МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(Наименование кафедры, центра, департамента)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

«Разработка программного обеспечения»
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения для учёта товаров строительного магазина»

Обучающийся С.А. Кузнецов _____
(Инициалы Фамилия) (личная подпись)

Руководитель доктор социологических наук, доцент, Е.В. Желнина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант кандидат филол. наук, доцент, М. В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

## Аннотация

Тема выпускной квалификационной работы: «Разработка программного обеспечения для учёта товаров строительного магазина».
Работа состоит из введения, трех глав, заключения и списка литературы.

В первой главе рассмотрены особенности строительной организации, существующие подходы к учёту товаров, обоснована необходимость разработки собственного решения и проведён анализ предметной области.

Во второй главе представлены архитектурные и технологические основы разрабатываемого программного обеспечения, описана структура базы данных, обоснован выбор используемых инструментов, приведены модели взаимодействия компонентов и диаграммы классов.

В третьей главе представлена реализация программного обеспечения: описаны пользовательские формы, элементы интерфейса и бизнес-логика. Подробно рассмотрены функции добавления, редактирования, удаления фильтрации и поиска товаров. Проведено ручное тестирование, подтверждающие исправность работы созданного программного обеспечения.

В заключении подведены итоги работы и сформулированы выводы о практической применимости разработанной системы.

Работа содержит 6 таблиц, 26 рисунков, список литературы, состоящий из 25 источников.

Общий объем выпускной квалификационной работы составляет 62 страницы.

## Annotation

The title of the graduation work is Development of an Application for Inventory Management in a Hardware Store.

The graduation work consists of an introduction, three chapters, a conclusion, 26 figures, 6 tables, and a list of 25 references.

The aim of this graduation work is to develop standalone inventory management software tailored for small hardware stores.

The object of the graduation work is the process of inventory management in small hardware retail businesses.

The subject of the graduation work is the design and implementation of an application that automates basic operations such as item entry, editing, deletion, search, and filtering.

The key issue of the graduation work is the creation of a user-friendly and autonomous software solution that does not require an internet connection and minimizes human errors in accounting.

The first chapter analyzes the specific features of hardware stores, reviews existing inventory systems, justifies the need for a custom solution, and describes the subject area.

The second chapter describes the architecture and technological foundation of the software, database structure, selected tools, and presents component interaction models and class diagrams.

The third chapter focuses on the implementation, describing the user interface, logic behind item operations, and testing procedures used to verify functionality.

In conclusion, the study confirms the feasibility and effectiveness of a custom lightweight application for small hardware stores. The developed software combines accessibility, stability, intuitive interface, and adaptability to specific retail conditions without the need for extensive training.

## Оглавление

# Введение

В условиях бурного развития цифровых технологий и усиления конкуренции на рынке розничной торговли точный и своевременный учёт товарных запасов становится критически важным элементом для организаций, занятых в сфере торговли. Особенно остро эта проблема стоит перед строительными магазинами, где ассортимент характеризуется широким разнообразием, частыми обновлениями и необходимостью строгого контроля остатков. Недочёты в учётных операциях могут привести как к нехватке нужных товаров, так и к их избытку на складе, что негативно сказывается как на финансовой устойчивости, так и на имидже предприятия.

На сегодняшний день на рынке представлено множество программных решений, предназначенных для автоматизации процессов торговли. Однако значительная часть этих продуктов является платной и не всегда адаптируется под нужды малого бизнеса или специфику конкретных торговых точек. Более того, высокая стоимость лицензионного программного обеспечения зачастую делает его недоступным для небольших организаций с ограниченным бюджетом.

Настоящая выпускная квалификационная работа направлена на создание программного обеспечения, ориентированного на потребности малых строительных магазинов в области учёта товарных позиций. Разрабатываемое приложение реализует базовый функционал, включающий добавление, редактирование, удаление и поиск товаров по различным критериям, а также предлагает интуитивно понятный графический интерфейс, рассчитанный на пользователей без глубоких технических знаний.

Объектом исследования выступает процесс учёта товаров в информационной системе управления строительным магазином.

Предмет исследования – методы автоматизации торгового учёта на основе современных информационных технологий с использованием языка

программирования C#, среды разработки Visual Studio, библиотеки Windows Forms и реляционной базы данных SQLite.

В процессе разработки учитываются актуальные принципы построения пользовательских интерфейсов и организации хранения данных. Особое внимание уделено вопросам удобства использования, адаптивности и расширяемости архитектурных решений, что обеспечивает гибкость системы и её пригодность к последующему масштабированию.

Практическая значимость проекта заключается в его применимости в реальных условиях розничной торговли для повышения производственной эффективности, минимизации ошибок при учёте товаров и оптимизации логистических процессов.

В ходе реализации поставленной цели решаются следующие задачи:

– проведение анализа предметной области и формулировка задач автоматизации,
– разработка архитектурной модели приложения и структуры базы данных,
– программная реализация на основе выбранных технологий,
– проведение тестирования и анализ эффективности работы разработанного решения.

# Глава 1 Анализ предметной области

## 1.1 Назначение и задачи учета товаров

Товарный учёт представляет собой комплекс организационно-технических мероприятий, направленных на систематизированное ведение, обновление и хранение информации о товарных позициях предприятия. В условиях розничной торговли, особенно в строительной сфере, отлаженный учёт играет ключевую роль в обеспечении прозрачности хозяйственной деятельности и стабильности бизнес-процессов [13].

Строительный магазин, как специфический объект розничной торговли, характеризуется высокой насыщенностью ассортимента. Здесь реализуются как стандартные товары повседневного спроса – гвозди, саморезы, шпаклёвки, – так и более сложные и специализированные наименования, включая электроинструмент, сантехнику, строительные смеси и отделочные материалы. Для каждого товарного элемента может быть указано множество атрибутов: размеры, вес, состав, производитель, артикул, страна происхождения, условия хранения и прочее. Такая специфика требует от системы учёта гибкости, точности и высокого уровня детализации.

Цель внедрения системы учёта в данном контексте заключается в создании надёжной и функциональной информационной среды, отражающей текущее состояние товарных остатков, а также предоставляющей возможности для их анализа, корректировки и обработки по заданным параметрам.

Основные задачи, решаемые системой учёта, включают:

– регистрацию и хранение полных сведений о каждом товаре с указанием его уникальных характеристик, цены, количества и дополнительной информации;

– обеспечение оперативного доступа к данным, позволяющего сотрудникам принимать обоснованные решения при осуществлении закупок,

перемещений или продаж [17];

– поддержку базовых операций (создание, просмотр, редактирование, удаление) в удобной, визуально понятной форме;

– контроль за актуальностью данных, предотвращающий расхождения между фактическими остатками и информацией в системе;

– фильтрацию и поиск по множеству критериев: названию, параметрам, принадлежности к группе и другим;

– автоматизацию рутинных действий, что снижает влияние человеческого фактора и уменьшает вероятность ошибок;

– формирование единой информационной среды для всех сотрудников, включая продавцов и управленческий персонал.

Функционал системы также играет стратегическую роль в управлении товарооборотом. При наличии актуальных и достоверных данных администрация магазина может:

– своевременно выявлять дефицит или избыток продукции;

– эффективно планировать закупки;

– анализировать рентабельность товарных категорий;

– проводить переоценку, списание или инвентаризацию.

Для обеспечения эффективности системы она должна учитывать особенности предприятия. В случае строительного магазина это, в частности, означает:

– поддержку различных единиц измерения (например, штуки, метры, литры);

– возможность логической группировки номенклатуры по категориям (инструменты, крепёж, отделочные материалы и т.д.);

– учёт сезонных колебаний спроса, а также сроков хранения или годности для специфичных групп товаров (например, лакокрасочных покрытий и клеевых смесей).

Таким образом, современная система учёта в строительной рознице выполняет не только функцию фиксации наличия товаров, но и становится

инструментом управленческого анализа, снижения операционных издержек и повышения конкурентоспособности торгового предприятия.

На основании рассмотренных особенностей можно выделить ряд ключевых функций, которые реализует система учёта товарных позиций в условиях строительного магазина. Визуальное представление данных задач приведено на рисунке 1.

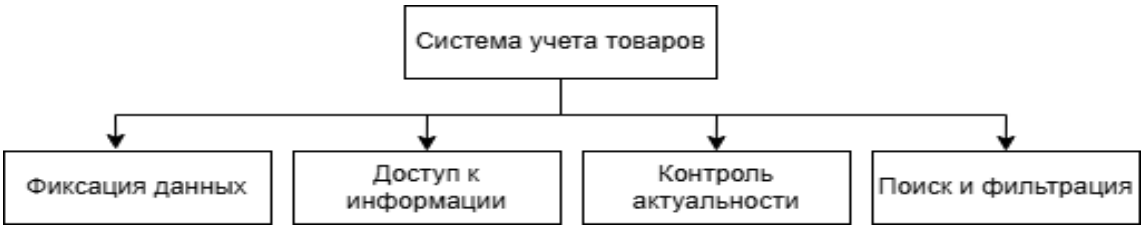



Рисунок 1 – Основные функции системы учета товаров

Реализация указанных функций обеспечивает замкнутый цикл управления товарными остатками и способствует автоматизации процессов, связанных с обработкой информации о продукции, её характеристиках и движении на складе.

### 1.2 Постановка проблемы и актуальность разработки

Современный рынок розничной торговли предъявляет всё более жёсткие требования к скорости обработки и точности учёта товарных запасов. Это особенно ощутимо в сфере строительной розницы, где ассортимент продукции отличается высокой сложностью: различные материалы, размеры, массы, цвета, бренды, единицы измерения и условия хранения формируют множество комбинаций характеристик. При этом значительная часть подобных торговых точек принадлежит к малому или среднему бизнесу и, как правило, не обладает достаточными ресурсами для внедрения полнофункциональных ERP-систем промышленного уровня.

Во многих магазинах по сей день используется ручной или полуавтоматический способ учёта – от бумажных журналов и блокнотов до таблиц в Excel. Такой подход уязвим к ошибкам, не обеспечивает актуальности данных для всех сотрудников и не подходит для быстрого поиска информации. По мере роста номенклатуры такие методы теряют управляемость и создают угрозу финансовых потерь, а также снижают доверие клиентов из-за возможных сбоев в обслуживании.

В то же время на рынке представлены комплексные решения, такие как 1С:Розница, МойСклад, Склад 365 и аналогичные системы. Однако использование таких продуктов в условиях малого бизнеса связано с рядом существенных ограничений:

– чрезмерный объём функциональности, который остаётся невостребованным;

– необходимость оплаты лицензий или подписки;

– высокая пороговая сложность освоения интерфейса;

– зависимость от интернет-соединения или наличия серверной инфраструктуры;

– ограниченная возможность настройки под уникальные процессы конкретного предприятия.

В совокупности эти факторы формируют актуальную проблему: дефицит доступных, адаптируемых и экономически оправданных программных решений для автоматизации учёта в строительных магазинах малого и среднего формата.

В связи с этим возникает практическая необходимость в разработке специализированного программного продукта, который бы позволил:

– отказаться от ручного ведения учёта в пользу автоматизированной системы;

– повысить точность и прозрачность операций, связанных с товарным оборотом;

– упростить работу персонала за счёт понятного и адаптированного

интерфейса;

– минимизировать затраты на внедрение, обучение и сопровождение;

– использовать автономное программное решение без внешней серверной зависимости.

Настоящая выпускная квалификационная работа направлена на создание настольного приложения с локальной базой данных, ориентированного на сотрудников строительного магазина. Разрабатываемая система предусматривает удобное выполнение операций по добавлению, поиску, редактированию и удалению товарных позиций, при этом интерфейс интуитивно понятен и не требует специальных знаний в сфере ИТ.

Программная реализация осуществляется на языке C# с применением технологии Windows Forms, что обеспечивает стабильность графического интерфейса и возможность запуска на любой машине с операционной системой Windows без предварительной настройки окружения [23]. В качестве СУБД выбрана SQLite – лёгкая и встраиваемая база данных, не требующая установки и идеально подходящая для локальных решений подобного масштаба [24].

Для работы с данными используется технология Entity Framework, реализующая объектно-реляционное отображение (ORM). Это позволяет повысить читаемость кода, облегчить поддержку и ускорить разработку, особенно в условиях ограниченных сроков и ресурсов.

Таким образом, создаваемое программное решение отвечает актуальным запросам реального сектора розничной торговли и может рассматриваться как пример эффективной ИТ-инициативы, направленной на поддержку малого бизнеса через создание недорогих, автономных и адаптируемых программных продуктов. Следует отметить, что создание специализированного программного обеспечения представляет собой своевременное и обоснованное решение, соответствующее требованиям конкретных бизнес-процессов в сегменте малого строительного ритейла. Разрабатываемое приложение демонстрирует, как с минимальными

затратами возможно реализовать эффективный инструмент для автоматизации учёта, обладающий простотой внедрения, гибкостью настройки и независимостью от внешних платформ [19]. Сравнительный анализ существующих подходов к организации товарного учёта приведён в таблице 1.

Таблица 1 – Сравнение подходов к ведению учета товаров

| Критерий | Ручной учет | Коммерческие ПО | Разрабатываемое ПО |
|---|---|---|---|
| Стоимость внедрения | Низкая | Высокая | Низкая |
| Удобства | Низкое | Среднее | Высокое |
| Требования к оборудованию | Минимальные | Среднее/высокие | Минимальные |
| Адаптация под конкретный бизнес | Трудная | Ограничена | Полная |

Как показывает проведённое сравнение, разработанное программное обеспечение удачно сочетает в себе сильные стороны ручного и коммерческого учёта, устраняя при этом их основные недостатки. Система проста в освоении, не требует затрат на инфраструктуру и остаётся гибкой в настройке, что особенно важно для небольших магазинов. За счёт доступности и адаптивности она становится удобным инструментом для повседневной работы, где на первом месте стоят надёжность и соответствие реальным потребностям бизнеса.

### 1.3 Обзор существующих решений

На рынке представлено множество программных решений, предназначенных для автоматизации учета товаров, управления складом и поддержки торговых операций. Наиболее известные из них – это универсальные коммерческие программные продукты, такие как 1С:Розница, МойСклад, TorgSoft, Склад 365, Класс365, а также отдельные веб-сервисы и

облачные платформы.

1С:Розница один из самых распространённых программных продуктов на территории России и стран СНГ. Предназначен для автоматизации розничной торговли, включает в себя мощные модули по учету товаров, кассовому обслуживанию, формированию отчетности и взаимодействию с онлайн-кассами.

Недостатки для малого бизнеса:

– высокая стоимость лицензии и обновлений;

– высокая зависимость от специалистов по настройке 1С;

– избыточный функционал, перегружающий интерфейс;

– отсутствие локальной лёгкой версии.

МойСклад облачный сервис для учета товаров и продаж, поддерживает работу с интернет-магазинами, интеграции с маркетплейсами и CRM. Недостатки:

– требует постоянного подключения к интернету;

– не предоставляет полной автономности;

– имеет ограничения в бесплатной версии;

– невозможность адаптации под локальные особенности магазина.

Склад 365 система, ориентированная на малый бизнес, работает через веб-интерфейс.

Недостатки:

– облачный доступ, что требует стабильного соединения;

– привязанность к подписке;

– ограниченные возможности кастомизации интерфейса. Excel и Google Таблицы

Некоторые магазины используют электронные таблицы как замену полноценным программам. Это позволяет быстро настроить базовый учёт, однако такой подход:

– подвержен ошибкам и дублированию данных;

– не масштабируется;

– не поддерживает параллельную работу нескольких пользователей;

– не обеспечивает целостности и безопасности информации.

Таким образом, существующие решения либо чрезмерно сложны и дорогие, либо не обеспечивают необходимого уровня автономности и адаптивности. Особенно остро эта проблема ощущается в небольших магазинах, где нет IT-персонала и выделенного бюджета на сопровождение программных продуктов.

Цель настоящей разработки – создать программное обеспечение, которое бы сочетало в себе преимущества:

– локальной установки;

– простоты интерфейса;

– достаточного функционала для учёта;

– возможности доработки без лицензирования.

Проведённый анализ демонстрирует, что ни одно из существующих решений не может в полной мере удовлетворить потребности небольшого строительного магазина, ориентированного на простоту, автономность и экономическую эффективность. Несмотря на наличие определённых достоинств, каждое из рассмотренных программных продуктов сопровождается существенными ограничениями – будь то высокая стоимость владения, зависимость от сетевой инфраструктуры или недостаточная гибкость в адаптации под индивидуальные процессы.

Таким образом, рассмотренные программные решения отражают общую рыночную тенденцию: приоритет отдан либо крупным торговым сетям с масштабируемыми ERP-системами, либо онлайн-коммерции с облачной архитектурой. В результате потребности небольших офлайн-магазинов с узкой специализацией, таких как строительные и хозяйственные точки розничной торговли, остаются в значительной степени неохваченными. Между тем именно для таких предприятий критически важны простота учёта, возможность гибко настраивать товарные характеристики и способность работать в автономном режиме без

постоянного подключения к интернету.

Большинство доступных решений предполагают либо значительные финансовые издержки на приобретение и обслуживание, либо требуют наличия ИТ-компетенций для настройки и сопровождения. Кроме того, облачные сервисы часто не предоставляют пользователю полного контроля над данными, что может противоречить требованиям к информационной безопасности, особенно в условиях внутреннего документооборота и локального хранения конфиденциальной информации.

В связи с этим предлагаемое в рамках данной выпускной квалификационной работы программное решение ориентировано на устранение указанных ограничений. Цель разработки – создание доступного, интуитивно понятного и независимого программного продукта, оптимально соответствующего задачам учёта в условиях небольшой торговой точки со специализированным ассортиментом и ограниченными ресурсами.

Для более наглядного представления ключевых характеристик, а также выявленных ограничений, основные параметры рассмотренных решений обобщены в таблице 2.

Таблица 2 – Сравнительный анализ существующих решений

| Название | Тип решения | Плюсы | Минусы |
|---|---|---|---|
| 1С:Розница | Локальное/сервер | Функционально, развитая система | Сложная, дорогая, требует обучения |
| МойСклад | Облачное | Удобство, отчёты, API | Требует интернет, платная версия |
| Склад 365 | Облачное | Быстрый старт | Подписка, слабая кастомизация |
| Разрабатываемое ПО | Локальное | Простота, автономность, бесплатно | Пока не интегрировано в бизнес-процессы |

В результате сравнительного анализа стало ясно, что большинство готовых решений рассчитано либо на крупные компании с развитой ИТ-инфраструктурой, либо на онлайн-форматы торговли.

## 1.4 Особенности учета в строительных магазинах

Строительные магазины обладают рядом особенностей, напрямую влияющих на организацию и реализацию процессов товарного учёта. В отличие от продовольственных или хозяйственных торговых точек, где ассортимент относительно стандартизирован, в строительной рознице представлен широкий спектр товаров, отличающихся по физическим характеристикам, единицам измерения, условиям хранения и скорости реализации.

Прежде всего, в номенклатуре строительного магазина одновременно присутствуют как мелкоштучные товары (например, гвозди, дюбели, саморезы), так и крупногабаритная продукция – листовые материалы, мешки со смесями, лестницы, двери и т.д. Это требует от системы учёта высокой степени гибкости, в частности – поддержки различных единиц измерения: штуки, килограммы, литры, погонные, квадратные и кубические метры.

Кроме того, многие товары имеют дополнительные характеристики, не ограничивающиеся стандартными параметрами вроде наименования, цены и количества. В строительной сфере критически важны такие свойства, как:

– габаритные размеры (длина, ширина, толщина);

– цветовая гамма или оттенок;

– марка прочности и эксплуатационные характеристики;

– допустимый температурный режим;

– состав и особенности нанесения (актуально для лакокрасочных и клеевых материалов);

– производитель и страна происхождения.

Эти данные должны храниться в структуре, позволяющей не только отображение, но и фильтрацию, а также выборку по заданным критериям. Наличие расширенного фильтра и контекстного поиска по характеристикам товара повышает точность обслуживания и снижает нагрузку на сотрудников магазина.

Существенную роль играют и сезонные колебания спроса. В тёплое время года увеличивается потребность в фасадных материалах, садовом инвентаре, грунтовках и клеевых смесях, тогда как в зимний период растёт спрос на утеплители, обогреватели и плиточные материалы. В этой связи полезной представляется реализация аналитических функций в системе учёта – например, формирование отчётов по динамике продаж, категориям товаров или сезонной активности.

Особенности строительной торговли также включают в себя целый ряд нестандартных сценариев:

– поступление товаров без маркировки или штрихкодов;

– проведение инвентаризации вручную;

– фасовка продукции непосредственно на точке реализации;

– частые случаи возврата без заводской упаковки.

В таких условиях особенно важно, чтобы программное обеспечение было одновременно функциональным и не перегруженным. Интерфейс системы должен оставаться понятным и доступным для пользователей, не обладающих техническими знаниями. Гибкость логики и возможность корректировки операций без глубоких ИТ-навыков позволяют свести к минимуму сопротивление со стороны персонала при переходе на автоматизированный учёт.

Наконец, не менее важным фактором является интерфейсное удобство. В большинстве строительных магазинов с программой работают продавцы и заведующие складом, а не ИТ-специалисты. Поэтому при проектировании необходимо учитывать практические аспекты: крупные кнопки, ясные формы ввода, ограниченное количество полей и простую навигацию. Показатели юзабилити, такие как интуитивность и доступность интерфейса, должны закладываться в систему на самых ранних этапах проектирования.

Таким образом, эффективная автоматизация товарного учёта в строительной рознице требует не универсального, а специализированного подхода [18]. Он должен включать гибкую модель данных, поддержку

различных характеристик и единиц измерения, а также адаптированный под повседневную практику интерфейс, ориентированный на удобство конечного пользователя.

С целью наглядной демонстрации разнообразия характеристик, присущих различным товарным категориям в строительной рознице, в таблице 3 приведены примеры параметров, которые рекомендуется учитывать при проектировании информационной системы учёта. Эти характеристики не только определяют выбор продукции со стороны покупателя, но и играют ключевую роль в процессе поиска, фильтрации и корректного отображения товарных позиций в системе.

Таблица 3 – Примеры товарных характеристик в строительном магазине

| Товар | Основные характеристики |
|---|---|
| Гвозди | Длина, диаметр, тип стали, антикоррозийное покрытие |
| Обои | Цвет, ширина рулона, длина рулона, рисунок, фактура, основа |
| Ламинат | Толщина, класс износостойкости, цвет, тип замка, производитель, упаковка |
| Шпаклёвка | Назначение (старт/финиш), вес, тип (гипсовая, цементная), время высыхания |
| Пена монтажная | Объём, рабочая температура, однокомпонентная/двухкомпонентная, срок хранения |
| Уголки пластиковые | Длина, ширина полок, цвет, гибкость, материал |
| ДСП/ОСБ плиты | Толщина, длина, ширина, класс эмиссии, влагостойкость |
| Электроинструмент | Мощность, напряжение, комплектация, производитель, тип питания |
| Краска фасадная | Цвет, объем, тип поверхности, устойчивость к УФ, расход на $м^2$ |
| Смесители для ванной | Тип установки, количество режимов, материал корпуса, диаметр подключения |

Проведённый анализ показывает, что в условиях строительной розницы универсальные подходы к учёту работают слабо – слишком много нюансов, от характеристик товара до форматов хранения. Чтобы система работала надёжно и не требовала постоянной доработки, важно изначально заложить возможность расширения структуры данных и гибкой настройки фильтров.

## 1.5 Информационные потоки в торговой точке

Чёткая организация информационных потоков внутри магазина – ключевое условие для эффективного и достоверного товарного учёта. Под информационным потоком в данном случае понимается процесс движения данных между участниками торговой цепочки: от поставщика – к складу и продавцу, далее – в систему учёта и обратно, если происходят изменения.

В условиях строительной розницы, где ассортимент разнообразен и постоянно обновляется, особенно важно, чтобы на каждом этапе информация была обработана точно и своевременно. Выделим основные этапы работы с данными:

Поступление товара: когда продукция поступает в магазин, она сопровождается первичными документами: бумажными или электронными. В этот момент фиксируются такие данные, как наименование, количество, цена, базовые характеристики. Эти сведения необходимо оперативно внести в учётную систему – вручную или через импорт, если используется электронный документооборот.

Ввод и структурирование данных: оператор или администратор магазина регистрирует товарные позиции в системе, заполняя ключевые поля: наименование, категория, параметры, цена, единица измерения и др. На этом этапе важно, чтобы интерфейс системы был логичным и не перегруженным – ведь ввод часто выполняется в условиях ограниченного времени и большого потока поступлений.

Хранение и обновление информации: после ввода данные хранятся в централизованной базе, что позволяет обращаться к ним с любого рабочего места. При изменении цен, остатков или характеристик – информация обновляется. Важно, чтобы система сохраняла историю изменений и обеспечивала актуальность в реальном времени.

Поиск и взаимодействие с клиентом: Продавец, используя интерфейс системы, ищет нужный товар, проверяет его наличие и предоставляет

информацию покупателю. Возможность быстро находить товары по различным параметрам – не просто удобство, а практическая необходимость, особенно при большом ассортименте.

Корректировки и возвраты: в процессе работы возникают ситуации возврата товара, списания повреждённых позиций, переоценки. Все подобные изменения также должны фиксироваться в системе, чтобы учёт оставался точным и актуальным.

Для более наглядного представления структуры движения данных в рамках торгового процесса, на рисунке 2 приведена блок-схема основных информационных потоков внутри строительного магазина. Схема иллюстрирует этапы поступления информации в систему, её обработки, последующего использования сотрудниками на разных уровнях, а также механизмов внесения корректировок при изменении условий или состояния товарных позиций.

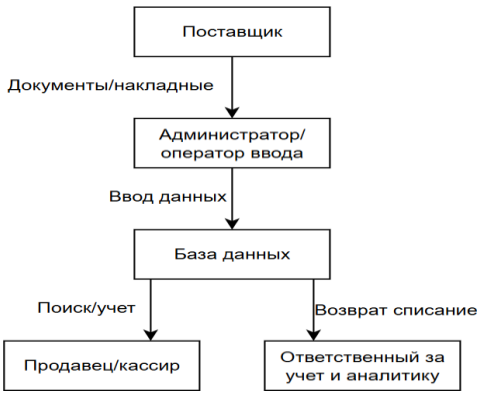



Рисунок 2 – Основные информационные потоки в магазине

Выводы по главе 1

В первой главе была рассмотрена предметная область, связанная с товарным учётом в строительных магазинах. Основное внимание уделялось тому, как ведётся учёт в небольших торговых точках, какие в этом процессе возникают сложности и почему автоматизация здесь становится особенно актуальной [18].

На практике выяснилось, что готовые программные решения часто оказываются неудобными: одни – слишком громоздкие и перегруженные функциями, другие – недоступны по стоимости. При этом ручной учёт, хоть и привычен, создаёт множество рисков: ошибки, потеря данных, отсутствие контроля [16].

Отдельно были отмечены особенности именно строительной торговли: большой ассортимент, разные единицы измерения, необходимость учитывать нестандартные характеристики товаров [6]. Всё это делает задачу автоматизации не универсальной, а требующей адаптации под конкретные условия.

Также была изучена организация работы внутри магазина – какие данные и в какой последовательности проходят через сотрудников. Это помогло понять, как должна быть устроена архитектура будущей системы, чтобы она действительно упрощала процесс, а не усложняла его.

По итогам анализа стало понятно: есть смысл разрабатывать собственное решение, ориентированное на конкретные задачи, с простым интерфейсом, локальной базой и возможностью настройки под конкретную торговую точку. Эти выводы стали отправной точкой для проектной части, где началась уже техническая реализация.

# Глава 2 Проектирование программного обеспечения

## 2.1 Постановка технического задания

Разработка программного обеспечения невозможна без предварительного анализа задач, которые должна решать создаваемая система. Как показал обзор в первой главе, типовые решения, представленные на рынке, не учитывают специфики малых строительных магазинов и, как правило, либо избыточны по функциональности, либо требуют значительных затрат на внедрение и сопровождение. В этих условиях возникает необходимость в создании автономного, компактного и интуитивно понятного программного продукта, ориентированного на конкретные условия эксплуатации.

Целью настоящего проекта является создание программного обеспечения для автоматизированного учёта товаров в строительном магазине. Приложение должно работать с локальной базой данных и иметь графический интерфейс, доступный для пользователей без специальной технической подготовки.

Система должна реализовывать следующие ключевые функции:

– ввод новой информации о товарных позициях;

– редактирование и удаление записей;

– быстрый поиск по множеству характеристик;

– фильтрация и сортировка по заданным параметрам.

Кроме того, программный продукт должен соответствовать ряду эксплуатационных критериев: устойчивость при работе, автономность (возможность функционировать без подключения к интернету), защищённость данных, эргономичный интерфейс и потенциальная масштабируемость, в частности, на уровне базы данных.

Обоснование необходимости разработки

Проведённый анализ текущего состояния автоматизации в малых

торговых точках строительного профиля выявил ряд типичных проблем:

– отсутствие возможности полноценного поиска по характеристикам товаров;
– ручной учёт остатков и операций (в блокнотах или Excel);
– сложности при работе с крупными таблицами и неструктурированными данными;
– высокая вероятность ошибок при переоценке или пересортице;
– отсутствие разграничения доступа, из-за чего даже простые действия сотрудников могут привести к потере или искажению данных.

Создание собственного программного решения позволяет устранить указанные недостатки и внедрить инструмент, полностью соответствующий реальным требованиям конкретного магазина, без избыточной функциональности и лишних зависимостей.

Общие требования к системе

На этапе проектирования определены следующие технические и функциональные параметры будущего программного продукта:

– Тип приложения: настольное (desktop), совместимое с ОС Windows;
– Язык разработки: C# с использованием платформы .NET;
– Графический интерфейс: Windows Forms (WinForms);
– Система управления базами данных: SQLite (встроенная, работает локально, не требует серверной части);
– ORM: Entity Framework – для реализации объектно-реляционного отображения;
– Архитектура: монолитная, с возможностью последующей модульной декомпозиции [9];
– Интерфейс: табличное отображение товарных позиций, кнопки действий (добавление, редактирование, удаление), панель поиска и фильтрации.

Для более полного представления архитектуры программного обеспечения и взаимодействия его основных компонентов, на рисунке 3

представлена расширенная контекстная диаграмма. Диаграмма отображает ключевые элементы системы: модули приложения, пользовательские роли, базу данных, а также конфигурационные файлы, с которыми осуществляется постоянный обмен. Отдельно обозначены зоны ответственности компонентов и их точки сопряжения, что позволяет наглядно очертить архитектурные границы проекта. Такое визуальное представление способствует более глубокому пониманию логики построения системы и роли каждого элемента в общем информационном процессе.

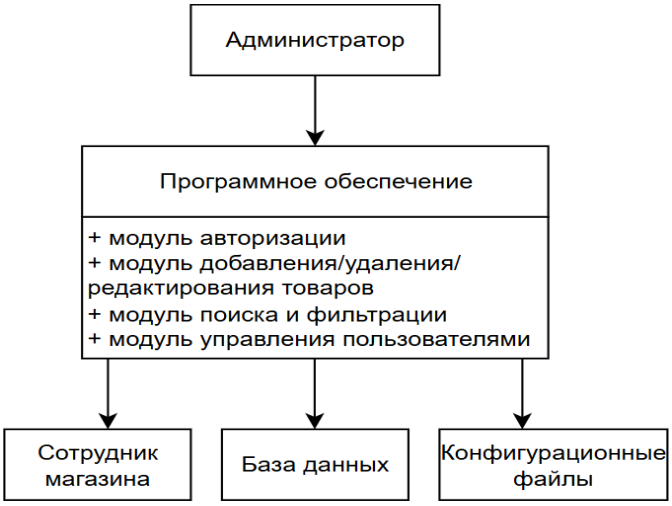



Рисунок 3 – Контекстная диаграмма архитектуры программного обеспечения

Система предполагает реализацию уровня пользовательского доступа, сотрудник магазина имеет доступ к просмотру и поиску товаров, добавление записей с фиксированным набором параметров, так же доступны функции удаление и редактирование. Эти функции необходимые для работы персоналу магазина для точного учёта товаров.

Архитектура разрабатываемого программного обеспечения строится на принципе модульности, предусматривающем логическое разделение функционала на отдельные компоненты [9]. Каждый модуль отвечает за выполнение конкретных задач, связанных с управлением товарными данными, взаимодействием с пользователями и отображением информации в

интерфейсе. Такой подход повышает читаемость и сопровождаемость кода, а также упрощает масштабирование и тестирование системы [8]. В таблице 4 приведён перечень основных модулей приложения с кратким описанием их назначения.

Таблица 4 – Основные модули программного обеспечения

| Модуль | Назначение |
|---|---|
| Учёт товаров | Добавление, удаление, редактирование записей |
| Поиск и фильтрация | Быстрый доступ к нужным позициям |
| База данных | Хранение сведений в SQLite |
| Интерфейс взаимодействия | Обеспечение простоты и доступности для сотрудника |

Разделение приложения на отдельные модули упрощает как процесс разработки, так и дальнейшее сопровождение системы. Каждый компонент отвечает за свою часть функционала, что делает архитектуру более понятной и предсказуемой. Такой подход позволяет не только быстрее вносить изменения, но и при необходимости расширять возможности программы – например, добавить экспорт данных, разграничение прав доступа или модуль формирования отчётов – без необходимости переделывать уже работающие части.

## 2.2 Требования к функциональности

Процесс проектирования программного обеспечения невозможен без чёткого структурирования требований, определяющих как функциональные, так и качественные характеристики системы. В рамках данной работы для формализации и систематизации требований используется модель FURPS++, получившая широкое распространение в индустриальной разработке ПО благодаря своей универсальности и охвату как функциональных, так и нефункциональных аспектов.

Модель FURPS++ позволяет упорядочить требования к программному обеспечению, распределяя их по тематическим категориям, каждая из которых отражает одно из ключевых качеств системы. Такой подход охватывает как поведение программы в стандартных и нестандартных условиях, так и удобство её использования, стабильность, производительность и потенциальную возможность масштабирования. Благодаря этой структуре удаётся взглянуть на проект всесторонне – не только с позиции функциональности, но и с учётом эксплуатационных рисков и требований к сопровождению.

С целью наглядного отображения требований, предъявляемых к разрабатываемому программному обеспечению, в настоящей работе применяется модель FURPS++. В таблице 5 представлена классификация этих требований по соответствующим категориям модели, что позволяет структурировано охватить как функциональные, так и нефункциональные характеристики системы. Подобное распределение способствует не только более глубокому анализу на этапе проектирования, но и упрощает дальнейшее развитие и сопровождение программного продукта.

Функциональность охватывает набор операций и возможностей, которые должна обеспечивать система в процессе своей работы. В рамках разработанного приложения реализованы следующие ключевые функции:

– выполнение базовых операций с товарными позициями: добавление, просмотр, редактирование и удаление записей;
– расширенный механизм поиска и фильтрации по различным параметрам – наименования, категории, производителю и другим характеристикам;
– защита критически важных действий посредством авторизации;
– локальное сохранение данных в базе SQLite без необходимости подключения к серверной инфраструктуре.

Такой функциональный набор покрывает основные потребности небольшого строительного магазина и обеспечивает стабильное выполнение

учётных операций.

Таблица 5 – Требования к системе по модели FURPS++

| Требование | Статус | Полезность | Риск | Стабильность |
|---|---|---|---|---|
| Functionality (Функциональность) | | | | |
| Добавление, редактирование и удаление товаров | Одобрено | Критическая | Средний | Высокая |
| Поиск товаров по названию и характеристикам | Одобрено | Критическая | Средний | Высокая |
| Разграничение доступа по ролям (админ/сотрудник) | Одобрено | Высокая | Средний | Средняя |
| Сохранение данных в SQLite | Одобрено | Высокая | Низкий | Высокая |
| Usability (Удобство использования) | | | | |
| Интуитивный интерфейс с крупными кнопками и читаемыми полями | Одобрено | Критическая | Низкий | Высокая |
| Поддержка русского языка | Одобрено | Высокая | Низкий | Высокая |
| Минимальное количество действий на стандартную операцию | Одобрено | Высокая | Низкий | Высокая |
| Reliability (Надёжность) | | | | |
| Устойчивость к некорректному вводу | Одобрено | Высокая | Средний | Высокая |
| Сохранность данных при аварийном завершении | Предложено | Высокая | Средний | Средняя |
| Performance (Производительность) | | | | |
| Время отклика при поиске – не более 1 секунды | Одобрено | Критическая | Средний | Высокая |
| Работа с 1000+ товаров без потери производительности | Одобрено | Высокая | Средний | Высокая |
| Supportability (Сопровождаемость) | | | | |
| Комментированный исходный код | Одобрено | Высокая | Низкий | Высокая |
| Чёткое разделение модулей | Одобрено | Высокая | Низкий | Высокая |
| +Security (Безопасность) | | | | |
| Ограничение доступа к базе данных извне | Предложено | Высокая | Средний | Средняя |
| ++Project Constraints (Проектные ограничения) | | | | |
| Завершение проекта за 3 месяца | Одобрено | Высокая | Средний | Средняя |
| Отсутствие внешних зависимостей кроме .NET Framework и SQLite | Предложено | Высокая | Низкий | Высокая |

Удобство использования является одним из ключевых факторов, учитывая, что конечными пользователями системы выступают сотрудники без специальной подготовки в области ИТ. Система должна быть

максимально понятной и адаптированной к условиям высокой нагрузки и ограниченного времени. Основные требования к интерфейсу включают:

– интуитивно понятную навигацию, не требующую предварительного обучения;
– визуально доступные элементы управления: крупные кнопки, читаемые поля ввода, простая логика переходов;
– логичноерасположение элементов, соответствующее привычному рабочему сценарию;
– полная локализация на русском языке.

Продуманный интерфейс способствует снижению количества ошибок, ускоряет выполнение повседневных задач и облегчает внедрение системы в рабочую среду магазина.

Надёжность приобретает особое значение в условиях локальной эксплуатации, при отсутствии резервных серверов и внешней поддержки. В связи с этим система должна соответствовать следующим критериям:

– устойчивость к некорректным действиям пользователя (например, попыткам ввести текст в числовое поле);
– сохранение целостности данных при неожиданном завершении работы;
– корректная обработка исключений без остановки интерфейса [22];
– возможность отмены не сохранённых изменений на этапе редактирования.

Производительность напрямую влияет на восприятие системы пользователем. Даже при значительном объёме данных – свыше 1000 записей – интерфейс должен оставаться отзывчивым и стабильным. Планируемые показатели:

– отклик на стандартные действия (поиск, фильтрация, открытие форм) – в пределах одной секунды;
– моментальное применение условий сортировки и фильтрации;
– стабильная работа на компьютерах с базовой офисной

конфигурацией. Сопровождаемость определяет удобство внесения изменений в программный код и его последующего развития. Для обеспечения долгосрочной жизнеспособности системы предусмотрены следующие меры:

– структурирование проекта по модулям с логическим разделением ответственности;
– использование комментариев, поясняющих работу ключевых участков кода;
– применение общепринятых и поддерживаемых технологий (в частности, WinForms и Entity Framework);
– простота установки, в том числе возможность запуска без дополнительной настройки (формат «из коробки»).

Дополнительные характеристики, выходящие за рамки основных категорий модели FURPS++, включают:

– автономность – система функционирует без подключения к интернету;
– портативность – возможность запуска с внешнего носителя без предварительной установки;
– совместимость – корректная работа в среде операционной системы Windows;

## 2.3 Архитектура и выбор технологий

Архитектура программного обеспечения играет решающую роль в процессе разработки, определяя как внутреннюю организацию компонентов, так и способность системы адаптироваться к изменениям, масштабироваться и эффективно сопровождаться в долгосрочной перспективе. Грамотно выстроенная архитектурная модель снижает связность между модулями, упрощает проведение модульного тестирования и обеспечивает устойчивую работу приложения при наращивании функциональности.

На основе анализа сформулированных ранее функциональных и нефункциональных требований в качестве архитектурного решения для программного обеспечения учёта товарных позиций в строительном магазине была выбрана трёхуровневая архитектура. Этот подход зарекомендовал себя как надёжный и гибкий вариант при разработке настольных приложений, позволяющий чётко разграничить ответственность между логическими слоями системы [21].

Уровень представления отвечает за взаимодействие с пользователем и реализован с использованием технологии Windows Forms (WinForms). Через графический интерфейс осуществляется ввод данных, их отображение и начальная обработка. Формы содержат элементы управления (поля ввода, списки, кнопки и пр.), которые обеспечивают выполнение основных пользовательских сценариев.

Особое внимание при разработке интерфейса уделялось его простоте и доступности – интерфейс интуитивно понятен и не требует технической подготовки [14]. Каждая форма ориентирована на выполнение конкретной задачи: добавление товара, редактирование информации, выполнение поиска по заданным параметрам и т. д.

На уровне бизнес-логике сосредоточена ключевая логика работы приложения: правила обработки информации, валидация пользовательских данных, контроль доступа, маршрутизация запросов. Данный слой выполняет функцию посредника между пользовательским интерфейсом и механизмами работы с базой данных [20].

Бизнес-логика реализована через отдельные классы, такие как ProductService, SearchManager, AccessController. Это позволяет централизованно управлять основными алгоритмами системы, устранять дублирование кода и поддерживать высокую читаемость проекта. В рамках этого слоя выполняются, в частности:

– проверка корректности вводимых пользователем значений;

– фильтрация товарных позиций по множеству параметров.

Уровень доступа к данным обеспечивает взаимодействие приложения с базой данных и реализован с использованием технологии Entity Framework, позволяющей манипулировать данными через объектную модель на языке C#, минуя прямую работу с SQL-запросами. Это упрощает разработку и повышает надёжность операций с данными.

Для наглядного представления внутренней структуры программного обеспечения на рисунке 4 отражена его трёхуровневая архитектура. Каждый уровень выполняет чётко очерченную функцию: слой представления обеспечивает взаимодействие пользователя с системой, уровень бизнес-логики обрабатывает запросы согласно встроенным правилам, а уровень доступа к данным отвечает за сохранность и извлечение информации из базы данных.

В пределах каждого слоя выделены основные компоненты, обеспечивающие выполнение ключевых операций системы учёта. Подобное логическое разделение способствует повышению модульности архитектуры, облегчает сопровождение и даёт возможность вносить изменения в один из уровней без затрагивания остальных частей системы.

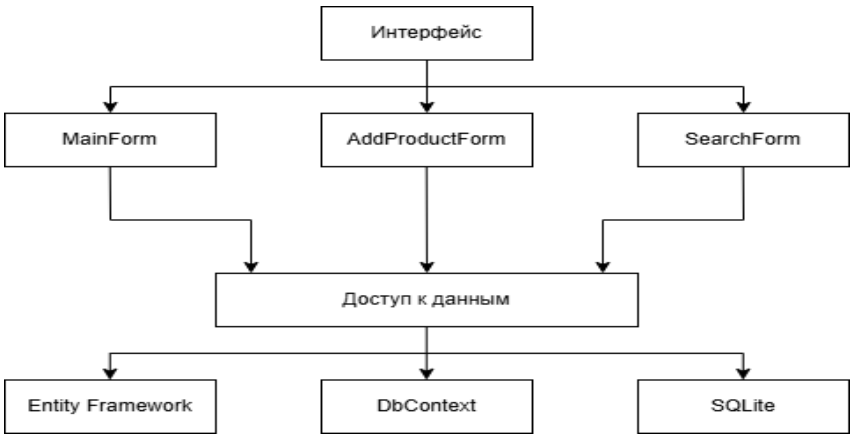



Рисунок 4 – Общая архитектура программного обеспечения

При разработке программного продукта были использованы технологии, обеспечивающие надёжность, стабильность и простоту

поддержки решения в условиях ограниченной инфраструктуры. Выбор инструментов основывался на их совместимости с архитектурой системы, доступностью, а также опытом их применения в аналогичных проектах.

В качестве основных средств разработки применялись:

– C# и .NET Framework – язык и платформа, обеспечивающие высокий уровень надёжности и широкие возможности для создания настольных приложений с графическим интерфейсом и доступом к базам данных [25];
– Windows Forms – классическая технология построения пользовательского интерфейса, проверенная временем и обеспечивающая достаточную гибкость для реализации типовых сценариев взаимодействия;
– Entity Framework – современная ORM-библиотека, позволяющая работать с базой данных через объектную модель, тем самым снижая количество шаблонного SQL-кода и упрощая поддержку приложения;
– SQLite – легкая встроенная СУБД, не требующая серверного развёртывания, что делает её оптимальным решением для локальных программ, ориентированных на использование в малом бизнесе.

Комплексное применение указанных технологий позволило создать автономную, легко настраиваемую и масштабируемую систему, адаптированную под нужды торгового предприятия и минимизирующую затраты на развёртывание и сопровождение.

Для более полного понимания логики функционирования приложения на рисунке 5 представлена схема взаимодействия основных компонентов системы. Диаграмма иллюстрирует последовательность обработки пользовательских действий, инициируемые через графический интерфейс команды передаются на уровень бизнес-логики, где осуществляется их обработка в соответствии с установленными правилами, после чего данные направляются в базу данных.


Рисунок 5 – Взаимодействие компонентов

В качестве системы хранения используется встроенная СУБД SQLite, не требующая отдельной серверной установки. Такой выбор обусловлен необходимостью обеспечить автономную работу программы на любом компьютере под управлением Windows без сложной предварительной настройки. База данных SQLite обладает всеми необходимыми характеристиками для малого бизнеса: она лёгкая, надёжная, поддерживает транзакции и без проблем справляется с объёмами данных, измеряемыми десятками тысяч записей [3].

Таким образом, выбранная архитектура и применённые технологии разработки обеспечивают создание надёжного и функционально прозрачного программного решения, полностью соответствующего требованиям учёта товарных позиций в условиях работы строительного магазина. Построенная система отличается хорошей масштабируемостью, удобством сопровождения и гибкостью, позволяющей адаптировать её под новые задачи без необходимости в глубокой переработке существующего программного кода.

### 2.4 Структура базы данных

База данных является ключевым элементом любой информационной системы, обеспечивая надёжное хранение, структурированный доступ и эффективную обработку данных. В рамках разрабатываемого приложения для учёта товарных позиций в строительном магазине используется встроенная реляционная система управления базами данных SQLite. Её

применение позволяет отказаться от серверной инфраструктуры, упростить развёртывание и обеспечить высокую производительность, что делает её оптимальным выбором для автономных настольных решений малого масштаба.

Основным требованием к базе данных является обеспечение быстрого и устойчивого доступа к информации о товарных единицах. Все ключевые действия пользователя – добавление, редактирование, удаление и поиск – реализуется через обращения к соответствующим таблицам, центральной из которых выступает таблица Products, содержащая основные характеристики товаров.

База данных в данном контексте представляет собой не просто хранилище, а критически важный компонент системы учёта, от качества проектирования которого напрямую зависят корректность инвентаризации, точность остатков, скорость отклика и возможность последующего масштабирования функциональности. Ошибки на уровне структуры – от отсутствующих ограничений до нарушений связей между таблицами – способны привести к сбоям в бизнес-процессах и негативным экономическим последствиям. Именно поэтому проектирование базы данных должно быть основано на принципах надёжности, целостности и адаптивности.

В качестве технологии взаимодействия с базой данных используется Entity Framework (EF) – объектно-реляционный маппер, позволяющий работать с таблицами как с объектами, минуя прямое написание SQL-запросов. В рамках данного проекта применяется подход Code First, при котором структура базы данных формируется автоматически на основе классов, описанных в программном коде. Это упрощает процесс разработки, снижает количество рутинных операций и делает проект более гибким в плане сопровождения: любые изменения в модели данных вносятся на уровне кода и затем автоматически отражаются в структуре базы.

Применение EF обеспечивает:

– автоматическую генерацию таблиц и связей на основании классов;

– валидацию данных на уровне модели;

– централизованную обработку запросов к базе;

– поддержку транзакционности и защиту от SQL-инъекций.

Основной сущностью системы является класс Product, описывающий характеристики товарной единицы. Помимо стандартных атрибутов, таких как наименование, цена и количество, в модель можно включить поле DateAdded по необходимости, позволяющее отслеживать дату внесения позиции и проводить временной анализ. В перспективе возможно добавление поля IsArchived для реализации логического удаления и фильтрации устаревших записей без их физического удаления из базы.

Взаимодействие между моделью и базой данных осуществляется через класс AppDbContext, унаследованный от DbContext. Этот компонент отвечает за конфигурацию подключения, отображение сущностей на таблицы, управление транзакциями и сохранением данных. Все операции – добавление, обновление, удаление и выборка – выполняются через методы данного контекста, что обеспечивает централизованное управление доступом к данным и позволяет гибко настраивать поведение EF с использованием Fluent API при необходимости.

Для визуализации структуры модели на рисунке 6 представлена диаграмма класса Product, отражающая соответствие между полями базы данных и свойствами программной сущности (см. рис. 6).

Несмотря на минималистичную текущую структуру базы данных, построенную без использования внешних ключей, такое архитектурное решение оправдано ориентиром на простоту, автономность и высокую скорость работы. В условиях ограниченного объёма данных и специфики настольного приложения подобная схема оказывается наиболее рациональной.

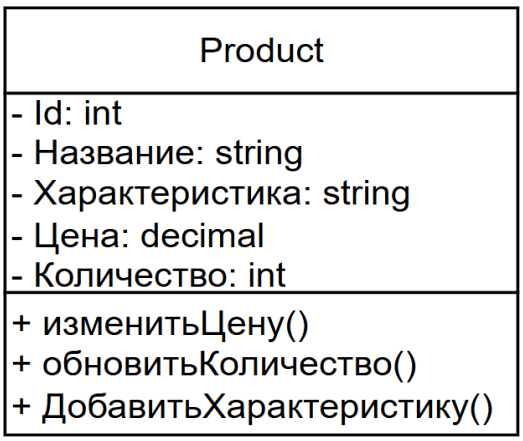



Рисунок 6 – Диаграмма класса Product

Вместе с тем архитектура системы изначально предусматривает возможность масштабирования: при необходимости структура базы может быть расширена за счёт добавления связанных таблиц, таких как Categories, Logs, Suppliers. Это позволит перейти к нормализованной модели с поддержкой связей «один-ко- многим» (например, категория – товары), механизма аудита изменений (фиксация даты и пользователя, внёсшего правку), а также отслеживания поставщиков.

Таким образом, структура базы данных в проекте не задаётся вручную через SQL-описания таблиц, а формируется на основе объектно-ориентированных моделей, определённых в коде. Entity Framework автоматически преобразует эти модели в соответствующие таблицы, обеспечивая высокую читаемость, устойчивость архитектуры и удобство сопровождения. Такой подход также закладывает основу для гибкой эволюции системы в будущем – без необходимости полной переработки существующей логики.

### 2.5 Моделирование предметной области

Процесс разработки программного обеспечения невозможно представить без предварительного и вдумчивого анализа предметной области.

Именно с этого этапа начинается формирование архитектуры будущей системы: через выявление ключевых сущностей, определение их свойств, поведения и взаимосвязей. Такой подход позволяет не только описать объекты, с которыми будет непосредственно взаимодействовать пользователь, но и сформировать внутреннюю логику приложения, включая способы хранения, обработки и передачи данных.

В рамках создания программного продукта для автоматизации учёта в строительном магазине в качестве базового инструмента моделирования использована диаграмма классов UML. Этот тип диаграмм широко применяется при объектно-ориентированном подходе к проектированию и позволяет детально представить как статические компоненты системы (классы с их атрибутами и методами), так и логические связи между ними: ассоциации, агрегации и композиции.

Ключевые элементы модели и предметная область в рассматриваемом проекте включает несколько центральных сущностей:

– Товар – единица учёта и продажи, характеризуемая наименованием, ценой, количеством, описанием, артикулом и поставщиком. Класс реализует методы для корректировки стоимости, обновления остатков и проверки наличия на складе;

– Сотрудник – описывает персонал магазина. Включает поля, отражающие ФИО, должность, стаж, зарплату и идентификатор. Поддерживает методы, связанные с продажами, приёмкой товаров и возвратами.

– Поставщик – юридическое или физическое лицо, осуществляющее поставку продукции. Атрибуты: название, контактные данные, ИНН, перечень предоставляемых товаров. Методы обеспечивают обновление сведений и выполнение поставки;

– Магазин – обобщающая сущность, в которой объединяются товары, сотрудники, клиенты, поставщики и заказы. Реализует функции управления персоналом, товарным ассортиментом и оформлением

заказов;

– Клиент – покупатель, совершающий транзакции. Описывается через ФИО, контактные данные, историю покупок, текущую скидку и баланс. Предусмотрены действия по оформлению заказа, оплате и проверке остатка средств;

– Заказ – фиксирует факт продажи. Основные поля: номер, дата, сумма, статус, перечень позиций. Методы позволяют оформить заказ, аннулировать его, изменить статус, а также рассчитать итоговую стоимость.

На диаграмме классов отражены логические связи между вышеуказанными объектами. Например:

– один магазин может включать в себя множество клиентов, сотрудников, заказов и товаров;

– один заказ ассоциируется с конкретным клиентом, но может включать несколько товарных позиций;

– поставщик, в свою очередь, может быть источником сразу нескольких товаров.

Для фиксации связей использованы стандартные элементы UML – ассоциации, композиции и указания кратности, что повышает читаемость и однозначность структуры.

Значение модели для проекта

Построенная модель предметной области играет ключевую роль в проектировании программной архитектуры. Она позволяет:

– сформировать структуру базы данных с учётом всех сущностных связей;

– описать бизнес-логику и пользовательские сценарии;

– обеспечить целостность и непротиворечивость системы на всех этапах её развития.

Кроме того, наличие чётко оформленной модели способствует лучшему пониманию проекта как разработчиками, так и заказчиками. Это

особенно актуально при командной работе, сопровождении системы или масштабировании функционала в будущем. Модель выступает своего рода

«каркасом» приложения, на который опираются все остальные компоненты.

В дополнение к структурной модели полезно визуализировать один из типовых бизнес-процессов, поддерживаемых системой. На рисунке 7 представлена диаграмма деятельности, иллюстрирующая последовательность действий при добавлении или редактировании товара.

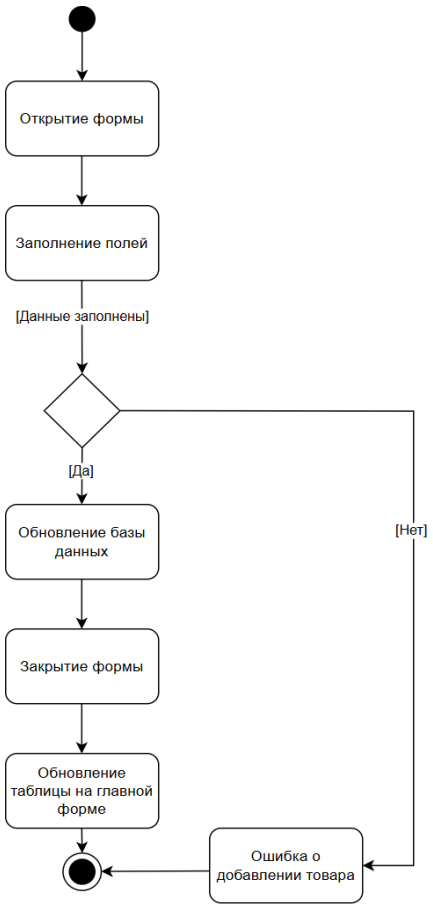



Рисунок 7 – Диаграмма деятельности добавления товара

Диаграмма отражает шаги, которые проходит пользователь: от открытия формы до ввода данных и их сохранения. Также показаны действия со стороны системы – проверка корректности, запись в базу и обновление интерфейса. Такая наглядная схема помогает лучше понять логику работы и

последовательность операций внутри приложения.

Для наглядного представления ключевых возможностей создаваемого программного обеспечения на рисунке 8 представлена диаграмма вариантов использования. С её помощью обобщённо отображаются доступные действия для пользователя. Диаграмма также отражает внешние взаимодействия, происходящие через графический интерфейс, включая операции по управлению данными и настройке системы.

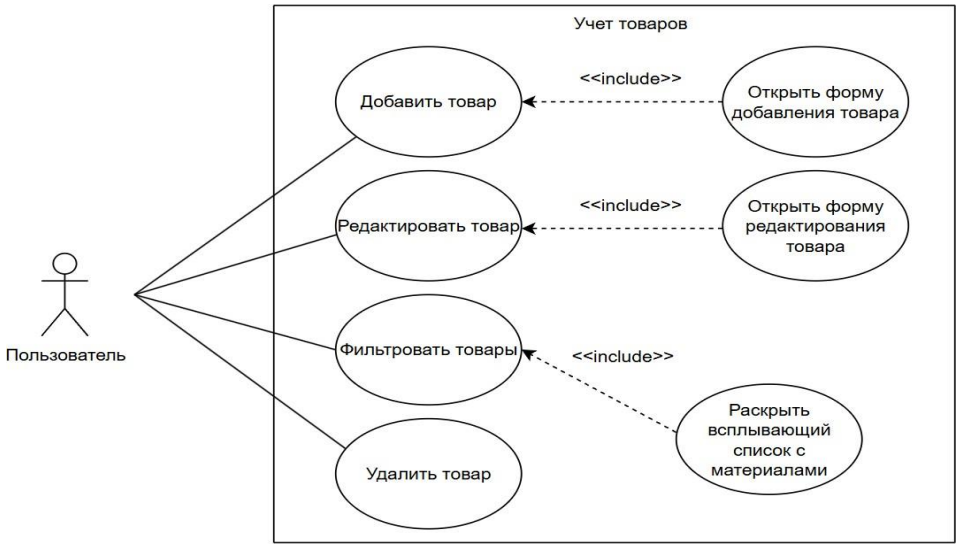



Рисунок 8 – Диаграмма вариантов использования модуля

На рисунке 9 приведена диаграмма классов, выполненная в нотации UML, иллюстрирующая логическую структуру ключевых элементов предметной области, связанной с функционированием разрабатываемого программного обеспечения. Схема охватывает набор основных классов с их характеристиками (атрибутами) и действиями (методами), а также описывает типы взаимосвязей между ними. Особое внимание уделено отражению механизмов учёта товарных позиций, процедуры формирования и обработки заказов, схемы взаимодействия с поставщиками, а также организации клиентского обслуживания.

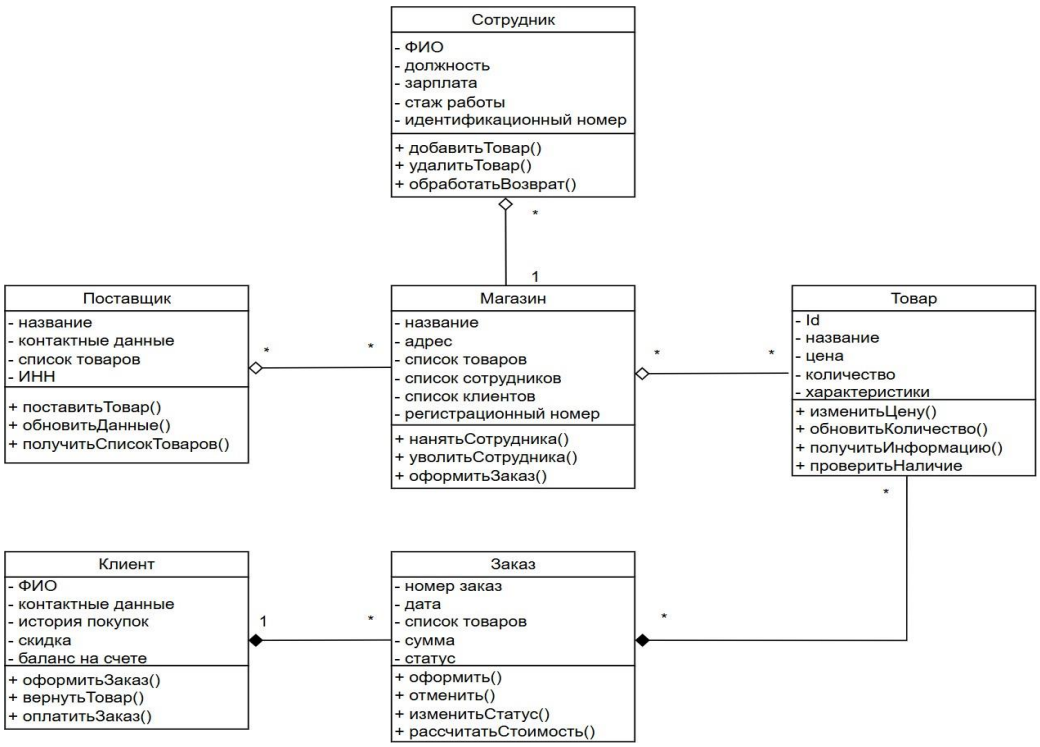



Рисунок 9 – Диаграмма классов предметной области

Этап моделирования предметной области играет ключевую роль в процессе проектирования программного обеспечения, так как именно на этом этапе закладывается логическая непротиворечивость и структурная целостность будущей системы. Применение диаграммы классов в рамках этого этапа даёт возможность не только идентифицировать основные сущности предметной области, но и формализовать их поведение, а также установить характер взаимосвязей между ними. Полученные в результате моделирования данные служат основой для конструирования структуры базы данных, определения архитектурных связей между модулями и проектирования пользовательских сценариев.

Выводы по главе 2

Во второй главе были рассмотрены основные проектные решения, от которых напрямую зависит архитектура и техническая реализация создаваемой системы. За основу была взята трёхуровневая архитектура, где

логика распределена между интерфейсом пользователя, бизнес-слоем и компонентом, отвечающим за работу с базой данных [7]. Подобная схема на практике показала свою надёжность и удобство: при необходимости можно менять отдельные элементы, не задевая всю систему целиком.

Технологии подбирались с учётом особенностей и предполагаемого сценария использования приложения. Язык программирования C# оказался оптимальным из-за его совместимости с выбранной платформой и широкого набора инструментов [8]. Интерфейс реализован на Windows Forms – это решение не требует лишних ресурсов и обеспечивает достаточную простоту для конечного пользователя [11]. В качестве СУБД использована SQLite – автономная и не требующая отдельного сервера база данных, что особенно важно для настольных решений [24]. Для связи с базой выбран Entity Framework: он позволяет обращаться к данным через объектную модель, без необходимости вручную формулировать SQL-запросы, что упрощает разработку и снижает риски ошибок.

Значительная часть внимания в этой главе была также уделена структуре базы данных и уточнению предметной области. Были выделены основные сущности (например, продукт), а их характеристики и связи описаны с помощью диаграммы классов UML [12]. Такая визуализация помогает не только в проектировании, но и в понимании логики системы в целом – особенно при переходе к этапу реализации.

# Глава 3 Реализация программного обеспечения

## 3.1 Основные формы и интерфейсы

В данной главе рассмотрен процесс практической реализации программного обеспечения, предназначенного для автоматизации учёта товарных позиций в строительном магазине. Основное внимание уделяется пользовательскому интерфейсу как ключевому элементу взаимодействия конечного пользователя с системой. От его удобства и логики зависит эффективность работы и общее восприятие приложения.

Пользовательский интерфейс

Интерфейс реализован с использованием технологии Windows Forms, которая предоставляет гибкий инструментарий для создания классических настольных приложений. Все формы выдержаны в едином визуальном стиле, ориентированном на простоту восприятия и минимальную нагрузку на пользователя. Элементы управления сгруппированы логически, что позволяет ускорить навигацию и облегчает выполнение повседневных операций.

Главная форма (MainForm)

Главное окно приложения служит отправной точкой для всех пользовательских действий. Здесь отображается список всех товаров, представленный в виде таблицы (DataGridView). Пользователю доступны основные функции:

– просмотр и сортировка товарных позиций;

– добавление новых записей;

– редактирование и удаление существующих данных;

– поиск по наименованию и другим характеристикам.

Панель инструментов содержит кнопки быстрого доступа к основным операциям. Кроме того, реализована строка поиска с возможностью фильтрации данных по ключевым полям. Из главной формы пользователь

может перейти ко всем остальным частям приложения. Представленный на рисунке 10 метод ApplySearch() реализует функциональность фильтрации списка товарных позиций на основе введённых пользователем параметров: наименования и материала. На первом этапе метод извлекает значения из текстового поля и выпадающего списка, приводит их к нижнему регистру и последовательно применяет фильтрацию к исходному списку товаров.

```
5 references
private void LoadProducts()
{
    products = Product.GetAllProducts();
    ApplySearch();
}

2 references
private void ApplySearch()
{
    var filtered = products;

    string nameSearch = textBoxSearchName.Text.Trim().ToLower();
    string materialSearch = comboBoxSearchMaterial.SelectedItem?.ToString()?.ToLower() ?? "";

    if (!string.IsNullOrWhiteSpace(nameSearch))
        filtered = filtered.Where(p => p.ProductName.ToLower().Contains(nameSearch)).ToList();

    if (!string.IsNullOrWhiteSpace(materialSearch))
        filtered = filtered.Where(p => p.Material.ToLower().Contains(materialSearch)).ToList();

    dataGridViewProducts.DataSource = null;
    dataGridViewProducts.DataSource = filtered;

    if (dataGridViewProducts.Columns["ProductName"] != null) dataGridViewProducts.Columns["ProductName"].HeaderText = "Название";
    if (dataGridViewProducts.Columns["Price"] != null) dataGridViewProducts.Columns["Price"].HeaderText = "Цена";
    if (dataGridViewProducts.Columns["Quantity"] != null) dataGridViewProducts.Columns["Quantity"].HeaderText = "Количество";
    if (dataGridViewProducts.Columns["Description"] != null) dataGridViewProducts.Columns["Description"].HeaderText = "Описание";
    if (dataGridViewProducts.Columns["Material"] != null) dataGridViewProducts.Columns["Material"].HeaderText = "Материал";
}
```

Рисунок 10 – Код, реализующий логику фильтрации данных по параметрам

Если поле поиска по наименованию не пустое, выполняется отбор элементов, в которых строка ProductName содержит указанное значение. Аналогично, при выборе материала производится фильтрация по полю Material. Полученный отфильтрованный список назначается в качестве нового источника данных для компонента DataGridView.

В завершение метод вручную переименовывает заголовки столбцов таблицы, заменяя системные имена на понятные пользователю названия на русском языке. Такая реализация обеспечивает интерактивную фильтрацию данных без необходимости обновления всей формы, что делает работу с системой более удобной и наглядной. На рисунке 11 приведён фрагмент

кода, в котором обрабатываются действия пользователя – переключение темы оформления и работа с товарами.

```
1 reference
private void checkBoxDarkTheme_CheckedChanged(object sender, EventArgs e)
{
    bool dark = checkBoxDarkTheme.Checked;
    Settings.DarkTheme = dark;
    ApplyTheme(dark);
}

1 reference
private void buttonAdd_Click(object sender, EventArgs e)
{
    var form = new AddProductForm(Settings.DarkTheme);
    form.Text = "Добавить товар";

    if (form.ShowDialog() == DialogResult.OK)
    {
        var newProduct = form.GetProduct();
        newProduct.Save();
        LoadProducts();
    }
}

1 reference
private void buttonEdit_Click(object sender, EventArgs e)
{
    if (dataGridViewProducts.CurrentRow?.DataBoundItem is Product product)
    {
        var form = new AddProductForm(Settings.DarkTheme);
        form.Text = "Редактировать товар";
        form.LoadProduct(product);

        if (form.ShowDialog() == DialogResult.OK)
        {
            var updated = form.GetProduct();
            updated.Id = product.Id;
            updated.Save();
            LoadProducts();
        }
    }
}
```

Рисунок 11 – Код, отвечающий за обработку событий переключение темы и операций добавления, редактирования товаров

Метод checkBoxDarkTheme_CheckedChanged отвечает за смену светлой и тёмной темы. При изменении состояния флажка новое значение сохраняется в настройках, а внешний вид приложения сразу обновляется.

Кнопки «Добавить» и «Редактировать» запускают одну и ту же форму (AddProductForm), но в разных режимах. Если пользователь добавляет новый товар – форма открывается пустой. При редактировании – в неё

автоматически подставляются данные выбранной записи. После сохранения изменений список товаров обновляется.

Этот участок кода отражает основные принципы работы интерфейса: реакцию на действия пользователя, настройку внешнего вида и базовые функции по учёту товаров.

Рисунок 12 иллюстрирует участок интерфейсного кода, задействованного в реализации базовых операций по управлению товарными записями.

```
1 reference
private void buttonDelete_Click(object sender, EventArgs e)
{
    if (dataGridViewProducts.CurrentRow?.DataBoundItem is Product product)
    {
        Product.Delete(product.Id);
        LoadProducts();
    }
}

1 reference
private void buttonFilterPrice_Click(object sender, EventArgs e)
{
    ApplySearch();
}

1 reference
private void buttonClearFilter_Click(object sender, EventArgs e)
{
    textBoxSearchName.Text = "";
    comboBoxSearchMaterial.SelectedIndex = 0;
    LoadProducts();
}
```

Рисунок 12 – Код, отвечающий за операцию удаления, фильтрацию и сброса фильтров

Метод buttonDelete_Click используется для удаления выбранной позиции из списка. Перед выполнением действия производится проверка, действительно ли в таблице выделена строка, после чего извлекается объект Product. Если условие выполнено, вызывается метод Product.Delete, и список данных обновляется, чтобы отразить изменения.

Следующий метод – buttonFilterPrice_Click – инициирует фильтрацию по заданным параметрам, передавая управление функции ApplySearch. Она

отбирает и отображает только те позиции, которые соответствуют критериям пользователя. При этом метод buttonClearFilter_Click выполняет противоположную задачу: очищает поля поиска, сбрасывает фильтр и вновь загружает исходный набор данных.

В совокупности эти действия охватывают основные этапы работы пользователя с приложением – от редактирования информации до удаления и поиска. Такая логика построения интерфейса обеспечивает завершённость пользовательского сценария и отвечает функциональным задачам, обозначенным при проектировании системы.

Форма добавления и редактирования (AddEditProductForm): создание и редактирование товарных записей осуществляется через отдельную универсальную форму. Она вызывается из главного окна по соответствующей кнопке и автоматически адаптируется под контекст: при добавлении – поля остаются пустыми, при редактировании – предварительно заполняются текущими значениями.

Интерфейс формы включает:

– текстовые поля для наименования, категории и описания;

– числовые поля для количества и цены с проверкой на корректность;

– выбор даты добавления с помощью календаря;

– кнопки "Сохранить" и "Отмена".

Изменения, внесённые пользователем, сразу сохраняются в базе данных и отображаются в общем списке. Рисунок 13 демонстрирует часть кода, относящегося к форме AddProductForm, а именно – её конструктор и метод LoadProduct, который используется при редактировании уже добавленного товара.

```
2 references
public AddProductForm(bool darkTheme = false)
{
    InitializeComponent();
    comboBoxMaterial.Items.AddRange(new object[] {
        "Дерево", "Железо", "Нержавеющая сталь", "Алюминий", "Смесь", "Пластмасса", "Жидкость"
    });
    comboBoxMaterial.SelectedIndex = 0;

    if (darkTheme)
    {
        this.BackColor = System.Drawing.Color.FromArgb(45, 45, 48);
        this.ForeColor = System.Drawing.Color.White;
        foreach (Control ctrl in this.Controls)
        {
            ctrl.BackColor = this.BackColor;
            ctrl.ForeColor = this.ForeColor;
        }
    }
}

1 reference
public void LoadProduct(Product product)
{
    textBoxName.Text = product.ProductName;
    numericUpDownPrice.Value = product.Price;
    numericUpDownQuantity.Value = product.Quantity;
    textBoxDescription.Text = product.Description;
    comboBoxMaterial.SelectedItem = product.Material;
}
```

Рисунок 13 – Фрагмент кода, отвечающий за загрузку данных товара

Конструктору передаётся параметр darkTheme, определяющий, в каком цветовом режиме будет работать интерфейс. Если активирована тёмная тема, под неё автоматически подстраивается оформление: меняется цвет фона, текста и элементов управления, чтобы обеспечить визуальное соответствие.

Кроме того, в конструкторе инициализируется список comboBoxMaterial, куда заранее загружаются возможные материалы, из которых может быть изготовлен товар. Это позволяет пользователю быстро выбрать нужный вариант из предложенного набора, что особенно удобно при заполнении формы.

Метод LoadProduct нужен для того, чтобы при открытии формы на редактирование значения нужного товара автоматически подставлялись в соответствующие поля: название, цена, количество, описание и материал. Такой подход позволяет избежать повторного ввода данных вручную и делает работу с формой более наглядной и быстрой.

На рисунке 14 показан обработчик события buttonSave_Click, который срабатывает при сохранении изменений в карточке товара

```
1 reference
private void buttonSave_Click(object sender, EventArgs e)
{
    string name = textBoxName.Text;
    decimal price = numericUpDownPrice.Value;
    int quantity = (int)numericUpDownQuantity.Value;
    string description = textBoxDescription.Text;

    string query = "UPDATE Products SET Name = @name, Price = @price, Quantity = @quantity, Description = @description WHERE Id = @id";
    using (var command = new SQLiteCommand(query, dbConnection))
    {
        command.Parameters.AddWithValue("@name", name);
        command.Parameters.AddWithValue("@price", price);
        command.Parameters.AddWithValue("@quantity", quantity);
        command.Parameters.AddWithValue("@description", description);
        command.Parameters.AddWithValue("@id", productId);

        command.ExecuteNonQuery();
    }

    MessageBox.Show("Product updated successfully!");
    this.DialogResult = DialogResult.OK; // Это для закрытия формы и обновления данных
}
```

Рисунок 14 – Фрагмент кода, отвечающий за события сохранения в бд

После нажатия кнопки метод собирает данные из полей формы и использует их для формирования SQL-запроса на обновление соответствующей строки в базе. Все значения передаются через объект SQLiteCommand с параметрами, что позволяет не вставлять данные напрямую в текст запроса.

Такой способ не только упрощает поддержку кода, но и защищает от SQL-инъекций – на практике это критично, особенно если система предполагает ввод данных от пользователя. После успешного выполнения запроса появляется сообщение, что товар обновлён, и форма закрывается с результатом DialogResult.OK. Это означает, что на основной форме можно сразу перезагрузить таблицу и отобразить уже изменённые данные. Фрагмент показывает, как интерфейс и база данных связаны между собой, и как реализуется один из часто встречающихся сценариев – редактирования и сохранение записи.

Валидация данных и контроль доступа. Для повышения надёжности работы интерфейс снабжён встроенной валидацией. Пользователь не может

ввести отрицательные значения, оставить обязательные поля пустыми или указать некорректную цену. При попытке сохранить неверные данные появляется поясняющее сообщение об ошибке.

Дополнительно реализована логика ограничения прав доступа: в случае запуска приложения без административных полномочий функции редактирования и удаления товаров становятся недоступными.

Удобство и адаптация. Интерфейс протестирован на наиболее типичных пользовательских сценариях. Использованы крупные шрифты, чёткие подписи и контрастные элементы управления – это особенно важно для работы в условиях загруженного магазина. Все формы лаконичны и не перегружены лишними элементами, что ускоряет выполнение операций и снижает вероятность ошибок.

### 3.2 Реализация операций учета

В этом разделе рассматриваются внутренние механизмы работы приложения – та часть, которая отвечает за обработку данных и реализацию ключевых функций учёта: добавление, редактирование, удаление и загрузку товарных позиций. В отличие от предыдущего раздела, где акцент делался на визуальный интерфейс, здесь внимание сосредоточено на структуре кода и принципах, заложенных в архитектуру проекта.

Центральным элементом бизнес-логики выступает класс Product, представляющий собой модель товарной единицы. Он включает свойства, соответствующие полям в таблице базы данных (Id, ProductName, Price, Quantity, Description, Material), а также методы для работы с этими данными. На рисунке 15 представлен фрагмент кода класса Product.

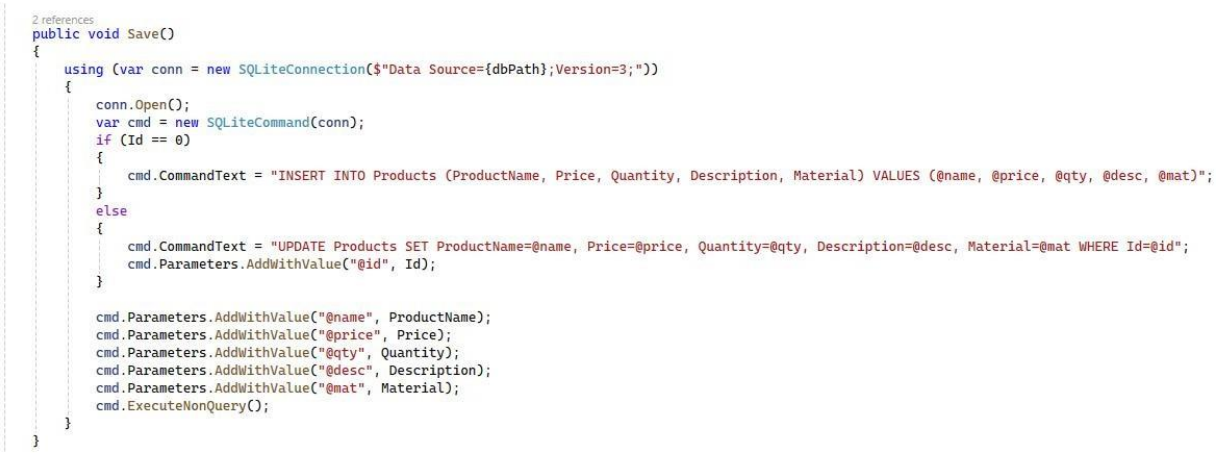
```
using System.Collections.Generic;
using System.Data.SQLite;
using System.IO;

namespace StoreInventoryApp
{
    12 references
    public class Product
    {
        6 references
        public int Id { get; set; }
        5 references
        public string ProductName { get; set; }
        4 references
        public decimal Price { get; set; }
        4 references
        public int Quantity { get; set; }
        4 references
        public string Description { get; set; }
        5 references
        public string Material { get; set; }

        private static string dbPath = "products.db";

        0 references
        static Product()
        {
            if (!File.Exists(dbPath))
            {
                SQLiteConnection.CreateFile(dbPath);
            }

            using (var conn = new SQLiteConnection($"Data Source={dbPath};Version=3;"))
            {
                conn.Open();
                var cmd = new SQLiteCommand(@"
                    CREATE TABLE IF NOT EXISTS Products (
                        Id INTEGER PRIMARY KEY AUTOINCREMENT,
                        ProductName TEXT NOT NULL,
                        Price REAL NOT NULL,
                        Quantity INTEGER NOT NULL,
                        Description TEXT,
                        Material TEXT NOT NULL
                    )", conn);
                cmd.ExecuteNonQuery();
            }
        }
```

Рисунок 15 – Класс Product с методами Save и Delete

Класс инкапсулирует логику сохранения и удаления записей. Метод Save() выполняет вставку или обновление в зависимости от того, задан ли идентификатор записи. Метод Delete() удаляет товар по его Id. Такой подход делает код повторно используемым и облегчает поддержку – особенно при дальнейшем расширении функциональности.

Метод Save() отвечает за добавление новой записи или обновление существующей. Он формирует SQL-запрос с параметрами и выполняет его через объект SQLiteCommand, используя установленное соединение с базой

данных. На рисунке 16 представлен фрагмент кода метода Save.



```
2 references
public void Save()
{
    using (var conn = new SQLiteConnection($"Data Source={dbPath};Version=3;"))
    {
        conn.Open();
        var cmd = new SQLiteCommand(conn);
        if (Id == 0)
        {
            cmd.CommandText = "INSERT INTO Products (ProductName, Price, Quantity, Description, Material) VALUES (@name, @price, @qty, @desc, @mat)";
        }
        else
        {
            cmd.CommandText = "UPDATE Products SET ProductName=@name, Price=@price, Quantity=@qty, Description=@desc, Material=@mat WHERE Id=@id";
            cmd.Parameters.AddWithValue("@id", Id);
        }

        cmd.Parameters.AddWithValue("@name", ProductName);
        cmd.Parameters.AddWithValue("@price", Price);
        cmd.Parameters.AddWithValue("@qty", Quantity);
        cmd.Parameters.AddWithValue("@desc", Description);
        cmd.Parameters.AddWithValue("@mat", Material);
        cmd.ExecuteNonQuery();
    }
}
```

Рисунок 16 – Метод Save с параметризированным SQL-запросом

Проверка вводимых данных осуществляется ещё на уровне пользовательского интерфейса, поэтому в метод попадают уже проверенные значения. Все параметры передаются через AddWithValue, что исключает возможность SQL-инъекций и упрощает структуру кода.

Метод Delete(int id) реализует удаление товарной позиции по идентификатору. Он формирует стандартный SQL-запрос DELETE и выполняет его через подключение к базе. На рисунке 17 изображен фрагмент кода, метода Delete.

```
        1 reference
        public static void Delete(int id)
        {
            using (var conn = new SQLiteConnection($"Data Source={dbPath};Version=3;"))
            {
                conn.Open();
                var cmd = new SQLiteCommand("DELETE FROM Products WHERE Id=@id", conn);
                cmd.Parameters.AddWithValue("@id", id);
                cmd.ExecuteNonQuery();
            }
        }
    }
}
```

Рисунок 17 – Метод Delete

Метод объявлен как static, что позволяет обращаться к нему напрямую, без создания экземпляра класса – это удобно при обработке событий в интерфейсе, где требуется быстрое удаление выбранного элемента.

Метод LoadProducts() используется для получения всех товарных записей и их отображения в главной таблице. Он выполняет SQL-запрос на чтение, преобразует полученные данные в список объектов Product, после чего этот список устанавливается как источник данных для DataGridView. На рисунке 18 представлен фрагмент кода метода LoadProducts.


Рисунок 18 – Метод LoadProducts

Дополнительно в этом методе происходит переименование заголовков столбцов, чтобы в интерфейсе отображались понятные пользователю русские названия.

Весь код, связанный с обработкой данных, вынесен за пределы форм – пользовательский интерфейс только вызывает нужные методы класса Product. Такой подход соответствует принципам архитектуры MVC (Model–View– Controller), где модель (в данном случае Product) несёт ответственность за данные и логику, а форма выполняет роль посредника между пользователем и бизнес-слоем.

Работа с базой организована исключительно через параметризованные SQL-запросы, что повышает безопасность и защищает от ошибок, связанных с вводом данных. Структура кода логична и расширяема: при необходимости можно легко добавить новые методы или изменить поведение существующих, не затрагивая остальную часть системы.

Таким образом, внутренняя логика приложения построена на чёткой модели данных и отделена от визуальной части. Это обеспечивает надёжную работу, простоту сопровождения и возможность масштабирования проекта без нарушения уже реализованной функциональности.

### 3.3 Тестирование и проверка работоспособности

Для оценки стабильности и корректности работы приложения было проведено ручное тестирование. Проверялись основные действия, которые выполняет пользователь при повседневной работе: добавление, редактирование, удаление товаров, фильтрация, переключение темы [4]. Цель тестов – убедиться, что система реагирует на действия предсказуемо и без сбоев. Добавление товара проверялось, сохраняется ли новая позиция.

Данные, которые вводились в ходе теста добавления товара:

– Бетон;

– 670.00;

– 423;

– Бетонная смесь М600, 10 кг. цена за шт.

После нажатия кнопки «Сохранить» товар появился в таблице. Всё отработало корректно: данные ушли в базу, интерфейс обновился. На рисунке 19 изображена форма добавления нового товара.

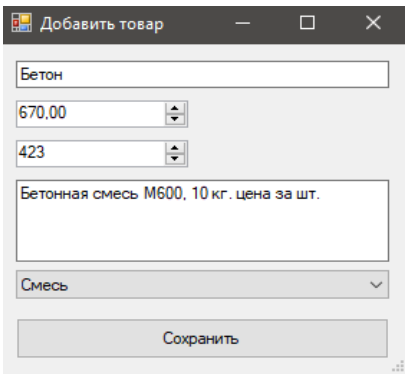



Рисунок 19 – Добавление нового товара

На рисунке 20 представлен добавленный товар в список.

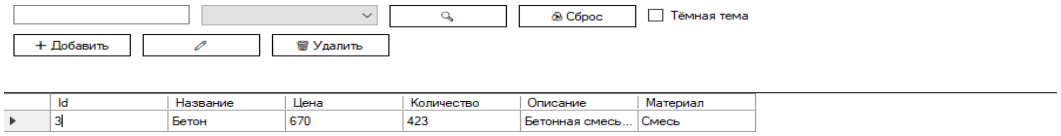



Рисунок 20 – Добавленный товар в список товаров

Для теста редактирование товара из списка выбран ранее добавленный товар. В открывшейся форме изменены поля «Цена» на 150 и «Количество» на 100. После сохранения изменения сразу отобразились в таблице. Метод обновления сработал как положено. На рисунке 21 изображена форма редактирования товара.

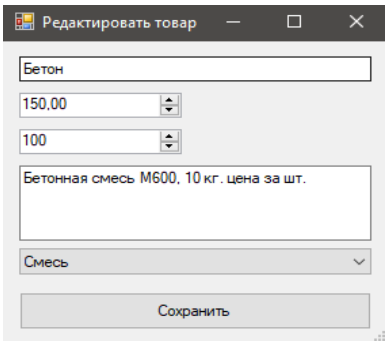



Рисунок 21 – Изменение существующей записи

На рисунке 22 изображен список товаров с редактированным товаром.

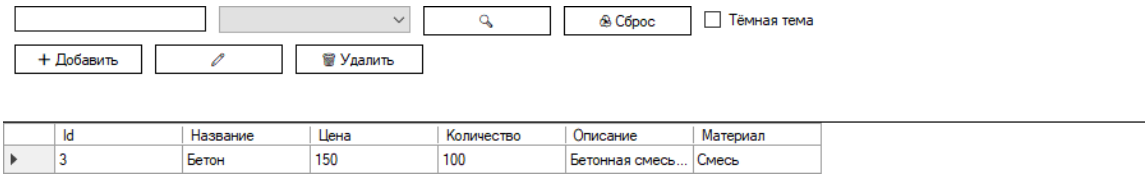



Рисунок 22 – Редактированный товар

Для тестирования удаление товара после выделения строки и нажатия кнопки «Удалить» запись исчезла из таблицы. После вызова метода Delete данные в таблице обновились. На рисунке 23 изображен список товаров с выделенным товаром для удаления.

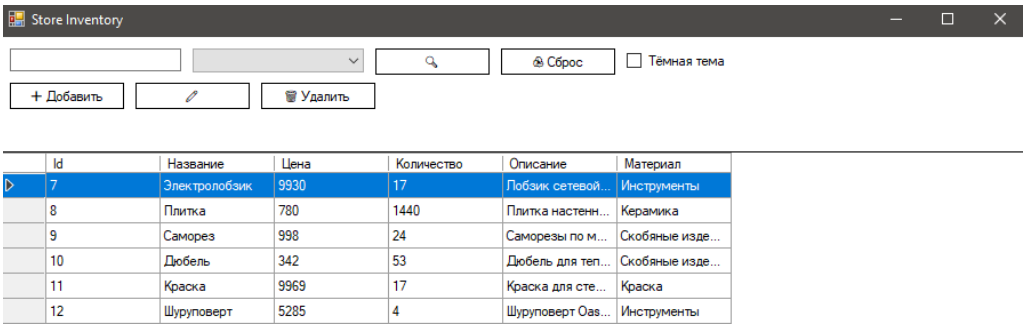



Рисунок 23 – Удаление записи из списка

На рисунке 24 изображена обновленная таблица без удаленной записи

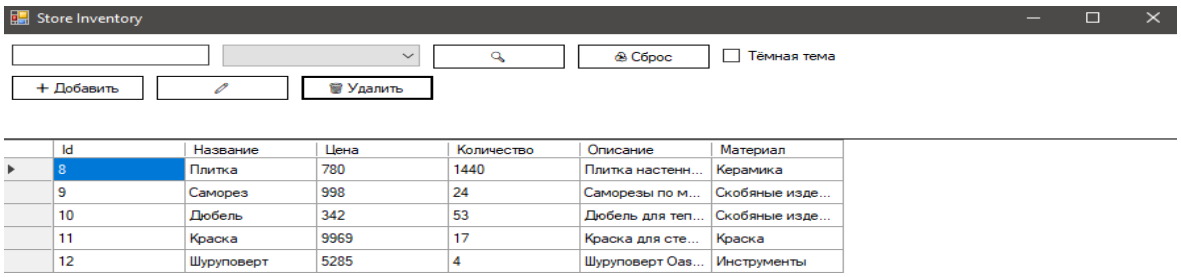



Рисунок 24 – Обновленный список товаров после удаления

Выполнена проверка фильтра по значению «Скобяные изделия». В таблице остались только те товары, у которых в поле «Материал» указано соответствующее значение. Интерфейс не завис, фильтрация прошла моментально. На рисунке 25 представлен итог фильтрации приложения.

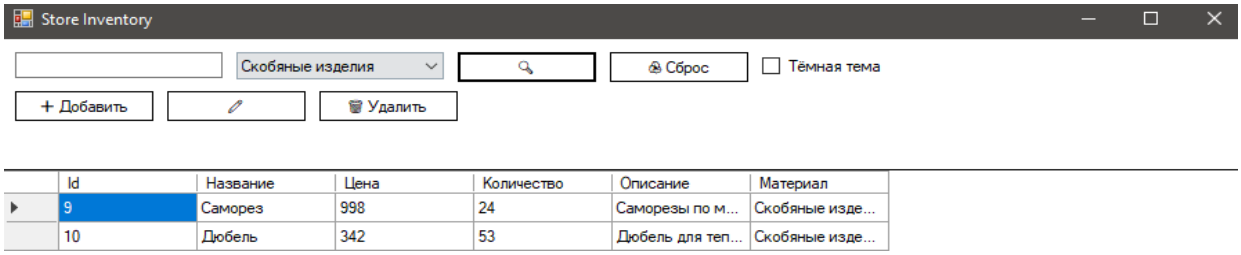



Рисунок 25 – Результат фильтрации

Включение тёмной темы изменило оформление всех форм. Переключение происходило мгновенно, без перезагрузки. Цветовая схема стала тёмной, шрифт остался читаемым. На рисунке 26 представлена темная тема программного обеспечения.

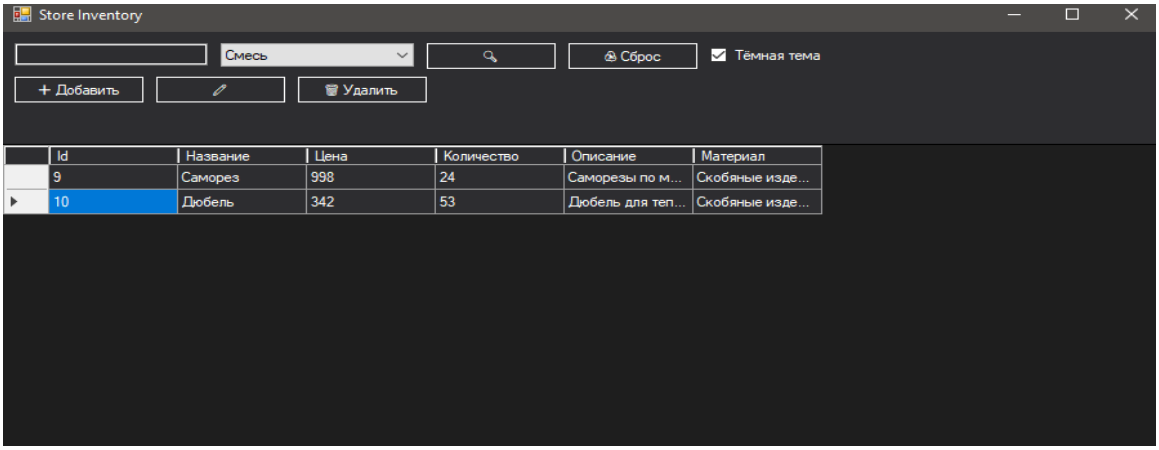



Рисунок 26 – Интерфейс в тёмной теме

Для систематизации результатов проведённого тестирования в таблице 6 приведены основные пользовательские сценарии, проверенные в процессе ручной проверки. Каждый тестовый случай включает описание входных

данных и ожидаемого результата, на основе которого оценивалась корректность работы соответствующих функций системы.

Таблица 6 – Результаты функций системы учета

| Тестовый сценарий | Входные данные | Ожидаемый результат |
|---|---|---|
| Добавление нового товара | Заполненные поля: наименование, цена, количество, описание материал | Товар добавлен и отображён в таблице |
| Редактирование товара | Изменены цена, количество, название, описание | Данные обновлены в таблице |
| Удаление товара | Выделен товар, нажата кнопка «Удалить» | Товар удалён из базы и таблицы |
| Фильтрация по материалу и названию. | В строке поиска написано название материала «Саморез» и выбран материал «Скобяные изделия» | Отображается только товар с выбранными характеристиками |
| Фильтрация по материалу | Выбран материал «Скобяные изделия» | Отображаются только товары с этим материалом |
| Переключение темы | Флажок тёмной темы включён | Интерфейс сменил цветовую схему |

Все проверенные функции работают стабильно. Ошибки обрабатываются корректно, действия пользователя выполняются без задержек, а интерфейс остаётся понятным. Программа не требует подключения к интернету, проста в использовании и полностью соответствует требованиям, предъявляемым к системе учёта для строительного магазина.

Выводы по главе 3

В третьей главе описан процесс реализации прикладного программного обеспечения, разработанного для учёта товаров в строительном магазине. Здесь основное внимание было сосредоточено на том, как устроены формы интерфейса, какие действия может выполнять пользователь, и каким образом реализованы ключевые функции внутри приложения.

В первую очередь рассмотрены основные операции: добавление новых

товаров, редактирование уже существующих записей, удаление, а также поиск и фильтрация по заданным параметрам. Эти действия лежат в основе ежедневного взаимодействия с системой. В коде они реализованы через события, привязанные к элементам интерфейса. Логика при этом отделена от визуальной части, что сделано для упрощения поддержки и дальнейшего расширения программы.

Связь с базой данных построена через SQL-запросы, вызываемые в нужных местах кода. Такой подход позволил обойтись без сторонних библиотек и держать структуру проекта как можно проще. Были реализованы механизмы для корректного сохранения, обновления и удаления данных. Тесты показали, что программа стабильно работает даже при повторных или ошибочных действиях пользователя.

Дополнительно в систему была добавлена поддержка тёмной темы – она переключается в зависимости от параметров, передаваемых при создании формы [10]. Также реализована базовая валидация полей: программа реагирует на пустые значения и не даёт сохранить запись с некорректными данными.

В процессе отладки было выявлено, что система устойчиво работает в стандартных сценариях и в условиях ошибочного ввода [2]. Программа быстро реагирует на действия, интерфейс не перегружен, а структура кода остаётся читаемой. Всё это говорит о том, что решение пригодно для использования на практике, и может быть внедрено в небольшой магазин без необходимости в дорогостоящем обслуживании или доработке.

Кроме того, реализация проекта показала, что даже при ограниченных ресурсах возможно создание устойчивого, функционального и удобного программного продукта. Применённые технологии обеспечили простую архитектуру, понятную логику и комфортную работу как для конечного пользователя, так и для будущих разработчиков. Полученные результаты подтверждают целесообразность индивидуальной разработки ПО, ориентированной на задачи малого бизнеса.

## Заключение

В условиях активного развития цифровых технологий и растущей конкуренции в розничной торговле автоматизация процессов учёта приобретает особую значимость [5]. Особенно актуально это для малых строительных магазинов, где ассортимент обширен, а ресурсы на внедрение сложных информационных систем ограничены. Разработка программного обеспечения, позволяет устранить типичные проблемы, связанные с ручным или неструктурированным учётом товаров.

В ходе выполнения выпускной квалификационной работы была последовательно реализована задача создания настольного приложения для учёта товарных позиций в строительном магазине. Проведён всесторонний анализ предметной области, исследованы существующие решения и обоснована необходимость разработки собственной системы, адаптированной под особенности небольших торговых точек. Были сформированы требования к функциональности и качеству, спроектирована архитектура системы и структура базы данных, а также реализован программный код с применением современных технологий: C#, Windows Forms, SQLite и Entity Framework [1].

Особое внимание было уделено удобству пользовательского интерфейса. Программа демонстрирует устойчивую работу, включает поддержку базовой валидации, фильтрации по характеристикам, а также поддержку тёмной темы оформления. Проведённое тестирование подтвердило корректность выполнения всех основных функций.

Разработанный программный продукт соответствует поставленным целям и требованиям и может быть рекомендован к использованию в условиях реальной торговой деятельности [15]. Он сочетает в себе доступность, простоту, надёжность и возможность адаптации под конкретные нужды магазина без необходимости в глубокой технической подготовке со стороны персонала.

## Список используемой литературы и используемых источников


1. Албахари Дж., Албахари Б. C# 9.0. Карманный справочник. – М.: Вильямс, 2021. – 480 с.
2. Андреев С.А. Практическое руководство по тестированию программного обеспечения. – СПб.: Питер, 2022. – 304 с.
3. Брин Д. Практика использования SQLite в .NET-приложениях. – М.: ДМК Пресс, 2020. – 240 с.
4. Веденяпин В.Г. Разработка баз данных в приложениях на C#. – М.: ДМК Пресс, 2022. – 296 с.
5. Гладкий Ю.В. Информационные технологии для автоматизации торговых предприятий. – М.: РГУТИС, 2020. – 328 с.
6. Глушаков В.А. Разработка информационных систем: учебник. – М.: Академия, 2021. – 352 с.
7. ГОСТ 19.101–2024. Единая система программной документации. Виды программ и программных документов. – М.: Стандартинформ, 2024. – 12 с.
8. ГОСТ 19.301–79. ЕСПД. Программа и методика испытаний. – М.: Изд-во стандартов, 1979. – 16 с.
9. ГОСТ 34.003–90. Информационная технология. Автоматизированные системы. Термины и определения. – М.: Изд-во стандартов, 1990. – 28 с.
10. Жуков В.А. Проектирование интерфейсов пользователя. – СПб.: Питер, 2022. – 288 с.
11. Каляев А.А. Разработка прикладных программ: Windows Forms. – М.: КноРус, 2021. – 304 с.
12. Куценко А.Н. Основы программной инженерии. – СПб.: БХВ-Петербург, 2021. – 416 с.
13. Лаптев В.Ю. Информационные системы управления. – М.: Финансы и статистика, 2020. – 272 с.

14. Левин В.А. Проектирование пользовательских интерфейсов: теория и практика. – М.: НИЦ Инфра-М, 2021. – 288 с.

15. Молчанов И.Ю. Разработка программного обеспечения: учебное пособие. – М.: Феникс, 2021. – 288 с.

16. Панкратова Н.Е. Проектирование программного обеспечения. – М.: Форум, 2020. – 320 с.

17. Трофимов В.А. Основы информационных систем. – М.: КноРус, 2019. – 360 с.

18. Шатунов В.П. Проектирование автоматизированных систем. – М.: Высшая школа, 2020. – 304 с.

19. Юрьев А.А. Разработка приложений для бизнеса. – СПб.: Питер, 2020. – 352 с.

20. Cooper A., Reimann R., Cronin D. About Face: The Essentials of Interaction Design. – 4th ed. – Indianapolis: Wiley, 2014. – 720 p.

21. Fowler M. Patterns of Enterprise Application Architecture. – Boston: Addison-Wesley, 2003. – 560 p.

22. ISO/IEC 25010:2011. Software Product Quality Requirements and Evaluation (SQuaRE). – Geneva: ISO, 2011. – 40 p.

23. MacDonald M. Pro .NET Windows Forëms. – Berkeley: Apress, 2005. – 1056 p.

24. Owens M. The Definitive Guide to SQLite. – 2nd ed. – Berkeley: Apress, 2010. – 368 p.

25. Ricardo J. Beginning C# and .NET: From Novice to Professional. – 2nd ed. – New York: Apress, 2021. – 780 p.