

        this.CancelCommand = new RelayCommand(param => this.Cancel(), param => true);
        this.AddCommand = new RelayCommand(param => this.Add(), param => true);
        this.SaveCommand = new RelayCommand(param => this.Save(), param => true);
        this.DeleteCommand = new RelayCommand(param => this.Delete(), param => true);
        this.SearchCommand = new RelayCommand(param => this.Search(), param => true);
        this.AddImageCommand = new RelayCommand(param => this.AddImage(), param => true);

        //Load data from database

        WarehouseDbContext ctx = new WarehouseDbContext();
        ProductList = new List<Product>(ctx.Products.ToList());
        Categories = new List<Category>(ctx.Categories.ToList());
    }

    private void AddImage()
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.InitialDirectory = @"C:\";
        openFileDialog.Title = "Open Files";
        openFileDialog.CheckFileExists = true;
        openFileDialog.CheckPathExists = true;
        openFileDialog.DefaultExt = ".jpg";
        openFileDialog.Filter = "image files (*.jpg)|*.png|All files (*.*)|*.*";
        openFileDialog.FilterIndex = 2;
        openFileDialog.RestoreDirectory = true;
        if (openFileDialog.ShowDialog() == true)
        {
            SelectedProduct.Image = File.ReadAllBytes(openFileDialog.FileName);
            OnPropertyChanged("SelectedProduct");
        }
    }

    private void Search()
    {
        WarehouseDbContext ctx = new WarehouseDbContext();
        if (SearchTerm != null)
        {
            ProductList = new List<Product>(ctx.Products.ToList().Where(u =>
                u.Name.Contains(SearchTerm) || u.Description.Contains(SearchTerm)));
        }
    }

    private void Delete()
    {

```

```

        if (SelectedProduct == null || SelectedProduct.ID == 0)
        {
            MessageBox.Show(CurrentWindows, "Please select a product before deleting.");
            return;
        }

        MessageBoxResult result = MessageBox.Show(CurrentWindows, "Are you sure you want to delete:\n" +
            SelectedProduct.Name, "Confirm delete", MessageBoxButton.OKCancel, MessageBoxImage.Warning);

        if (result == MessageBoxResult.OK)
        {
            try
            {
                WarehouseDbContext ctx = new WarehouseDbContext();
                var product = ctx.Products.Single(u => u.ID == SelectedProduct.ID);
                ctx.Products.Remove(product);
                ctx.SaveChanges();

                ProductList = new List<Product>(ctx.Products.ToList());
            }
            catch (SqlException ex)
            {
                MessageBox.Show(CurrentWindows, ex.Message, "Database error", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
    }

    private void Save()
    {
        WarehouseDbContext ctx = new WarehouseDbContext();
        if (!IsNew)
        {
            if (SelectedProduct == null)
            {
                MessageBox.Show(CurrentWindows, "Please select a product before editing.");
                return;
            }

            var cat = ctx.Categories.SingleOrDefault(x => x.Name == SelectedCategory);
            if (cat == null)
            {
                cat = ctx.Categories.Add(new Category { Name = SelectedCategory });
                ctx.SaveChanges();
            }

            var product = ctx.Products.Single(u => u.ID == SelectedProduct.ID);

            product.Name = SelectedProduct.Name;
            product.Category = cat;
            product.Price = SelectedProduct.Price;
            product.Description = SelectedProduct.Description;
            product.Image = SelectedProduct.Image;

            ctx.SaveChanges();
        }
        else
        {
            var cat = ctx.Categories.SingleOrDefault(x => x.Name == SelectedCategory);

            if (cat == null)
            {
                cat = ctx.Categories.Add(new Category { Name = SelectedCategory });
            }
        }
    }

```

```

        ctx.SaveChanges();
    }

    SelectedProduct.Category = cat;
    ctx.Products.Add(SelectedProduct);
    ctx.SaveChanges();

}

//Reload data from database
ProductList = new List<Product>(ctx.Products.ToList());
Categories = new List<Category>(ctx.Categories.ToList());
IsNew = false;

MessageBox.Show(CurrentWindows, "Updated successfully !");

}

private void Add()
{
    SelectedProduct = new Product();
    IsNew = true;
}

private void Cancel()
{
    this.CurrentWindows.Close();
}
}
}

```