

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИ-
ТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»
Специальность 1-40 05 01-01 Информационные системы и технологии (в проек-
тировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Объектно-ориентированное программирование»

на тему: **«WPF ПРИЛОЖЕНИЕ ДЛЯ УЧЁТА ПОСТУПЛЕНИЯ ТОВАРОВ
НА СКЛАДЫ ПРЕДПРИЯТИЯ»**

Исполнитель: студент гр. ИТП-22
Расшивалов Н.И.

Руководитель: доцент
Курочка К.С.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____
Оценка работы: _____

Подписи членов комиссии
по защите курсового проекта: _____

Гомель 2021

СОДЕРЖАНИЕ

Введение.....	4
1 Особенность программно-технических средств для решения поставленной задачи.....	5
1.1 Особенности среды разработки	5
1.2 Особенности языка программирования <i>c#</i>	6
1.3 Технология <i>windows presentation foundation</i>	8
1.4 Особенности объектно-реляционной СУБД	10
2 Архитектура программного комплекса по учету поступления товаров на склады предприятия.....	12
2.1 Общая схема программного комплекса по учету поступления товаров на склады предприятия.....	12
2.2 Структура программного комплекса по учету .поступления товаров на склады предприятия.....	14
3 Структура и верификация программного обеспечения по учету поступления товаров на склады предприятия	18
3.1 Функционально-модульная структура программы по учету поступления товаров на склады предприятия.....	18
3.2 Модульное тестирование программного комплекса по учету поступления товаров на склады предприятия.....	19
3.3 Функциональное тестирование программного комплекса по учету поступления товаров на склады предприятия.....	20
Заключение	30
Список использованных источников	31
Приложение А Листинг программы.....	32
Приложение Б Функциональная схема приложения.....	97
Приложение В Руководство системного администратора.....	98
Приложение Г Руководство программиста	100
Приложение Д Руководство пользователя	101

ВВЕДЕНИЕ

Для эффективной работы любого из предприятий важен учет и модернизация операций, производимых на складе. Автоматизация складских операций позволяет добиться лучшей производительности. Правильная и отлаженная работа склада определяет, насколько эффективно работает предприятие, как рационально используются все необходимые ресурсы производства. Благодаря этому, готовый продукт становится намного качественнее.

Учёт товаров на складе является одним из главных компонентов в ведении торговой деятельности предприятия и не только. Компании, которые не осуществляют торговой или производственной деятельности, имеют на складе какие-либо активы. Их так же нужно учитывать.

Компьютерный учет имеет свои особенности и преимущества перед бумажным. Автоматизация системы ведения учета, позволяет повысить эффективность работы, достичь возможностей, недоступных ранее при использовании «ручного» метода обработки документации.

Простейшим примером компьютерного учета служит приложение *Microsoft Excel*. Но эффективность его использования крайне мала в наше время, поскольку обработка информации занимает длительное время, отсутствие бизнес логики и целостности данных, сложности в работе нескольких пользователей, невозможность хранить большие объемы информации и ограниченный функционал.

Актуальность проекта заключается в увеличении объема информации поступления товаров на склады и их отгрузке, быстрый и точный подсчет данных.

Цель курсового проекта – создание программного комплекса для упрощения учета поступления товаров на склады предприятия.

Задачи курсового проекта заключаются в следующем – создание программного комплекса на языке программирования *C#* и системы управления базами данных (СУБД) *PostgreSQL* для учета поступления товаров на склады предприятия, а также написание блок-схем и тестов для этого комплекса.

1 ОБЗОР ПРОГРАММНО-ТЕХНИЧЕСКИХ СРЕДСТВ ДЛЯ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Особенности среды разработки

Visual Studio 2019 – это чистый код, продуктивность и командная совместная работа. VS 2019 ускоряет разработку программного обеспечения, включая автоматический ввод и завершение, рефакторинг и улучшения инструментов.

Visual Studio 2019 использует искусственный интеллект, чтобы предсказать, какой код будет написан дальше. Экономит время на копирование и вставку кода.

IntelliCode (вспомогательное средство, используемое во время проверки кода для выделения изменений, требующих внимания) не только повышает интеллект *IDE*, предлагая и рекомендуя фрагменты кода, но также выполняет поиск на *GitHub* тысяч проектов с открытым исходным кодом. В сочетании с контекстом кода список завершения адаптируется к распространенным практикам.

IntelliCode не ограничивается завершением оператора. Справка по подписи также рекомендует наиболее вероятную перегрузку для контекста.

Бывают случаи, когда в определенной точке появляется нужда в помощи на месте или коллег, которые работают из другого места. Обычно используется *Webex* и подобные инструменты. Но теперь *Visual Studio 2019* поставляется с кнопкой «ПОДЕЛИТЬСЯ». Нажимая на нее, создается ссылка, которой можно поделиться с другими. Другой человек просто должен установить любую версию *Visual Studio* или *Visual Studio Code*. В отличие от *Webex*, пользователь и его коллега могут одновременно работать над кодом, а другой человек также может отлаживать код. Это сэкономит много времени.

CodeLens (информационные индикаторы, которые отображаются в коде и могут в лаконичной форме показать прямо в тексте программы важную информацию, которая затем пригодится при работе с кодом приложения) помогает легко найти важную информацию о коде, например, какие изменения были внесены в код, влияние этих изменений и был ли метод модульно протестирован.

Visual Studio 2019 очистит код в соответствии с определенными правилами одним щелчком мыши. Можно установить эти правила в соответствии со стандартами кодирования, что может помочь сэкономить время.

Предоставляет такие функции, как преобразование операторов *Foreach* в *LINQ* и удаление неиспользуемых функций одним щелчком мыши.

Новая панель поиска экономит много времени. Можно легко создавать новые проекты, добавлять ссылки на проекты и многое другое с помощью этого текстового поля поиска.

Если есть огромный проект с множеством подпроектов, с помощью фильтра решения можно определить, какие проекты решения загрузить. Это помогает сэкономить время на открытии ненужных проектов.

Можно легко извлекать проекты из *GIT*.

Visual Studio обеспечивает быструю отладку. Можно искать различные ключевые элементы во время отладки приложения.

Помимо вышеперечисленных функций, есть изменения в пользовательском интерфейсе и многие другие функции.

1.2 Особенности языка программирования C#

Язык C# – это современный объектно-ориентированный язык программирования общего назначения, разработанный *Microsoft* по инициативе Андреса Хейлсберга.

Язык программирования C# очень прост для изучения. Он полностью основан на языках C и C++.

Однако синтаксис языка очень выразителен, но при этом он прост и легок в освоении.

C# упрощает многие из сложностей C++ и предоставляет мощные функции, такие как типы значений, допускающие значение *null*, делегаты, перечисления, лямбда-выражения и прямой доступ к памяти, которых нет, например, в *Java*.

Раньше *Microsoft* создавала этот язык только для приложений *Windows*, но после этого, этот язык начал использоваться для консоли, *Android* и *iOS*, кроме того, C# стал использоваться с программным обеспечением для машинного обучения [1, с.51].

Основное преимущество языка в том, что приложение, написанное на C#, может быть развернуто в любой операционной системе, например, *Android* или *iOS*, *Windows* или облачной платформе. Это называется кроссплатформенностью.

В языке C# есть много полезных функций, которые делают его более удобным и уникальным по сравнению с другими языками.

C# – простой язык. Это дает структурированный подход к разбивке проблемы на части. Кроме того, он имеет богатый набор библиотечных функций и типов данных.

Язык C# – это объектно-ориентированный язык программирования. Это так же упрощает разработку и сопровождение по сравнению с процедурно-ориентированным языком программирования [2, с.83].

Кроме того, программирование на C# поддерживает инкапсуляцию данных, наследование, полиморфизм, интерфейсы.

C# – это безопасный код, который может обращаться только к области памяти и имеет разрешение на выполнение. Следовательно, это повышает безопасность программы.

В языке невозможно выполнять небезопасные преобразования, такие как преобразование *double* в логическое значение. Его типы значений (примитивные типы) инициализируются нулями, а ссылочные типы (объекты и классы) автоматически инициализируются компилятором значением *null*.

Совместимость – это процесс, который позволяет программам на *C#* делать практически все, что может делать собственное приложение *C++*. Языковая совместимость – это способность кода взаимодействовать с кодом, написанным с использованием другого языка программирования. Это может помочь максимально увеличить повторное использование кода и повысить эффективность процесса разработки.

Язык *C#* обеспечивает поддержку использования *COM*-объектов, независимо от того, на каком языке они были созданы. Однако он также поддерживает специальную функцию, которая позволяет программе вызывать любой собственный *API*.

C# – это компьютеризированный масштабируемый язык программирования с возможностью обновления.

В программировании на *C#* установлена очень эффективная система, которая автоматически собирает и стирает мусор, присутствующий в системе. Язык очень эффективен в управлении системой, потому что он не создает беспорядка в системе, и система не зависает во время выполнения.

Приложение, написанное на *.NET*, будет иметь лучшую интеграцию и интерпретируемость по сравнению с другими *NET*-технологиями. Программирование на *C#* выполняется в среде *CLR*, что упрощает интеграцию с компонентами, написанными на других языках [3, с.321].

Стоимость обслуживания меньше и безопаснее в использовании по сравнению с другими языками. На языке *C#* можно разрабатывать собственные приложения для *iOS*, *Android* и *Windows Phone* с помощью платформы *Xamarin*.

Вот некоторые преимущества языка *C#* над языком *Java*, которые делают его более полезным.

Поскольку язык *C#* является языком *.NET*, он поддерживает взаимодействие языков. Он может получить доступ к коду, написанному на любом языке, совместимом с *.NET*, а также наследовать классы, написанные на этих языках. Но на языке *Java* это невозможно.

Переносимый исполняемый файл языка *C#* может содержать любое количество классов, в то время как файл *.class* на языке *Java* содержит только один класс. Язык содержит такие функции, как «Свойства» и «Индексаторы», поддерживает перегрузку оператора, структуры и директивы препроцессора, тогда как язык *Java* не имеет их.

В программировании на C# пространства имен не связаны с каталогами, в то время как пакеты в программировании на Java напрямую связаны с именами каталогов.

Код C# при компиляции генерирует файл «.exe» или «.dll», который также называется переносимым исполняемым файлом. Эти файлы содержат код *Microsoft Intermediate Language (MSIL)*. В то время как код Java при компиляции генерирует файл .class, который содержит байт-код.

В программировании на C# методы не являются виртуальными по умолчанию, тогда как в программировании на Java методы являются виртуальными по умолчанию, что снижает производительность.

В языке C# особое внимание уделяется управлению версиями для обеспечения совместимости программ и библиотек при их изменении. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы *virtual* и *override*, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

1.3 Технология *Windows Presentation Foundation*

Технология *WPF (Windows Presentation Foundation)* является частью экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе *WinForms* за отрисовку элементов управления и графики отвечали такие части ОС Windows, как *User32* и *GDI+*, то приложения *WPF* основаны на *DirectX*. В этом состоит ключевая особенность рендеринга графики в *WPF*: используя *WPF*, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложится на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Единственное наиболее важное различие между *WinForms* и *WPF* заключается в том, что *WinForms* – это просто слой поверх стандартных элементов управления Windows (например, текстовое поле), *WPF* создается с нуля и почти во всех ситуациях не полагается на стандартные элементы управления Windows.

Еще одной из важных особенностей является использование языка декларативной разметки интерфейса *XAML*, основанного на *XML*: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C# и VB.NET, либо совмещать и то, и другое.

Преимущества *WPF*:

- использование традиционных языков .NET-платформы-C# и VB.NET для создания логики приложения;

- возможность декларативного определения графического интерфейса с помощью специального языка разметки *XAML*, основанном на *XML* и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать *XAML* и *C#/VB.NET*;
- независимость от разрешения экрана: поскольку в *WPF* все элементы измеряются в независимых от устройства единицах, приложения на *WPF* легко масштабируются под разные экраны с разным разрешением;
- новые возможности, которых сложно было достичь в *WinForms*, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и другие;
- хорошее взаимодействие с *WinForms*, благодаря чему, например, в приложениях *WPF* можно использовать традиционные элементы управления из *WinForms*;
- богатые возможности по созданию различных приложений: это и мультимедиа, и двумерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое;
- аппаратное ускорение графики – вне зависимости от того, работаете ли вы с *2D* или *3D*, графикой или текстом, все компоненты приложения транслируются в объекты, понятные *Direct3D*, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной;
- создание приложений под множество *ОС* семейства *Windows* – от *Windows XP* до *Windows 10*.

Схематически архитектуру *WPF* представлена на рисунке 1.1.

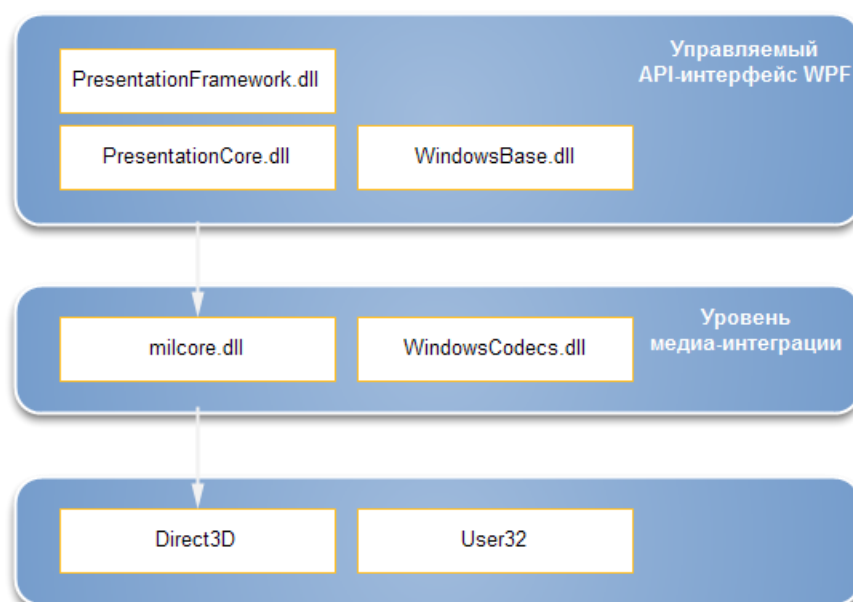


Рисунок 1.1 – Архитектура *WPF*

В тоже время *WPF* имеет определенные ограничения. Несмотря на поддержку трехмерной визуализации, для создания приложений с большим количеством трехмерных изображений, прежде всего игр, лучше использовать другие средства – *DirectX* или специальные фреймворки, такие как *Monogame* или *Unity*. Также стоит учитывать, что по сравнению с приложениями на *Windows Forms* объем программ на *WPF* и потребление ими памяти в процессе работы в среднем несколько выше [4]. Но это с лихвой компенсируется более широкими графическими возможностями и повышенной производительностью при отрисовке графики.

1.4 Особенности объектно-реляционной СУБД

PostgreSQL – это популярная свободная объектно-реляционная система управления базами данных (БД). *PostgreSQL* базируется на языке *SQL* и поддерживает многочисленные возможности.

Преимущества *PostgreSQL*:

- поддержка БД неограниченного размера;
- мощные и надёжные механизмы транзакций и репликации;
- расширяемая система встроенных языков программирования и поддержка загрузки *C*-совместимых модулей;
- наследование;
- легкая расширяемость.

Текущие ограничения *PostgreSQL*:

- нет ограничений на максимальный размер базы данных;
- нет ограничений на количество записей в таблице;
- нет ограничений на количество индексов в таблице;
- максимальный размер таблицы – 32 терабайта;
- максимальный размер записи – один терабайт;
- максимальный размер поля – один гигабайт;
- максимум полей в записи 250 – 1600 (в зависимости от типов полей).

Функции в *PostgreSQL* являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом *SQL*, реализация дополнительной логики, например, условных переходов и циклов, выходит за рамки собственно *SQL* и требует использования некоторых языковых расширений. Функции могут писаться с использованием различных языков программирования. *PostgreSQL* допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса. Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя. Иногда функции отождествляются с хранимыми процедурами, однако между этими понятиями есть различие [5, с.56].

Триггеры в *PostgreSQL* определяются как функции, инициируемые *DML*-операциями. Например, операция *INSERT* может запускать триггер, проверяющий добавленную запись на соответствия определённым условиям. При написании функций для триггеров могут использоваться различные языки программирования. Триггеры ассоциируются с таблицами. Множественные триггеры выполняются в алфавитном порядке.

Механизм правил в *PostgreSQL* представляет собой механизм создания пользовательских обработчиков не только *DML*-операций, но и операции выборки. Основное отличие от механизма триггеров заключается в том, что правила срабатывают на этапе разбора запроса, до выбора оптимального плана выполнения и самого процесса выполнения. Правила позволяют переопределять поведение системы при выполнении *SQL*-операции к таблице [6].

Индексы в *PostgreSQL* следующих типов: *B*-дерево, хэш, *R*-дерево, *GiST*, *GIN*. При необходимости можно создавать новые типы индексов, хотя это далеко не тривиальный процесс.

Многоверсионность поддерживается в *PostgreSQL* – возможна одновременная модификация БД несколькими пользователями с помощью механизма *Multiversion Concurrency Control (MVCC)*. Благодаря этому соблюдаются требования *ACID*, и практически отпадает нужда в блокировках чтения.

Расширение *PostgreSQL* для собственных нужд возможно практически в любом аспекте. Есть возможность добавлять собственные преобразования типов, типы данных, домены (пользовательские типы с изначально наложенными ограничениями), функции (включая агрегатные), индексы, операторы (включая переопределение уже существующих) и процедурные языки.

Наследование в *PostgreSQL* реализовано на уровне таблиц. Таблицы могут наследовать характеристики и наборы полей от других таблиц (родительских). При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице.

В разработке простых сайтов *PostgreSQL* используется несколько реже, чем *MySQL* / *MariaDB*, но всё же эта пара с заметным отрывом опережает по частоте использования остальные системы управления базами данных. При этом в разработке сложных сайтов и веб-приложений *PostgreSQL* опережает по использованию *MySQL* и *MariaDB*. Большинство фреймворков (например, *Ruby on Rails*, *Yii*, *Symfony*, *Django*) поддерживают использование *PostgreSQL* в разработке.

У *PostgreSQL* множество возможностей. Созданный с использованием объектно-реляционной модели, он поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных. Он обеспечивает расширенную ёмкость данных и заслужил доверие бережным отношением к целостности данных.

2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА ПО УЧЕТУ ПОСТУПЛЕНИЯ ТОВАРОВ НА СКЛАДЫ ПРЕДПРИЯТИЯ

2.1 Общая схема программного комплекса по учету поступления товаров на склады предприятия

Для работы с программным обеспечением была разработана база данных в СУБД *PostgreSQL*. В качестве сущностей БД вступают:

- клиенты;
- расходные накладные;
- продукты;
- приходные накладные;
- роли;
- пользователи;
- склады.

Чтобы избежать связи «многие-ко-многим» были спроектированы дополнительные сущности:

- продукты на складе;
- позиции расходной;
- позиции накладной.

Благодаря этим таблицам, можно просматривать детализацию накладных, и видеть информацию о товарах, хранящихся на разных складах.

Данные каждой таблицы соответствуют требованиям нормализации реляционных баз данных. Таблицы находятся в первой нормальной форме, так как все их поля являются простыми, в таблице нет повторяющихся полей и каждый столбец хранит одно единственное значение. Каждый столбец не является ни списком, ни множеством значений.

Таблицы находятся во второй нормальной форме, так как они находятся в первой нормальной форме, имеют простые первичные ключи и каждое не ключевое поле функционально зависит от первичного ключа.

Таблицы находятся в третьей нормальной форме, т.к. они находятся во второй нормальной форме и все не ключевые поля являются взаимно-независимыми. Каждая таблица имеет идентификационное поле (первичный ключ). Данное поле имеет тип данных *integer* и является автоинкрементным. Все первичные ключи в сущностях спроектированной структуры являются простыми и однозначно идентифицируют запись.

Диаграмма базы данных содержит в себе описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных. Так же на диаграмме баз данных изображены связи между таблицами.

Была составлена диаграмма базы данных по учету поступления товаров на склады предприятия, включая в себя все сущности и связи между ними.

2.2 Структура программного комплекса по учету поступления товаров на склады предприятия

Логическая структура реляционной базы данных – это адекватное отображение полученной информационно-реляционной модели предметной области.

Разрабатываемый софт должен выполнять следующие функции:

- возможность добавлять новые товары и осуществления их отгрузки, а также возможность изменения количества товаров на складе;
- формирование отчетов в формате *XML* и сохранение на диск;
- защиту данных от несанкционированного доступа, с помощью аутентификации пользователей в системе;
- регистрация новых пользователей и изменение уже существующих.

Для работы с данными были выделены семь основных и три вспомогательных сущностей:

Roles, Users, Stocks, Customers, Expenditure_Invoices, Receipt_Invoices, Products, Products_in_Stok, Receipt_Positions, Expenditure_Positions.

Сущность *Roles* содержит информацию о ролях.

Сущность *Users* содержит информацию о пользователях.

Сущность *Stocks* содержит информацию о складах.

Сущность *Customers* содержит информацию о клиентах.

Сущность *Expenditure_Invoices* содержит информацию о расходных накладных.

Сущность *Receipt_Invoices* содержит информацию о приходных накладных.

Сущность *Products* содержит информацию о продуктах.

Сущность *Products_in_Stok* содержит информацию о продуктах, хранящихся на складе.

Сущность *Receipt_Positions* содержит детализацию приходных накладных.

Сущность *Expenditure_Positions* содержит детализацию расходных накладных.

Сущность – это объект, находящийся в базе данных, в котором хранится определенная информация. Сущность может представлять собой нечто вещественное (предмет) или абстрактное (операция). В физической модели сущность называется таблицей.

Для получения доступа к объектам базы данных используется класс *User*. Благодаря этому классу происходит аутентификация пользователей и отображение предусмотренных согласно роли, элементов интерфейса. Каждый пользователь имеет пароль и логин, только администратор может добавлять и редактировать пользователей.

Для получения соединения с базой данных используется паттерн «*Singleton*». *Singleton* – порождающий паттерн проектирования который гарантирует, что класс имеет только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру. Одиночка позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан.

При планировании информационной системы проявляются некоторые слои, отвечающие за взаимодействие различных модулей системы. Соединение с базой данных является одной из важнейших компонентов приложения. Всегда выделяется часть кода, модуль, отвечающий за передачу запросов в БД и обработку полученных от неё ответов. В общем случае, определение *Data Access Object* описывает модуль как прослойку между БД и системой. *DAO* абстрагирует сущности системы и делает их отображение на БД, определяет общие методы использования соединения, его получение, закрытие и (или) возвращение *Connection Pool* [7].

Интерфейс *IService* является главным компонентом иерархии *DAO* в разработанном ПО для описания общих методов, которые будут использоваться при взаимодействии с базой данных. В интерфейсе *IService* это методы поиска, удаление по ключу, обновление и т.д. Диаграмма этих классов представлена на рисунке 2.2.

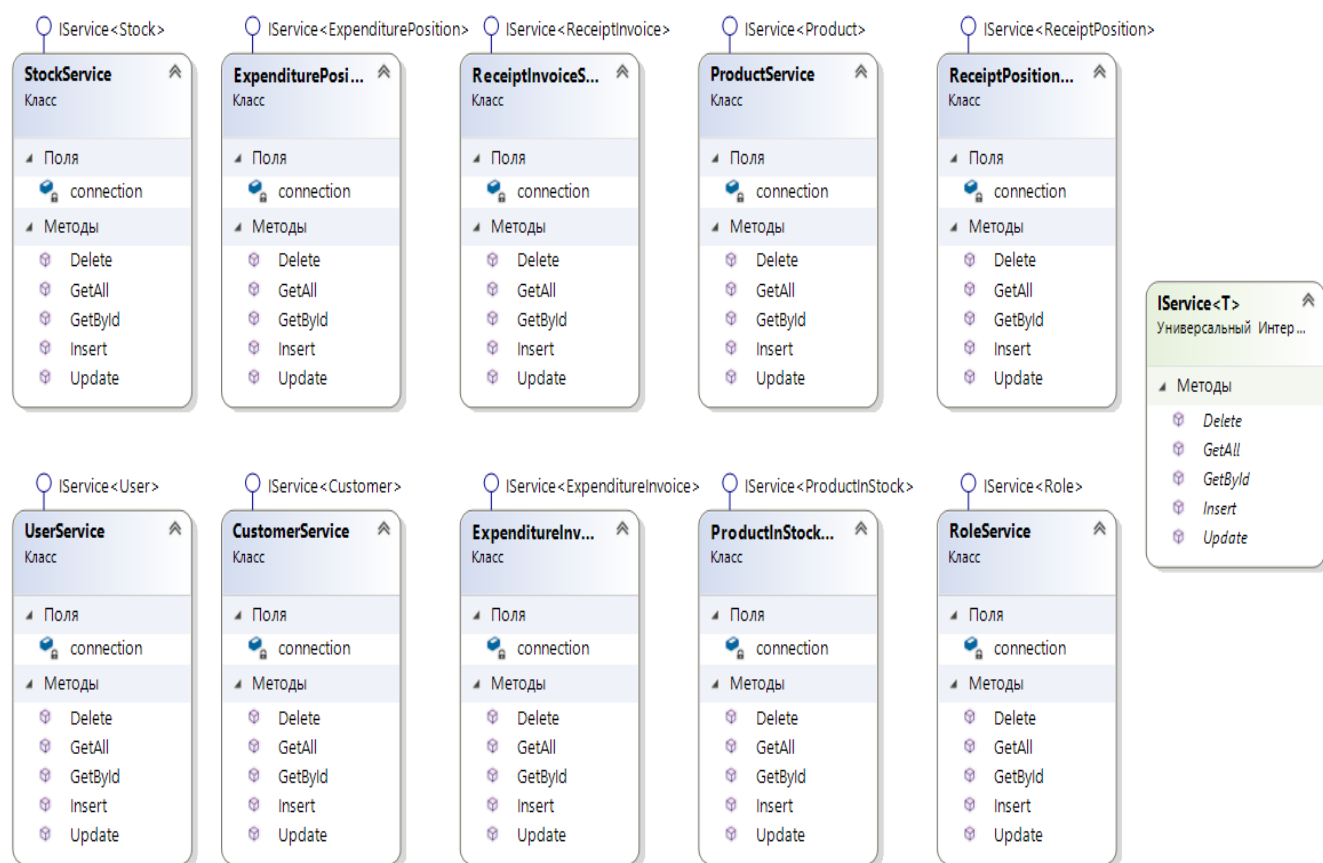


Рисунок 2.2 Диаграмма классов, наследуемых от *IService*

Был разработан интерфейс *IReportService*, отвечающий за получение данных для формирования отчетов. В нем содержится четыре метода для формирования различных видов отчетов. Далее был создан класс *Reports* реализующий этот интерфейс.

Отчеты делятся на четыре типа:

- отчет, полученный по конкретному складу;
- отчет, полученный по всем складам;
- отчет о наиболее доходных товарах;
- отчет о полученной прибыли по складам.

В этом классе описан каждый метод для получения конкретного вида отчета.

Во всех методах формируется строка запроса, которая отправляется в базу данных. После БД отправляет таблицу готовых данных, необходимых для формирования отчета.

Интерфейс и реализующий его класс показаны на рисунке 2.3.

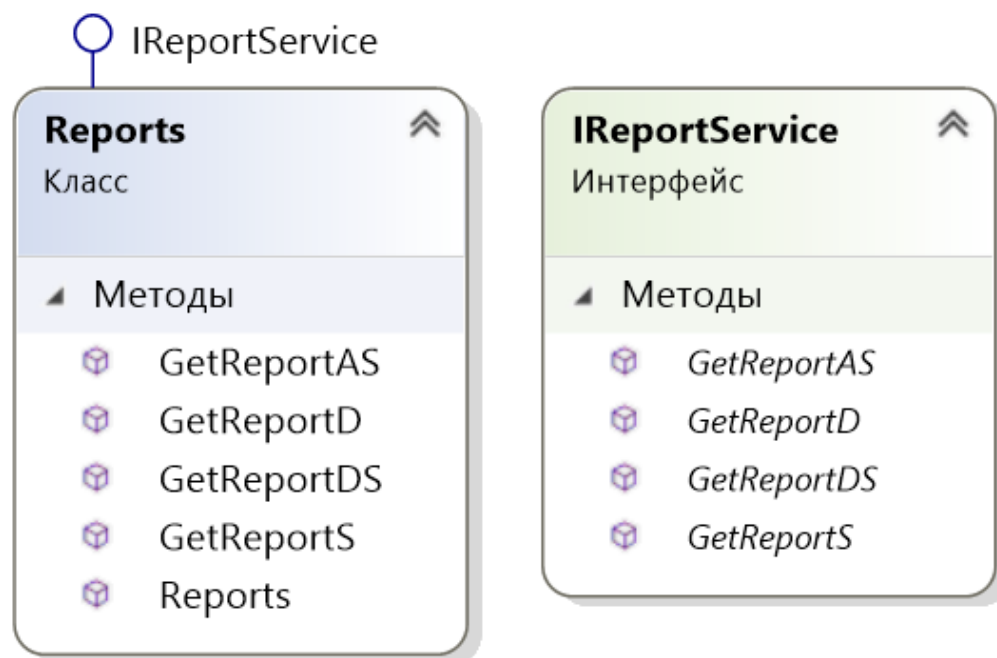


Рисунок 2.3 – Интерфейс и реализующий его класс

Все запросы, отправляемые в базу данных, являются параметрическими. Это служит для того, чтобы избежать попытку внедрения *sql* инъекции и сохранения целостности данных.

Была создана схема взаимодействия классов приложения.

На схеме отображены все классы и как они взаимодействуют друг с другом.

Схема взаимодействия классов изображена на рисунке 2.4

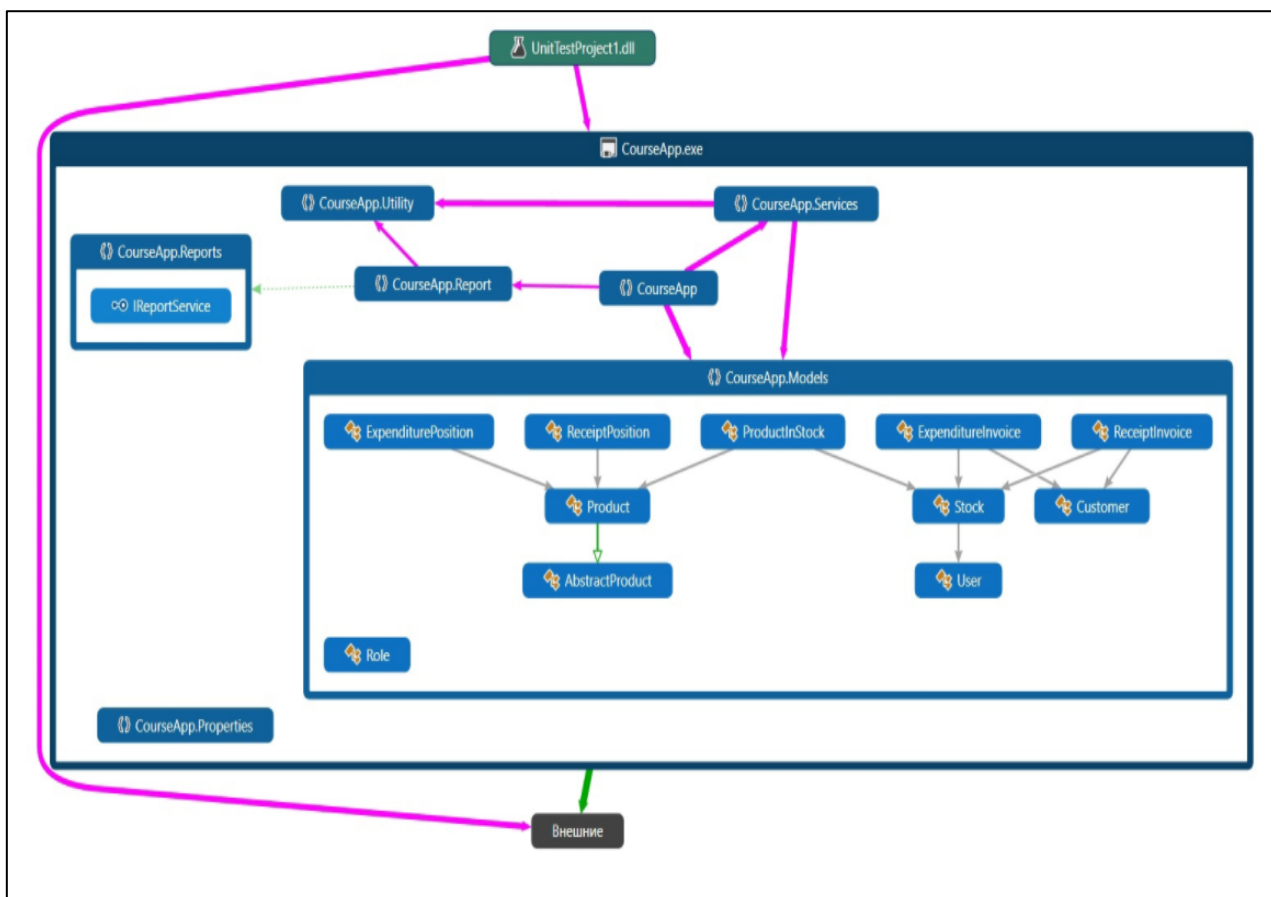


Рисунок 2.4 – Схема взаимодействия классов

На схеме видно, что сущность *ProductInStock*, является связующим звеном между сущностями *Stock* и *Product*. Эта сущность является необходимым компонентом в архитектуре приложения, так как благодаря ей можно посчитать общее количество продуктов на складе и получить необходимую информацию об этих продуктах.

Далее рассматриваются сущности *ExpenditurePosition* и *ReceiptPosition*. Они являются посредниками между накладными и продуктами.

Эти сущности были выбраны для того, чтобы избавиться от связи «многие-ко-многим» из таблиц *ReceiptInvoice* и *ExpenditureInvoice*, которые связаны с таблицей *Products*.

Это позволило формировать позиции приходной и расходной накладной из многих продуктов. Если не создавать посредников, то одна накладная будет содержать один продукт, что является бессмысленным.

3 СТРУКТУРА И ВЕРИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПО УЧЕТУ ПОСТУПЛЕНИЯ ТОВАРОВ НА СКЛАДЫ ПРЕДПРИЯТИЯ

3.1 Функционально-модульная структура программы по учету поступления товаров на склады предприятия

Реализация программного обеспечения производилось на языке программирования C#.

Анализируя поставленную задачу в папке *Models* были созданы 10 моделей:

- *Customer*, описывающая клиента;
- *ExpenditureInvoice*, описывающая расходную накладную;
- *ReceiptInvoice*, описывающая приходную накладную;
- *Role*, описывающая роль пользователя;
- *Stock*, описывающая склад;
- *User*, описывающая пользователя;
- *PdocuctsInStock*, описывающая продукты на складе;
- *ReceiptPositions*, описывающая позиции приходной накладной;
- *ExpenditurePositions*, описывающая позиции расходной накладной;
- *Product*, описывающая продукт.

Также была создана папка *Services*, в которой расположены: один интерфейс и 10 классов, реализующих этот интерфейс. Интерфейс содержит следующие методы:

- *List<T> GetAll()* – для получения всех записей, где *T* – универсальный параметр, вместо него используются модели, описанные выше;
- *bool Update(T entity)* – для обновления записи;
- *bool Delete(T entity)* – для удаления записи;
- *T GetById(int id)* – для получения записи по *Id*;
- *bool Insert(T entity)* – для вставки данных.

Была создана папка *Utility*, в которой расположен один класс *DbConnection*, который содержит следующие поля и методы:

- поле *NpgsqlConnection connection*, которое является строкой подключения к базе данных;
- метод *NpgsqlConnection GetConnection()*, который получает соединение с базой данных;
- метод *CloseConnection()*, который закрывает соединение с базой данных (обязательное приложение А, листинг программного класса *DbConnnection*, страница 43).

Была создана папка *Report*, в которой расположен интерфейс *IReportService* и класс *Reports*.

Интерфейс *IReportService* содержит следующие методы:

- *void GetReportS(int stockId, Combobox reportComboBox)* – для получения информации о конкретном складе, где *stockId* – *Id* склада, *ReportComboBox* – выбранный элемент из *ReportComboBox*;
- *void GetReportAS()* – получение информации обо всех складах;
- *void GetReportDS(string dataFrom, string dataTo)* – получение прибыли по складам за конкретный период, где *dataFrom* – период с, *dataTo* – период по;
- *Void GetReportD()* – получение наиболее доходных товаров.

Класс *Reports* реализует интерфейс *IReportService* (обязательное приложение А, листинг программного класса *Reports*, страница 38).

3.2 Модульное тестирование программного комплекса по учету поступления товаров на склады предприятия

Цель модульного тестирования является изолирование отдельных частей программы чтобы показать, что по отдельности эти части работоспособны. Результат выполнения модульного тестирования представлен на рисунке 3.1.

Тестирование	Длительности	При
✓ CourseAppTests (22)	1 с	
✓ CourseAppTests (22)	1 с	
✓ CustomerServiceTest (5)	1 с	
✓ AGetAllTest	982 мс	
✓ BInsertTest	19 мс	
✓ CUpdateTest	4 мс	
✓ DGetByldTest	2 мс	
✓ EDeleteTest	3 мс	
▷ ✓ ExpenditureInvoiceServiceTest (2)	5 мс	
▷ ✓ ProductServiceTest (5)	14 мс	
▷ ✓ RoleServiceTest (2)	2 мс	
▷ ✓ StockServiceTest (3)	2 мс	
▷ ✓ UserServiceTest (5)	8 мс	

Рисунок 3.1 – Результат *unit*-тестирования ПО

Были протестированы методы для конкретных моделей, описанных в предыдущей подглаве:

- метод *GetAll()*;
- метод *Update(T entity)*;
- метод *GetById(int id)*;
- метод *Insert(T entity)*.

В каждом *unit*-тесте для конкретного класса создавались соответствующие экземпляры этих классов с начальными данными для тестирования.

В ходе проверки всех методов, в БД отправлялись тестовые данные и затем извлекались из нее, возвращая результат. Если результат, полученный в ходе выполнения метода, соответствует начальным данным, то тест считается успешным, если данные различны, то тест провален,

3.3 Функциональное тестирование программного комплекса по учету поступления товаров на склады предприятия

При запуске приложения появляется окно авторизации, в котором необходимо ввести логин и пароль. При неверном вводе данных, программа выдаст сообщение об ошибке с информацией «Логин или пароль введены не верно!».

Сообщение с информацией об ошибке изображено на рисунке 3.2.

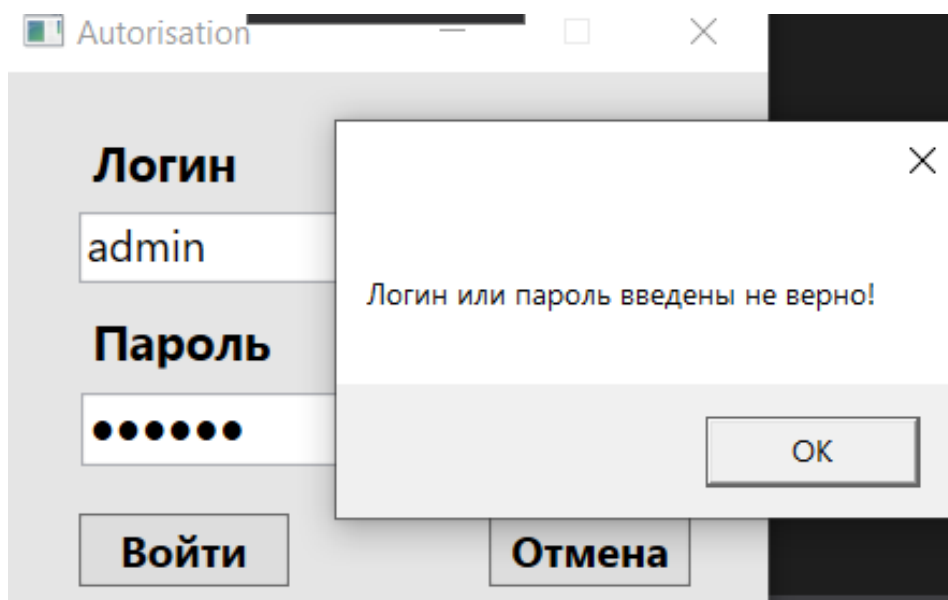


Рисунок 3.2 – Ошибка при введении неправильного логина или пароля

Получение логина и пароля происходит путем ввода данных и отправки их на сервер. Методом *Equals()*, проверяется есть ли в базе такая комбинация логина и пароля, если есть, то открывается главное меню, если нет, то выводит соответствующее сообщение.

При успешном вводе логина и пароля открывается главное меню программы, которое представлено на рисунке 3.3.

Организации	Пользователи	Продукция	Приходные накладные	Товары на складе	Отходные накладные	Приход/Отгрузка	Отчеты	Выход
-------------	--------------	-----------	---------------------	------------------	--------------------	-----------------	--------	-------

ID	Наименование организации	Описание
1	ooo молоко	молочная продукция
4	ООО Савушкин продукт	молочная продукция

Наименование

Описание

Удалить

Очистить

Сохранить

Рисунок 3.3 – Главное меню программы

Функционал меню для каждой роли различается. Администратор имеет полные права доступа и функционал.

Кладовщик может оформлять накладные и просматривать информацию по своему складу, формировать отчет по конкретному складу.

Менеджер может просматривать информацию о складах, накладных и формировать отчеты.

Приложение скрывает ненужные вкладки либо ограничивает функционал для ролей, не имеющих право пользования этими вкладками.

При нажатии на вкладку «Организации» появляется информация, содержащая данные об организациях.

Вкладка «Организации» изображена на рисунке 3.4.

Организации	Пользователи	Продукция	Приходные накладные	Товары на складе	Отходные накладные	Приход/Отгрузка	Отчеты	Выход
-------------	--------------	-----------	---------------------	------------------	--------------------	-----------------	--------	-------

ID	Наименование организации	Описание
1	ooo молоко	молочная продукция
4	ООО Савушкин продукт	молочная продукция

Наименование

Описание

Удалить

Очистить

Сохранить

Рисунок 3.4 – Вкладка «Организации»

На вкладке «Организации» администратор может добавлять и удалять организации.

Информация об организации содержит следующие поля:

- наименование организации;
- описание организации.

Чтобы удалить организацию, нужно выбрать ее в таблице и нажать кнопку удалить.

При нажатии на вкладку «Пользователи» появляется информация, содержащая данные о пользователях.

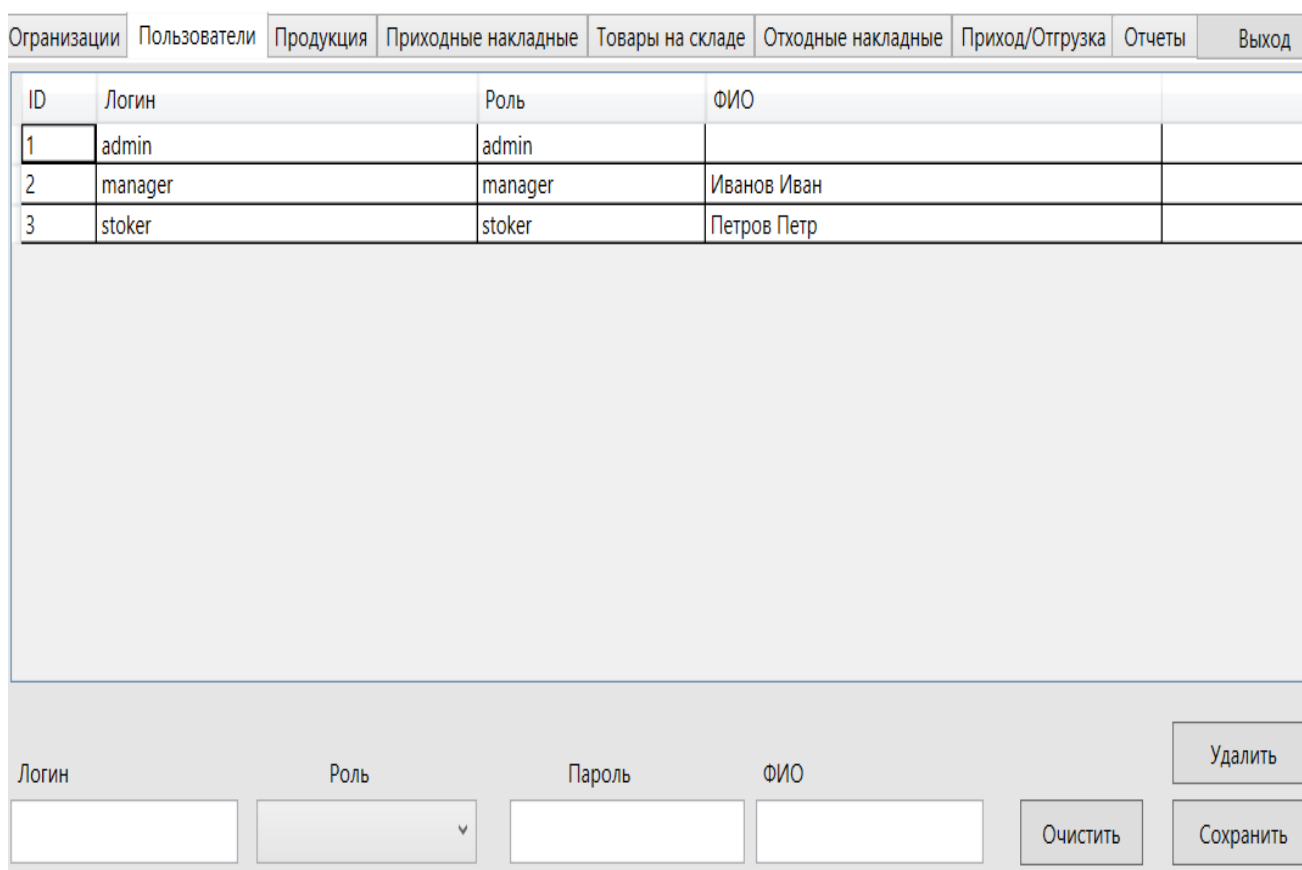
На вкладке «Пользователи» администратор может добавлять и удалять пользователей.

Чтобы добавить пользователя, необходимо заполнить соответствующие поля и нажать кнопку «Сохранить».

Чтобы удалить пользователя, необходимо выбрать его из списка и нажать кнопку «Удалить».

Чтобы изменить информацию о пользователе, необходимо выбрать его из списка доступных, заполнить соответствующие поля и нажать кнопку «Сохранить».

Вкладка «Пользователи» изображена на рисунке 3.5.



ID	Логин	Роль	ФИО
1	admin	admin	
2	manager	manager	Иванов Иван
3	stoker	stoker	Петров Петр

Логин	Роль	Пароль	ФИО	Удалить
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="Очистить"/>
				<input type="button" value="Сохранить"/>

Рисунок 3.5 – Вкладка «Пользователи»

Главного администратора нельзя удалить, нельзя изменить роль, но при необходимости можно изменить пароль и добавить ФИО.

У главного администратора *Id* всегда равен одному.

При нажатии на вкладку «Продукция» появляется информация, содержащая данные о продуктах.

На вкладке «Продукция» администратор может добавлять различную продукцию.

Вкладка «Продукция» изображена на рисунке 3.6.

ID	Наименование	Цена
1	молоко	12
2	Сметана	15
3	сыр	15
4	ряженка	20
6	кефир	12.5
7	сливки	15.5

Наименование	Цена	Очистить	Удалить	Сохранить
<input type="text"/>	<input type="text"/>	<input type="button" value="Очистить"/>	<input type="button" value="Удалить"/>	<input type="button" value="Сохранить"/>

Рисунок 3.6 – Вкладка «Продукция»

При нажатии на вкладку «Приходные накладные» появляется информация, содержащая данные о приходных накладных.

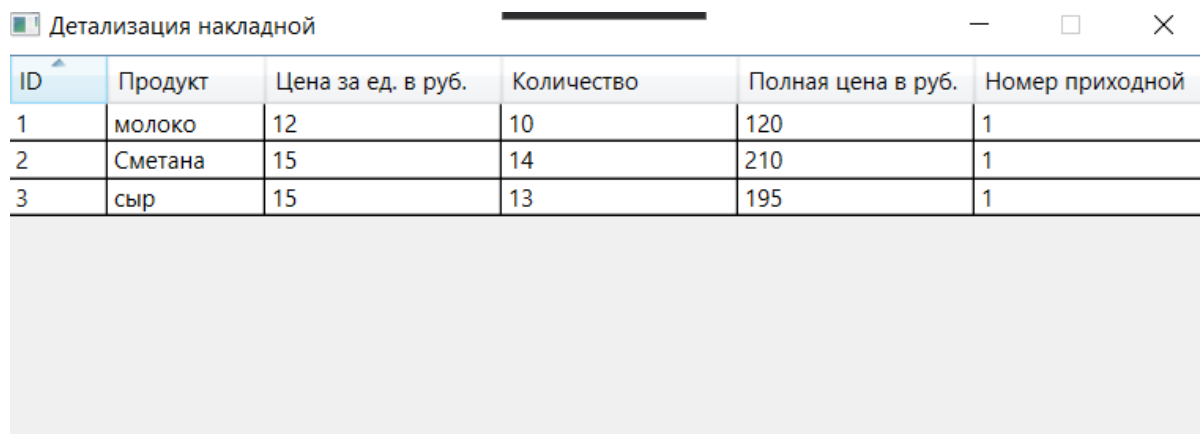
Вкладка «Приходные накладные» изображена на рисунке 3.7.

ID	Дата	Поставщик	Склад
38	2021-05-09	ooo молоко	первый склад
39	2021-05-06	ooo молоко	первый склад
40	2021-04-30	ooo молоко	первый склад
41	2021-05-01	ooo молоко	первый склад
42	2021-05-06	ooo молоко	первый склад
43	2021-05-02	ooo молоко	первый склад
44	2021-05-13	ooo молоко	первый склад
45	2021-05-01	ooo молоко	первый склад
46	2021-05-22	ooo молоко	первый склад

Рисунок 3.7 – Вкладка «Приходные накладные»

На вкладке «Приходные накладные» содержится список всех приходных накладных. Чтобы просмотреть детализацию приходной накладной, необходимо выбрать нужную накладную из таблицы и нажать на нее два раза. После нажатия появляется окно, содержащее детализацию накладной.

Окно содержащее детализацию приходной накладной изображено на рисунке 3.8.



ID	Продукт	Цена за ед. в руб.	Количество	Полная цена в руб.	Номер приходной
1	молоко	12	10	120	1
2	Сметана	15	14	210	1
3	сыр	15	13	195	1

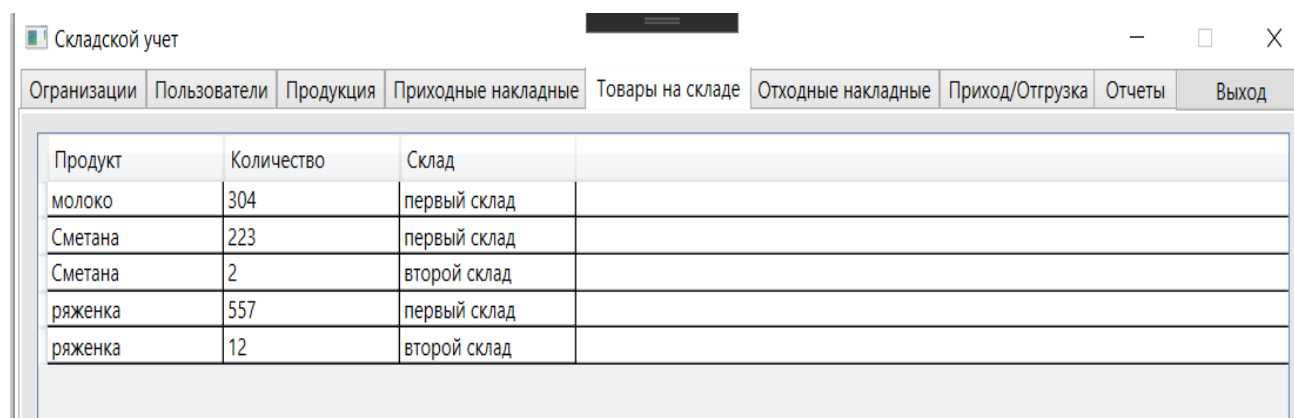
Рисунок 3.8 – Вкладка «Детализация приходной накладной»

Детализация накладной содержит следующие поля:

- *Id* позиции;
- название продукта;
- цену за единицу продукта;
- количество продукта;
- полную цену всего количества продуктов одного названия;
- номер приходной накладной.

При выборе вкладки «Товары на складе», появляется информация, содержащая сведения о имеющихся товарах на складе.

Вкладка «Товары на складе» изображена на рисунке 3.9.



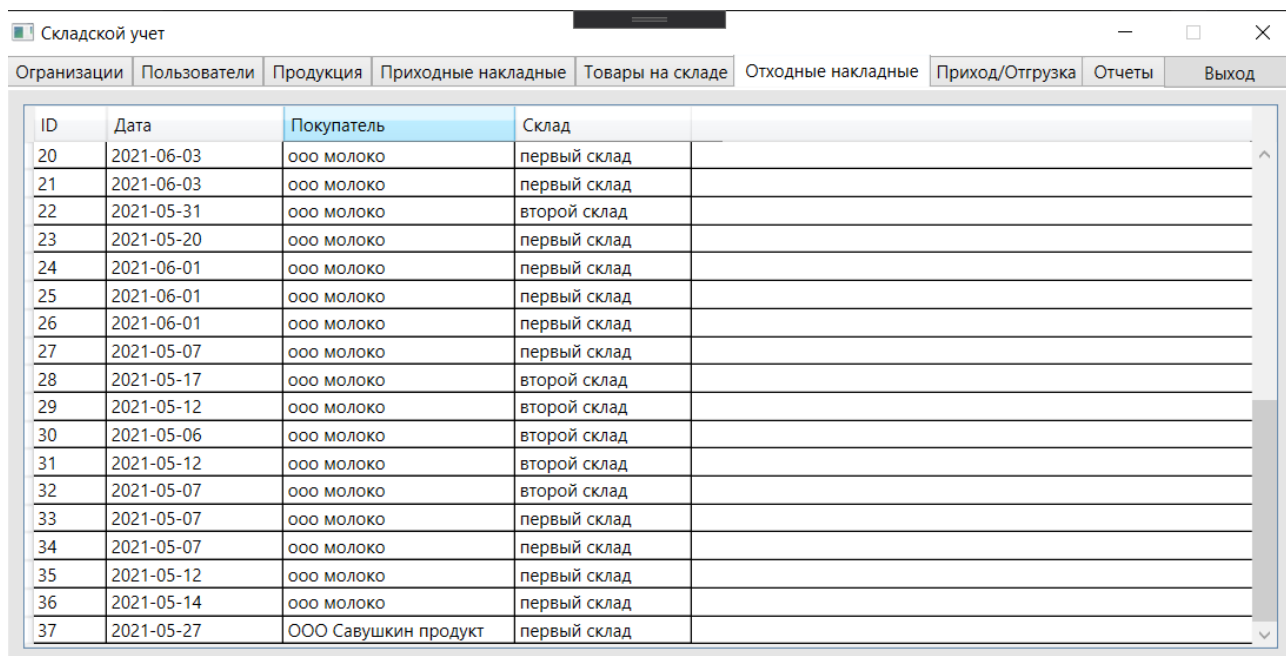
Продукт	Количество	Склад	
молоко	304	первый склад	
Сметана	223	первый склад	
Сметана	2	второй склад	
ряженка	557	первый склад	
ряженка	12	второй склад	

Рисунок 3.9 – Вкладка «Товары на складе»

При проведении прихода, количество товаров на конкретном складе увеличивается. При проведении отгрузки, количество товара на конкретном складе уменьшается.

При нажатии на вкладку «Отходные накладные», появляется информация, содержащая данные об отходных накладных.

Вкладка «Отходные накладные» изображена на рисунке 3.10.

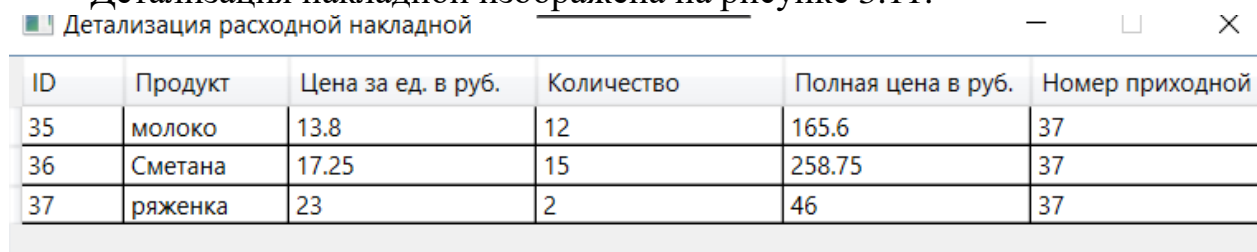


ID	Дата	Покупатель	Склад	
20	2021-06-03	ooo молоко	первый склад	
21	2021-06-03	ooo молоко	первый склад	
22	2021-05-31	ooo молоко	второй склад	
23	2021-05-20	ooo молоко	первый склад	
24	2021-06-01	ooo молоко	первый склад	
25	2021-06-01	ooo молоко	первый склад	
26	2021-06-01	ooo молоко	первый склад	
27	2021-05-07	ooo молоко	первый склад	
28	2021-05-17	ooo молоко	второй склад	
29	2021-05-12	ooo молоко	второй склад	
30	2021-05-06	ooo молоко	второй склад	
31	2021-05-12	ooo молоко	второй склад	
32	2021-05-07	ooo молоко	второй склад	
33	2021-05-07	ooo молоко	первый склад	
34	2021-05-07	ooo молоко	первый склад	
35	2021-05-12	ooo молоко	первый склад	
36	2021-05-14	ooo молоко	первый склад	
37	2021-05-27	ООО Савушкин продукт	первый склад	

Рисунок 3.10 – Вкладка «Отходные накладные»

На вкладке «Отходные накладные» содержится список всех отходных накладных. Чтобы просмотреть детализацию отходной накладной, необходимо выбрать нужную накладную из таблицы и нажать на нее два раза. После нажатия появляется окно, содержащее детализацию накладной.

Детализация накладной изображена на рисунке 3.11.



ID	Продукт	Цена за ед. в руб.	Количество	Полная цена в руб.	Номер приходной
35	молоко	13.8	12	165.6	37
36	Сметана	17.25	15	258.75	37
37	ряженка	23	2	46	37

Рисунок 3.11 – Детализация расходной накладной

Детализация приходной накладной содержит список товаров, входящих в накладную, а также информацию об этих товарах:

- наименование продукта;
- цена за единицу продукта;

- количество продукта;
- полную цену продуктов;
- номер приходной накладной.

При нажатии на вкладку «Приход/отгрузка», появляется окно для формирования прихода или отгрузки.

Вкладка «Приход/Отгрузка» изображена на рисунке 3.12.

Рисунок 3.12 – Формирование прихода/отгрузки

На вкладке «Приход\Отгрузка» можно формировать приходные и расходные накладные.

Чтобы сформировать приход или отгрузку необходимо выбрать тип операции, склад, где будет проводится выбранная операция, организацию и дату.

Для операции приход, поле «Организации» означает поставщика, а для операции отгрузка – покупателя.

При нажатии на кнопку «Добавить продукты в накладную», открывается окно для добавления продуктов, которое изображено на рисунке 3.13.

Рисунок 3.13 – Добавление продуктов в накладную

В окно «Введите количество» нельзя вводит символы или отрицательные числа.

По нажатию на кнопку «Добавить», выбранный продукт добавится в позицию накладной.

При нажатии на вкладку «Отчеты», появляется окно для формирования отчетов.

Вкладка «Отчеты» изображена на рисунке 3.14.

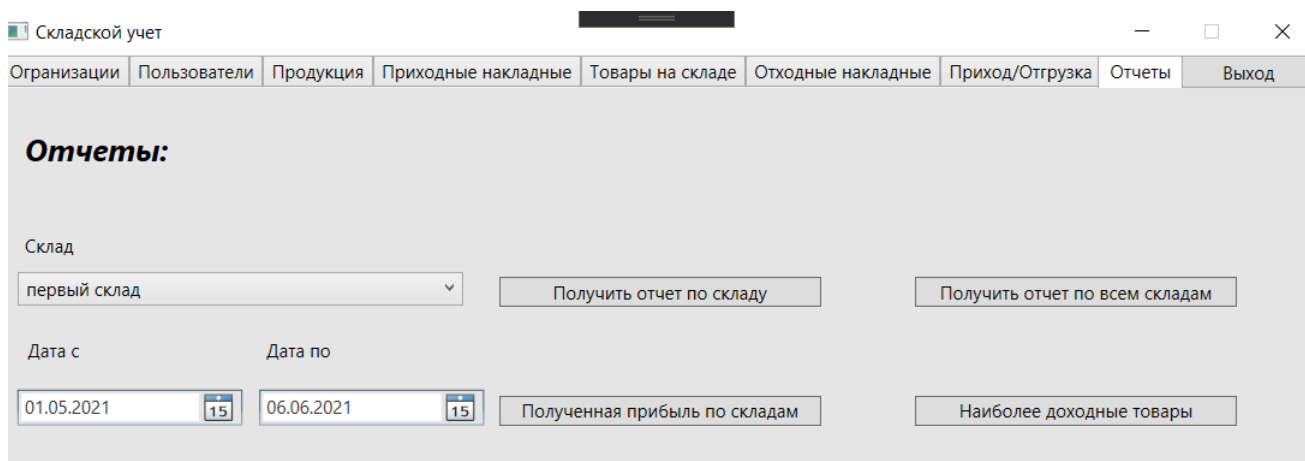


Рисунок 3.14 – Вкладка отчеты

На вкладке «Отчеты» можно сформировать отчеты в виде XML файла по:

- конкретному складу;
- по всем складам;
- по прибыли, полученной со всех складов;
- о наиболее доходных товарах.

Полученную прибыль со всех складов можно получить в любой период времени. Для этого необходимо выбрать дату с какого по какое число получить отчет.

Пример отчета по складу изображен на рисунке 3.15.

```
<?xml version="1.0" encoding="utf-8"?>
<Склад>
  <Склад Наименование="первый склад">
    <Продукт>молоко</Продукт>
    <Цена>12</Цена>
    <Количество>292</Количество>
  </Склад>
  <Склад Наименование="первый склад">
    <Продукт>ряженка</Продукт>
    <Цена>20</Цена>
    <Количество>555</Количество>
  </Склад>
  <Склад Наименование="первый склад">
    <Продукт>Сметана</Продукт>
    <Цена>15</Цена>
    <Количество>220</Количество>
  </Склад>
</Склад>
```

Рисунок 3.15 – Пример отчета по складу

Все отчеты поддерживают открытие в формате *XLSX* в приложении *Microsoft Excel*.

Пример открытия *XML*-файла с отчетом в *Microsoft Excel* изображен на рисунке 3.16.

Наименование ▼	Продукт ▼	Цена ▼	Количество ▼
первый склад	молоко	12	292
первый склад	ряженка	20	555
первый склад	Сметана	15	220

Рисунок 3.15 – Пример отчета по складу открытый в приложении *Microsoft Excel*

Отчет содержит следующие поля:

- наименование склада;
- наименование товара;
- цену товара
- количество товара.

Пример отчета в формате *XLSX* о наиболее доходных товарах изображен на рисунке 3.16.

Наименование ▼	Стоимость ▼
молоко	138000
сыр	2530
Сметана	258,75
сливки	213,9
ряженка	200
кефир	30

Рисунок 3.16 – Отчет о наиболее доходных товарах

Таблица содержит сведения о наиболее доходных товарах, расположенных в порядке возрастания цены.

Любые отчеты формата *XLSX* поддерживают сортировку по названию (в алфавитном порядке), если поле является строкой, либо по возрастанию или по убыванию, если поле является числом.

В отчете формата *XLSX* все поля соответствуют тегам и атрибутам файла формата *XML*.

Все отчеты формировались согласно требованию, описанному в листе задания.

При нажатии на кнопку выход, происходит закрытие основного меню и открытие меню авторизации, где можно сменить пользователя.

Чтобы выйти из приложения необходимо нажать на крестик в правом верхнем углу, либо кнопку «Отмена» в меню авторизации. После любой из этих процедур, приложение закроется.

После закрытия приложения не остается открытых соединений с базой данных.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта было разработано программное обеспечение для учета поступления товаров на склады предприятия при использовании современных средств *PostgreSQL*.

В современных условиях руководству предприятий или организаций приходится иметь дело с огромным количеством информации. Эта информация растет и изменяется так быстро, что её часто становится просто невозможно обрабатывать «вручную». Кроме того, на больших предприятиях с большими оборотами продукции существует необходимость учёта и контроля большого объёма финансовой, производственной, закупочно-сбытовой, маркетинговой информации.

Для этого и создаются автоматизированные системы для сбора, обработки и хранения информации. Такие информационные системы должны облегчить процесс работы с информацией, циркулирующей на предприятии.

Таким образом, была решена поставленная задача и построена гибкая модель базы данных, в которой легко просмотреть необходимые данные, отредактировать их так как это нужно, получить нужный отчет. Данные представлены в удобном для пользователя виде. Интерфейс программы построен без излишков и настроен на максимальное удобство пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рихтер Дж. *CLR VIA C#*. Программирование на платформе *Microsoft.NET Framework 4.5* на языке *C#* / Рихтер Дж. 4-е изд., перераб. и доп. – Питер: 2019 – 896 с.
2. Шарп Джон *Microsoft Visual C#*: учебное пособие / Шарп Джон – 8-е изд., – Питер, 2017. – 848
3. Троелсен Э., Джепикс Ф. Язык программирования *C# 7* и платформы *.NET* и *.NET Core* / Троелсен Э., Джепикс Ф. 8-е изд., перераб. и доп. – Санкт-Петербург: 2018 – 1328 с.
4. Особенности платформы *WPF*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: <https://metanit.com/sharp/wpf/1.php> – Дата доступа: 12.04.2021
5. Лузанов П.В. *PostgreSQL* для начинающих: учебное пособие П.В. Лузанов, Е.В. Рогов – 3-е изд., – Москва, 2017. – 218 с.
6. СУБД *PostgreSQL*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: <http://bourabai.kz/dbt/servers/postgresql.htm> – Дата доступа: 12.04.2021
7. *Data Access Object*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: <https://habr.com/ru/post/262243/> – Дата доступа: 12.04.2021

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

Класс *Customer*

```
namespace CourseApp.Models
{
    /// <summary>
    /// Класс поставщик\покупатель
    /// </summary>
    public class Customer
    {
        /// <summary>
        /// Id покупателя\поставщика
        /// </summary>
        public int CustomerId { get; set; }
        /// <summary>
        /// Имя покупателя\поставщика
        /// </summary>
        public string CustomerName { get; set; }
        /// <summary>
        /// Описание покупателя\поставщика
        /// </summary>
        public string Description { get; set; }
    }
}
```

Класс *ExpenditureInvoice*

```
using NpgsqlTypes;
namespace CourseApp.Models
{
    /// <summary>
    /// Класс расходной накладной
    /// </summary>
    public class ExpenditureInvoice
    {
        /// <summary>
        /// Id расходной
        /// </summary>
        public int ExpenditureInvoiceId { get; set; }
        /// <summary>
        /// Дата расходной
        /// </summary>
        public NpgsqlDate? ExpenditureInvoiceDate { get; set; }
        /// <summary>
        /// Id покупателя
        /// </summary>
        public int? CustomerId { get; set; }
        /// <summary>
        /// Id склада
        /// </summary>
        public int? StockId { get; set; }
        /// <summary>
```

```

        /// Покупатель
        /// </summary>
        public Customer Customer { get; set; }
        /// <summary>
        /// Склад
        /// </summary>
        public Stock Stock { get; set; }
    }
}

```

Класс *ExpenditurePosition*

```

namespace CourseApp.Models
{
    /// <summary>
    /// Класс позиций расходной
    /// </summary>
    public class ExpenditurePosition
    {
        /// <summary>
        /// Id позиции
        /// </summary>
        public int ExpenditurePositionId { get; set; }
        /// <summary>
        /// Id продукта
        /// </summary>
        public int? ProductId { get; set; }
        /// <summary>
        /// Продукт
        /// </summary>
        public Product Product { get; set; }
        /// <summary>
        /// Количество продукта
        /// </summary>
        public double CountProduct { get; set; }
        /// <summary>
        /// Id расходной
        /// </summary>
        public int ExpenditureInvoiceId { get; set; }
        /// <summary>
        /// Цена продукта
        /// </summary>
        public double ProductPrice { get; set; }
        /// <summary>
        /// Полная цена
        /// </summary>
        public double FullPrice { get; set; }
    }
}

```

Класс *ProductInStock*

```

namespace CourseApp.Models
{
    /// <summary>
    /// Класс продукта на складе

```



```

/// </summary>
public class ProductInStock
{
    /// <summary>
    /// Id
    /// </summary>
    public int Id { get; set; }
    /// <summary>
    /// Id продукта
    /// </summary>
    public int ProductId { get; set; }
    /// <summary>
    /// Id склада
    /// </summary>
    public int StockId { get; set; }
    /// <summary>
    /// Количество продукта
    /// </summary>
    public double CountProduct { get; set; }
    /// <summary>
    /// Продукт
    /// </summary>

    public Product Product { get; set; }
    /// <summary>
    /// Склад
    /// </summary>
    public Stock Stock { get; set; }
}
}

```

Класс *ReceiptInvoice*

```

using NpgsqlTypes;
namespace CourseApp.Models
{
    /// <summary>
    /// Приходная накладная
    /// </summary>
    public class ReceiptInvoice
    {
        /// <summary>
        /// ID накладной
        /// </summary>
        public int ReceiptInvoiceId { get; set; }
        /// <summary>
        /// Дата накладной
        /// </summary>
        public NpgsqlDate? ReceiptInvoiceDate { get; set; }
        /// <summary>
        /// Id поставщика
        /// </summary>
        public int? CustomerId { get; set; }
        /// <summary>
        /// ID склада
        /// </summary>
    }
}

```

```

        public int StockId { get; set; }
        /// <summary>
        /// Поставщик
        /// </summary>
        public Customer Customer { get; set; }

        /// <summary>
        /// Склад
        /// </summary>
        public Stock Stock { get; set; }
    }
}

```

Класс *ReceiptPosition*

```

namespace CourseApp.Models
{
    /// <summary>
    /// Класс позиция накладной
    /// </summary>
    public class ReceiptPosition
    {
        /// <summary>
        /// Id позиции
        /// </summary>
        public int PositionId { get; set; }
        /// <summary>
        /// Id накладной
        /// </summary>
        public int? ReceiptInvoiceId { get; set; }
        /// <summary>
        /// Id продукта
        /// </summary>
        public int? ProductId { get; set; }
        /// <summary>
        /// Продукт
        /// </summary>
        public Product Product { get; set; }
        /// <summary>
        /// Количество продукта
        /// </summary>
        public double CountProduct { get; set; }
        /// <summary>
        /// Полная цена
        /// </summary>
        public double FullPrice { get; set; }
    }
}

```

Класс *Role*

```

namespace CourseApp.Models
{
    /// <summary>
    /// Класс роль
    /// </summary>

```

```

public class Role
{
    /// <summary>
    /// Роль
    /// </summary>
    public string RoleName { get; set; }
    /// <summary>
    /// Ключ
    /// </summary>
    public string RoleKey { get; set; }
}
}

```

Класс *Stock*

```

namespace CourseApp.Models
{
    /// <summary>
    /// Класс склад
    /// </summary>
    public class Stock
    {
        /// <summary>
        /// Id склада
        /// </summary>
        public int StockId { get; set; }
        /// <summary>
        /// Имя склада
        /// </summary>
        public string StockName { get; set; }
        /// <summary>
        /// Описание
        /// </summary>
        public string Description { get; set; }
        /// <summary>
        /// Наценка
        /// </summary>
        public double Markup { get; set; }
        /// <summary>
        /// UserId
        /// </summary>
        public int? UserId { get; set; }
        /// <summary>
        /// User
        /// </summary>
        public User User { get; set; }
    }
}

```

Класс *User*

```

namespace CourseApp.Models
{
    /// <summary>

```

```

/// Класс пользователей
/// </summary>
public class User
{
    /// <summary>
    /// Id
    /// </summary>
    public int UserId { get; set; }
    /// <summary>
    /// RoleKey
    /// </summary>
    public string RoleKey { get; set; }
    /// <summary>
    /// Логин
    /// </summary>
    public string UserName { get; set; }
    /// <summary>
    /// Пароль
    /// </summary>
    public string UserPass { get; set; }
    /// <summary>
    /// Полное имя
    /// </summary>
    public string FullName { get; set; }
}
}

```

Класс *AbstractProduct*

```

namespace CourseApp.Models
{
    /// <summary>
    /// Класс абстрактного продукта
    /// </summary>
    public abstract class AbstractProduct
    {
        /// <summary>
        /// Id продукта
        /// </summary>
        public int EntityId { get; set; }
        /// <summary>
        /// Название продукта
        /// </summary>
        public string ProductName { get; set; }
        /// <summary>
        /// Цена продукта
        /// </summary>
        public double ProductPrice { get; set; }
        /// <summary>
        /// Получение полной цены
        /// </summary>
        /// <returns></returns>
        public abstract double GetFullPrice();
    }
}

```

Интерфейс *IReportService*

```
using System.Windows.Controls;
namespace CourseApp.Reports
{
    /// <summary>
    /// Интерфейс для получения отчетов
    /// </summary>
    public interface IReportService
    {
        /// <summary>
        /// Получить информацию по конкретному складу
        /// </summary>
        /// <param name="stockId">id склада</param>
        /// <param name="reportComboBox">элемент комбобокса</param>
        void GetReportS(int stockId, ComboBox reportComboBox);
        /// <summary>
        /// Получить информацию обо всех складах
        /// </summary>
        void GetReportAS();
        /// <summary>
        /// Получить прибыль по складам
        /// </summary>
        /// <param name="dataFrom">дата с какой брать данные</param>
        /// <param name="dataTo">дата по какую брать данные</param>
        void GetReportDS(string dataFrom, string dataTo);
        /// <summary>
        /// Получение доходных товаров
        /// </summary>
        void GetReportD();
    }
}
```

Класс *Reports*

```
using Microsoft.Win32;
using Npgsql;
using System.Windows;
using System.Xml.Linq;
using CourseApp.Utility;
using System.Windows.Controls;
using CourseApp.Reports;

namespace CourseApp.Report
{
    public class Reports:IReportService
    {
        public Reports()
        {
        }

        /// <summary>
        /// Получение отчета по складу
        /// </summary>
        public void GetReportS(int stockId, ComboBox ReportComboBox)
```

```

{
    if (ReportComboBox == null ||
        ReportComboBox.SelectedItem == null)
    {
        MessageBox.Show("Выберите склад");
    }
    else
    {
        DbConnection connection = new DbConnection();
        try
        {
            string query = "select stock_name, product_name, count_product, product_price from products_in_stock " +
                " inner join stocks on products_in_stock.stock_id = stocks.stock_id " +
                "inner join products on products_in_stock.product_id = products.product_id where prod-
ucts_in_stock.stock_id =" + stockId;

            NpgsqlCommand command = new NpgsqlCommand(query, connection.GetConnection());
            NpgsqlDataReader reader = command.ExecuteReader();

            XmlDocument xdoc = new XmlDocument();
            XElement stocks = new XElement("Склад");

            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    XElement stock = new XElement("Склад");
                    XAttribute istockNameAttr = new XAttribute("Наименование", reader["stock_name"].ToString());
                    XElement istockProductElem = new XElement("Продукт", reader["product_name"].ToString());
                    XElement istockCountElem = new XElement("Количество", reader["count_product"].ToString());
                    XElement istockSumElem = new XElement("Цена", reader["product_price"].ToString());
                    stock.Add(istockNameAttr);
                    stock.Add(istockProductElem);
                    stock.Add(istockSumElem);
                    stock.Add(istockCountElem);
                    stocks.Add(stock);
                }
                reader.Close();
            }
            xdoc.Add(stocks);

            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "XML-File | *.xml";
            if (saveFileDialog.ShowDialog() == true)
            {
                xdoc.Save(saveFileDialog.FileName);
                MessageBox.Show("Файл сохранен");
            }
        }
        catch (NpgsqlException ex)
        {
            MessageBox.Show(ex.Message.ToString());
        }
        finally
        {

```

```

        connection.CloseConnection();
    }
}
}
/// <summary>
/// Получение отчета по всем складам
/// </summary>
public void GetReportAS()
{
    DbConnection connection = new DbConnection();
    try
    {
        string query = "select stock_name, product_name,count_product,product_price from products_in_stock" +
            " inner join stocks on products_in_stock.stock_id = stocks.stock_id " +
            "inner join products on products_in_stock.product_id = products.product_id ";

        NpgsqlCommand command = new NpgsqlCommand(query, connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        XmlDocument xdoc = new XmlDocument();
        XElement stocks = new XElement("Склады");

        if (reader.HasRows)
        {
            while (reader.Read())
            {
                XElement stock = new XElement("Склад");
                XAttribute istockNameAttr = new XAttribute("Наименование", reader["stock_name"].ToString());
                XElement istockProductElem = new XElement("Продукт", reader["product_name"].ToString());
                XElement istockCountElem = new XElement("Количество", reader["count_product"].ToString());
                XElement istockSumElem = new XElement("Цена", reader["product_price"].ToString());
                stock.Add(istockNameAttr);
                stock.Add(istockProductElem);
                stock.Add(istockSumElem);
                stock.Add(istockCountElem);
                stocks.Add(stock);
            }
            reader.Close();
        }
        xdoc.Add(stocks);

        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.Filter = "XML-File | *.xml";
        if (saveFileDialog.ShowDialog() == true)
        {
            xdoc.Save(saveFileDialog.FileName);
            MessageBox.Show("Файл сохранен");
        }
    }
    catch (NpgsqlException ex)
    {
    }
    finally
    {
        connection.CloseConnection();
    }
}

```

```

    }
}
/// <summary>
/// Полученная прибыль по складам
/// </summary>
/// <param name="dataFrom"></param>
/// <param name="dataTo"></param>
public void GetReportDS(string dataFrom, string dataTo)
{
    DbConnection connection = new DbConnection();
    try
    {
        string query = "select stock_name, sum(product_price*count_product) from expenditure_invoices" +
            " inner join expenditure_positions on expenditure_invoices.expenditure_invoice_id = expenditure_posi-
tions.expenditure_invoice_id " +
            "inner join stocks on stocks.stock_id = expenditure_invoices.stock_id" +
            " WHERE expenditure_invoice_date BETWEEN '" + dataFrom + "' AND '" + dataTo + "'" +
            " group by stock_name";

        NpgsqlCommand command = new NpgsqlCommand(query, connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        XmlDocument xdoc = new XmlDocument();
        XElement stocks = new XElement("Склады");

        if (reader.HasRows)
        {
            while (reader.Read())
            {
                XElement stock = new XElement("Склад");
                XAttribute istockNameAttr = new XAttribute("Наименование", reader["stock_name"].ToString());
                XElement istockPriceElem = new XElement("Прибыль", reader["sum"].ToString());
                stock.Add(istockNameAttr);
                stock.Add(istockPriceElem);
                stocks.Add(stock);
            }
            reader.Close();
        }
        xdoc.Add(stocks);

        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.Filter = "XML-File | *.xml";
        if (saveFileDialog.ShowDialog() == true)
        {
            xdoc.Save(saveFileDialog.FileName);
            MessageBox.Show("Файл сохранен");
        }
    }
    catch (NpgsqlException ex)
    {
    }
    finally
    {
        connection.CloseConnection();
    }
}

```



```

    }

    /// <summary>
    /// Наиболее доходные товары
    /// </summary>
    public void GetReportD()
    {
        DbConnection connection = new DbConnection();
        try
        {
            string query = "select products.product_name, max(count_product*expenditure_positions.product_price)" +
                "from expenditure_positions" +
                " inner join products on expenditure_positions.product_id = products.product_id" +
                " group by products.product_name " +
                "order by max desc ";
            NpgsqlCommand command = new NpgsqlCommand(query, connection.GetConnection());
            NpgsqlDataReader reader = command.ExecuteReader();
            XmlDocument xdoc = new XmlDocument();
            XElement products = new XElement("Товары");
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    XElement product = new XElement("Товар");
                    XAttribute iproductNameAttr = new XAttribute("Наименование", reader["product_name"].ToString());
                    XElement iproductPriceElem = new XElement("Стоимость", reader["max"].ToString());
                    product.Add(iproductNameAttr);
                    product.Add(iproductPriceElem);
                    products.Add(product);
                }
                reader.Close();
            }
            xdoc.Add(products);

            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "XML-File | *.xml";
            if (saveFileDialog.ShowDialog() == true)
            {
                xdoc.Save(saveFileDialog.FileName);
                MessageBox.Show("Файл сохранен");
            }
        }
        catch (NpgsqlException ex)
        {
        }
        finally
        {
            connection.CloseConnection();
        }
    }
}

```

Класс *DbConnection*

```
using Npgsql;
using System.Configuration;
using System.Data;

namespace CourseApp.Utility
{
    /// <summary>
    /// Подключение к БД.
    /// </summary>
    public class DbConnection
    {
        private readonly NpgsqlConnection connection = new NpgsqlConnection(ConfigurationManager.ConnectionStrings["MyDatabase"].ConnectionString);
        /// <summary>
        /// Получение соединения.
        /// </summary>
        /// <returns></returns>
        public NpgsqlConnection GetConnection()
        {
            // Проверяем, закрыто ли соединение.
            if (connection.State == ConnectionState.Closed)
            {
                // Если закрыто, то открываем.
                connection.Open();
            }
            return connection;
        }

        /// <summary>
        /// Закрытие соединения.
        /// </summary>
        public void CloseConnection()
        {
            // Проверяем, открыто ли соединение.
            if (connection.State == ConnectionState.Open)
            {
                // Если да, то закрываем.
                connection.Close();
            }
        }
    }
}
```

Интерфейс *IService*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CourseApp.Services
{
    /// <summary>
    /// Интерфейс со стандартными операциями.
    /// </summary>
    /// <typeparam name="T"></typeparam>
```

```

public interface IService<T>
{
    /// <summary>
    /// Получение всех записей.
    /// </summary>
    /// <returns></returns>
    List<T> GetAll();

    /// <summary>
    /// Обновление.
    /// </summary>
    /// <param name="entity"></param>
    /// <returns></returns>
    bool Update(T entity);

    /// <summary>
    /// Удаление.
    /// </summary>
    /// <param name="entity"></param>
    /// <returns></returns>
    bool Delete(T entity);

    /// <summary>
    /// Получение по id.
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    T GetById(int id);

    /// <summary>
    /// Добавление.
    /// </summary>
    /// <param name="entity"></param>
    /// <returns></returns>
    bool Insert(T entity);
}
}

```

Класс *CustomerService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System.Collections.Generic;

namespace CourseApp.Services
{
    /// <summary>
    /// Сервис для работы с организациями.
    /// </summary>
    public class CustomerService : IService<Customer>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Получение всех кастомеров.
        /// </summary>

```

```

/// <returns></returns>
public List<Customer> GetAll()
{
    try
    {
        List<Customer> customers = new List<Customer>();

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM customers ORDER BY cus-
tomer_id", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            Customer customer = new Customer();

            customer.CustomerId = reader.GetInt32(0);
            customer.CustomerName = reader.GetString(1);
            customer.Description = reader.GetString(2);

            customers.Add(customer);
        }

        connection.CloseConnection();
        return customers;
    }
    catch (NpgsqlException ex)
    {
    }

    return null;
}

/// <summary>
/// Обновление кастомера.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(Customer entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE customers SET cus-
tomer_name=@name, description=@description WHERE customer_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.CustomerId);
        command.Parameters.AddWithValue("@name", entity.CustomerName);
        command.Parameters.AddWithValue("@description", entity.Description);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }
}

```

```

        return true;
    }

    /// <summary>
    /// Удаление кастомера.
    /// </summary>
    /// <param name="entity"></param>
    /// <returns></returns>
    public bool Delete(Customer entity)
    {
        try
        {
            NpgsqlCommand command = new NpgsqlCommand("DELETE FROM customers WHERE cus-
tomer_id=@id;", connection.GetConnection());

            command.Parameters.AddWithValue("@id", entity.CustomerId);

            command.ExecuteNonQuery();
            connection.CloseConnection();
        }
        catch (NpgsqlException ex)
        {
            return false;
        }

        return true;
    }

    /// <summary>
    /// Получение кастомера по id.
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    public Customer GetById(int id)
    {
        Customer customer = null;

        try
        {
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM customers WHERE cus-
tomer_id=@id;", connection.GetConnection());
            command.Parameters.AddWithValue("@id", id);

            NpgsqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                customer = new Customer()
                {
                    CustomerId = reader.GetInt32(0),
                    CustomerName = reader.GetString(1),
                    Description = reader.GetString(2)
                };
            }

            connection.CloseConnection();

```

```

    }
    catch (NpgsqlException ex)
    {

    }

    return customer;
}

/// <summary>
/// Добавление кастомера.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(Customer entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO customers (customer_name, description) VALUES (@name, @description);", connection.GetConnection());

        command.Parameters.AddWithValue("@name", entity.CustomerName);
        command.Parameters.AddWithValue("@description", entity.Description);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}

```

Класс *ExpenditureInvoiceService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System.Collections.Generic;

namespace CourseApp.Services
{
    /// <summary>
    /// Сервис для работы с отходными накладными.
    /// </summary>
    public class ExpenditureInvoiceService : IService<ExpenditureInvoice>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Удаление отходной накладной.
    }
}

```

```

/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Delete(ExpenditureInvoice entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("DELETE FROM expenditure_invoices WHERE ex-
penditure_invoice_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.ExpenditureInvoiceId);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

/// <summary>
/// Получение всех отходных накладных.
/// </summary>
/// <returns></returns>
public List<ExpenditureInvoice> GetAll()
{
    try
    {
        List<ExpenditureInvoice> entities = new List<ExpenditureInvoice>();

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM expenditure_invoices OR-
DER BY expenditure_invoice_id", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            ExpenditureInvoice entity = new ExpenditureInvoice()
            {
                ExpenditureInvoiceId = reader.GetInt32(0),
                ExpenditureInvoiceDate = reader.GetDate(1),
                CustomerId = reader.GetInt32(2),
                StockId = reader.GetInt32(3)
            };

            entities.Add(entity);
        }

        connection.CloseConnection();
        return entities;
    }
    catch (NpgsqlException ex)
    {
    }
}

```

```

        return null;
    }

    /// <summary>
    /// Получение отходной накладной по id.
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    public ExpenditureInvoice GetById(int id)
    {
        ExpenditureInvoice entity = null;

        try
        {
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM expenditure_invoices WHERE ex-
penditure_invoice_id=@id;", connection.GetConnection());
            command.Parameters.AddWithValue("@id", id);

            NpgsqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                entity = new ExpenditureInvoice()
                {
                    ExpenditureInvoiceId = reader.GetInt32(0),
                    ExpenditureInvoiceDate = reader.GetDate(1),
                    CustomerId = reader.GetInt32(2),
                    StockId = reader.GetInt32(3)
                };
            }

            connection.CloseConnection();
        }
        catch (NpgsqlException ex)
        {
        }

        return entity;
    }

    /// <summary>
    /// Добавление отходной накладной.
    /// </summary>
    /// <param name="entity"></param>
    /// <returns></returns>
    public bool Insert(ExpenditureInvoice entity)
    {
        try
        {
            NpgsqlCommand command = new NpgsqlCommand("INSERT INTO expenditure_invoices " +
                "(expenditure_invoice_date, customer_id, stock_id) VALUES (@expenditure_in-
voice_date, @customer_id, @stock_id);",
                , connection.GetConnection());

            command.Parameters.AddWithValue("@expenditure_invoice_date", entity.ExpenditureInvoiceDate);

```



```

        command.Parameters.AddWithValue("@customer_id", entity.CustomerId);
        command.Parameters.AddWithValue("@stock_id", entity.StockId);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return false;
    }

    return true;
}

/// <summary>
/// Обновление отходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(ExpenditureInvoice entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE expenditure_invoices " +
            "SET expenditure_invoice_date=@expenditure_invoice_date, " +
            "customer_id=@customer_id, " +
            "stock_id=@stock_id " +
            "WHERE expenditure_invoice_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.ExpenditureInvoiceId);
        command.Parameters.AddWithValue("@expenditure_invoice_date", entity.ExpenditureInvoiceDate);
        command.Parameters.AddWithValue("@customer_id", entity.CustomerId);
        command.Parameters.AddWithValue("@stock_id", entity.StockId);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}
}

```

Класс *ExpenditurePositionService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System;
using System.Collections.Generic;

namespace CourseApp.Services
{
    public class ExpenditurePositionService : IService<ExpenditurePosition>
    {

```

```

DbConnection connection = new DbConnection();

/// <summary>
/// Удаление позиции отходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Delete(ExpenditurePosition entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("DELETE FROM expenditure_positions WHERE ex-
penditure_position_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.ExpenditurePositionId);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

/// <summary>
/// Получение всех позиций отходных накладных.
/// </summary>
/// <returns></returns>
public List<ExpenditurePosition> GetAll()
{
    try
    {
        List<ExpenditurePosition> entities = new List<ExpenditurePosition>();

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM expenditure_positions OR-
DER BY expenditure_position_id", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            ExpenditurePosition entity = new ExpenditurePosition()
            {
                ExpenditurePositionId = reader.GetInt32(0),
                ProductId = reader.GetInt32(1),
                CountProduct = reader.GetDouble(2),
                ExpenditureInvoiceId = reader.GetInt32(3),
                ProductPrice = Math.Round(reader.GetDouble(4),3)
            };

            entities.Add(entity);
        }

        connection.CloseConnection();
    }
}

```

```

        return entities;
    }
    catch (NpgsqlException ex)
    {

    }

    return null;
}

/// <summary>
/// Получение позиции отходной накладной по id.
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public ExpenditurePosition GetById(int id)
{
    ExpenditurePosition entity = null;

    try
    {
        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM expenditure_positions WHERE expenditure_position_id=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@id", id);

        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            entity = new ExpenditurePosition()
            {
                ExpenditurePositionId = reader.GetInt32(0),
                ProductId = reader.GetInt32(1),
                CountProduct = reader.GetDouble(2),
                ExpenditureInvoiceId = reader.GetInt32(3),
                ProductPrice = reader.GetDouble(4)
            };
        }

        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {

    }

    return entity;
}

/// <summary>
/// Добавление позиции отходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(ExpenditurePosition entity)
{
    try

```

```

    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO expenditure_positions " +
            "( product_id, count_product,expenditure_invoice_id,product_price) VAL-
UES (@product_id, @count_product,@expenditure_invoice_id,@product_price);"
            , connection.GetConnection());

        command.Parameters.AddWithValue("@product_id", entity.ProductId);
        command.Parameters.AddWithValue("@count_product", entity.CountProduct);
        command.Parameters.AddWithValue("@expenditure_invoice_id", entity.ExpenditureInvoiceId);
        command.Parameters.AddWithValue("@product_price", entity.ProductPrice);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return false;
    }

    return true;
}

/// <summary>
/// Обновление позиции отходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(ExpenditurePosition entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE expenditure_positions " +
            "SET product_id=@product_id, " +
            "count_product=@count_product, " +
            "expenditure_invoice_id=@expenditure_invoice_id " +
            "product_price=@product_price " +
            "WHERE expenditure_position_id=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@product_id", entity.ProductId);
        command.Parameters.AddWithValue("@count_product", entity.CountProduct);
        command.Parameters.AddWithValue("@expenditure_invoice_id", entity.ExpenditureInvoiceId);
        command.Parameters.AddWithValue("@product_price", entity.ProductPrice);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}
}

```

Класс *ProductInStockService*

using CourseApp.Models;

```

using CourseApp.Utility;
using Npgsql;
using System.Collections.Generic;
using System.Diagnostics;
using System.Windows;

namespace CourseApp.Services
{
    public class ProductInStockService : IService<ProductInStock>
    {
        DbConnection connection = new DbConnection();
        /// <summary>
        /// Удаление товаров со склада
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(ProductInStock entity)
        {
            try
            {
                NpgsqlCommand command = new NpgsqlCommand("DELETE FROM products_in_stock WHERE id=@id;", connection.GetConnection());

                command.Parameters.AddWithValue("@id", entity.Id);

                command.ExecuteNonQuery();
                connection.CloseConnection();
            }
            catch (NpgsqlException ex)
            {
                return false;
            }

            return true;
        }
        /// <summary>
        /// Получение всех товаров со склада
        /// </summary>
        /// <returns></returns>
        public List<ProductInStock> GetAll()
        {
            try
            {
                List<ProductInStock> entities = new List<ProductInStock>();

                NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM products_in_stock ORDER BY product_id", connection.GetConnection());
                NpgsqlDataReader reader = command.ExecuteReader();

                while (reader.Read())
                {
                    ProductInStock entity = new ProductInStock()
                    {
                        Id = reader.GetInt32(0),
                        ProductId = reader.GetInt32(1),
                        StockId = reader.GetInt32(2),
                        CountProduct = reader.GetDouble(3)
                    }
                }
            }
        }
    }
}

```

```

        };

        entities.Add(entity);
    }

    connection.CloseConnection();
    return entities;
}
catch (NpgsqlException ex)
{
}

return null;
}
/// <summary>
/// Получение товаров со склада по id
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public ProductInStock GetById(int id)
{
    throw new System.NotImplementedException();
}

/// <summary>
/// Вставка данных в таблицу
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(ProductInStock entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO products_in_stock(prod-
uct_id, stock_id,count_product) VALUES (@product_id, @stock_id, @count_product);", connection.GetConne-
ction());

        command.Parameters.AddWithValue("@product_id", entity.ProductId);
        command.Parameters.AddWithValue("@stock_id", entity.StockId);
        command.Parameters.AddWithValue("@count_product", entity.CountProduct);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
/// <summary>
/// Обновление количества товаров на складе
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>

```

```

public bool Update(ProductInStock entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE products_in_stock SET count_product=@count_product WHERE id=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@id", entity.Id);
        command.Parameters.AddWithValue("@count_product", entity.CountProduct);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}
}

```

Класс *ProductService*

```

using System.Collections.Generic;
using CourseApp.Utility;
using CourseApp.Models;
using Npgsql;
using System.Windows;
using System.Diagnostics;

namespace CourseApp.Services
{
    /// <summary>
    /// Сервис для работы с продукцией.
    /// </summary>
    public class ProductService : IService<Product>
    {
        DbConnection connection = new DbConnection();
        /// <summary>
        /// Удаление продукта
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(Product entity)
        {
            try
            {
                NpgsqlCommand command = new NpgsqlCommand("DELETE FROM products WHERE product_id=@id;", connection.GetConnection());

                command.Parameters.AddWithValue("@id", entity.EntityId);

                command.ExecuteNonQuery();
                connection.CloseConnection();
            }
        }
    }
}

```

```

        catch (NpgsqlException ex)
        {
            Debug.WriteLine(ex.Message);
            return false;
        }

        return true;
    }
    /// <summary>
    /// Получение всех продуктов
    /// </summary>
    /// <returns></returns>
    public List<Product> GetAll()
    {
        try
        {
            List<Product> entities = new List<Product>();

            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM products ORDER BY product_id", connection.GetConnection());
            NpgsqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                Product entity = new Product()
                {
                    EntityId = reader.GetInt32(0),
                    ProductName = reader.GetString(1),
                    ProductPrice = reader.GetDouble(2)
                };

                entities.Add(entity);
            }

            connection.CloseConnection();
            return entities;
        }
        catch (NpgsqlException ex)
        {
        }

        return null;
    }
    /// <summary>
    /// Получение продукта по id
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    public Product GetById(int id)
    {
        Product entity = null;

        try
        {
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM products WHERE product_id=@id;", connection.GetConnection());

```



```

        command.Parameters.AddWithValue("@id", id);

        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            entity = new Product()
            {
                EntityId = reader.GetInt32(0),
                ProductName = reader.GetString(1),
                ProductPrice = reader.GetDouble(2)
            };
        }

        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
    }

    return entity;
}
/// <summary>
/// вставка продукта
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(Product entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO products(product_name, product_price) VALUES (@name, @product_price);", connection.GetConnection());

        command.Parameters.AddWithValue("@name", entity.ProductName);
        command.Parameters.AddWithValue("@product_price", entity.ProductPrice);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
        Debug.WriteLine(ex.Message);
    }

    return true;
}
/// <summary>
/// обновление продукта
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(Product entity)
{
    try

```

```

    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE products SET product_name=@name, product_price=@product_price WHERE product_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.EntityId);
        command.Parameters.AddWithValue("@name", entity.ProductName);
        command.Parameters.AddWithValue("@product_price", entity.ProductPrice);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}
}

```

Класс *ReceiptInvoiceService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System.Collections.Generic;

namespace CourseApp.Services
{
    /// <summary>
    /// Сервис для работы с приходными накладными.
    /// </summary>
    public class ReceiptInvoiceService : IService<ReceiptInvoice>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Удаление приходной накладной.
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(ReceiptInvoice entity)
        {
            try
            {
                NpgsqlCommand command = new NpgsqlCommand("DELETE FROM receipt_invoices WHERE receipt_invoice_id=@id;", connection.GetConnection());

                command.Parameters.AddWithValue("@id", entity.ReceiptInvoiceId);

                command.ExecuteNonQuery();
                connection.CloseConnection();
            }
            catch (NpgsqlException ex)
            {

```

```

        return false;
    }

    return true;
}

/// <summary>
/// Получение всех приходных накладных.
/// </summary>
/// <returns></returns>
public List<ReceiptInvoice> GetAll()
{
    try
    {
        List<ReceiptInvoice> entities = new List<ReceiptInvoice>();

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM receipt_invoices ORDER BY receipt_invoice_id", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            ReceiptInvoice entity = new ReceiptInvoice()
            {
                ReceiptInvoiceId = reader.GetInt32(0),
                ReceiptInvoiceDate = reader.GetDate(1),
                CustomerId = reader.GetInt32(2),
                StockId = reader.GetInt32(3)
            };

            entities.Add(entity);
        }

        connection.CloseConnection();
        return entities;
    }
    catch (NpgsqlException ex)
    {
    }

    return null;
}

/// <summary>
/// Получение приходной накладной по id.
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public ReceiptInvoice GetById(int id)
{
    ReceiptInvoice entity = null;

    try
    {
        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM receipt_invoices WHERE receipt_invoice_id=@id;", connection.GetConnection());

```

```

command.Parameters.AddWithValue("@id", id);

NpgsqlDataReader reader = command.ExecuteReader();

while (reader.Read())
{
    entity = new ReceiptInvoice()
    {
        ReceiptInvoiceId = reader.GetInt32(0),
        ReceiptInvoiceDate = reader.GetDate(1),
        CustomerId = reader.GetInt32(2),
        StockId = reader.GetInt32(4)
    };
}

connection.CloseConnection();
}
catch (NpgsqlException ex)
{
}

return entity;
}

/// <summary>
/// Добавление приходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(ReceiptInvoice entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO receipt_invoices " +
            "(receipt_invoice_date, customer_id, stock_id) " +
            "VALUES (@receipt_invoice_date, @customer_id, @stock_id);",
            connection.GetConnection());

        command.Parameters.AddWithValue("@receipt_invoice_date", entity.ReceiptInvoiceDate);
        command.Parameters.AddWithValue("@customer_id", entity.CustomerId);
        command.Parameters.AddWithValue("@stock_id", entity.StockId);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

/// <summary>
/// Обновление приходной накладной.

```

```

/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(ReceiptInvoice entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE receipt_invoices " +
            "SET receipt_invoice_date=@receipt_invoice_date, " +
            "customer_id=@customer_id, " +
            "stock_id=@stock_id, " +
            "WHERE receipt_invoice_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.ReceiptInvoiceId);
        command.Parameters.AddWithValue("@receipt_invoice_date", entity.ReceiptInvoiceDate);
        command.Parameters.AddWithValue("@customer_id", entity.CustomerId);
        command.Parameters.AddWithValue("@stock_id", entity.StockId);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}

```

Класс *ReceiptPositionService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CourseApp.Services
{
    public class ReceiptPositionService: IService<ReceiptPosition>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Удаление позиции приходной накладной.
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(ReceiptPosition entity)
        {
            try
            {

```

```

        Npgsql.NpgsqlCommand command = new NpgsqlCommand("DELETE FROM receipt_posi-
tions WHERE position_id=@id;", connection.GetConnection());

```

```

        command.Parameters.AddWithValue("@id", entity.PositionId);
        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

```

```

/// <summary>
/// Получение всех позиций приходных накладных.
/// </summary>
/// <returns></returns>
public List<ReceiptPosition> GetAll()

```

```

{
    try
    {
        List<ReceiptPosition> entities = new List<ReceiptPosition>();

```

```

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM receipt_positions ORDER BY posi-
tion_id", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

```

```

        while (reader.Read())
        {
            ReceiptPosition entity = new ReceiptPosition()
            {
                PositionId = reader.GetInt32(0),
                CountProduct = reader.GetDouble(1),
                ProductId = reader.GetInt32(2),
                ReceiptInvoiceId = reader.GetInt32(3)
            };

            entities.Add(entity);
        }

        connection.CloseConnection();
        return entities;
    }
    catch (NpgsqlException ex)
    {
    }

    return null;
}

```

```

/// <summary>
/// Получение приходной накладной по id.
/// </summary>
/// <param name="id"></param>

```

```

/// <returns></returns>
public ReceiptPosition GetById(int id)
{
    ReceiptPosition entity = null;

    try
    {
        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM receipt_positions WHERE position_id=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@id", id);

        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            entity = new ReceiptPosition()
            {
                PositionId = reader.GetInt32(0),
                CountProduct = reader.GetDouble(1),
                ProductId = reader.GetInt32(2),
                ReceiptInvoiceId = reader.GetInt32(3)
            };
        }

        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
    }

    return entity;
}

/// <summary>
/// Добавление приходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(ReceiptPosition entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO receipt_positions " +
            "(product_id, count_product,receipt_invoice_id) " +
            "VALUES (@product_id, @count_product,@receipt_invoice_id);",
            connection.GetConnection());
        command.Parameters.AddWithValue("@product_id", entity.ProductId);
        command.Parameters.AddWithValue("@count_product", entity.CountProduct);
        command.Parameters.AddWithValue("@receipt_invoice_id", entity.ReceiptInvoiceId);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
    }
}

```

```

        return false;

    }

    return true;
}

/// <summary>
/// Обновление приходной накладной.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(ReceiptPosition entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE receipt_positions " +
            "SET count_product=@count_product " +
            "WHERE position_id=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@id", entity.PositionId);
        command.Parameters.AddWithValue("@count_product", entity.CountProduct);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}

```

Класс *RoleService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System;
using System.Collections.Generic;

namespace CourseApp.Services
{
    public class RoleService : IService<Role>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Удаление роли.
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(Role entity)
        {
            try

```



```

    {
        NpgsqlCommand command = new NpgsqlCommand("DELETE FROM roles WHERE role_key=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.RoleKey);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

/// <summary>
/// Получение всех ролей.
/// </summary>
/// <returns></returns>
public List<Role> GetAll()
{
    try
    {
        List<Role> entities = new List<Role>();

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM roles", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            Role entity = new Role()
            {
                RoleName = reader.GetString(0),
                RoleKey = reader.GetString(1)
            };

            entities.Add(entity);
        }

        connection.CloseConnection();
        return entities;
    }
    catch (NpgsqlException ex)
    {
    }

    return null;
}

/// <summary>
/// Получение роли по id.
/// </summary>
/// <param name="id"></param>

```

```

/// <returns></returns>
public Role GetById(int id)
{
    Role entity = null;

    try
    {
        NpgsqlCommand command = new NpgsqlCommand("SE-
LECT * FROM roles WHERE role_key=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@id", id);

        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            entity = new Role()
            {
                RoleName = reader.GetString(0),
                RoleKey = reader.GetString(1)
            };
        }

        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
    }

    return entity;
}

/// <summary>
/// Добавление роли.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(Role entity)
{
    throw new NotImplementedException();
}

/// <summary>
/// Обновление роли.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(Role entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UP-
DATE roles SET role_name=@name WHERE role_key=@key;", connection.GetConnection());

        command.Parameters.AddWithValue("@key", entity.RoleKey);
        command.Parameters.AddWithValue("@name", entity.RoleName);
    }
}

```

```

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}
}

```

Класс *StockService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System.Collections.Generic;

namespace CourseApp.Services
{
    /// <summary>
    /// Сервис для работы со складами.
    /// </summary>
    public class StockService : IService<Stock>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Удаление склада.
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(Stock entity)
        {
            try
            {
                NpgsqlCommand command = new NpgsqlCommand("DE-
LETE FROM stocks WHERE stock_id=@id;", connection.GetConnection());

                command.Parameters.AddWithValue("@id", entity.UserId);

                command.ExecuteNonQuery();
                connection.CloseConnection();
            }
            catch (NpgsqlException ex)
            {
                return false;
            }

            return true;
        }

        /// <summary>
        /// ПОЛУЧЕНИЕ всех складов.
        /// </summary>

```

```

/// <returns></returns>
public List<Stock> GetAll()
{
    try
    {
        List<Stock> entities = new List<Stock>();

        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM stocks ORDER BY stock_id", connection.GetConnection());
        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            Stock entity = new Stock()
            {
                StockId = reader.GetInt32(0),
                StockName = reader.GetString(1),
                Markup = reader.GetDouble(3)
            };

            entities.Add(entity);
        }

        connection.CloseConnection();
        return entities;
    }
    catch (NpgsqlException ex)
    {
    }

    return null;
}

/// <summary>
/// Получение склада по id.
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public Stock GetById(int id)
{
    Stock entity = null;

    try
    {
        NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM stocks WHERE stock_id=@id;", connection.GetConnection());
        command.Parameters.AddWithValue("@id", id);

        NpgsqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            entity = new Stock()
            {
                StockId = reader.GetInt32(0),
                StockName = reader.GetString(1),

```

```

        Description = reader.GetString(2),
        Markup = reader.GetDouble(3)
    };
}

connection.CloseConnection();
}
catch (NpgsqlException ex)
{
}

return entity;
}

/// <summary>
/// Добавление склада.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(Stock entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO stocks(stock_name, markup) VALUES(@name, @markup);", connection.GetConnection());

        command.Parameters.AddWithValue("@name", entity.StockName);
        //command.Parameters.AddWithValue("@description", entity.Description);
        command.Parameters.AddWithValue("@markup", entity.Markup);
        //command.Parameters.AddWithValue("@user_id", entity.UserId);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

/// <summary>
/// Обновление склада.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Update(Stock entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE stocks SET stock_name=@name, description=@description, markup=@markup, user_id=@user_id WHERE stock_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.StockId);
        command.Parameters.AddWithValue("@name", entity.StockName);

```

```

        command.Parameters.AddWithValue("@description", entity.Description);
        command.Parameters.AddWithValue("@markup", entity.Markup);
        command.Parameters.AddWithValue("@user_id", entity.UserId);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}
}

```

Класс *UserService*

```

using CourseApp.Models;
using CourseApp.Utility;
using Npgsql;
using System.Collections.Generic;

namespace CourseApp.Services
{
    /// <summary>
    /// Сервис для работы с пользователями.
    /// </summary>
    public class UserService : IService<User>
    {
        DbConnection connection = new DbConnection();

        /// <summary>
        /// Удаление юзера.
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public bool Delete(User entity)
        {
            try
            {
                NpgsqlCommand command = new NpgsqlCommand("DELETE FROM users WHERE user_id=@id;", connection.GetConnection());

                command.Parameters.AddWithValue("@id", entity.UserId);

                command.ExecuteNonQuery();
                connection.CloseConnection();
            }
            catch (NpgsqlException ex)
            {
                return false;
            }
        }
    }
}

```

```

        return true;
    }

    /// <summary>
    /// Получение всех юзеров.
    /// </summary>
    /// <returns></returns>
    public List<User> GetAll()
    {
        try
        {
            List<User> entities = new List<User>();

            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM users ORDER BY user_id", connection.GetConnection());
            NpgsqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                User entity = new User()
                {
                    UserId = reader.GetInt32(0),
                    RoleKey = reader.GetString(1),
                    UserName = reader.GetString(2),
                    UserPass = reader.GetString(3),
                    FullName = reader.GetString(4)
                };

                entities.Add(entity);
            }

            connection.CloseConnection();
            return entities;
        }
        catch (NpgsqlException ex)
        {
        }

        return null;
    }

    /// <summary>
    /// Получение юзера по Id.
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    public User GetById(int id)
    {
        User entity = null;

        try
        {
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM users WHERE user_id=@id;", connection.GetConnection());
            command.Parameters.AddWithValue("@id", id);

```

```

NpgsqlDataReader reader = command.ExecuteReader();

while (reader.Read())
{
    entity = new User()
    {
        UserId = reader.GetInt32(0),
        RoleKey = reader.GetString(1),
        UserName = reader.GetString(2),
        UserPass = reader.GetString(3),
        FullName = reader.GetString(4)
    };
}

connection.CloseConnection();
}
catch (NpgsqlException ex)
{
}

return entity;
}

/// <summary>
/// Добавление юзера.
/// </summary>
/// <param name="entity"></param>
/// <returns></returns>
public bool Insert(User entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("INSERT INTO us-
ers (role_key, user_name, user_pass, full_name) VALUES(@role, @name, @pswd, @fullName);", connec-
tion.GetConnection());

        command.Parameters.AddWithValue("@role", entity.RoleKey);
        command.Parameters.AddWithValue("@name", entity.UserName);
        command.Parameters.AddWithValue("@pswd", entity.UserPass);
        command.Parameters.AddWithValue("@fullName", entity.FullName);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}

/// <summary>
/// Обновление юзера.
/// </summary>

```



```

/// <param name="entity"></param>
/// <returns></returns>
public bool Update(User entity)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand("UPDATE users SET role_key=@role, user_name=@name, user_pass=@pswd, full_name=@fullName WHERE user_id=@id;", connection.GetConnection());

        command.Parameters.AddWithValue("@id", entity.UserId);
        command.Parameters.AddWithValue("@role", entity.RoleKey);
        command.Parameters.AddWithValue("@name", entity.UserName);
        command.Parameters.AddWithValue("@pswd", entity.UserPass);
        command.Parameters.AddWithValue("@fullName", entity.FullName);

        command.ExecuteNonQuery();
        connection.CloseConnection();
    }
    catch (NpgsqlException ex)
    {
        return false;
    }

    return true;
}
}

```

Класс *MainWindow*

```

using System.Linq;
using System.Windows;
using CourseApp.Models;
using CourseApp.Services;

namespace CourseApp
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        IService<User> _userService = new UserService();
        public MainWindow()
        {
            InitializeComponent();
        }

        private void ButtonLoginPress_Click(object sender, RoutedEventArgs e)
        {
            // Получение пользователя с заданным логином и паролем.
            var user = _userService.GetAll()
                .SingleOrDefault(u => u.UserName.Equals(textBoxLogin.Text) && u.UserPass.Equals(passwordBoxPass.Password));

            if (user == null)

```

```

    {
        // Если не найден, то выводим соответствующее сообщение.
        MessageBox.Show("Логин или пароль введены не верно!");
        passwordBoxPass.Password = "";
    }
    else
    {
        // Если пользователь найден, то открываем рабочее окно.
        WorkWindow workWindow = new WorkWindow(user.RoleKey, user.UserId);
        workWindow.Show();
        this.Close();
        //this.Hide();
    }
}

/// <summary>
/// Закрытие окна
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CancelButton_Click(object sender, RoutedEventArgs e)
{
    this.Close();
}
}
}

```

Класс *WorkWindow*

```

using NpgsqlTypes;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using CourseApp.Services;
using CourseApp.Models;

namespace CourseApp
{
    /// <summary>
    /// Логика взаимодействия для WorkWindow.xaml
    /// </summary>
    public partial class WorkWindow : Window
    {
        /// <summary>
        /// Инициализация полей
        /// </summary>
        private int userId;
        private string roleKey = "";
        /// <summary>
        /// Для вызова отчетов
    }
}

```

```

/// </summary>
private Report.Reports reports = new Report.Reports();
private User selectedUser = null;
private Product selectedProduct = null;
private Customer selectedCustomer = null;

// Инициализация сервисов для работы с таблицами в БД.
IService<User> _userService = new UserService();
IService<Role> _roleService = new RoleService();
IService<Stock> _stockService = new StockService();
IService<Product> _productService = new ProductService();
IService<Customer> _customerService = new CustomerService();
IService<ReceiptInvoice> _receiptInvoiceService = new ReceiptInvoiceService();
IService<ExpenditureInvoice> _expenditureInvoiceService = new ExpenditureInvoiceService();
IService<ProductInStock> _productInStockService = new ProductInStockService();

/// <summary>
/// Инициализация главного окна
/// </summary>
/// <param name="roleKey"></param>
/// <param name="userId"></param>
public WorkWindow(string roleKey, int userId)
{
    InitializeComponent();
    this.roleKey = roleKey;
    this.userId = userId;
    confidantility();
}

/// <summary>
/// Видимость элементов на интерфейсе по ролям (конфиденциальность)
/// </summary>
private void confidantility()
{
    idCustomer.Visibility = roleKey.Equals("admin") ? Visibility.Visible : Visibility.Hidden;
    idUser.Visibility = roleKey.Equals("admin") ? Visibility.Visible : Visibility.Hidden;
    idProduct.Visibility = roleKey.Equals("admin") ? Visibility.Visible : Visibility.Hidden;
    labelExpenditureReceiptDate.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    datePickerExpenditureReceiptDate.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    labelExpenditureReceiptStock.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    comboBoxExpenditureReceiptCompany.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    stockCb.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    labelExpenditureReceiptOperation.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    comboBoxExpenditureReceiptOperation.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    idExpenditureReceipt.Visibility = !roleKey.Equals("manager") ? Visibility.Visible : Visibility.Hidden;
    AddProductsBtn.Visibility = Visibility.Hidden;
    ExpenditureBtn.Visibility = Visibility.Hidden;
}

/// <summary>
/// Привязка организаций в таблицу.
/// </summary>
/// <param name="sender"></param>

```

```

/// <param name="e"></param>
private void IdCustomer_MouseEnter(object sender, MouseEventArgs e)
{
    dataGridCustomer.ItemsSource = _customerService.GetAll();
}

/// <summary>
/// Добавление/обновление организации.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonCustomer_Click(object sender, RoutedEventArgs e)
{
    if (selectedCustomer != null)
    {
        selectedCustomer.CustomerName = textBoxCustomerName.Text;
        selectedCustomer.Description = textBoxCustomerDescription.Text;

        _customerService.Update(selectedCustomer);

        selectedCustomer = null;
        textBoxCustomerName.Text = "";
        textBoxCustomerDescription.Text = "";
    }
    else
    {
        _customerService.Insert(new Customer
        {
            CustomerName = textBoxCustomerName.Text,
            Description = textBoxCustomerDescription.Text
        });

        textBoxCustomerName.Text = "";
        textBoxCustomerDescription.Text = "";
    }

    dataGridCustomer.ItemsSource = _customerService.GetAll();
}

/// <summary>
/// Отображение организации в контролы при нажатии.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DataGridCustomer_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (dataGridCustomer.SelectedItem != null)
    {
        selectedCustomer = (Customer)dataGridCustomer.SelectedItem;
        textBoxCustomerName.Text = selectedCustomer.CustomerName;
        textBoxCustomerDescription.Text = selectedCustomer.Description;
    }
}

/// <summary>
/// Удаление организации
/// </summary>

```

```

/// <param name="sender"></param>
/// <param name="e"></param>
private void DellOrganisationBtn_Click(object sender, RoutedEventArgs e)
{
    if (dataGridCustomer.SelectedItem != null)
    {
        _customerService.Delete((Customer)dataGridCustomer.SelectedItem);
        dataGridCustomer.Items.Refresh();
        dataGridCustomer.ItemsSource = _customerService.GetAll();
    }
    else
    {
        MessageBox.Show("Выберете строку");
    }
}

/// <summary>
/// Очистка контролов организации.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonCustomerClear_Click(object sender, RoutedEventArgs e)
{
    selectedCustomer = null;
    textBoxCustomerName.Text = "";
    textBoxCustomerDescription.Text = "";
}

/// <summary>
/// Привязка юзеров в таблицу и привязка ролей в листбокс.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void IdUser_MouseEnter(object sender, MouseEventArgs e)
{
    // Получение всех пользователей и привязка их в таблицу.
    dataGridUser.ItemsSource = _userService.GetAll();
    // Получение ролей и привязка их в листбокс.
    comboBoxUserRole.ItemsSource = _roleService.GetAll()?.Select(o => o.RoleKey);
}

/// <summary>
/// Добавление/обновление юзера.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonUser_Click(object sender, RoutedEventArgs e)
{
    if (comboBoxUserRole.SelectedItem != null)
    {
        if (selectedUser != null)
        {
            selectedUser.UserName = textBoxUserName.Text;
            selectedUser.FullName = textBoxUserFullName.Text;
            selectedUser.UserPass = passwordBoxUser.Password;
            selectedUser.RoleKey = comboBoxUserRole.SelectedItem.ToString();
        }
    }
}

```

```

        _userService.Update(selectedUser);

        selectedUser = null;
        textBoxUserName.Text = "";
        textBoxUserFullName.Text = "";
        passwordBoxUser.Password = "";
    }
    else
    {
        _userService.Insert(new User
        {
            UserName = textBoxUserName.Text,
            FullName = textBoxUserFullName.Text,
            UserPass = passwordBoxUser.Password,
            RoleKey = comboBoxUserRole.SelectedItem.ToString()
        });

        selectedUser = null;
        textBoxUserName.Text = "";
        textBoxUserFullName.Text = "";
        passwordBoxUser.Password = "";
    }
}
else
{
    MessageBox.Show("Выберите роль!");
}

dataGridUser.ItemsSource = _userService.GetAll();
comboBoxUserRole.ItemsSource = _roleService.GetAll().Select(o => o.RoleKey);
}

/// <summary>
/// Отображение выделенного юзера в контроллы.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DataGridUser_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (dataGridUser.SelectedItem != null)
    {
        selectedUser = (User)dataGridUser.SelectedItem;
        textBoxUserName.Text = selectedUser.UserName;
        textBoxUserFullName.Text = selectedUser.FullName;
        passwordBoxUser.Password = selectedUser.UserPass;
        comboBoxUserRole.SelectedItem = selectedUser.RoleKey;
    }
}

/// <summary>
/// Очистка контролов, связанных с юзером.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonUserClear_Click(object sender, RoutedEventArgs e)
{

```

```

        selectedUser = null;
        textBoxUserName.Text = "";
        textBoxUserFullName.Text = "";
        passwordBoxUser.Password = "";
    }

    /// <summary>
    /// Удаление юзера
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void DelUserBtn_Click(object sender, RoutedEventArgs e)
    {

        if (dataGridUser.SelectedItem != null)
        {
            if (dataGridUser.SelectedIndex == 0)
            {

                MessageBox.Show("Этот элемент нельзя удалить");
            }
            else
            {
                _userService.Delete((User)dataGridUser.SelectedItem);
                dataGridUser.Items.Refresh();
                dataGridUser.ItemsSource = _userService.GetAll();
            }
        }
        else
        {
            MessageBox.Show("Выберете строку");
        }
    }

    /// <summary>
    /// Очистка контролов продуктов.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonProductClear_Click(object sender, RoutedEventArgs e)
    {
        selectedProduct = null;
        textBoxProductName.Text = "";
        priceTb.Text = "";
    }

    /// <summary>
    /// Добавление/обновление продукта.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonProduct_Click(object sender, RoutedEventArgs e)
    {
        if (selectedProduct != null)
        {
            selectedProduct.ProductName = textBoxProductName.Text;

```

```

        selectedProduct.ProductPrice = Convert.ToDouble(priceTb.Text);

        _productService.Update(selectedProduct);

        selectedProduct = null;
        textBoxProductName.Text = priceTb.Text = "";
    }
    else
    {
        _productService.Insert(new Product
        {
            ProductName = textBoxProductName.Text,
            ProductPrice = Convert.ToDouble(priceTb.Text)
        });

        selectedProduct = null;
        textBoxProductName.Text = priceTb.Text = "";
    }

    dataGridProduct.ItemsSource = _productService.GetAll();
}

/// <summary>
/// Отображение продукта в контролы при выделении.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DataGridProduct_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (dataGridProduct.SelectedItem != null)
    {
        selectedProduct = (Product)dataGridProduct.SelectedItem;
        textBoxProductName.Text = selectedProduct.ProductName;
    }
}

/// <summary>
/// Привязка продуктов к таблице
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void IdProduct_MouseEnter(object sender, MouseEventArgs e)
{
    dataGridProduct.ItemsSource = _productService.GetAll();
}

/// <summary>
/// Удаление продукта
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DelProdBtn_Click(object sender, RoutedEventArgs e)
{
    if (dataGridProduct.SelectedItem != null)
    {
        if (_productService.Delete((Product)dataGridProduct.SelectedItem) == false)
        {

```



```

        Message-
Box.Show($"Удаление запрещено, другие записи имеют ссылку на выбранный продукт.", "Информация", Mes-
sageBoxButton.OK, MessageBoxImage.Error);
    }
    else
    {
        _productService.Delete((Product)dataGridProduct.SelectedItem);
        dataGridProduct.Items.Refresh();
        dataGridProduct.ItemsSource = _productService.GetAll();
    }
}
else
{
    MessageBox.Show("Выберете строку");
}
}

/// <summary>
/// Привязка складов к таблице.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void IdStock_MouseEnter(object sender, MouseEventArgs e)
{
    var stokeList = _stockService.GetAll() ?? new List<Stock>();

    foreach (var item in stokeList)
    {
        item.User = _userService.GetById((int)item.UserId);
    }
}

/// <summary>
/// Привязка приходных накладных к таблице.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void IdReceiptInvoices_MouseEnter(object sender, MouseEventArgs e)
{
    List<ReceiptInvoice> reseiptInvoices = _receiptInvoiceService.GetAll() ?? new List<ReceiptInvoice>();

    foreach (var item in reseiptInvoices)
    {
        item.Customer = _customerService.GetById((int)item.CustomerId);
        item.Stock = _stockService.GetById((int)item.StockId);
    }
    dataGridReceiptInvoices.ItemsSource = reseiptInvoices;
}

/// <summary>
/// Привязка отходных накладных к таблице.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void IdExpenditureInvoices_MouseEnter(object sender, MouseEventArgs e)

```

```

{
    List<ExpenditureInvoice> expenditureInvoices = _expenditureInvoiceService.GetAll() ?? new List<ExpenditureInvoice>();

    foreach (var item in expenditureInvoices)
    {
        item.Customer = _customerService.GetById((int)item.CustomerId);
        item.Stock = _stockService.GetById((int)item.StockId);
    }

    dataGridExpenditureInvoices.ItemsSource = expenditureInvoices;
}

/// <summary>
/// Инициализация контролов.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void IdExpenditureReceipt_MouseEnter(object sender, MouseEventArgs e)
{
    var stocks = _stockService.GetAll();
    stockCb.ItemsSource = _stockService.GetAll()?.Select(p => p.StockName);
    comboBoxExpenditureReceiptCompany.ItemsSource = _customerService.GetAll()?.Select(c => c.CustomerName);
    comboBoxExpenditureReceiptOperation.ItemsSource = new List<string>() { "Приход", "Отгрузка" };
    ReportComboBox.ItemsSource = stockCb.ItemsSource;
}

/// <summary>
/// Добавление приходных
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void AddProductsBtn_Click(object sender, RoutedEventArgs e)
{
    var stkId = _stockService.GetAll()
        .SingleOrDefault(p => p.StockName.Equals(stockCb.SelectedItem.ToString())).StockId;

    AddProdWindow add = new AddProdWindow(stkId);

    if (stockCb.SelectedItem == null && comboBoxExpenditureReceiptCompany.SelectedItem == null && datePickerExpenditureReceiptDate.SelectedDate == null)
    {
        MessageBox.Show("Заполните все данные");
    }
    else
    {
        _receiptInvoiceService.Insert(new ReceiptInvoice
        {
            ReceiptInvoiceDate = (NpgsqlDate)datePickerExpenditureReceiptDate.SelectedDate,
            CustomerId = _customerService.GetAll()
                .SingleOrDefault(p => p.CustomerName.Equals(comboBoxExpenditureReceiptCompany.SelectedItem.ToString())).CustomerId,
            StockId = _stockService.GetAll()
                .SingleOrDefault(p => p.StockName.Equals(stockCb.SelectedItem.ToString())).StockId
        });
        add.ShowDialog();
    }
}

```

```

    }

}

/// <summary>
/// Детализация приходной
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void dataGridReceiptInvoices_PreviewMouseDoubleClick(object sender, MouseButtonEventArgs e)
{
    var stockItem = (ReceiptInvoice)dataGridReceiptInvoices.SelectedItem;
    InvoiceDetailisationWindow details = new InvoiceDetailisationWindow(stockItem);
    details.ShowDialog();
}

/// <summary>
/// Изменение типа накладной.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ComboBoxExpenditureReceiptOperation_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (comboBoxExpenditureReceiptOperation.SelectedItem != null &&
        comboBoxExpenditureReceiptOperation.SelectedItem.ToString() == "Отгрузка")
    {
        stockCb.Visibility = Visibility.Visible;
        AddProductsBtn.Visibility = Visibility.Hidden;
        ExpenditureBtn.Visibility = Visibility.Visible;
    }
    else
    {
        stockCb.Visibility = Visibility.Visible;
        AddProductsBtn.Visibility = Visibility.Visible;
        ExpenditureBtn.Visibility = Visibility.Hidden;
    }
}

/// <summary>
/// Получение детализации расходной
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void dataGridExpenditureInvoices_PreviewMouseDoubleClick(object sender, MouseButtonEventArgs e)
{
    var stockItem = (ExpenditureInvoice)dataGridExpenditureInvoices.SelectedItem;
    EInvoiceDetailisationWindow edetails = new EInvoiceDetailisationWindow(stockItem);
    edetails.ShowDialog();
}

/// <summary>
/// Формирование расходной
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>

```

```

private void ExpenditureBtn_Click(object sender, RoutedEventArgs e)
{
    double markup = _stockService.GetAll()
        .SingleOrDefault(p => p.StockName.Equals(stockCb.SelectedItem.ToString()))
        .Markup;

    int stockId = _stockService.GetAll()
        .SingleOrDefault(p => p.StockName.Equals(stockCb.SelectedItem.ToString()))
        .StockId;

    EAddProdWindow add = new EAddProdWindow(markup, stockId);
    if (stockCb.SelectedItem == null && comboBoxExpenditureReceiptCompany.SelectedItem == null && date-
PickerExpenditureReceiptDate.SelectedDate == null)
    {
        MessageBox.Show("Заполните все данные");
    }
    else
    {
        _expenditureInvoiceService.Insert(new ExpenditureInvoice
        {
            ExpenditureInvoiceDate = (NpgsqlDate)datePickerExpenditureReceiptDate.SelectedDate,
            CustomerId = _customerService.GetAll()
                .SingleOrDefault(p => p.CustomerName.Equals(comboBoxExpenditureReceiptCompany.SelectedI-
tem.ToString())).CustomerId,
            StockId = _stockService.GetAll()
                .SingleOrDefault(p => p.StockName.Equals(stockCb.SelectedItem.ToString())).StockId
        });
        add.ShowDialog();
    }
}

/// <summary>
/// Инициализация товаров на складе
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ProductInStock_MouseEnter(object sender, MouseEventArgs e)
{
    List<ProductInStock> prodInStock = _productInStockService.GetAll() ?? new List<ProductInStock>();
    foreach (var item in prodInStock)
    {
        item.Product = _productService.GetById((int)item.ProductId);
        item.Stock = _stockService.GetById((int)item.StockId);
    }
    prodInStockGrid.ItemsSource = prodInStock;
}

/// <summary>
/// Получение отчета по складу
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonReportC_Click(object sender, RoutedEventArgs e)
{

```

```

        var stockId = _stockService.GetAll().SingleOrDefault(p => p.StockName.Equals(ReportComboBox.SelectedItem)).StockId;
        reports.GetReportS(stockId, ReportComboBox);
    }
    /// <summary>
    /// Получение отчета по всем складам
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonReportCA_Click(object sender, RoutedEventArgs e)
    {
        reports.GetReportAS();
    }
    /// <summary>
    /// Получение отчета
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonReportCK_Click(object sender, RoutedEventArgs e)
    {
        string dataFrom = datePickerExpenditureReceiptDateFrom.DisplayDate.ToString("yyyy-MM-dd");
        string dataTo = datePickerExpenditureReceiptDateTo.DisplayDate.ToString("yyyy-MM-dd");
        reports.GetReportDS(dataFrom, dataTo);
    }
    /// <summary>
    /// Получение отчета о наиболее доходных товарах
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>

    private void ButtonReportCL_Click(object sender, RoutedEventArgs e)
    {
        reports.GetReportD();
    }
    /// <summary>
    /// Выход в окно регистрации
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ExitBtn_Click(object sender, RoutedEventArgs e)
    {
        MainWindow main = new MainWindow();
        main.Show();
        this.Close();
    }
}
}

```

Класс *AddProdWindow*

```

using CourseApp.Models;
using CourseApp.Services;
using System.Linq;
using System.Windows;

namespace CourseApp
{

```

```

/// <summary>
/// Логика взаимодействия для AddProdWindow.xaml
/// </summary>
public partial class AddProdWindow : Window
{
    /// <summary>
    /// Инициализация сервисов
    /// </summary>
    IService<Product> _productService = new ProductService();
    IService<ReceiptPosition> _receiptPositionService = new ReceiptPositionService();
    IService<ReceiptInvoice> _receiptInvoiceService = new ReceiptInvoiceService();
    IService<ProductInStock> _productInStockService = new ProductInStockService();

    public int StckId { get; set; }
    /// <summary>
    /// Инициализация окна
    /// </summary>
    public AddProdWindow(int stockId)
    {
        InitializeComponent();
        this.StckId = stockId;
        ProductComboBox.ItemsSource = _productService.GetAll().Select(o => o.ProductName).ToList();
    }
    /// <summary>
    /// Кнопка добавления товаров в позицию
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void AddProdBtn_Click(object sender, RoutedEventArgs e)
    {
        int receiptInvoiceId = _receiptInvoiceService.GetAll().Select(p => p.ReceiptInvoiceId).Last();

        int productId = _productService.GetAll()
            .SingleOrDefault(p => p.ProductName.Equals(ProductComboBox.SelectedItem.ToString()))
            .EntityId;
        _receiptPositionService.Insert(new ReceiptPosition
        {
            CountProduct = double.Parse(CountBox.Text),
            ProductId = productId,
            ReceiptInvoiceId = receiptInvoiceId
        });

        if (_productInStockService.GetAll().Where(o => o.StckId == StckId && o.ProductId == productId).Count() == 0)
        {
            _productInStockService.Insert(new ProductInStock
            {
                ProductId = productId,
                CountProduct = double.Parse(CountBox.Text),
                StockId = StckId
            });
        }
        else
        {

```

```

        var id = _productInStockService.GetAll().SingleOrDefault(
            o => o.StockId.Equals(StockId) && o.ProductId.Equals(productId)).Id;
        double countProductsInStock = _productInStockService.GetAll().Where(o => o.StockId == StockId && o.ProductId == productId).Select(o => o.CountProduct).FirstOrDefault();
        _productInStockService.Update(new ProductInStock
        {
            Id = id,
            CountProduct = double.Parse(CountBox.Text) + countProductsInStock
        });
    }

    CountBox.Text = "";
    ProductComboBox.SelectedIndex = 0;
}
}
}

```

Класс *EAddProdWindow*

```

using CourseApp.Models;
using CourseApp.Services;
using System;
using System.Linq;
using System.Windows;

namespace CourseApp
{
    /// <summary>
    /// Логика взаимодействия для EAddProdWindow.xaml
    /// </summary>
    public partial class EAddProdWindow : Window
    {
        /// <summary>
        /// Инициализация сервисов
        /// </summary>
        IService<Product> _productService = new ProductService();
        IService<ExpenditurePosition> _expenditurePositionService = new ExpenditurePositionService();
        IService<ExpenditureInvoice> _expenditureInvoiceService = new ExpenditureInvoiceService();
        IService<ProductInStock> _productInStockService = new ProductInStockService();
        double Markup { get; set; }
        /// <summary>
        /// не нужно
        /// </summary>
        int StockId { get; set; }
        /// <summary>
        /// Инициализация окна
        /// </summary>
        public EAddProdWindow(double markup, int stockId)
        {
            InitializeComponent();
            this.Markup = markup;
            this.StockId = stockId;
            ProductComboBox.ItemsSource = _productService.GetAll().Select(o => o.ProductName).ToList();
        }
        /// <summary>

```

```

/// Кнопка добавления товаров в позицию
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void AddProdBtn_Click(object sender, RoutedEventArgs e)
{
    double productPrice = _productService.GetAll()
        .SingleOrDefault(p => p.ProductName.Equals(ProductComboBox.SelectedItem.ToString()))
        .ProductPrice;
    int expenditureInvoiceId = _expenditureInvoiceService.GetAll().Select(p => p.ExpenditureInvoiceId).Last();

    int productId = _productService.GetAll()
        .SingleOrDefault(p => p.ProductName.Equals(ProductComboBox.SelectedItem.ToString()))
        .EntityId;

    double countProductsInStock = _productInStockService.GetAll().Where(o => o.StockId == StockId && o.ProductId == productId).Select(o => o.CountProduct).FirstOrDefault();

    if (countProductsInStock < double.Parse(CountBox.Text))
    {
        MessageBox.Show($"Недостаточно товара в количестве {double.Parse(CountBox.Text) - countProductsInStock}");
    }
    else
    {
        _expenditurePositionService.Insert(new ExpenditurePosition
        {
            CountProduct = double.Parse(CountBox.Text),
            ProductId = productId,
            ExpenditureInvoiceId = expenditureInvoiceId,
            ProductPrice = (Math.Round(productPrice + productPrice * (Markup / 100.0), 3))
        });

        var id = _productInStockService.GetAll().SingleOrDefault(o => o.StockId.Equals(StockId) && o.ProductId.Equals(productId)).Id;

        _productInStockService.Update(new ProductInStock
        {
            Id = id,
            CountProduct = countProductsInStock - double.Parse(CountBox.Text)
        });
        CountBox.Text = "";
        ProductComboBox.SelectedIndex = 0;
    }
}

```

Класс *InvoiceDetailisationWindow*

```

using CourseApp.Models;
using CourseApp.Services;

```



```

using System.Collections.Generic;
using System.Linq;
using System.Windows;

namespace CourseApp
{
    /// <summary>
    /// Логика взаимодействия для InvoiceDetalisationWindow.xaml
    /// </summary>
    public partial class InvoiceDetalisationWindow : Window
    {
        /// <summary>
        /// Объявление сервисов для работы с детализацией накладных
        /// </summary>
        IService<ReceiptPosition> _receiptPositionService = new ReceiptPositionService();
        IService<Product> _productService = new ProductService();
        /// <summary>
        /// Инициализация окна для работы с накладными
        /// </summary>
        /// <param name="stockItem"></param>
        public InvoiceDetalisationWindow(ReceiptInvoice stockItem)
        {
            InitializeComponent();
            List<ReceiptPosition> positions = _receiptPositionService.GetAll()?.Where(o => o.Re-
ceiptInvoiceId == stockItem.ReceiptInvoiceId).ToList() ?? new List<ReceiptPosition>();

            foreach (var item in positions)
            {
                item.Product = _productService.GetById((int)item.ProductId);
                var id = _productService.GetAll().SingleOrDefault(o => o.EntityId.Equals(item.ProductId));
                item.FullPrice = id.ProductPrice * item.CountProduct;
            }
            InvoicePositionsGrid.ItemsSource = positions;
            InvoicePositionsGrid.IsReadOnly = true;
        }
    }
}

```

Класс *EInvoiceDetalisationWindow*

```

using CourseApp.Models;
using CourseApp.Services;
using System.Collections.Generic;
using System.Linq;
using System.Windows;

namespace CourseApp
{
    /// <summary>
    /// Логика взаимодействия для EInvoiceDetalisationWindow.xaml
    /// </summary>
    public partial class EInvoiceDetalisationWindow : Window
    {
        /// <summary>
        /// Инициализация сервисов
        /// </summary>

```

```

IService<ExpenditurePosition> _expenditurePositionService = new ExpenditurePositionService();
IService<Product> _productService = new ProductService();
/// <summary>
/// Инициализация позиций расходной
/// </summary>
/// <param name="stockItem"></param>
public EInvoiceDetailisationWindow(ExpenditureInvoice stockItem)
{
    InitializeComponent();
    List<ExpenditurePosition> positions = _expenditurePositionService.GetAll()?.Where(o => o.ExpenditureInvoiceId == stockItem.ExpenditureInvoiceId).ToList() ?? new List<ExpenditurePosition>();
    foreach (var item in positions)
    {
        item.Product = _productService.GetById((int)item.ProductId);
        item.FullPrice = item.ProductPrice * item.CountProduct;
    }
    EInvoicePositionsGrid.ItemsSource = positions;
    EInvoicePositionsGrid.IsReadOnly = true;
}
}
}

```

Класс *CustomerServiceTest*

```

using System;
using System.Linq;
using CourseApp.Models;
using CourseApp.Services;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace CourseAppTests
{
    [TestClass]
    public class CustomerServiceTest
    {
        IService<Customer> service = new CustomerService();
        static Customer testCustomer = new Customer() { CustomerName = "TestCustomerName", Description = "TestCustomerDescription" };

        [TestMethod]
        public void AGetAllTest()
        {
            Assert.AreEqual(true, service.GetAll().Count > 0);
        }

        [TestMethod]
        public void BInsertTest()
        {
            Assert.AreEqual(true, service.Insert(testCustomer));
        }

        [TestMethod]
        public void CUpdateTest()
        {
            var updateItem = service.GetAll()
                .FirstOrDefault(c => c.CustomerName == testCustomer.CustomerName && c.Description == testCustomer.Description);

```

```

        updateItem.CustomerName = "NewTestName";

        Assert.AreEqual(true, service.Update(updateItem));

        testCustomer.CustomerId = updateItem.CustomerId;
        updateItem.CustomerName = testCustomer.CustomerName;

        service.Update(updateItem);
    }

    [TestMethod]
    public void DGetByIdTest()
    {
        var findItem = service.GetById(testCustomer.CustomerId);

        Assert.AreEqual(findItem.CustomerName, testCustomer.CustomerName);
        Assert.AreEqual(findItem.CustomerId, testCustomer.CustomerId);
        Assert.AreEqual(findItem.Description, testCustomer.Description);
    }

    [TestMethod]
    public void EDeleteTest()
    {
        Assert.AreEqual(true, service.Delete(testCustomer));
    }
}

```

Класс *ExpenditureInvoiceServiceTest*

```

using System;
using CourseApp.Models;
using CourseApp.Services;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NpgsqlTypes;

namespace CourseAppTests
{
    [TestClass]
    public class ExpenditureInvoiceServiceTest
    {
        IService<ExpenditureInvoice> service = new ExpenditureInvoiceService();

        static ExpenditureInvoice testExpend = new ExpenditureInvoice()
        {
            ExpenditureInvoiceDate = (NpgsqlDate)DateTime.Now,
            CustomerId = 1,
            StockId = 2
        };

        [TestMethod]
        public void BGetAllTest()
        {
            Assert.AreEqual(true, service.GetAll().Count > 0);
        }
    }
}

```

```

[TestMethod]
public void EDeleteTest()
{
    Assert.AreEqual(true, service.Delete(testExpend));
}
}
}

```

Класс *ProductServiceTest*

```

using System;
using System.Linq;
using CourseApp.Models;
using CourseApp.Services;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace CourseAppTests
{
    [TestClass]
    public class ProductServiceTest
    {
        IService<Product> service = new ProductService();
        static Product testProd = new Product() { ProductName = "TestName" };

        [TestMethod]
        public void AGetAllTest()
        {
            Assert.AreEqual(true, service.GetAll().Count > 0);
        }

        [TestMethod]
        public void BInsertTest()
        {
            Assert.AreEqual(true, service.Insert(testProd));
        }

        [TestMethod]
        public void CUpdateTest()
        {
            var updateItem = service.GetAll()
                .FirstOrDefault(c => c.ProductName == testProd.ProductName);

            updateItem.ProductName = "NewTestName";

            Assert.AreEqual(true, service.Update(updateItem));

            testProd.EntityId = updateItem.EntityId;
            updateItem.ProductName = testProd.ProductName;

            service.Update(updateItem);
        }

        [TestMethod]
        public void DGetByIdTest()

```

```

    {
        var findItem = service.GetById(testProd.EntityId);

        Assert.AreEqual(findItem.ProductName, testProd.ProductName);
        Assert.AreEqual(findItem.EntityId, testProd.EntityId);
    }

    [TestMethod]
    public void EDeleteTest()
    {
        Assert.AreEqual(true, service.Delete(testProd));
    }
}

```

Класс *RoleServiceTest*

```

using System;
using System.Linq;
using CourseApp.Models;
using CourseApp.Services;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace CourseAppTests
{
    [TestClass]
    public class RoleServiceTest
    {
        IService<Role> service = new RoleService();
        static Role testRole = new Role() { RoleName = "TestName", RoleKey = "test" };

        [TestMethod]
        public void AGetAllTest()
        {
            Assert.AreEqual(true, service.GetAll().Count > 0);
        }

        [TestMethod]
        public void EDeleteTest()
        {
            Assert.AreEqual(true, service.Delete(testRole));
        }
    }
}

```

Класс *StockServiceTest*

```

using System;
using System.Linq;
using CourseApp.Models;
using CourseApp.Services;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace CourseAppTests
{
    [TestClass]

```

```

public class StockServiceTest
{
    IService<Stock> service = new StockService();
    Stock testStock = new Stock() { StockId = 1, StockName = "TestName", Markup = 10 };
    [TestMethod]
    public void BInsertTest()
    {
        Assert.AreEqual(true, service.Insert(testStock));
    }
    [TestMethod]
    public void BInsertTest2()
    {
        Assert.AreEqual(true, service.Insert(testStock));
    }

    [TestMethod]
    public void BInsertTest3()
    {
        Assert.AreEqual(true, service.Insert(testStock));
    }
}

```

Класс *UserServiceTest*

```

using System;
using System.Linq;
using CourseApp.Models;
using CourseApp.Services;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace CourseAppTests
{
    [TestClass]
    public class UserServiceTest
    {
        IService<User> service = new UserService();
        static User testUser = new User()
        {
            RoleKey = "admin",
            UserName = "TestCustomerDescription",
            UserPass = "TestCustomerDescription",
            FullName = "TestCustomerDescription",
        };

        [TestMethod]
        public void AGetAllTest()
        {
            Assert.AreEqual(true, service.GetAll().Count > 0);
        }

        [TestMethod]
        public void BInsertTest()
        {
            Assert.AreEqual(true, service.Insert(testUser));
        }
    }
}

```

```

    }

[TestMethod]
public void CUpdateTest()
{
    var updateItem = service.GetAll()
        .FirstOrDefault(c => c.UserName == testUser.UserName && c.UserPass == testUser.UserPass);

    updateItem.UserName = "NewTestName";

    Assert.AreEqual(true, service.Update(updateItem));

    testUser.UserId = updateItem.UserId;
    updateItem.UserName = testUser.UserName;

    service.Update(updateItem);
}

[TestMethod]
public void DGetByIdTest()
{
    var findItem = service.GetById(testUser.UserId);

    Assert.AreEqual(findItem.UserName, testUser.UserName);
    Assert.AreEqual(findItem.UserId, testUser.UserId);
    Assert.AreEqual(findItem.UserPass, testUser.UserPass);
}

[TestMethod]
public void EDeleteTest()
{
    Assert.AreEqual(true, service.Delete(testUser));
}
}

```

ПРИЛОЖЕНИЕ Б
(обязательное)
Функциональная схема приложения

ПРИЛОЖЕНИЕ В

(обязательное)

Руководство системного администратора

1. Общие сведения о программе.

Программное приложение предназначено для автоматизации работы с документами и с пользователями. Программное обеспечение имеет возможности просмотра, создания и редактирования. Из аппаратной части для работы приложения необходим сервер, на котором будет располагаться само приложение. Минимальные системные требования к серверу следующие:

- 4 ГБ оперативной памяти (рекомендуется 4 ГБ и выше);
- 32 или 64-разрядные системы: компьютер, оборудованный процессором не ниже *Intel Pentium* (рекомендуется *Intel Core i3* и выше), рекомендуется применять процессоры с частотой не менее 2.0 ГГц;
- приложение может исполняться на любых ОС на которых установлен *.NET Framework* версии 4.0 и выше, а также снабженные сетевыми протоколами: *TCP/IP*, поддерживающими именованные каналы;
- для работы приложения необходим доступ к базе данных, для этого нужен сервер для хранения самой базы, и система управления базой данных. Минимальные системные требования к серверу базы данных:
 - 2 ГБ оперативной памяти (рекомендуется 4 ГБ и выше);
 - 32 или 64-разрядные системы: компьютер, оборудованный процессором не ниже *Intel Pentium* (рекомендуется *Intel Core i3* и выше), рекомендуется применять процессоры с частотой не менее 2.0 ГГц;
 - 10 ГБ свободного места на жестком диске (рекомендуется 50 ГБ и более).

2. Настройка программы.

Перед началом установки приложения и базы данных на сервер необходимо убедиться в том, что они удовлетворяют минимальным системным требованиям. После чего необходимо скопировать все файлы приложения на сервер. Если готовая база данных отсутствует, то необходимо войти в систему управления базой данных и создать пустую базу. Если готовая база данных уже есть, и она не находится на удаленном сервере, ее необходимо подключить к серверу.

Если база данных располагается на удаленном сервере, то ни каких действий производить не нужно. После загрузки всех файлов приложения на сервер и завершения работы с базой данных необходимо отредактировать файл *App.config*.

В файле *App.config* необходимо изменить показанный на рисунке В.1 код в соответствии с настройками базы данных.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="MyDatabase" connectionString="Server=localhost;Port=5432;Database=testDb;
      User Id=postgres;Password=1;" providerName="PostgreSQL" />
  </connectionStrings>
</configuration>
```

Рисунок В.1 – Фрагмент кода для изменения

После проведения этих настроек можно запускать приложение. В случае если не использовалась готовая база данных, приложение подключится к пустой базе и создаст необходимые таблицы. После создания таблицы нужно заполнить данными, указав как минимум одного администратора.

3. Проверка программы

Программа прошла тестирование, а также проверку валидации и верификации данных. Результаты тестирования отражены в пункте 3.1 и 3.2, что подтверждает работоспособность приложения.

4. Структура программы

Разработанное программное обеспечение состоит из двух модулей: модуль для работы с документами и модуль для работы с сотрудниками. Оба модуля имеют набор функций для выполнения соответствующих работ. Разработанное приложение на прямую взаимодействует лишь с *WPF* и не взаимодействует с другими приложениями, однако для удобства просмотра результатов может понадобиться приложение для просмотра файлов в формате *xml*.

ПРИЛОЖЕНИЕ Г

(обязательное)

Руководство программиста

1. Назначение и условия применения программы.

Программное приложение предназначено для автоматизации работы с документами и пользователями. Программное обеспечение имеет возможности просмотра, создания, редактирования и печати текстовых документов с использованием подстановки данных и цифровой подписи. Для работы с приложением необходимы:

- объем оперативной памяти устройства не менее 2 ГБ;
- приложение для просмотра *xml* файлов (обычно встроено в браузер);
- любое устройство с установленной *Visual Studio* (ноутбук, нетбук или стационарный компьютер).

2. Обращение к программе.

Для просмотра и редактирования исходного кода необходимо открыть файл *CourseApp.sln* с помощью среды разработки *Microsoft Visual Studio*.

3. Входные и выходные данные.

Входными данными в процессе работы приложения являются данные введенные пользователями и данные полученные из базы данных либо файлового хранилища.

Выходными данными являются текстовые и графические данные, отображаемые на экране или выводимые на печать с помощью принтера.

Кодирование информации производится стандартными средствами языка программирования *C#*.

ПРИЛОЖЕНИЕ Д

(обязательное)

Руководство пользователя

1. Введение.

Приложение применяется для ведения складского учета предприятия.

Программа позволяет вести удобный учет товаров, поступающих на предприятие и не требует особых знаний для пользования.

2. Условия применения.

Для работы с приложением необходимы:

- объем оперативной памяти устройства не менее 2 ГБ;
- приложение для просмотра *xml* файлов (обычно встроено в браузер);
- любое устройство с установленным приложением (ноутбук, нетбук или стационарный компьютер).

3. Подготовка к работе.

Перед началом работы необходимо установить полный софт, зайти в приложение и использовать его по назначению. Больше никаких действий со стороны пользователя не требуется.

4. Описание операций.

В ходе работы с приложением необходимо пройти регистрацию. Для этого нужно обратиться к системному администратору для получения персонального логина и пароль. После получения этих данных нужно запустить приложение и авторизоваться под своим логином и паролем.

В зависимости от роли (администратор, менеджер, кладовщик), пользователь получает различный функционал, который описан в пункте 3.3.

5. Аварийные ситуации.

При возникновении какой-либо аварийной ситуации следует обратиться к программисту либо к системному администратору.

6. Рекомендации по освоению.

Приложение применяется только для конкретной предметной области – ведение складского учета на предприятии.

Перед эксплуатацией приложения рекомендуется изучить его предметную область.