

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ СИСТЕМЫ УЧЁТА...	8
1.1 Этапы разработки системы учёта .....	8
1.2 Участники разработки системы учёта .....	11
1.2 Анализ приложений на рынке программного обеспечения.....	13
1.3 Компоненты системы учёта .....	18
1.4 Выбор языка программирования .....	20
1.5 Анализ и выбор среды разработки.....	22
ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ УЧЁТА .....	26
2.1 Разработка дизайна системы учёта.....	26
2.2 Разработка функциональной части системы учёта .....	43
ГЛАВА 3. ОЦЕНКА ЭКОНОМИЧЕСКОЙ ЦЕЛЕСООБРАЗНОСТИ РАЗРАБОТКИ СИСТЕМЫ УЧЁТА.....	64
3.1 Цель и задачи экономического раздела.....	64
3.2 Экономический эффект и эффективность .....	69
ЗАКЛЮЧЕНИЕ .....	70
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ИНФОРМАЦИИ.....	72

## ВВЕДЕНИЕ

Современные условия функционирования торговых предприятий требуют от организаций не только высокой скорости обслуживания клиентов, но и эффективного управления внутренними бизнес-процессами. Особое значение приобретает автоматизация процессов на предприятиях с большой номенклатурой товаров и значительным оборотом продукции — к числу которых относятся строительные гипермаркеты.

Подобные организации ежедневно обрабатывают множество заказов, осуществляют внутреннюю логистику, контролируют перемещение материалов и координируют действия различных подразделений. Ручное ведение учёта и управление процессами в таких условиях становится не только затруднительным, но и неэффективным, что может привести к потерям, ошибкам и снижению общей производительности компании.

В условиях высокой конкуренции на рынке строительных материалов, одним из ключевых факторов успеха становится цифровизация — внедрение современных информационных систем, которые позволяют оптимизировать процессы управления, повысить прозрачность операций, сократить временные затраты на выполнение рутинных задач и обеспечить контроль за движением материальных ресурсов в реальном времени. Автоматизированные системы учета позволяют не только вести достоверный контроль остатков и перемещений, но и предоставляют руководству необходимые аналитические данные для принятия управленческих решений.

Актуальность темы выпускной квалификационной работы определяется растущими требованиями к цифровизации бизнес-процессов в строительной отрасли и необходимостью повышения прозрачности логистики и эффективности работы персонала. Предлагаемая система может быть адаптирована для любых крупных торговых предприятий и имеет высокую практическую значимость.

Объектом настоящей выпускной квалификационной работы является процесс учёта и управления логистикой в условиях строительного гипермаркета. Предметом исследования выступает автоматизированная система, предназначенная для повышения эффективности и прозрачности внутренней логистики, упрощения учёта заказов и оптимизации работы сотрудников.

Целью выпускной квалификационной работы является разработка и внедрение автоматизированной информационной системы, обеспечивающей удобный и надёжный учёт заказов, материалов и логистических операций на базе программной платформы WPF с использованием языка программирования C# и реляционной базы данных Microsoft SQL Server.

Для достижения поставленной цели в рамках ВКР необходимо решить следующие задачи:

1. Провести анализ предметной области и существующих аналогичных решений;
2. Определить функциональные и технические требования к системе;
3. Разработать архитектуру программного обеспечения и спроектировать структуру базы данных;
4. Разработать пользовательский интерфейс в соответствии с принципами UX-дизайна;
5. Реализовать основные модули приложения: учёт заказов, управление материалами, контроль исполнения;
6. Осуществить тестирование работоспособности системы и устранение выявленных ошибок;
7. Оценить экономическую эффективность внедрения программного продукта на предприятии;
8. Подготовить систему к внедрению и эксплуатации в реальных условиях.

Структура выпускной квалификационной работы включает три

основные главы.

В первой главе рассмотрены теоретические основы автоматизации процессов и обзор технологий, используемых при разработке систем учёта.

Вторая глава посвящена проектированию и реализации системы учёта для строительного гипермаркета.

В третьей главе производится экономическое обоснование внедрения программного продукта, включая расчёт затрат и ожидаемый эффект.

# ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ СИСТЕМЫ УЧЁТА

## 1.1 Этапы разработки системы учёта

Разработка современной системы учёта представляет собой многоуровневый, итеративный процесс, включающий в себя аналитическую, проектную, техническую и организационную составляющие. Каждый из этапов данного процесса имеет стратегически важное значение и напрямую влияет на итоговое качество создаваемого программного продукта, его функциональность, удобство использования, надёжность и перспективы масштабирования в будущем. Особенно актуальным является поэтапный подход при создании информационных систем для крупных предприятий, таких как строительные гипермаркеты, деятельность которых охватывает широкий спектр операций, включая учёт материалов, логистику, работу с заказами и внутренний документооборот.

Разработка системы, как правило, начинается с анализа предметной области. Данный этап включает в себя глубокое исследование внутренних бизнес-процессов компании, выявление проблемных зон, определение задач, которые необходимо автоматизировать.

На этом этапе проводится сбор информации о текущем состоянии системы управления предприятием, изучаются рабочие процессы, используется интервьюирование сотрудников, анализируются документы, отчёты, нормативные акты и регламенты.

Цель этапа – понять, какие функции наиболее востребованы, какие узкие места мешают эффективной работе, и какие показатели необходимо улучшить.

Следующим этапом является формирование технического задания – одного из ключевых документов, на базе которого будет осуществляться проектирование и разработка. Техническое задание формализует требования заказчика, описывает структуру, интерфейсы, функции, модули, алгоритмы работы, требования к безопасности, масштабируемости и производительности. Оно согласуется с руководством предприятия, подписывается ответственными сторонами и служит правовой и организационной основой для разработки.

После утверждения технического задания начинается этап проектирования. Он включает в себя создание архитектуры программного обеспечения, проектирование логики функционирования отдельных модулей, построение схемы базы данных. Также в рамках проектирования прорабатывается взаимодействие между компонентами системы (frontend, backend, СУБД), создаются UML-диаграммы, схемы экранов и процессов. На данном этапе принимаются решения о выборе архитектурной модели (например, модель клиент-сервер), способе обмена данными (например, через SQL-запросы), а также определяются основные интерфейсные элементы будущего программного продукта.

Особое внимание уделяется выбору технологий и инструментов разработки. От этого напрямую зависит, насколько гибкой, надёжной и удобной в эксплуатации окажется система. В дипломной работе был выбран стек технологий, включающий WPF (Windows Presentation Foundation) и язык C#, а также реляционную СУБД Microsoft SQL Server. Такой выбор обусловлен необходимостью создания настольного приложения с богатым пользовательским интерфейсом, высокой производительностью и возможностью быстрой интеграции в локальную сеть предприятия.

Следующий этап – разработка пользовательского интерфейса. Интерфейс – это первая точка взаимодействия пользователя с программой. Именно от него зависит, будет ли система удобной, понятной, логичной и приятной в использовании. При проектировании интерфейса учитываются требования эргономики, стандарты UX/UI-дизайна, создаются прототипы, макеты и интерактивные модели. Интерфейс должен быть интуитивно понятен для сотрудников гипермаркета, в том числе для пользователей без специальной подготовки.

Затем наступает этап программной реализации, в рамках которого происходит написание исходного кода. Реализация ведётся поэтапно: разрабатываются отдельные модули, которые постепенно объединяются в единую систему. Программный код структурируется по архитектурным слоям – представление, логика и данные. На этом этапе активно применяются средства контроля версий, используются шаблоны проектирования (например, MVVM), подключаются библиотеки, обеспечивается связь между интерфейсом и базой данных. В случае использования WPF интерфейс создаётся в XAML, логика – на языке C#.

Немаловажным этапом является тестирование. Оно позволяет выявить и устранить ошибки, проверить соответствие системы требованиям ТЗ, убедиться в стабильности и корректности работы всех функций. Тестирование проводится как вручную, так и автоматизированными средствами. Проверяются различные сценарии работы системы, включая граничные значения, ошибочные вводы, корректность валидации, производительность при высокой нагрузке. По итогам тестирования формируются отчёты, проводится устранение дефектов.

Далее следует этап внедрения, в рамках которого система учёта интегрируется в производственную среду предприятия. Происходит установка системы на рабочие станции, настройка параметров, загрузка начальных данных, обучение пользователей, инструктаж администраторов. Также на этом этапе возможно проведение пилотного запуска в ограниченном сегменте, после которого осуществляется масштабирование на всё предприятие.

Завершающим этапом является техническое сопровождение и обслуживание. Оно включает мониторинг работы системы, устранение выявленных неполадок, обновление программного обеспечения, добавление нового функционала по мере необходимости, адаптацию к изменяющимся условиям работы компании. Поддержка может быть, как внутренней, так и осуществляться сторонними подрядчиками.

Таким образом, процесс разработки системы учёта включает в себя множество взаимосвязанных этапов, каждый из которых вносит важный вклад в создание полноценного программного продукта. Последовательное и качественное выполнение всех этапов обеспечивает не только соответствие системы требованиям, но и её надёжность, расширяемость, эффективность и удобство в повседневной эксплуатации.

## 1.2 Участники разработки системы учёта

Процесс создания системы учёта представляет собой скоординированную работу специалистов различных направлений, объединённых общей целью – разработкой эффективного, надёжного и удобного программного решения. Количество и специализация участников зависит от масштаба проекта, уровня автоматизации, финансовых ресурсов заказчика и технических требований к системе. В современных условиях всё чаще применяется командная модель разработки, основанная на распределении ролей и ответственности между участниками проекта.

Наиболее типичные роли участников разработки включают:



## 1. Аналитик

Бизнес-аналитик — это ключевая фигура на начальных этапах проекта. Он осуществляет сбор и структурирование требований, анализирует бизнес-процессы предприятия, выявляет проблемы и определяет цели разработки. Именно аналитик формирует основу технического задания и координирует процесс перехода от бизнес-целей к технической реализации. Его задача — выступить связующим звеном между заказчиком и командой разработчиков, перевести бизнес-язык в язык технических требований.

## 2. Архитектор/системный проектировщик

Этот специалист определяет архитектуру программного продукта: логическую структуру, распределение ответственности между модулями, схему взаимодействия между слоями, способ обмена данными и хранения информации. Архитектор закладывает фундамент системы, от которого зависит её стабильность, масштабируемость и поддерживаемость в будущем. Он также принимает участие в выборе технологического стека, моделей безопасности, резервного копирования и других инфраструктурных аспектов.

## 3. UI/UX-дизайнер

Дизайнер пользовательского интерфейса играет важнейшую роль в обеспечении удобства использования системы учёта. Он разрабатывает макеты экранов, цветовую палитру, иерархию элементов управления, структуру меню и форм. На основе проведённых исследований и опросов целевой аудитории дизайнер формирует интерфейс, который будет понятен и доступен даже для пользователей без специальной подготовки. Особое внимание уделяется навигации, логике переходов, читаемости текста и визуальному стилю.

## 4. Разработчики (frontend и backend)

Программисты являются основной производительной силой проекта.

Frontend-разработчик (в случае десктопных приложений — WPF-разработчик) реализует интерфейс, обрабатывает взаимодействие с пользователем, реализует визуальные элементы и события.

Backend-разработчик отвечает за серверную логику, обработку запросов, работу с базой данных, реализацию бизнес-процессов и соблюдение требований безопасности.

Оба направления тесно взаимодействуют друг с другом, обеспечивая целостную работу приложения.

#### 5. Тестировщик (QA-инженер)

Качество программного продукта в значительной степени зависит от работы тестировщика. Он разрабатывает тест-кейсы, проводит ручное и автоматизированное тестирование, отслеживает дефекты, проверяет функциональность, производительность, надёжность и безопасность системы. Именно тестировщик обеспечивает соответствие продукта требованиям технического задания и ожиданиям заказчика.

#### 6. Системный администратор / DevOps-инженер

Этот специалист занимается настройкой серверной части, развёртыванием среды, обеспечением работы баз данных, резервным копированием, организацией обновлений и мониторингом системы. В современных условиях его обязанности нередко включают автоматизацию процессов поставки (CI/CD), настройку виртуальных машин, Docker-контейнеров и других инструментов инфраструктуры.

#### 7. Менеджер проекта

Руководитель проекта (Project Manager) координирует деятельность всех участников, контролирует соблюдение сроков, бюджета и ресурсов.

Он планирует спринты, следит за прогрессом, управляет рисками, выстраивает коммуникации с заказчиком и внутри команды, решает конфликтные ситуации и отвечает за итоговый результат проекта.

### 1.2 Анализ приложений на рынке программного обеспечения

На современном этапе развития информационных технологий рынок программного обеспечения (ПО) демонстрирует стремительный рост и разнообразие решений, направленных на автоматизацию учётных и

управленческих процессов. Особенно актуальны такие решения для предприятий торговой и логистической направленности, где эффективность бизнес-деятельности напрямую зависит от точности, оперативности и прозрачности учёта. Строительные гипермаркеты, как представители крупного ритейла, являются яркими примерами таких предприятий, где использование автоматизированных систем учёта становится не просто рекомендованным, а жизненно необходимым элементом управления.

Современный рынок программного обеспечения предлагает широкий спектр готовых решений для учёта, логистики, складирования, управления персоналом и продажами. Однако выбор наиболее подходящего продукта требует всестороннего анализа: необходимо учитывать особенности бизнеса, специфику отрасли, возможности масштабирования, стоимость владения, уровень поддержки и возможность кастомизации системы под индивидуальные потребности.

Анализ программных решений можно условно разделить на несколько направлений: универсальные ERP-системы, специализированные отраслевые продукты, модульные платформы, облачные сервисы и разработки индивидуального характера. Ниже приведён подробный обзор каждой группы с акцентом на их применение в условиях, приближенных к задачам предприятия ООО «К1-Строй».

ERP (Enterprise Resource Planning) — это сложные интегрированные системы управления предприятием, охватывающие практически все бизнес-процессы: от снабжения и производства до логистики, кадрового учёта и финансового анализа. Наиболее известными представителями данного класса являются:

- SAP ERP;

- Oracle NetSuite;
- 1С:Предприятие 8;
- Microsoft Dynamics NAV (Business Central).

Эти решения зарекомендовали себя на крупных предприятиях, включая торговые сети, производственные объединения и логистические корпорации. Они предлагают высокий уровень надёжности, масштабируемости, наличия готовых модулей и поддержки сложных бизнес-сценариев.

Однако внедрение ERP-систем, как правило, связано с рядом ограничений и трудностей:

- высокая стоимость лицензий и сопровождения;
- сложность адаптации и необходимость привлечения внешних специалистов;
- избыточность функционала для средних и малых предприятий;
- длительный срок внедрения и обучения персонала.

Для предприятий уровня ООО «К1-Строй», где необходима фокусировка на конкретных задачах (учёт заказов, материалов, логистика), использование тяжеловесных ERP-систем может оказаться нецелесообразным с экономической и организационной точки зрения.

На рынке также представлены системы, ориентированные на конкретные сферы бизнеса: розничную торговлю, логистику, строительные и дистрибьюторские компании. К таким решениям относятся:

- МойСклад;
- Торговля+Склад;
- Управление торговлей от 1С;
- Склад365;
- АСУТП и логистика строительных объектов (в рамках ведомственных решений).

Данные системы обеспечивают функции, близкие к задачам

гипермаркета: учёт поступлений, контроль остатков, ведение заказов, генерация отчётности, работа с клиентской базой. Некоторые из них предлагают мобильные приложения, интеграцию с онлайн-кассами и банковскими терминалами.

Однако и здесь существует ряд ограничений:

- фиксированная структура данных, ограниченная гибкость при попытке изменить логику работы системы;
- зависимость от внешнего провайдера;
- ограниченный контроль над базой данных;
- регулярные обновления, которые могут нарушить индивидуально настроенные процессы.

Кроме того, многие подобные программы построены как SaaS-сервисы (software as a service), что требует постоянного подключения к интернету и абонентской платы, что не всегда приемлемо для локальных систем учёта.

В последнее десятилетие на рынке программных решений всё большую популярность приобретают облачные платформы, предоставляющие доступ к учётной системе через веб-браузер или мобильное приложение. Среди таких решений можно отметить:

- Bitrix24 (CRM и складской модуль);
- Zoho Inventory;
- Мегатлан;
- Odoo (модуль учёта и логистики).

Облачные системы обладают следующими достоинствами:

- доступ из любой точки мира;
- минимальные затраты на оборудование;
- регулярные обновления и резервное копирование данных;
- масштабируемость и интеграция с другим облачными сервисами.

Однако при всех плюсах такие решения имеют важные недостатки:

- невозможность полной автономной работы;
- зависимость от стабильности интернет-соединения;
- отсутствие контроля над физическим хранением данных;
- сложность интеграции с локальными бухгалтерскими и производственными программами.

Для строительного гипермаркета, где критичны оффлайн-доступ, надёжность хранения данных и необходимость работы в закрытой корпоративной сети, облачные сервисы могут быть не лучшим выбором.

На фоне рассмотренных решений всё чаще компании малого и среднего бизнеса делают выбор в пользу разработки индивидуального программного обеспечения, полностью адаптированного под их бизнес-логику, специфику документооборота и структуру взаимодействия между отделами. Такой подход позволяет:

- учесть уникальные процессы предприятия;
- выстроить интерфейс под конкретные задачи пользователей;
- самостоятельно контролировать базу данных и логику работы;
- обеспечить полную автономность и безопасность.

Разработка системы учёта «с нуля» требует наличия квалифицированных специалистов, но при этом позволяет достичь максимальной точности и гибкости. В рамках дипломного проекта был реализован именно такой подход: создана уникальная система учёта на базе WPF и Microsoft SQL Server, с проработанным интерфейсом, логикой учёта заказов, материалов и задач сотрудников, а также модулями аналитики и отчётности.

Рынок программного обеспечения предлагает широкую палитру решений для автоматизации учёта, однако ни одно из готовых приложений не обеспечивало полной адаптации под потребности ООО «К1-Строй». Универсальные ERP-системы оказались избыточными, облачные решения — слишком зависимыми от внешней инфраструктуры, а типовые отраслевые продукты — недостаточно гибкими.

По этой причине было принято обоснованное решение о разработке собственной системы учёта, позволяющей реализовать индивидуальные бизнес-процессы предприятия, исключить лишний функционал и повысить эффективность работы сотрудников. Такой подход обеспечил полную кастомизацию, высокую производительность и логическую завершённость проекта.

### 1.3 Компоненты системы учёта

Система учёта как программно-техническая система строится по модульному принципу и включает в себя ряд взаимосвязанных компонентов, каждый из которых выполняет строго определённые функции. Совокупность этих компонентов формирует архитектуру системы, обеспечивающую ее работоспособность, устойчивость и адаптивность.

В классическом понимании архитектура системы учёта включает три ключевых уровня:

- клиентский (интерфейс пользователя),
- логический (обработка данных и бизнес-логика),
- уровень хранения данных (база данных).

Такая структура соответствует трёхзвенной модели (three-tier architecture), обеспечивающей изоляцию слоёв, надёжность, возможность масштабирования и упрощение сопровождения.

Клиентский уровень представляет собой визуальную оболочку системы учёта — пользовательский интерфейс, с помощью которого осуществляется взаимодействие между человеком и системой. В случае разработки на платформе WPF (Windows Presentation Foundation) интерфейс создаётся на языке XAML и обрабатывается при помощи C#.

Клиентская часть отвечает за:

- отображение информации, полученной с сервера;
- приём и отправку пользовательских команд (нажатие кнопок, ввод данных и пр.);
- валидацию данных до их отправки на сервер;
- генерацию элементов визуализации (графики, таблицы, формы);
- соблюдение принципов доступности, эргономики и визуальной иерархии.

Преимущества WPF заключаются в богатых возможностях анимации, поддержке шаблонов, использовании шаблонизированных контролов и связях с данными (data binding), что существенно упрощает построение сложных интерфейсов.

Серверный уровень в рамках данного проекта реализован средствами языка C# и ADO.NET. Он отвечает за обработку запросов от клиентской части, выполнение бизнес-логики, проверку прав доступа и взаимодействие с базой данных.

Функции backend:

- аутентификация и авторизация пользователей;
- выполнение CRUD-операций (создание, чтение, обновление, удаление);
- обработка сложных пользовательских сценариев (например, фильтрация, сортировка, отчётность);
- реализация ролевой модели безопасности;



- логирование действий и событий в системе;
- контроль целостности данных.

Архитектура backend может включать слои логики, репозитории и интерфейсы доступа, что соответствует паттерну «чистой архитектуры» и улучшает модульность кода.

Третий уровень — это подсистема хранения данных, реализованная на базе Microsoft SQL Server. Здесь хранятся все сущности предметной области: сотрудники, материалы, заказы, статусы, склады, маршруты, отчёты и т. д.

База данных спроектирована в соответствии с принципами нормализации, обеспечивающими:

- отсутствие избыточности информации;
- логические связи между таблицами (через первичные и внешние ключи);
- оптимизацию индексов и быстродействие.

Также важно, что база данных защищена от прямого внешнего доступа: доступ осуществляется только через контролируемый слой backend. Это гарантирует безопасность и централизованное управление данными.

#### 1.4 Выбор языка программирования

Выбор языка программирования — это важнейшее архитектурное и стратегическое решение при разработке программного обеспечения. Он определяет не только синтаксис и технические возможности реализации проекта, но и уровень поддержки, гибкость, надёжность, производительность и совместимость с другими компонентами. При создании программного обеспечения для предприятий, особенно в области автоматизации логистики и учёта, выбор языка напрямую влияет на устойчивость и масштабируемость создаваемой системы.

В рамках данной выпускной квалификационной работы в качестве основного языка разработки выбран C# (си шарп) — современный, объектно-ориентированный язык программирования, разработанный корпорацией Microsoft и активно применяемый в построении Windows-приложений различного уровня сложности.

Причины выбора C#:

#### 1. Интеграция с Windows-платформой

Так как программный продукт предназначен для запуска на настольных компьютерах под управлением Windows, использование C# является логичным решением. Он обладает полной совместимостью с Windows API, средствами WPF и .NET Framework, что обеспечивает высокую производительность и доступ ко всем необходимым системным функциям.

#### 2. Совместимость с WPF

C# идеально сочетается с технологией WPF (Windows Presentation Foundation), которая используется в проекте для создания графического пользовательского интерфейса. Язык поддерживает двустороннюю привязку данных (data binding), событийно-ориентированную модель, стили, шаблоны и анимации.

#### 3. Объектно-ориентированная парадигма

C# предоставляет широкие возможности объектно-ориентированного моделирования: наследование, полиморфизм, инкапсуляция. Это упрощает реализацию сложных логических схем и модульность кода, что особенно важно при реализации крупных систем, включающих множество взаимосвязанных сущностей.

#### 4. Поддержка LINQ и ADO.NET

Для работы с базой данных в проекте применяется технология ADO.NET, а также LINQ — мощный язык запросов, встроенный в синтаксис C#. Это позволяет эффективно взаимодействовать с реляционными данными, выполнять фильтрацию, сортировку, агрегацию и группировку информации непосредственно в коде приложения.

## 5. Безопасность и обработка исключений

C# обладает развитой системой обработки ошибок и исключений, встроенными средствами безопасности, включая строгую типизацию, сборку мусора и защиту от переполнения буфера. Эти особенности делают его предпочтительным выбором для разработки надёжного ПО в условиях корпоративной среды.

## 6. Поддержка инструментов разработки

Среда разработки Microsoft Visual Studio предоставляет мощные инструменты для написания, отладки, профилирования и тестирования C#-приложений. Наличие автодополнения, встроенной поддержки Git, дизайнеров интерфейса и средств анализа кода значительно ускоряет процесс разработки.

В качестве альтернатив могли рассматриваться такие языки, как Java или Python. Однако они либо менее интегрированы с Windows-интерфейсами (в случае Python), либо требуют дополнительных библиотек и настроек для создания настольных приложений (в случае Java). Поэтому для реализации данной системы C# является оптимальным и технологически обоснованным решением.

### 1.5 Анализ и выбор среды разработки

Среда разработки играет ключевую роль в реализации программного обеспечения, обеспечивая программисту все необходимые инструменты для написания, отладки, компиляции, тестирования и внедрения приложения. Выбор подходящей интегрированной среды разработки (IDE) оказывает непосредственное влияние на эффективность разработки, скорость реализации проекта, удобство сопровождения и расширяемость системы. При проектировании системы учёта для автоматизации процессов в ООО «К1-Строй» в качестве основной среды разработки была выбрана Microsoft Visual Studio.

Данный выбор обусловлен спецификой проекта: приложение разрабатывается с использованием языка программирования C# и технологии WPF, что предполагает глубокую интеграцию с .NET Framework и широкие возможности, предоставляемые экосистемой Microsoft. Ни одна другая среда разработки не обладает столь мощной и стабильной поддержкой всех требуемых компонентов для реализации корпоративных десктопных приложений под Windows.

Microsoft Visual Studio — это ведущая среда разработки от компании Microsoft, предоставляющая полный набор инструментов для разработки, отладки и развертывания приложений на платформах .NET, .NET Core и .NET MAUI. В контексте рассматриваемого проекта были особенно актуальны следующие её преимущества:

#### 1. Интеграция с .NET и WPF

Visual Studio предлагает полноценную поддержку технологии WPF, включая:

- визуальные редакторы XAML;
- удобную систему навигации по связям между представлением и логикой;
- инструменты отладки интерфейса и событий;
- автогенерацию шаблонов и модулей (MVVM, ResourceDictionary и др.).

Эта интеграция обеспечивает удобство построения пользовательского интерфейса, тесно связанного с бизнес-логикой приложения.

#### 2. Поддержка ADO.NET и работы с базами данных

В проекте активно используется связка C# и Microsoft SQL Server. Visual Studio позволяет:

- подключать базы данных прямо из IDE;
- выполнять SQL-запросы, создавать и модифицировать таблицы, процедуры и представления;

- визуализировать структуру данных;
  - использовать Entity Framework и ADO.NET без необходимости сторонних плагинов.

Благодаря этому снижается время на настройку среды и облегчается взаимодействие с реляционной моделью данных.

### 3. Инструменты анализа и отладки

Visual Studio предоставляет обширный арсенал средств отладки, включая:

- точечные остановки (breakpoints), пошаговое выполнение;
- просмотр стеков вызовов, переменных, замеров производительности;
- отладку как на клиентской, так и на серверной стороне;
- интеграцию с системой логирования.

Это существенно упрощает диагностику ошибок и обеспечивает высокое качество кода на всех этапах реализации.

### 4. Встроенные системы контроля версий

В Visual Studio встроена поддержка систем контроля версий, таких как Git, Azure DevOps и TFS. Разработчик может:

- создавать репозитории;
- отслеживать изменения в коде;
- выполнять слияние веток;
- просматривать историю коммитов.

Это способствует командной работе, удобной организации кода и возможности отката изменений в случае ошибок.

### 5. Богатая экосистема расширений

Marketplace Visual Studio предлагает тысячи расширений, включая:

- генераторы кода (ReSharper, CodeMaid);
- проверку соответствия стиля;

- автоформатирование и диагностику;
- создание пользовательских шаблонов;
- интеграцию с Figma, Postman, Docker и другими инструментами.

Эти расширения позволяют адаптировать IDE под конкретные задачи проекта, повысив производительность работы разработчика.

Помимо Microsoft Visual Studio, существуют и другие среды, которые потенциально могли бы быть использованы для разработки WPF-приложений. Среди них можно выделить JetBrains Rider — кроссплатформенную среду, поддерживающую C# и .NET. Она отличается удобным интерфейсом и рядом полезных инструментов, но требует отдельной настройки для корректной работы с WPF и уступает Visual Studio в плане встроенной интеграции с Windows и SQL Server.

Другой вариант — Visual Studio Code, обладающий модульной архитектурой и лёгкостью. Однако этот редактор не предназначен для разработки полноценных настольных приложений на WPF: его функциональность сильно ограничена при работе с XAML и отсутствует визуальный редактор интерфейсов.

Также можно отметить SharpDevelop, который ранее использовался как лёгкая альтернатива Visual Studio. Однако в настоящее время данный проект признан устаревшим и не поддерживается, что делает его непригодным для современных задач.

В результате сравнительного анализа очевидно, что ни одна из альтернатив не может предложить такого уровня поддержки, стабильности и интеграции с экосистемой .NET, как Microsoft Visual Studio. Именно поэтому она была выбрана как основная среда разработки в рамках реализации данной системы учёта.

## ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ УЧЁТА

### 2.1 Разработка дизайна системы учёта

На этапе проектирования пользовательского интерфейса большое значение имело предварительное создание макета системы учёта. Для этих целей использовался онлайн-инструмент Figma, который позволил визуализировать структуру будущего программного продукта, проработать логику расположения элементов, а также провести базовую проверку удобства и понятности интерфейсов до начала реализации.

Важно отметить, что представленные в Figma макеты не являются окончательным вариантом интерфейса. Они использовались исключительно как черновые прототипы и впоследствии были доработаны, переработаны и частично заменены в процессе реальной разработки. Финальная реализация интерфейса в приложении, разработанном с использованием WPF и C#, значительно отличается как в плане визуального оформления, так и в логике поведения элементов, их динамике, а также адаптации под реальные роли пользователей и бизнес-сценарии.

Тем не менее, макеты в Figma сыграли важную роль на этапе концептуального моделирования интерфейса: они позволили сформировать общее представление о структуре системы, определить перечень необходимых страниц, навигационных блоков и элементов управления.

Ниже приводится подробное описание основных экранов системы учёта, с акцентом на их реальную реализацию в WPF-приложении, а не на визуальную составляющую макета.

#### Окно авторизации

Одним из первых и наиболее значимых экранов системы является окно авторизации. Это именно тот элемент, с которого начинается работа любого сотрудника, и который формирует первое впечатление от всей системы в целом. Основная задача данного интерфейса — обеспечить быстрый,

безопасный и понятный вход в систему учёта. Финальная реализация макета страницы авторизации выполнена в минималистичном, но строгом и деловом стиле, соответствующем корпоративному стилю компании ООО «К1-Строй». Цветовая гамма основана на светлых и нейтральных тонах с акцентами синего цвета, который используется для выделения активных элементов, кнопок и логотипа.

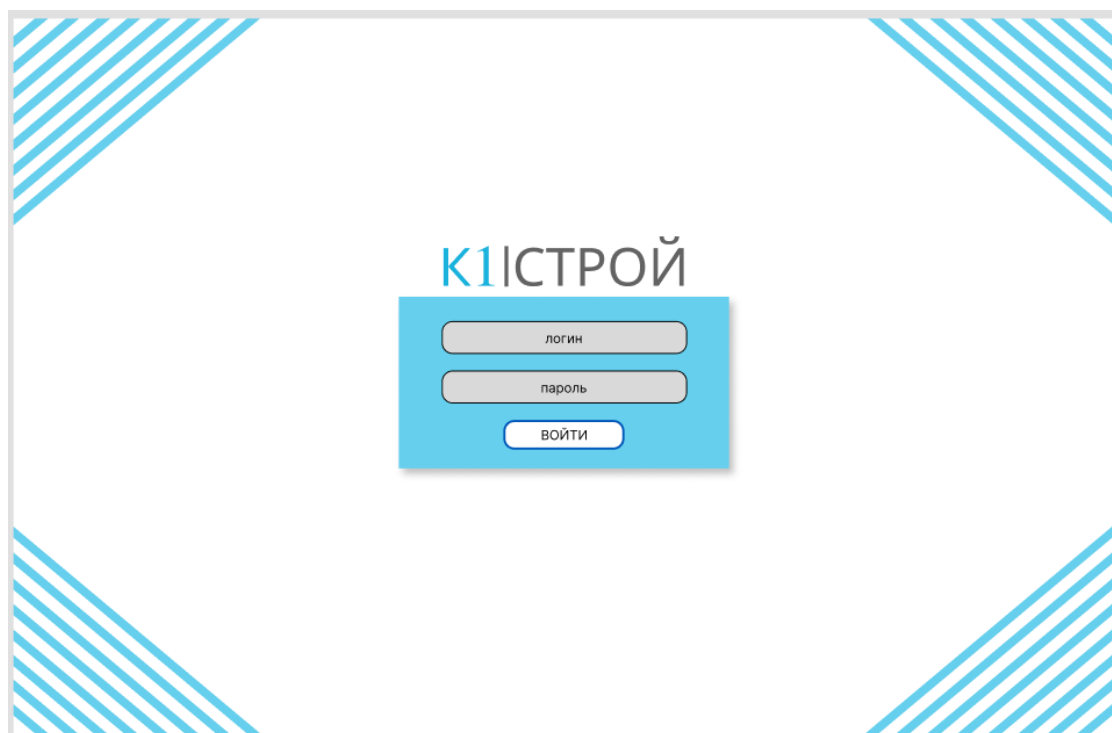


Рисунок 1 - Макет страницы авторизации

### Главная страницы

После прохождения этапа авторизации пользователь попадает на главную страницу системы учёта — так называемую панель управления. Это центральный интерфейс, откуда осуществляется доступ ко всем ключевым разделам системы (рисунок 1). В визуальном и структурном плане данное окно было спроектировано с ориентацией на предельную простоту, однозначность выбора и отсутствие визуальных перегрузок.



Интерфейс разбит на две основные зоны: боковая панель и основное рабочее пространство, что соответствует логике традиционных корпоративных приложений. Такое зонирование повышает читаемость, помогает пользователю быстро сориентироваться в интерфейсе и обеспечивает чёткое разделение навигации и содержательного взаимодействия.

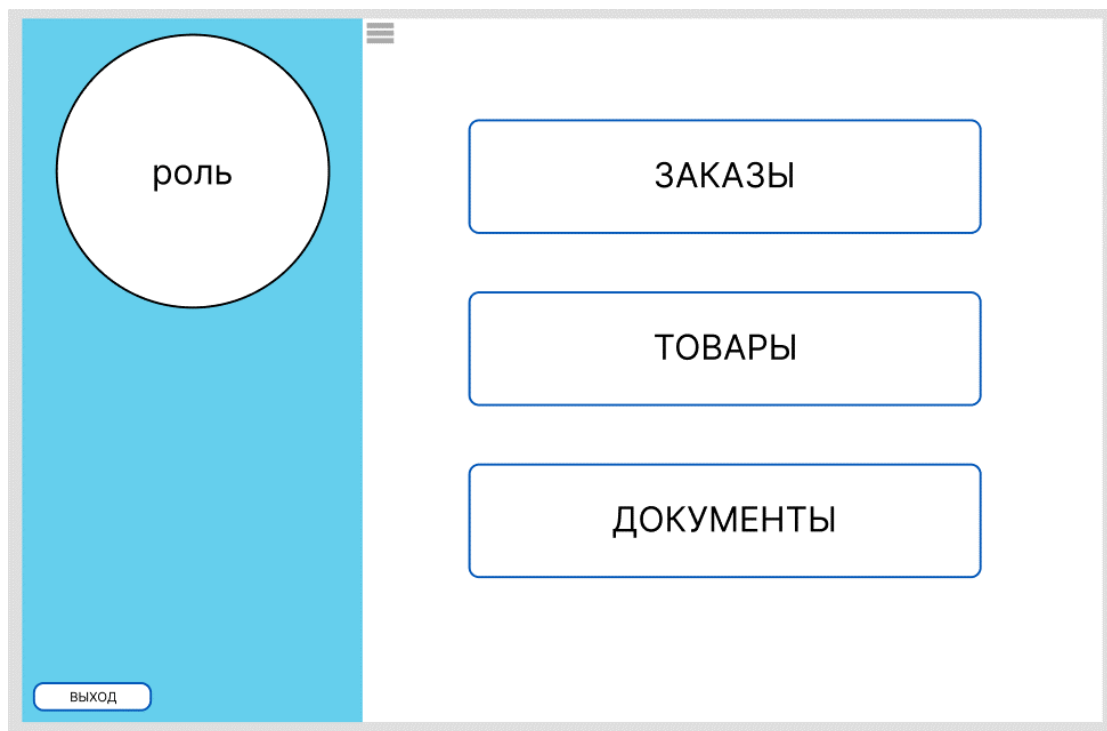


Рисунок 2 - Макет главной страницы

Слева размещён вертикальный блок голубого цвета, в котором выделено место для отображения текущей роли пользователя (рисунок 2). Элемент визуализирован в виде большого круга с надписью «роль» — в финальной реализации на этом месте должно было выводиться конкретное значение «Администратор», «Менеджер» или «Сотрудник». Такое решение не только информативно, но и способствует визуальной идентификации уровня доступа. В последствии было решено отказаться от данного решения в пользу возможности загрузки пользовательских изображений — аватаров.

Ниже размещена кнопка «Выход», выполненная в виде прямоугольной формы с закруглёнными углами. Её расположение в левом нижнем углу является традиционным и интуитивно ожидаемым для большинства пользователей, что упрощает процесс завершения сеанса работы с системой. Кнопка визуально выделена благодаря рамке и цвету, но не доминирует — её оформление подчёркивает вспомогательный характер действия.

Правую часть окна занимает основная навигационная зона, включающая три крупные кнопки:

- «ЗАКАЗЫ»
- «ТОВАРЫ»
- «ДОКУМЕНТЫ»

#### Страница заказов

Страница заказов представляет собой основной интерфейс для работы с заявками в системе учёта. Дизайн реализован в виде вертикального списка карточек, каждая из которых содержит ключевую информацию о заказе: наименование, производителя, описание, цену, статус и приоритет. Дополнительно отображаются дата добавления и срок выполнения, размещённые в правом верхнем углу карточки (рисунок 3).

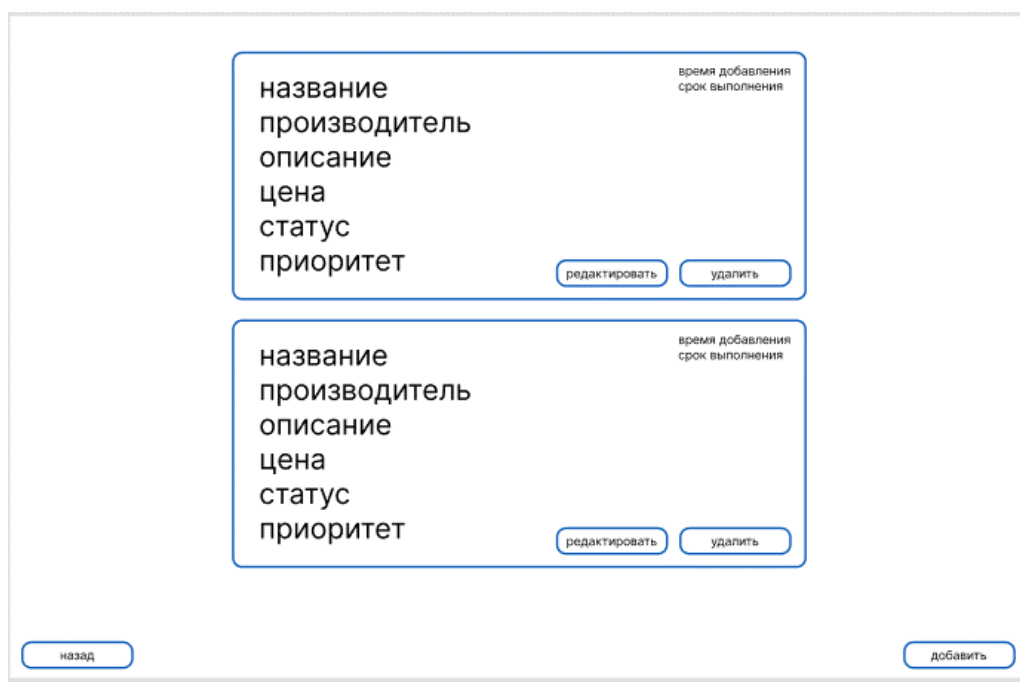


Рисунок 3 - Макет страницы заказов

Каждая карточка визуально отделена рамкой и имеет две основные кнопки: «Редактировать» и «Удалить», расположенные в нижней части. Кнопки оформлены лаконично, без лишних декоративных элементов, и находятся на одном уровне, что упрощает взаимодействие.

Внизу страницы расположены:

- кнопка «Назад» — возвращает в главное меню;
- кнопка «Добавить» — открывает форму создания нового заказа.

Интерфейс выполнен в светлой цветовой гамме с синими акцентами. Элементы размещены симметрично и равномерно, что повышает читаемость и удобство работы. Страница визуально не перегружена и ориентирована на быстрое выполнение типовых операций с заказами.

#### Страница товаров

Страница товаров используется для отображения и управления перечнем продукции, доступной в системе учёта (рисунок 4). Визуально каждый товар представлен в виде отдельной карточки, содержащей изображение (фото), наименование, производителя, описание и цену. Информация структурирована по вертикали, что упрощает восприятие.

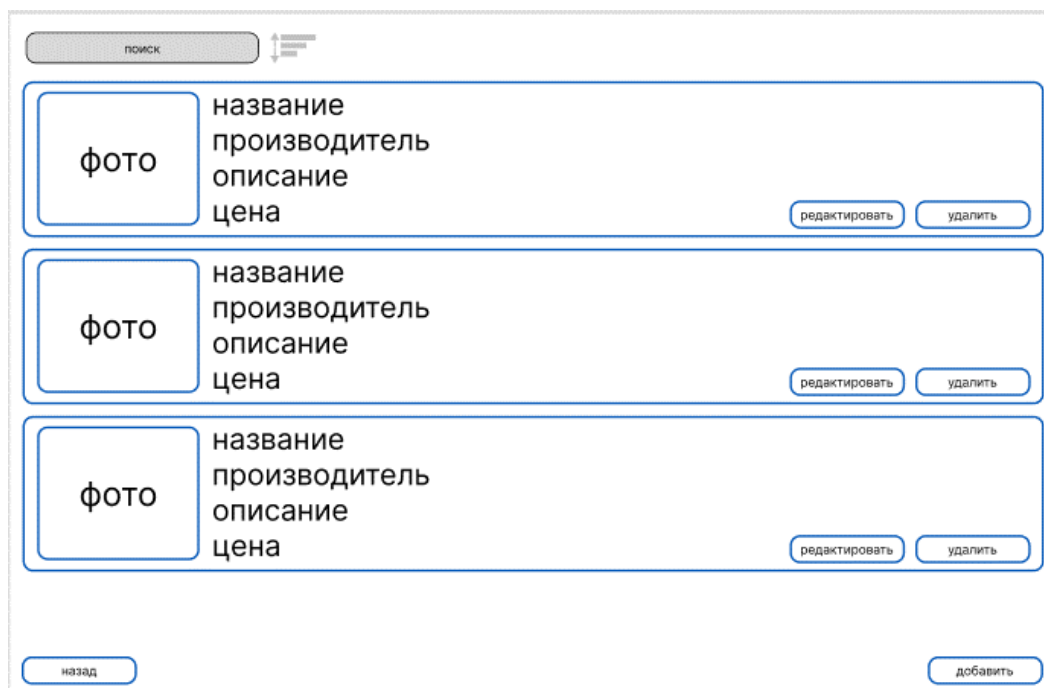


Рисунок 4 - Макет страницы товаров

Слева в каждой карточке размещён визуальный элемент — фотография товара. Это позволяет сотрудникам быстро идентифицировать позиции без необходимости чтения текста. Справа от изображения располагается текстовая информация, оформленная стандартным шрифтом с равномерными отступами ( рисунок 4)..

В нижней части каждой карточки размещены кнопки «Редактировать» и «Удалить», выполненные в едином стиле: рамка, минимальный размер, визуальная симметрия ( рисунок 4).

В верхней части страницы реализована строка поиска и кнопка фильтрации, что позволяет быстро находить необходимые товары по заданным критериям ( рисунок 4)..

Управление страницей осуществляется через две кнопки:

- «Назад» — возвращение к главному меню;
- «Добавить» — переход к форме создания нового товара.

Дизайн страницы повторяет общую стилистику системы: светлый фон, синие акценты, простая структура. Интерфейс направлен на оперативную работу с товарными позициями и не содержит лишних элементов.

Как уже упоминалось ранее, макеты, созданные в Figma, использовались исключительно в качестве прототипов и основы для формирования логики интерфейса. В процессе реализации системы на базе WPF были внесены многочисленные изменения, касающиеся как визуального оформления, так и структуры расположения элементов. Разработка финального дизайна велась с учётом требований корпоративного стиля, технических ограничений и практического удобства для пользователей.

На данном этапе начинается рассмотрение итогового интерфейса, реализованного в финальной версии приложения. При этом некоторые окна визуально и функционально были переработаны по сравнению с прототипами, а также были добавлены новые страницы, не предусмотренные изначально на этапе проектирования в Figma.

Итоговая версия страницы авторизации почти не отличается от своего прототипа из Figma ( рисунок 5)..



Рисунок 5 - Итоговая страница авторизации

Главная страница открывается сразу после авторизации и служит стартовой точкой для работы пользователя ( рисунок 6). В верхней части располагается панель управления: логотип компании, поле поиска, фильтры по категориям и производителям, а также кнопка раскрытия бокового меню.

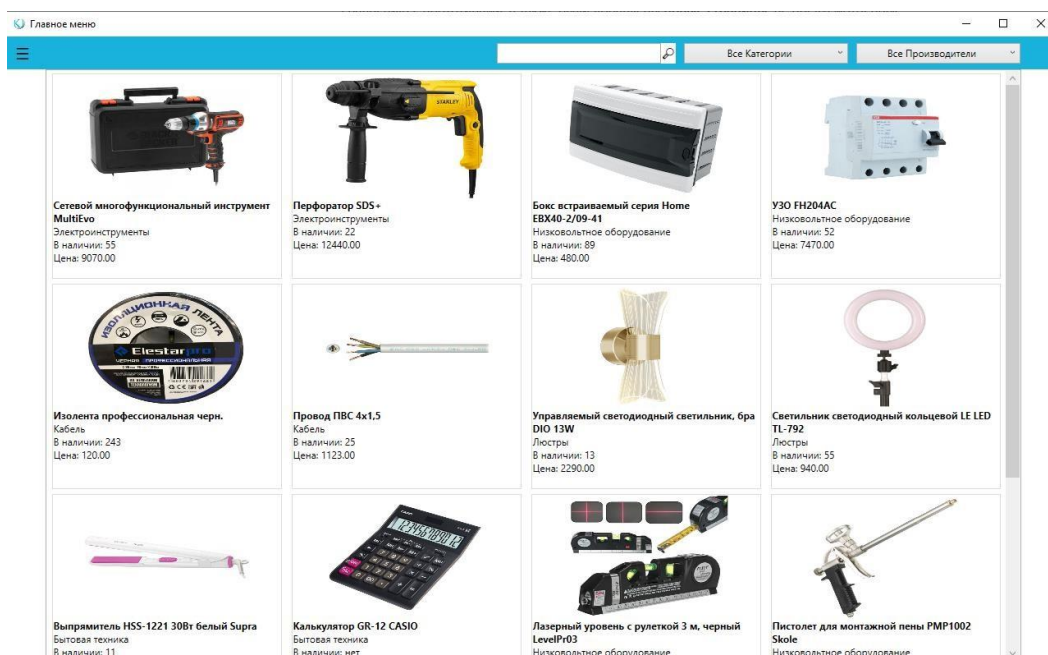



Рисунок 6 - Итоговая главная страница

Ниже представлена товарная сетка, выполненная в виде карточек с изображениями. Каждая карточка включает название товара, категорию, количество в наличии и цену. Интерфейс ориентирован на визуальное восприятие и позволяет быстро находить нужные позиции.

При нажатии на любую карточку с товаром открывается окно просмотра информации о нем. Данное окно отображает полную информацию о выбранном товаре: наименование, категорию, производителя, артикул, цену, наличие на складе, а также подробное описание и технические характеристики (рисунок 7).

Справа выводится изображение товара. В нижней части размещена кнопка создания заказа на данный товар, а также кнопка закрытия окна. Интерфейс оформлен лаконично и симметрично, данные структурированы в две колонки для удобства восприятия.

Наименование: Провод ПВС 4х1,5	Количество на складе: 25
Производитель: ККЗ	Описание: Провод со скрученными медными жилами с ПВХ изоляцией, с ПВХ оболочкой, гибкий, на
Категория: Кабель	Характеристики: Максимальное рабочее напряжение, Вольт 450/750; Длина бухты, м 100
Тип продукта: Силовой кабель ПВС, КГ	Изображение:
Артикул: 01-8206-1	
Стоимость: 1123,00	

Создать заказ с Провод ПВС 4х1,5?

Создать      Закрыть

Рисунок 7 - Окно просмотра информации о товаре

Снизу окна расположены две кнопки с верхней подписью предлагающей пользователю создать заказ с текущим товаром. При нажатии на кнопку создать откроется окно создания заказа.

Окно предназначено для оформления нового заказа. Пользователь заполняет поля: выбранный товар, статус, даты заказа и выполнения, контактные данные клиента, заказчик, склад отправления, адрес доставки и количество (рисунок 8). Также отображаются автоматические поля — создатель заказа и текущая дата. Интерфейс построен в единой вертикальной структуре с двумя управляющими кнопками внизу: «Сохранить» и «Отмена». Все поля упорядочены и подписаны, что обеспечивает удобное и быстрое заполнение.

Рисунок 8 - Окно создания заказа

При нажатии на символ с тремя полосками на главной странице открывается боковая панель. Отображается она слева и включает информацию о пользователе:

ФИО, аватар, а также элементы управления.

Менеджеру доступны действия:

- Параметры профиля — переход к настройке учётной записи;
- Просмотр заказов — доступ к списку оформленных заказов;
- Выход — завершение сеанса.

Панель оформлена в корпоративном синем цвете и остаётся доступной при навигации, обеспечивая быстрый доступ к основным функциям (рисунок 9).

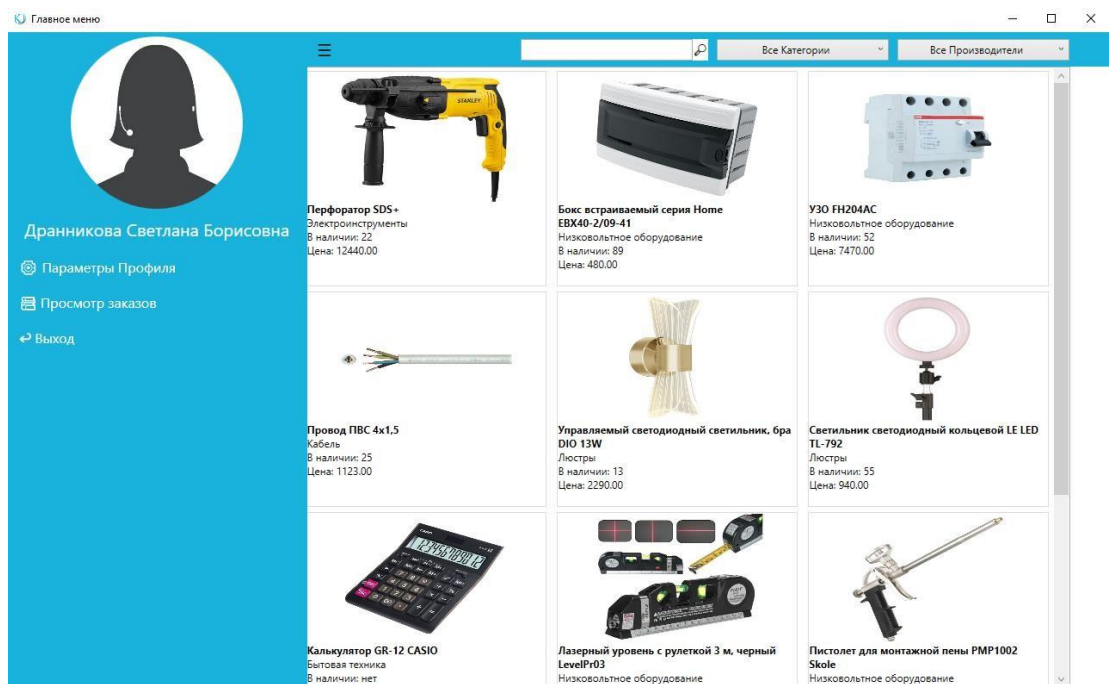
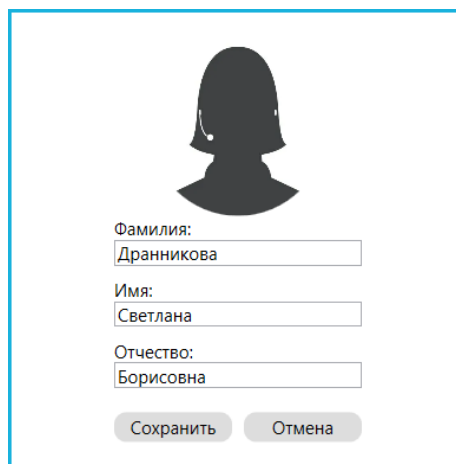


Рисунок 9 - Главная страница с боковой панелью менеджера

При нажатии на кнопку «параметры профиля» появится форма редактирования профиля которая содержит поля для изменения фамилии, имени и отчества пользователя. В верхней части отображается аватар, который можно заменить вручную — при нажатии открывается окно выбора изображения. Ниже расположены кнопки «Сохранить» и «Отмена», оформленные в общем стиле интерфейса(рисунок 10). Окно позволяет быстро обновить персональные данные и визуальную идентификацию пользователя в системе.





Фамилия:  
Дранникова

Имя:  
Светлана

Отчество:  
Борисовна

Сохранить Отмена

Рисунок 10 - Окно параметров профиля пользователя

Страница просмотра заказов предназначена для просмотра всех оформленных заказов. Доступ к ней осуществляется через боковую панель менеджера — пункт «Просмотр заказов». После перехода открывается список, в котором каждая строка содержит изображение товара, его характеристики, ФИО оформителя, даты оформления и завершения, контактные данные, склад отправки, адрес назначения, количество и текущий статус заказа.

Каждую запись можно отредактировать через кнопку «Изменить информацию», расположенную справа. В нижней части экрана доступны кнопки «Назад» и «Оформить заказ». Интерфейс упрощает управление большим числом заказов и обеспечивает быстрый доступ к ключевым данным (рисунок 11).

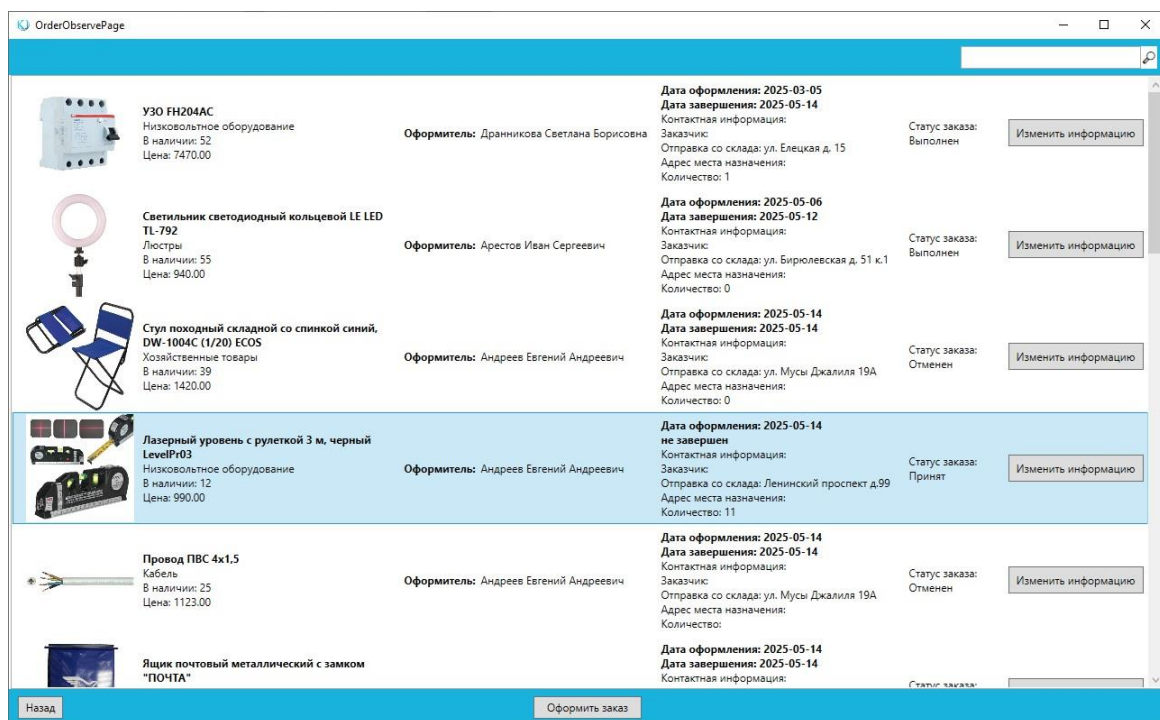


Рисунок 11 - Страница просмотра заказов

## Главная страница администратора и боковая панель

На главной странице администратора отображается общий каталог товаров в стандартной сетке. Слева размещена боковая панель с расширенным набором функций:

- Параметры профиля — изменение личных данных;
- Настройка пользователей — управление аккаунтами сотрудников;
- Добавить продукт — переход к форме добавления нового товара;
- Просмотр заказов — переход к списку заказов;
- Выход — завершение сеанса.

Интерфейс выдержан в общем стиле системы, но панель администратора дополнена функциональностью для управления пользователями и товарной номенклатурой (рисунок 12).

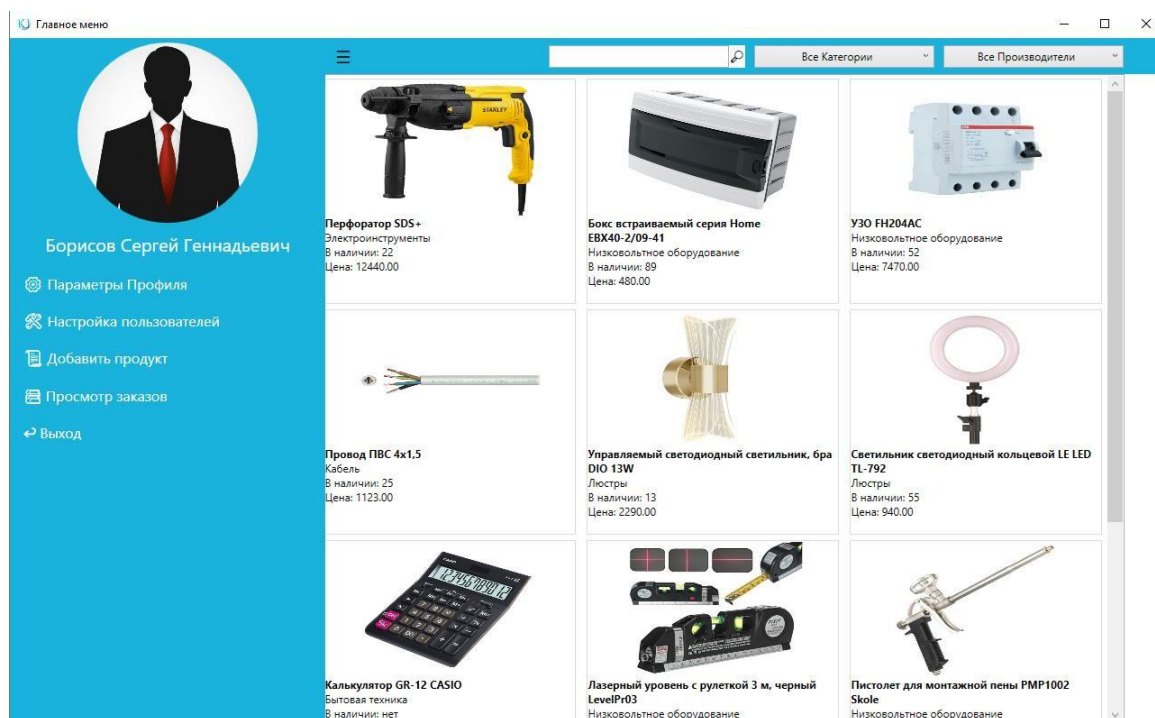


Рисунок 12 - Боковое меню администратора

Переход на страницу настройки пользователей осуществляется из бокового меню администратора через пункт «Настройка пользователей».

На экране отображается список всех зарегистрированных пользователей с указанием ФИО, роли, логина и пароля. Слева представлен аватар, справа — служебные данные и кнопка «Редактировать» для изменения информации. В нижней части расположены кнопки «Назад» и «Добавить пользователя», позволяющие вернуться в главное меню или создать новую учётную запись.

Интерфейс страницы адаптирован для быстрой административной работы с учетными записями сотрудников (рисунок 13).

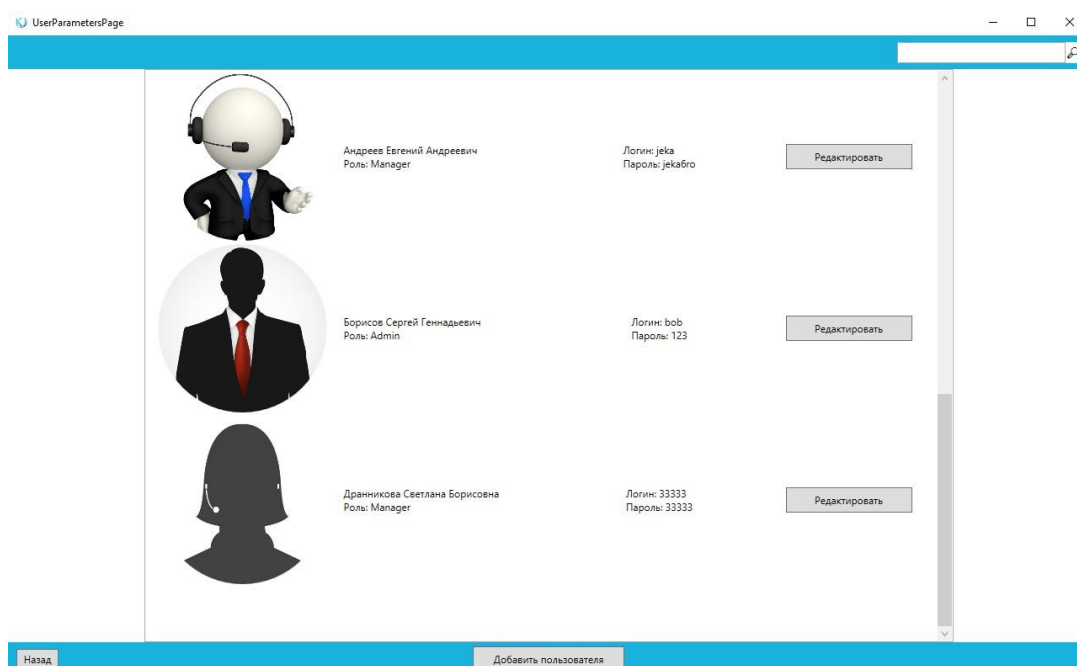


Рисунок 13 - Страница настройки пользователей

Переход к окну редактирования осуществляется с страницы настройки пользователей по нажатию кнопки «Редактировать» возле нужной учётной записи (рисунок 14).

Имя:

Фамилия:

Отчество:

Логин:

Пароль:

Роль:

Рисунок 14 - Окно редактирования пользователя

Переход к окну добавления нового пользователя выполняется нажатием кнопки «Добавить пользователя» в нижней части той же страницы (рисунок 15).

Имя:

Фамилия:

Отчество:

Логин:

Пароль:

Роль:

Сохранить Отмена

Рисунок 15 - Окно добавления пользователя

Оба окна имеют схожую структуру: поля для ввода имени, фамилии, отчества, логина, пароля и выбора роли (Manager/Admin). В окне редактирования дополнительно доступна кнопка «Удалить». Кнопки «Сохранить» и «Отмена» находятся внизу и оформлены в едином стиле системы. Интерфейс лаконичен и интуитивно понятен для администратора.

При нажатии на пункт «Добавить продукт» в боковом меню администратора осуществляется переход на страницу, предназначенную для добавления новой позиции в товарный каталог.

На этой странице размещена форма, позволяющая ввести полные сведения о товаре. Слева находятся поля для ввода наименования, артикула, стоимости, а также выпадающие списки для выбора производителя, категории и типа продукта. Справа расположены поля для указания количества на складе, описания и характеристик (рисунок 16).

Наименование: <input type="text"/>	Количество на складе: <input type="text"/>
Производитель: <input type="text"/>	Описание: <input type="text"/>
Категория: <input type="text"/>	Характеристики: <input type="text"/>
Тип продукта: <input type="text"/>	Изображение: <div></div>
Артикул: <input type="text"/>	<input type="button" value="Сохранить"/>
Стоимость: <input type="text"/>	


Рисунок 16 - Страница добавления товара

Ниже размещён блок с возможностью загрузки изображения — при клике по области пользователь может выбрать фото с устройства. Завершает форму большая кнопка «Сохранить», а также кнопка «Назад» для возврата к предыдущему экрану.

Интерфейс окна упрощён и структурирован по колонкам, что обеспечивает удобное и быстрое внесение данных администратором.

Доступ к добавлению имеет только Администратор.

При нажатии на карточку товара осуществляется переход на страницу, предназначенную для редактирования информации о выбранной позиции.

<p>Наименование: Провод ПВС 4х1,5</p> <p>Производитель: ККЗ</p> <p>Категория: Кабель</p> <p>Тип продукта: Силовой кабель ПВС, КГ</p> <p>Артикул: 01-8206-1</p> <p>Стоимость: 1123,00</p>	<p>Количество на складе: 25</p> <p>Описание: Провод со скрученными медными жилами с ПВХ изоляцией, с ПВХ оболочкой, гибкий.</p> <p>Характеристики: Максимальное рабочее напряжение, Вольт 450/750; Длина бухты, м 100.</p> <p>Изображение: </p> <p>Сохранить</p>
--	---

Назад

Удалить

Рисунок 17 - Страница редактирования товара

На этой странице отображаются уже заполненные поля: наименование, артикул, стоимость, категория, производитель, тип продукта, количество на складе, описание, характеристики и изображение. Пользователь может внести изменения в любые из доступных полей, а затем сохранить их нажатием кнопки «Сохранить». Также доступны кнопки «Удалить» — для удаления товара из базы, и «Назад» — для возврата к каталогу (рисунок 17).

Интерфейс повторяет структуру окна добавления товара, что обеспечивает единый пользовательский опыт и минимизирует необходимость адаптации к новому окну.

При попытке закрытия приложения пользователю отображается модальное окно с запросом подтверждения действия. Система предупреждает о том, что все несохранённые данные будут удалены (рисунок 18).

Пользователь может выбрать одно из двух действий:

- «Закрыть» — завершить работу приложения;
- «Отмена» — вернуться к текущему сеансу без выхода.

Такое решение обеспечивает защиту от случайного завершения работы и потери данных.

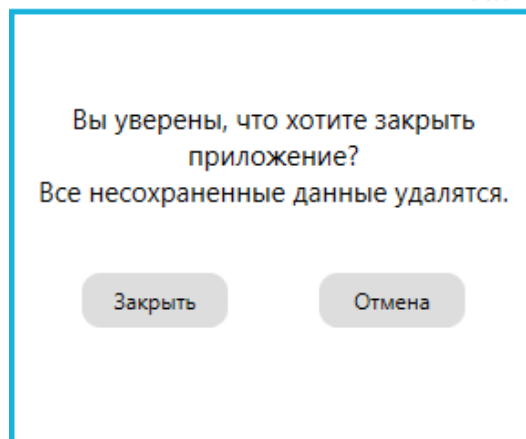


Рисунок 18 - Окно подтверждения выхода

## 2.2 Разработка функциональной части системы учёта

После завершения этапа проектирования визуального интерфейса и утверждения макетной структуры приложения была осуществлена непосредственная реализация логики функционирования системы учёта. Основное внимание в данном этапе уделяется не только построению стабильной архитектуры приложения, но и обеспечению надёжной связи между пользовательским интерфейсом, внутренними обработчиками событий, а также базой данных, в которой хранятся все ключевые сведения о заказах, товарах, пользователях и складах.

Программа создавалась с учётом требований масштабируемости, читаемости и разделения ответственности между компонентами. В основу архитектуры легла типичная для WPF модель, разделяющая представление и логику взаимодействия: страницы (Pages), диалоговые окна (Dialogs), классы и вспомогательные обработчики данных (Data, Classes). На начальном этапе реализация была сосредоточена в рамках единого проекта под названием *klstroy*, структура которого представлена в обозревателе решений Visual Studio.

На изображении ниже представлена структура проекта в Visual Studio, демонстрирующая логическую организацию всех файлов приложения (рисунок 19).

Проект содержит следующие ключевые папки и элементы:



- Dialogs — включает в себя модальные окна и всплывающие формы, используемые для подтверждения действий, редактирования заказов, добавления пользователей и настройки профиля.
- Pages — основной набор страниц приложения, реализующих функциональность для добавления, редактирования, просмотра и управления данными.
- Data — отвечает за работу с базой данных. Здесь хранятся модели сущностей и доступ к хранилищу.
- Classes — вспомогательные классы и утилиты, обеспечивающие вспомогательные функции.
- App.xaml / MainWindow.xaml — стартовая точка WPF- приложения, определяющая глобальные стили и начальную навигацию.
- App.config — конфигурационный файл с параметрами подключения к базе данных и другими настройками.

Таким образом, структура проекта построена по принципу логического разграничения: каждая папка отвечает за отдельный аспект функциональности, что обеспечивает удобство сопровождения и расширения системы.

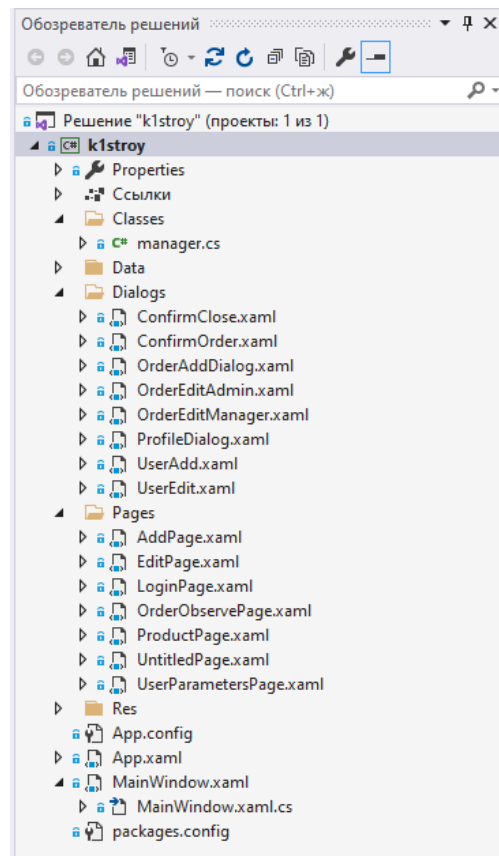


Рисунок 19 - Структура проекта

Система построена по принципу одного главного окна, в котором происходит переключение между различными функциональными страницами. Такой подход реализуется с помощью элемента `Frame`, встроенного в структуру `MainWindow.xaml`. Это решение позволяет избежать открытия множества отдельных окон и обеспечивает централизованную навигацию, что особенно важно в корпоративных приложениях с различными ролями пользователей.

В разметке окна, представленной в `MainWindow.xaml`, задаётся базовая конфигурация, в том числе габариты окна, заголовок, а главное — объявляется `Frame` с именем `MainFrame`, куда в дальнейшем и загружаются страницы (рисунок 20).

```

<Window x:Class="k1stroy.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:k1stroy"
        mc:Ignorable="d" WindowStartupLocation="CenterScreen"
        Title="{Binding ElementName=MainFrame, Path=Content.Title}" Height="800" Width="1300"
        MinHeight="650" MinWidth="1000"
        MaxHeight="1080" MaxWidth="1920">
    <Grid>
        <Frame x:Name="MainFrame" NavigationUIVisibility="Hidden"></Frame>
    </Grid>
</Window>

```

Рисунок 20 - Структура окна MainWindow.xaml

Как видно из разметки, MainFrame размещён в корневом элементе Grid, имеет скрытую панель навигации (NavigationUIVisibility="Hidden") и гибко адаптируется под размеры экрана. Это позволяет менять отображаемые страницы без перезагрузки всего окна, что особенно удобно для систем с множеством различных ролей и действий.

В коде MainWindow.xaml.cs происходит начальная инициализация интерфейса. В методе конструктора вызывается InitializeComponent(), записывается лог запуска системы, инициализируется глобальный MainFrame, а затем осуществляется переход на страницу авторизации.

Кроме того, в обработчике события закрытия окна MainWindow\_Closing реализована логика подтверждения выхода. Если переменная Leave не равна нулю (то есть пользователь находится в сеансе), система вызывает модальное окно ConfirmClose, предупреждая о возможной потере данных. В случае подтверждения — сеанс завершается, иначе — отменяется закрытие окна (рисунок 21).

```

namespace k1stroy
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    Ссылка: 2
    public partial class MainWindow : Window
    {
        Ссылка: 0
        public MainWindow()
        {
            InitializeComponent();
            //Classes.manager.GetImageData();
            string logline = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} - запуск клиента";
            File.AppendAllText("log.txt", logline + Environment.NewLine);
            Classes.manager.MainFrame = MainFrame;
            Classes.manager.MainFrame.Navigate(new Pages.LoginPage());
            this.Closing += MainWindow_Closing;
        }

        Ссылка: 1
        private void MainWindow_Closing(object sender, CancelEventArgs e)
        {
            if (Classes.manager.Leave != 0)
            {
                var dialog = new Dialogs.ConfirmClose();
                dialog.Owner = this;
                dialog.ShowDialog();
                if (!dialog.IsConfirmed)
                {
                    e.Cancel = true;
                }
                else
                {
                    string logline = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} - клиент закрыт";
                    File.AppendAllText("log.txt", logline + Environment.NewLine);
                }
            }
            else
            {
                string logline = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} - клиент закрыт";
                File.AppendAllText("log.txt", logline + Environment.NewLine);
            }
        }
    }
}

```

Рисунок 21 - Логика работы главного окна (MainWindow.xaml.cs)

Ключевым элементом в обеспечении доступа к MainFrame из любых других компонентов системы является класс manager, размещённый в пространстве имён k1stroy.Classes. Этот вспомогательный класс содержит три публичных статических свойства: счётчик Leave, текущий пользователь CurrentUser, и главное — публичный доступ к MainFrame, что и позволяет выполнять навигацию из любого окна или диалогового элемента системы (рисунок 22).

```

namespace k1stroy.Classes
{
    Ссылка: 43
    public class manager
    {
        Ссылка: 7
        public static int Leave;
        Ссылка: 32
        public static Data.Users CurrentUser { get; set; }
        public static Frame MainFrame { get; set; }
    }
}

```

Рисунок 22 - Класс manager с глобальным доступом к MainFrame

Такой подход обеспечивает гибкую, централизованную и масштабируемую структуру проекта, позволяя эффективно переключаться между функциональными модулями системы — будь то авторизация, работа с заказами, управление товарами или администрирование пользователей.

Основой любой системы учёта является надёжное и структурированное хранилище данных. В рамках разрабатываемого приложения взаимодействие с базой данных Microsoft SQL Server осуществляется с использованием технологии Entity Framework в режиме Database First. Это позволило автоматически сгенерировать модель данных на основе уже существующей базы данных, обеспечив высокую скорость разработки и точность соответствия между структурой БД и классами C#.

Для создания и управления моделью данных в проект был добавлен EDMX-файл<sup>1</sup> под названием Model1.edmx, расположенный в директории Data. После подключения к базе данных в Visual Studio был выполнен импорт всех необходимых таблиц, связей и ограничений, на основе которых была автоматически сгенерирована объектная модель (рисунок 23).

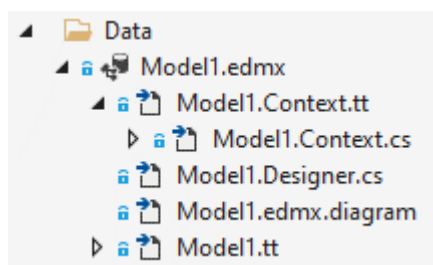


Рисунок 23 - Структура EDMX-модели и автосгенерированных файлов

Файл Model1.Context.tt содержит реализацию контекста базы данных, необходимого для выполнения запросов, а .Designer.cs и .tt генерируют классы сущностей, соответствующих таблицам.

Связь между системой и базой данных настраивается в конфигурационном файле App.config путём добавления строки подключения с именем k1stroyDBEntities. Строка указывает на ресурс модели, имя

---

<sup>1</sup> EDMX-файл (Entity Data Model XML) — файл, содержащий описание концептуальной модели, модели хранения и отображения (mapping) для Entity Framework. Используется при визуальном проектировании модели данных в .NET-приложениях.

источника данных, название базы (k1stroyDB), параметры безопасности и провайдер (System.Data.EntityClient) (рисунок 24).

```
<connectionStrings>
  <add name="k1stroyDBEntities"
        connectionString="metadata=res://*/Data.Model1.csdl|res://*/Data.Model1.ssdl|res://*/Data.Model1.msl;provider=System.Data.SqlClient;provider
        connection string=&quot;data source=XTEND0;initial catalog=k1stroyDB;integrated
        security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;;" providerName="System.Data.EntityClient" />
</connectionStrings>
```

Рисунок 24 - Строка подключения в App.config

Таким образом, вся работа с БД осуществляется через именованный контекст, позволяющий Entity Framework автоматически определять путь к модели и необходимые метаданные.

Для упрощения и оптимизации доступа к базе данных в проекте реализован паттерн Singleton. Вместо создания нового экземпляра контекста при каждом обращении, используется единый статический метод GetContext(), который возвращает уже существующий экземпляр контекста или создаёт новый при первом вызове (рисунок 25).

```
public partial class k1stroyDBEntities : DbContext
{
    private static k1stroyDBEntities _context;

    public static k1stroyDBEntities GetContext()
    {
        if (_context == null)
        {
            _context = new k1stroyDBEntities();
        }
        return _context;
    }
}
```

Рисунок 25 - Статический доступ к базе данных через GetContext()

Этот подход позволяет:

- избежать лишней нагрузки на память и ресурсы SQL-сервера;
- централизованно контролировать все изменения в базе;
- удобно использовать контекст из любой точки приложения без необходимости повторного подключения.

Таким образом, архитектура работы с базой данных построена на современных подходах и сочетает в себе удобство, производительность и расширяемость. Entity Framework<sup>2</sup> позволяет работать с данными как с

---

<sup>2</sup> Entity Framework — это объектно-реляционный сопоставитель (ORM) с открытым исходным кодом для .NET

объектами, а единый доступ через `GetContext()` делает код более чистым и управляемым.

Функциональность авторизации реализована в обработчике события `loginBut_Click`, который срабатывает при нажатии пользователем кнопки «Войти». На начальном этапе производится базовая проверка введенных данных на пустоту. Если логин или пароль не введены, пользователю выводится соответствующее предупреждение (рисунок 26).

```
ссылка: 1
private void loginBut_Click(object sender, RoutedEventArgs e)
{
    StringBuilder err = new StringBuilder();
    if (String.IsNullOrEmpty(loginTB.Text) || loginTB.Text == "логин")
    {
        err.AppendLine("Пожалуйста введите логин");
    }
    if (String.IsNullOrEmpty(passwordPB.Password))
    {
        err.AppendLine("Пожалуйста введите пароль");
    }
    if (err.Length > 0)
    {
        MessageBox.Show(err.ToString(), "Внимание!", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }
}
```

Рисунок 26 - Проверка на пустые значения

После валидации введенные данные сравниваются с записями в таблице `Users` базы данных. Для этого используется контекст `k1stroyDBEntities.GetContext()` и LINQ-запрос<sup>3</sup>. Если найдено соответствие логина и пароля, текущий пользователь сохраняется в `Classes.manager.CurrentUser`, после чего система выполняет навигацию на соответствующую страницу в зависимости от роли пользователя (администратор, менеджер) (рисунок 27).

```
if (Data.k1stroyDBEntities.GetContext().Users
    .Any(d => d.Login == loginTB.Text &&
        d.Password == passwordPB.Password))
{
    var user = Data.k1stroyDBEntities.GetContext().Users
        .Where(d => d.Login == loginTB.Text
            && d.Password == passwordPB.Password).FirstOrDefault();
    Classes.manager.CurrentUser = user;
    MessageBox.Show($"Добро пожаловать, {user.Surname} {user.Firstname} {user.Patronymic}!", "Выполнен вход в учетную запись", MessageBoxButton.OK, MessageBoxImage.Information);
    string logline = $"{DateTime.Now:yyyy-MM-dd HH:mm:ss} - выполнен вход в учетную запись ({user.Surname} {user.Firstname} {user.Patronymic})";
    File.AppendAllText("log.txt", logline + Environment.NewLine);
    switch (user.Roles.Role)
    {
        case "Admin":
            Classes.manager.MainFrame.Navigate(new Pages.ProductPage(user));
            break;
        case "Manager":
            Classes.manager.MainFrame.Navigate(new Pages.ProductPage(user));
            break;
    }
}
```

Рисунок 27 - Логика авторизации

<sup>3</sup> LINQ (Language Integrated Query) — компонент платформы .NET, позволяющий выполнять запросы к данным в виде синтаксически встроенных выражений C# или VB.NET.

Также реализована запись в лог-файл события входа в систему с фиксированием даты, времени и данных пользователя. При успешной авторизации выводится приветственное окно, а навигация между страницами выполняется с помощью `Classes.manager.MainFrame.Navigate(...)`

После прохождения авторизации пользователи системы автоматически перенаправляются на главную страницу приложения, реализованную в файле `ProductPage.xaml`. Переход осуществляется с помощью фрейма `MainFrame`, что позволяет динамически загружать содержимое в пределах одного окна, не открывая дополнительные формы. Таким образом, главная страница служит центральным узлом для взаимодействия с товарной базой.

На этапе инициализации страницы в конструкторе `ProductPage` происходит передача текущего пользователя и привязка данных к интерфейсу. Также задаётся переменная `Leave`, отвечающая за корректное завершение работы сессии. Далее вызывается метод `RoleChecker()`, который определяет, какие элементы интерфейса будут доступны в зависимости от роли пользователя (рисунок 28). Например, менеджеру недоступна кнопка добавления товара, в то время как администратору — весь функционал открыт.

Рисунок 28 - Инициализация главной страницы

```
public partial class ProductPage : Page
{
    Ссылка: 11
    public Data.Users currentUser { get; set; }
    Ссылка: 3
    public ProductPage(Data.Users user)
    {
        currentUser = user;
        DataContext = this;
        Classes.manager.Leave = 1;

        InitializeComponent();
        RoleChecker();
        NoFounds.Visibility = Visibility.Collapsed;
        LoadProducts();
    }
}
```

Визуальная часть страницы оформлена с использованием стандартного WPF-интерфейса. Навигационное меню реализовано с помощью анимации раскрытия, управляемой методом `HamburgerButton_Click`. При нажатии кнопки гамбургера выполняется анимация смещения панели, которая плавно



появляется или скрывается с левой стороны экрана (рисунок 29).

```
private void HamburgerButton_Click(object sender, RoutedEventArgs e)
{
    _isMenuOpen = !_isMenuOpen;

    var animation = new ThicknessAnimation
    {
        Duration = TimeSpan.FromSeconds(0.3),
        EasingFunction = new QuadraticEase { EasingMode = EasingMode.EaseInOut }
    };
    var animation_but = new ThicknessAnimation
    {
        Duration = TimeSpan.FromSeconds(0.3),
        EasingFunction = new QuadraticEase { EasingMode = EasingMode.EaseInOut }
    };

    animation.To = _isMenuOpen ? new Thickness(0, 0, 0, 0) : new Thickness(-350, 0, 0, 0);
    SideMenu.BeginAnimation(Border.MarginProperty, animation);

    animation.To = _isMenuOpen ? new Thickness(350, 0, 0, 0) : new Thickness(0, 0, 0, 0);
    HamburgerButton.BeginAnimation(Border.MarginProperty, animation);
}
```

Рисунок 29 - Анимация появления бокового меню

Загрузка информации о товарах осуществляется методом LoadProducts(). В его рамках из базы данных извлекаются коллекции всех товаров, категорий и производителей. Далее эти данные привязываются к соответствующим визуальным элементам интерфейса: списку товаров (ProductsListView) и выпадающим спискам категорий и производителей. Для удобства поиска пользователь может выбрать фильтры «Все категории» и «Все производители», которые добавляются программно в начало каждого списка (рисунок 30).

```
private void LoadProducts()
{
    var products = Data.kistroyDBEntities.GetContext().Products.ToList();
    Products = new ObservableCollection<Data.Products>(products);
    ProductsListView.ItemsSource = Products;

    var categoryList = Data.kistroyDBEntities.GetContext().ProductCategory.ToList();
    categoryList.Insert(0, new Data.ProductCategory { CategoryName = "Все Категории" });
    CategoryComboBox.ItemsSource = categoryList;
    CategoryComboBox.SelectedIndex = 0;

    var manufacturerList = Data.kistroyDBEntities.GetContext().Manufacturers.ToList();
    manufacturerList.Insert(0, new Data.Manufacturers { ManufacturerName = "Все Производители" });
    ManufacturerComboBox.ItemsSource = manufacturerList;
    ManufacturerComboBox.SelectedIndex = 0;
}
```

Рисунок 30 - Загрузка данных и фильтров в интерфейс

Для расширенного управления данными реализована возможность поиска и фильтрации. Метод Update() обрабатывает текстовые запросы и параметры, установленные в комбинированных списках. Результаты фильтрации отображаются в том же списке, при этом блок NoFounds становится видимым, если результат отсутствует. Таким образом, обеспечивается высокое удобство работы с каталогом продукции (рисунок 31).

```

public List<Data.Products> _products = Data.kistroyDBEntities.GetContext().Products.ToList();
Ссылка: 6
public void Update()
{
    try
    {
        ListGrid.Visibility = Visibility.Visible;
        NoFounds.Visibility = Visibility.Collapsed;

        _products = Data.kistroyDBEntities.GetContext().Products.ToList();

        if (!string.IsNullOrEmpty(SearchTextBox.Text))
        {
            _products = Data.kistroyDBEntities.GetContext().Products
                .Where(p => p.ProductName.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Description.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Manufacturers.ManufacturerName.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Price.ToString().Contains(SearchTextBox.Text))
                .ToList();
        }

        var selectedCategory = CategoryComboBox.SelectedItem as Data.ProductCategory;
        if (selectedCategory != null && selectedCategory.CategoryName != "Все Категории")
        {
            _products = _products
                .Where(p => p.ProductCategory != null && p.ProductCategory.ID == selectedCategory.ID)
                .ToList();
        }

        var selectedManufacturer = ManufacturerComboBox.SelectedItem as Data.Manufacturers;
        if (selectedManufacturer != null && selectedManufacturer.ManufacturerName != "Все Производители")
        {
            _products = _products
                .Where(p => p.Manufacturers != null && p.Manufacturers.ID == selectedManufacturer.ID)
                .ToList();
        }

        ProductsListView.ItemsSource = _products;

        if (_products.Count == 0)
        {
            ListGrid.Visibility = Visibility.Collapsed;
            NoFounds.Visibility = Visibility.Visible;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}

```

Рисунок 31 - Логика фильтрации

Ключевым элементом системы является корректная обработка выхода пользователя. Метод Logout() обнуляет текущего пользователя, очищает ресурсы изображения профиля, а также инициирует сбор мусора (GC.Collect), чтобы освободить используемую память (рисунок 32). Это особенно важно для настольных приложений, в которых критична стабильность и долгосрочная работа без утечек памяти.

```
private void Logout()
{
    if (PFP?.Source is BitmapImage bitmap)
    {
        bitmap.StreamSource?.Dispose();
        bitmap.UriSource = null;
        PFP.Source = null;
    }

    Classes.manager.CurrentUser = null;

    GC.Collect();
    GC.WaitForPendingFinalizers();
}
```

Рисунок 32 - очистка мусора и выход из учетной записи

Таким образом, главная страница объединяет в себе функции отображения, фильтрации и управления товарной информацией с учётом прав доступа, обеспечивая централизованный и гибкий интерфейс для различных ролей в системе учёта.

Боковая панель формируется динамически в зависимости от роли текущего пользователя, информация о котором содержится в объекте CurrentUser. При инициализации главной страницы вызывается метод RoleChecker(), определяющий доступные элементы интерфейса.

В верхней части панели отображаются данные авторизованного пользователя: фотография (если задана) и полное имя. Ниже размещаются кнопки управления, список которых различается для ролей администратор и менеджер (рисунок 33). Таким образом обеспечивается персонализированный доступ к функциям системы.

```

private void RoleChecker()
{
    switch (CurrentUser.Roles.Role)
    {
        case "Admin":
            Settings.Visibility = Visibility.Visible;
            Add.Visibility = Visibility.Visible;
            OrderCheck.Visibility = Visibility.Visible;
            AllSettings.Visibility = Visibility.Visible;
            break;
        case "Manager":
            Settings.Visibility = Visibility.Visible;
            Add.Visibility = Visibility.Collapsed;
            OrderCheck.Visibility = Visibility.Visible;
            AllSettings.Visibility = Visibility.Collapsed;
            break;
    }
}

```

Рисунок 33 - Обработка боковой панели пользователя

После нажатия на карточку конкретного товара на главной странице осуществляется переход к окну подтверждения заказа. Это окно открывается с передачей двух аргументов: выбранного товара и текущего пользователя.

Внутри конструктора окна происходит инициализация интерфейса и заполнение всех текстовых полей на основании данных переданного объекта Product. Отображаются такие сведения, как наименование, артикул, цена, количество на складе, описание, технические характеристики, а также связанная информация о категории, типе и производителе товара. (рисунок 34).

```

public ConfirmOrder(Data.Products selectedProduct, Data.Users CurrUser)
{
    InitializeComponent();

    CurrentEl = selectedProduct;
    CurrentUser = CurrUser;
    ProductSin = CurrentEl.ProductName.ToString();
    DataContext = this;

    NameTB.Text = CurrentEl.ProductName;
    ArticleTB.Text = CurrentEl.Article;
    PriceTB.Text = CurrentEl.Price.ToString();
    InStockTB.Text = CurrentEl.InStock.ToString();
    DescTB.Text = CurrentEl.Description;
    CharTB.Text = CurrentEl.Characteristics;
    ManufacturerComboBox.Text = CurrentEl.Manufacturers.ManufacturerName;
    CategoryCB.Text = CurrentEl.ProductCategory.CategoryName;
    TypeCB.Text = CurrentEl.ProductType.TypeName;
}

```

Рисунок 34 - Логика окна подтверждения заказа

Окно создания нового заказа реализовано в виде отдельного диалогового окна OrderAddDialog.xaml, вызываемого при оформлении нового заказа с карточки товара. Логика взаимодействия сосредоточена в одноимённом классе.

Инициализация окна производится через конструктор, принимающий в качестве параметров объект выбранного продукта, текущего пользователя, а также делегат обратного вызова и строку статуса. В конструкторе производится установка контекста данных, а также инициализация текстовых и комбинированных полей на основе переданного объекта (рисунок 35).

```
public OrderAddDialog(Data.Products selectedProduct, Data.Users CurrUser, Action Show, String _status)
{
    InitializeComponent();

    CurrentEl = selectedProduct;
    CurrentUser = CurrUser;
    DataContext = this;
    ShowWindow = Show;
    Status = _status;

    StatusCheck();
}
```

Рисунок 35 - Инициализация окна создания заказа

После инициализации окна выполняется заполнение выпадающих списков (ComboBox) актуальными значениями из базы данных: список статусов, заказчиков, товаров и складов (рисунок 36).

```
StatusCB.ItemsSource = Data.k1stroyDBEntities.GetContext().Status.ToList();
CustomerCB.ItemsSource = Data.k1stroyDBEntities.GetContext().Users.ToList();
ProductCB.ItemsSource = Data.k1stroyDBEntities.GetContext().Products.ToList();
StorageFromTB.ItemsSource = Data.k1stroyDBEntities.GetContext().Storages.ToList();
```

Рисунок 36 - Наполнение выпадающих списков

После ввода необходимых данных пользователем и нажатия кнопки подтверждения, выполняется формирование нового объекта заказа. В объекте указываются идентификаторы товара, заказчика, склада-отправителя, данные клиента, количество, дата оформления, а также устанавливается начальный статус заказа.

Далее объект сохраняется в базу данных через контекст Entity Framework, вызовом SaveChanges (рисунок 37).

```

var newOrder = new Data.Orders
{
    OrderCreatorID = selectedCreator.ID,
    OrderProductID = selectedProduct.ID,
    ContactData = ContactDataTB.Text,
    CustomerName = CustomerNameTB.Text,
    StorageFrom = selectedStorage.ID,
    StorageTo = StorageToTB.Text,
    Count = CountRes,
    OrderDate = DateTime.Now,
    OrderCompleteDate = null,
    OrderStatusID = selectedStatus.ID
};

var context = Data.k1stroyDBEntities.GetContext();
context.Orders.Add(newOrder);
context.SaveChanges();

```

Рисунок 37 - Формирование и сохранение нового заказа

Такой подход позволяет динамически создавать заказы с учетом выбранных данных и текущего пользователя. Все данные при этом валидируются перед добавлением.

Следующей рассмотренной страницей является окно просмотра заказов, доступное как менеджеру, так и администратору. Попасть на данную страницу можно через боковое меню, выбрав пункт «Просмотр заказов».

При открытии страницы вызывается конструктор OrderObservePage, в котором задаются роль пользователя и его идентификатор. Далее скрываются элементы, отвечающие за отсутствие данных, и вызывается метод LoadProducts() — инициализация списка заказов из базы данных (рисунок 38)



```

public OrderObservePage(string RoleCheck, int CurrrentUserID)
{
    role = RoleCheck;
    UserID = CurrrentUserID;
    InitializeComponent();
    NoOrders.Visibility = Visibility.Collapsed;
    NoFoundOrders.Visibility = Visibility.Collapsed;
    LoadProducts();
}

```

Рисунок 38 - Инициализация страницы просмотра заказов

Реализация фильтрации осуществляется через метод Update(), который подгружает все заказы и, в зависимости от введенного текста в поле поиска, выполняет LINQ-запрос с множеством условий: поиск по названию товара, ФИО оформителя, дате, складу, статусу и другим параметрам (рисунок 39).

```

private void Update()
{
    _orders = Data.k1stroyDBEntities.GetContext().Orders.ToList();

    if(role == "Manager" || role == "Admin")
    {
        if (!string.IsNullOrEmpty(SearchTextBox.Text))
        {
            _orders = Data.k1stroyDBEntities.GetContext().Orders
                .Where(p => p.Products.ProductName.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Users.Firstname.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Users.Surname.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Users.Patronymic.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.ID.ToString().Contains(SearchTextBox.Text.ToLower()) ||
                    p.OrderDate.ToString().Contains(SearchTextBox.Text.ToLower()) ||
                    p.OrderCompleteDate.ToString().Contains(SearchTextBox.Text.ToLower()) ||
                    p.ContactData.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.CustomerName.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.StorageTo.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                    p.Products.Price.ToString().Contains(SearchTextBox.Text))
                .ToList();
        }
    }
}

```

Рисунок 39 - Метод Update со множественной фильтрацией

Метод Update() вызывается также при каждом изменении текста в поле поиска (SearchTextBox\_TextChanged), что обеспечивает мгновенный отклик интерфейса (рисунок 40).

ссылка: 1

```

private void SearchTextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    Update();
}

```

Рисунок 40 - Вызов метода Update при изменении текста в строке поиска

В зависимости от роли пользователя при нажатии кнопки «Изменить информацию» вызываются разные диалоговые окна: для администратора — OrderEditAdmin, для менеджера — OrderEditManager. В каждом случае происходит передача выбранного заказа, привязка окна к текущему и обновление интерфейса (рисунок 41).

```
ссылка: 1
private void EditOrderButton_Click(object sender, RoutedEventArgs e)
{
    if(role == "Manager")
    {
        var OrderObservePage = Classes.manager.MainFrame.Content as OrderObservePage;
        if (sender is Button button && button.DataContext is Data.Orders selectedOrder)
        {
            var dialog = new Dialogs.OrderEditManager(selectedOrder);
            dialog.Owner = Application.Current.Windows.OfType<Window>()
                .FirstOrDefault(w => w.IsActive);
            dialog.WindowStartupLocation = WindowStartupLocation.CenterOwner;
            dialog.ShowDialog();
            OrdersView.Items.Refresh();
        }
    }
    if(role == "Admin")
    {
        var OrderObservePage = Classes.manager.MainFrame.Content as OrderObservePage;
        if (sender is Button button && button.DataContext is Data.Orders selectedOrder)
        {
            var dialog = new Dialogs.OrderEditAdmin(selectedOrder);
            dialog.Owner = Application.Current.Windows.OfType<Window>()
                .FirstOrDefault(w => w.IsActive);
            dialog.WindowStartupLocation = WindowStartupLocation.CenterOwner;
            dialog.ShowDialog();
            OrdersView.Items.Refresh();
        }
    }
}
```

Рисунок 41 - проверка роли для вызова окна редактирования

Кроме того, реализована возможность изменения статуса заказа напрямую через выпадающий список. При выборе нового значения статуса происходит обновление записи в базе и сохранение изменений (рисунок 42).



```

Ссылка: 0
private void StatusCB_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (sender is ComboBox comboBox &&
        comboBox.DataContext is Data.Orders selectedOrder &&
        comboBox.SelectedItem is Data.Status selectedStatus)
    {
        selectedOrder.OrderStatusID = selectedStatus.ID;
        selectedOrder.Status = selectedStatus;

        Data.k1stroyDBEntities.GetContext().SaveChanges();
        OrdersView.Items.Refresh();
    }
}

```

Рисунок 42 - Обновление статуса заказа в списке

Страница настройки пользователей предназначена для управления учётными записями сотрудников внутри системы. Доступ к данной странице предоставляется исключительно администраторам. Интерфейс включает таблицу со списком пользователей, строку поиска, а также кнопки для добавления и редактирования записей (рисунок 33).

```

Ссылка: 1
public UserParametersPage()
{
    InitializeComponent();
    LoadUsers();
}

Ссылка: 3
private void LoadUsers()
{
    UsersList.ItemsSource = Data.k1stroyDBEntities.GetContext().Users.ToList();
}

```

Рисунок 43 - Инициализация страницы настройки пользователей и загрузка списка

При открытии страницы автоматически подгружается информация обо всех пользователях из базы данных. Каждый элемент списка содержит сведения: имя, фамилию, отчество, логин, пароль и роль.

В верхней части размещена строка поиска, позволяющая фильтровать записи. Поиск осуществляется сразу по нескольким полям: имени, фамилии, роли, логину, паролю и отчеству.

Для добавления новой записи предусмотрена кнопка, открывающая модальное окно. В нём администратор может ввести данные нового пользователя, указав все необходимые поля (рисунок 44).

```
private void AddButton_Click(object sender, RoutedEventArgs e)
{
    var OrderObservePage = Classes.manager.MainFrame.Content as OrderObservePage;

    ...
    var dialog = new Dialogs.UserAdd(LoadUsers);
    dialog.Owner = Application.Current.Windows.OfType<Window>()
    ...
        .FirstOrDefault(w => w.IsActive);
    dialog.WindowStartupLocation = WindowStartupLocation.CenterOwner;
    dialog.ShowDialog();
    ...
}
```

Рисунок 44 - Открытие окна добавления пользователя

Каждую запись можно отредактировать. Для этого используется отдельная кнопка, при нажатии на которую появляется окно редактирования с возможностью изменить любую информацию о выбранном пользователе (рисунок 45).

```
ссылка: 1
private void EditUserButton_Click(object sender, RoutedEventArgs e)
{
    var OrderObservePage = Classes.manager.MainFrame.Content as OrderObservePage;
    if (sender is Button button && button.DataContext is Data.Users selectedUser)
    {
        var dialog = new Dialogs.UserEdit(selectedUser, LoadUsers);
        dialog.Owner = Application.Current.Windows.OfType<Window>()
        ...
            .FirstOrDefault(w => w.IsActive);
        dialog.WindowStartupLocation = WindowStartupLocation.CenterOwner;
        dialog.ShowDialog();
    }
}
```

Рисунок 45 - Открытие окна редактирования пользователя

После внесения изменений основной список обновляется, что позволяет сразу видеть актуальные данные, а также снова реализован поиск (рисунок 46).

Рисунок 46 - Обновление списка и поиск

```
ссылка: 2
public List<Data.Users> _users = Data.k1stroyDBEntities.GetContext().Users.ToList();
private void Update()
{
    _users = Data.k1stroyDBEntities.GetContext().Users.ToList();

    ...
    if (!string.IsNullOrEmpty(SearchTextBox.Text))
    {
        ...
        _users = Data.k1stroyDBEntities.GetContext().Users
            .Where(p => p.Firstname.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                p.Surname.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                p.Roles.Role.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                p.Login.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                p.Password.ToLower().Contains(SearchTextBox.Text.ToLower()) ||
                p.Patronymic.ToLower().Contains(SearchTextBox.Text.ToLower()))
            .ToList();
        ...
    }
    UsersList.ItemsSource = _users;
}
```

При инициализации страницы (см. Рисунок 47 – Инициализация страницы добавления) происходит вызов метода `InitializeComponent()`, а также передаётся делегат `_Update`, необходимый для последующего обновления списка товаров после добавления (рисунок 47).

```
public AddPage(Action Update)
{
    InitializeComponent();
    _Update = Update;

    CategoryCB.ItemsSource = Data.k1stroyDBEntities.GetContext().ProductCategory.ToList();
    TypeCB.ItemsSource = Data.k1stroyDBEntities.GetContext().ProductType.ToList();
    ManufacturerComboBox.ItemsSource = Data.k1stroyDBEntities.GetContext().Manufacturers.ToList();

    BitmapImage bitmap = new BitmapImage(new Uri("/Res/imagenull.png", UriKind.Relative));
    ProductPhotoSelector.Source = bitmap;
}
```

Рисунок 47 - Инициализация страницы добавления

Далее загружаются данные из базы для выпадающих списков (категории, типы, производители), а в `ProductPhotoSelector` загружается изображение по умолчанию.

Перед сохранением выполняется валидация всех полей с помощью `StringBuilder`, где при наличии пустых полей формируется список ошибок и выводится сообщение пользователю (рисунок 48).

```

private void SaveButton_Click(object sender, RoutedEventArgs e)
{
    StringBuilder err = new StringBuilder();
    if (String.IsNullOrEmpty(NameTB.Text))
    {
        err.AppendLine("Введите Наименование!");
    }
    if (String.IsNullOrEmpty(ArticleTB.Text))
    {
        err.AppendLine("Введите Артикул!");
    }
    if (String.IsNullOrEmpty(PriceTB.Text))
    {
        err.AppendLine("Введите Стоимость!");
    }
    if (String.IsNullOrEmpty(InStockTB.Text))
    {
        err.AppendLine("Введите Количество продукта!");
    }
    if (String.IsNullOrEmpty(ManufacturerComboBox.Text))
    {
        err.AppendLine("Выберите Производителя!");
    }
    if (String.IsNullOrEmpty(CategoryCB.Text))
    {
        err.AppendLine("Выберите Категорию!");
    }
    if (String.IsNullOrEmpty(TypeCB.Text))
    {
        err.AppendLine("Выберите Тип продукта!");
    }
    if (err.Length > 0)
    {
        MessageBox.Show(err.ToString(), "Внимание!", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }
}

```

Рисунок 48 - Валидация полей при добавлении товара

Если все поля заполнены корректно, создаётся новый объект Products, данные которого берутся из элементов интерфейса. Далее он добавляется в базу данных через контекст и сохраняется (рисунок 49).

```

var newProduct = new Data.Products
{
    ProductName = NameTB.Text,
    ManufacturerID = ((Data.Manufacturers)ManufacturerComboBox.SelectedItem).ID,
    CategoryID = ((Data.ProductCategory)CategoryCB.SelectedItem).ID,
   TypeID = ((Data.ProductType)TypeCB.SelectedItem).ID,
    Article = ArticleTB.Text,
    Price = Decimal.Parse(PriceTB.Text),
    InStock = Int32.Parse(InStockTB.Text),
    Description = DescTB.Text,
    Characteristics = CharTB.Text,
    ProductPhotoName = Path,
    Photo = photoBytes
};

var context = Data.k1stroyDBEntities.GetContext();
context.Products.Add(newProduct);
context.SaveChanges();

```

Рисунок 49 - Добавление товара в базу данных

## ГЛАВА 3. ОЦЕНКА ЭКОНОМИЧЕСКОЙ ЦЕЛЕСООБРАЗНОСТИ РАЗРАБОТКИ СИСТЕМЫ УЧЁТА

### 3.1 Цель и задачи экономического раздела

В настоящем разделе выпускной квалификационной работы производится расчёт затрат на Разработку и внедрение системы учёта для автоматизации управления процессами в строительном гипермаркете ООО «К1-Строй».

Целью данного раздела является расчёт:

- Себестоимости проекта;
- Экономической части проекта.

Для подсчёта себестоимости проекта и экономической эффективности данного программного продукта, нужно знать следующие составляющие: расчёт затрат на энергоресурсы;

- амортизационные отчисления;
- расчёт фонда заработной платы;
- прочие (накладные) расходы.

### 3.1 Исходные данные для расчётов

Таблица 1 - Исходные данные

Показатели	Единицы измерения	Значение
Страховые взносы по единому тарифу	%	30
Заработная плата специалиста	руб./мес	175 000
Фонд рабочего времени в 2025 году при 40-часовой рабочей неделе*	часов	1972
Стоимость компьютера	руб.	90 000
Срок полезного использования компьютера	лет	3
Амортизационные отчисления (33,3% в год)	%	33,3
Потребляемая мощность компьютера (в час)	кВт	0,51
Тариф на электроэнергию	руб./кВт·ч	6,43
Затраты на текущий и профилактический ремонт**	руб.	2 497
Затраты на расходные материалы	руб.	3 000

\* Всего в 2025 году будет 365 дней, из которых 247 рабочих и 118 выходных. Рабочее время при 40-часовой неделе составит 1972 часа.

\*\* Затраты=Стоимость ОС \* Норму амортизации/12 месяцев Для расчёта стоимости проделанной работы требуется узнать:

- среднюю заработную плату программиста на предприятии;
- затраты на оборудование (амортизация);
- затраты на электроэнергию.

Среднее значение заработной платы Fullstack разработчика необходимо вычислить при помощи анализа заработной платы с нескольких предприятий по г. Москве и Московской области:

Заработная плата в размере 175 000 рублей.

Источник: <https://hh.ru/vacancy/94845652>

Заработная плата в размере 250 000 рублей.

Источник: <https://hh.ru/vacancy/93657260>

Заработная плата в размере 150 000 рублей.

Источник: <https://hh.ru/vacancy/93657260>

Заработная плата в размере 80 000 рублей.

Источник: <https://hh.ru/vacancy/99840460>

Заработная плата в размере 220 000 рублей.

Источник: <https://hh.ru/vacancy/98920438>

Высчитываем среднее значение:

$(175\,000 + 250\,000 + 150\,000 + 80\,000 + 220\,000) / 5 = 175\,000$  рублей – средняя заработная плата у Fullstack разработчика<sup>4</sup>.

Вычисляем почасовую оплату. Для этого нужно разделить среднюю заработную плату за год на фонд рабочего времени в 2025 году:

$175\,000 \text{ рублей} \times 12 \text{ месяцев} / 1972 \text{ часов} = 1064 \text{ рубль}$  (среднегодовая почасовая оплата)

---

<sup>4</sup> Fullstack разработчик — это специалист, который владеет как **frontend**- (пользовательская часть), так и **backend**-разработкой (серверная логика и базы данных), способен создавать полнофункциональные веб-приложения от интерфейса до сервера.

На создание предоставленного в дипломном проекте приложения в общем объеме ушло 86 часов.

$1064 * 86 = 91\,504$  рублей оплата всех часов работы.

Общие затраты на оплаты труда составили:

Заработная плата + Страховые взносы во внебюджетные фонды (30%)

$91\,504 + 91\,504 * 30\% = 91\,504 + 27\,451 = 118\,955$  рублей.

Рассчитаем примерные затраты на электроэнергию, узнав тариф на электроэнергию;

Тариф на 1 кВт/ч = 6,43 рублей.

Примерное использование кВт в час составляет 0,51. Следовательно, на электроэнергию будет потрачено:

$0,51 \text{ кВт/ч} * 6,43 \text{ руб.} = 3,28$  рубля (в час).

$3,28 \text{ руб.} * 86 \text{ ч} = 282$  рубля – затраты на электроэнергию.

Таким образом, себестоимость разработки приложения для составила 124 734 рублей.

Таблица 2 – Расчёт себестоимости и цены программного продукта

Показатели	Условное обозначение	Значение
Заработная плата	ЗП	91 504
Страховые взносы	СтрВзн	27 451
Амортизация и ремонт	Аморт	2 497
Затраты на материалы	Мат	3 000
Электроснабжение	Эл	282
Итого прямые затраты	ОбщЗ	124 734
Плановая прибыль (20%)	П	24 947
Цена без НДС	Ц	149 681
НДС (20%)	НДС	29 936



Цена продажи с НДС	РознЦ	179 617
--------------------	-------	---------

### 3.2 Экономический эффект и эффективность

Стоимость программного продукта для конечного пользователя соотнесена с затратами и ожидаемыми выгодами от внедрения.

Внедрение системы учёта в строительный гипермаркет ООО «К1-Строй» обеспечит:

1. Повышение эффективности логистических процессов;
2. Сокращение времени обработки заказов;
3. Повышение скорости исполнения заказов;
4. Улучшение контроля за движением материалов и ресурсов.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана информационная система учёта для автоматизации управления процессами в строительном гипермаркете ООО «К1-Строй». Основной целью проекта являлось создание удобного и функционального программного решения, позволяющего повысить эффективность логистических процессов, ускорить обработку заказов и улучшить внутреннее взаимодействие между сотрудниками.

В процессе реализации проекта был выполнен анализ бизнес-процессов компании, определены ключевые требования к системе, разработано техническое задание и сформирован стек используемых технологий. В качестве инструментов разработки были выбраны язык программирования C#, платформа WPF, а также система управления базами данных Microsoft SQL Server. Такой выбор обеспечил стабильную и производительную работу приложения, удобную поддержку пользовательского интерфейса и надёжную обработку хранимой информации.

В рамках выпускной квалификационной работы была реализована система учёта, включающая модули авторизации пользователей, ведения заказов, работы со складами и учёта товаров. Интерфейс приложения спроектирован с учётом потребностей сотрудников различных ролей — от менеджеров до администраторов. Это позволило упростить рабочие процессы, уменьшить количество рутинных операций и обеспечить доступ к ключевой информации в несколько кликов. Для каждого типа пользователя были реализованы собственные страницы, соответствующие их обязанностям и уровню доступа.

Также в ходе разработки были учтены вопросы безопасности, модульности и масштабируемости системы. Использование архитектурного подхода с единым MainFrame, реализующего навигацию между страницами, позволило централизовать управление интерфейсом и облегчить сопровождение проекта.

Практическая реализация проекта подтвердила его работоспособность и эффективность: приложение успешно функционирует на тестовых данных, корректно обрабатывает заказы, обеспечивает взаимодействие с базой данных и предоставляет возможности для дальнейшего расширения. При необходимости система может быть доработана и интегрирована с другими внешними ресурсами — бухгалтерскими или складскими системами, что делает её гибкой и адаптируемой под реальные условия эксплуатации.

Таким образом, все поставленные цели и задачи были достигнуты. Данная работа позволила применить на практике теоретические знания, полученные в процессе обучения, и приобрести ценный опыт проектирования и разработки полноценного десктопного программного продукта. Разработанная информационная система обладает реальной практической ценностью и может быть рекомендована к внедрению на предприятии ООО «К1-Строй».

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ИНФОРМАЦИИ

### Законодательные и нормативные акты:

1. Федеральный закон "О персональных данных" от 27.07.2006 №152-ФЗ (в ред. от 01.03.2024) [Электронный ресурс]. – Режим доступа: свободный. URL: [https://www.consultant.ru/document/cons\\_doc\\_LAW\\_61801/](https://www.consultant.ru/document/cons_doc_LAW_61801/)
2. Федеральный закон "Об информации, информационных технологиях и о защите информации" от 27.07.2006 №149-ФЗ (в ред. от 15.02.2024) [Электронный ресурс]. – Режим доступа: свободный. URL: [https://www.consultant.ru/document/cons\\_doc\\_LAW\\_61798/](https://www.consultant.ru/document/cons_doc_LAW_61798/)

### Книжные (печатные) издания:

3. Яковлев С. А. Основы программирования на языке C# / С. А. Яковлев. – М. : Юрайт, 2024. – 432 с.
4. Климов С. В. Проектирование интерфейсов в WPF / С. В. Климов. – СПб. : БХВ-Петербург, 2024. – 336 с.
5. Попов А. В. Информационные системы и технологии управления / А. В. Попов. – М. : Инфра-М, 2024. – 410 с.
6. Захарова Н. Н. Разработка приложений на платформе .NET / Н. Н. Захарова. – М. : Эксмо, 2023. – 388 с.
7. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.8 / Д. Рихтер. – М. : Вильямс, 2022. – 912 с.
8. Павлов А. Н. Архитектура клиент-серверных приложений / А. Н. Павлов. – СПб. : Питер, 2023. – 256 с.
9. Мельников И. А. Безопасность информационных систем / И. А. Мельников. – М. : Академия, 2024. – 320 с.
10. Абрамов А. Г. Основы работы с базами данных SQL Server / А. Г. Абрамов. – М. : ДМК Пресс, 2024. – 348 с.

### Интернет-источники:

11. Документация по WPF – Microsoft Docs [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/> (дата обращения: 26.05.2024)
12. Документация по C# – Microsoft Learn [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 28.05.2024)
13. Работа с базами данных SQL Server [Электронный ресурс]. – URL: <https://metanit.com/sharp/adonet/> (дата обращения: 01.06.2024)
14. Стилизация интерфейсов в WPF [Электронный ресурс]. – URL: <https://wpf-tutorial.com/> (дата обращения: 29.05.2024)
15. Принципы разработки пользовательских интерфейсов [Электронный ресурс]. – URL: <https://uxdesign.cc/> (дата обращения: 30.05.2024)
16. Хабр. Разработка приложений на C# и WPF [Электронный ресурс]. – URL: <https://habr.com/ru/hub/csharp/> (дата обращения: 02.06.2024)
17. SQL Server 2022 – официальный сайт Microsoft [Электронный ресурс]. – URL: <https://www.microsoft.com/ru-ru/sql-server> (дата обращения: 03.06.2024)
18. Использование паттерна MVVM в WPF [Электронный ресурс]. – URL: <https://metanit.com/sharp/wpf/> (дата обращения: 02.06.2024)
19. Руководство по работе с LiveCharts [Электронный ресурс]. – URL: <https://lvcharts.net/App/examples/v1/wpf> (дата обращения: 01.06.2024)
20. Платформа Visual Studio 2022 – Microsoft [Электронный ресурс]. – URL: <https://visualstudio.microsoft.com/ru/> (дата обращения: 06.06.2024)