**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании 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

Установа адукацыі «Гомельскі дзяржаўны   
тэхнічны ўніверсітэт імя П.В. Сухога»

Факультэт аўтаматызаваных і інфармацыйных сістэм

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РЭЦЭНЗІЯ**

**на курсавы праект (работу)**

па дысцыпліне \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,

выканана студэнтам \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

групы \_\_\_\_\_\_\_\_\_\_

I. Пералік заўваг па тэксту курсавога праекта (работы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

II. Агульная характарыстыка работы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[Введение 2](#_Toc65968606)

[1 Обзор программго-технических средств для решения поставленной задачи 4](#_Toc65968607)

[1.1 Особенности языка программирования *С#* 4](#_Toc65968608)

[1.2 Особенности среды разработки 6](#_Toc65968609)

[1.3 Особенности объектно-реляционной СУБД 7](#_Toc65968610)

# **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 работы заключается в увеличении объема информации поступлении товаров на склады и их отгрузке, быстрый и точный подсчет данных.

Цель курсовой работы – создание программного комплекса для упрощения учета поступления товаров на склады предприятия.

Задачи курсовой работы заключаются в следующем – создание программного комплекса на языке программирования *С#* и СУБД *PostgreSQL* для учета поступления товаров на склады предприятия, а также написание блок-схем и тестов для этого комплекса.

1. **ОБЗОР ПРОГРАММГО-ТЕХНИЧЕСКИХ СРЕДСТВ ДЛЯ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ**

## **Особенности языка программирования *С#***

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 *C#* 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.

Основное преимущество языка *C#* 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.

Язык очень быстрый, его компиляция и выполнение слишком малы.

*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.

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 *C#* 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. Языкочень 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.

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 *C#* содержит такие функции, как «Свойства» и «Индексаторы», хотя они недоступны в языке *Java*.

*C#* 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.

*C#* полностью основан на платформе *Microsoft* .*Net*, поэтому этот язык не является гибким.

В *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.

## **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*.

*VS* 2019 обеспечивает более быструю отладку. Можно искать различные ключевые слова во время отладки.

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

## **Особенности объектно-реляционной СУБД**

*PostgreSQL* – объектно-реляционная система управления базами данных

*PostgreSQL* – это популярная свободная объектно-реляционная система управления базами данных. *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 Тбайт;
* максимальный размер записи — 1,6 Тбайт;
* максимальный размер поля — 1 Гбайт;
* 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.

**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.

**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*](https://web-creator.ru/articles/mysql) / *MariaDB*, но всё же эта пара с заметным отрывом опережает по частоте использования остальные [системы управления базами данных](https://web-creator.ru/articles/about_databases). При этом в разработке сложных сайтов и веб-приложений *PostgreSQL* опережает по использованию *MySQL* и *MariaDB*. Большинство [фреймворков](https://web-creator.ru/articles/about_frameworks) (например, [*Ruby* *on* *Rails*](https://web-creator.ru/articles/about_ruby_on_rails), [*Yii*](https://web-creator.ru/articles/yii), [*Symfony*](https://web-creator.ru/articles/symfony), [*Django*](https://web-creator.ru/articles/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.

Сравнение популярнейших СУБД Таблица 1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СУБД | Тип | Операционные системы | Лицензия | Исходный код | Поддержка |
| *Oracle* *Database* | Мульти-модельная | *Linux, Microsoft Windows, Oracle Solaris, IBM AIX, HP-UX* | Коммерческая | Закрытый | Платная |
| *MySQL* | Реляционная | *Linux, Microsoft Windows, Oracle Solaris, macOS, FreeBSD* | GNU GPL и коммерческая | Открытый | Платная |
| *SQL* *Server* | Реляционная | *Linux, Microsoft Windows* | Коммерческая | Закрытый | Бесплатная |
| *PostgreSQL* | Объектно-реляционная | *Linux, Microsoft Windows, Oracle Solaris, IBM AIX, macOS, HP-UX, QNX* | Открытое ПО, разрешительная лицензия | Открытый | Платная |

# **АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА ПО УЧЕТУ ПОСТУПЛЕНИЯ ТОВАРОВ НА СКЛАДЫ ПРЕДПРИЯТИЯ**

## **2.1 Предметная область комплекса программ для учета поступления товаров на склады предприятия**

Необходимо разработать приложение для учета поступления товаров на склады предприятия. Товары поступают по приходным накладным на каждый конкретный склад, а отгружаются по расходным. В накладных содержится информация о дате прихода или отгрузки, кому отгружается и с какого склада, наименование товаров, количество единиц и их стоимость. Нельзя отгрузить товар, не имеющийся на складе. Ввод и оформление накладных осуществляют кладовщики. При выписке расходной накладной закладывается наценка от 10% до 25% в зависимости от склада. Необходимо сформировать отчеты по остаткам на каждом складе и сводный предприятию. За указанный период вывести данные о полученной прибыли каждым складом, найти наиболее доходный товар. Все отчеты сохранить в виде *XML-*файла с возможностью открытия в электронных таблицах. Обеспечить многопользовательский доступ к системе: администратор, кладовщик, менеджер. Администратор имеет полные права доступа, кладовщик может вносить данные только по своему складу, менеджер может видеть информацию о товарах на всех складах.

В качестве источника данных использовать СУБД *PostgreSQL* и технологию доступа *ADO*. Для реализации использовать средства языка программирования *C*# *WPF Application* и шаблоны проектирования. Для доступа к данным использовать *LINQ*. Для тестирования разработанных классов использовать модульные тесты.

## **2.2 Общая схема программного комплекса по учету поступления товаров на склады предприятия**

Для работы с программным обеспечением была составлена база данных в СУБД *PostgreSQl* [4]. В качестве сущностей БД вступают:

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

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, которая изображена на рисунке 2.1.

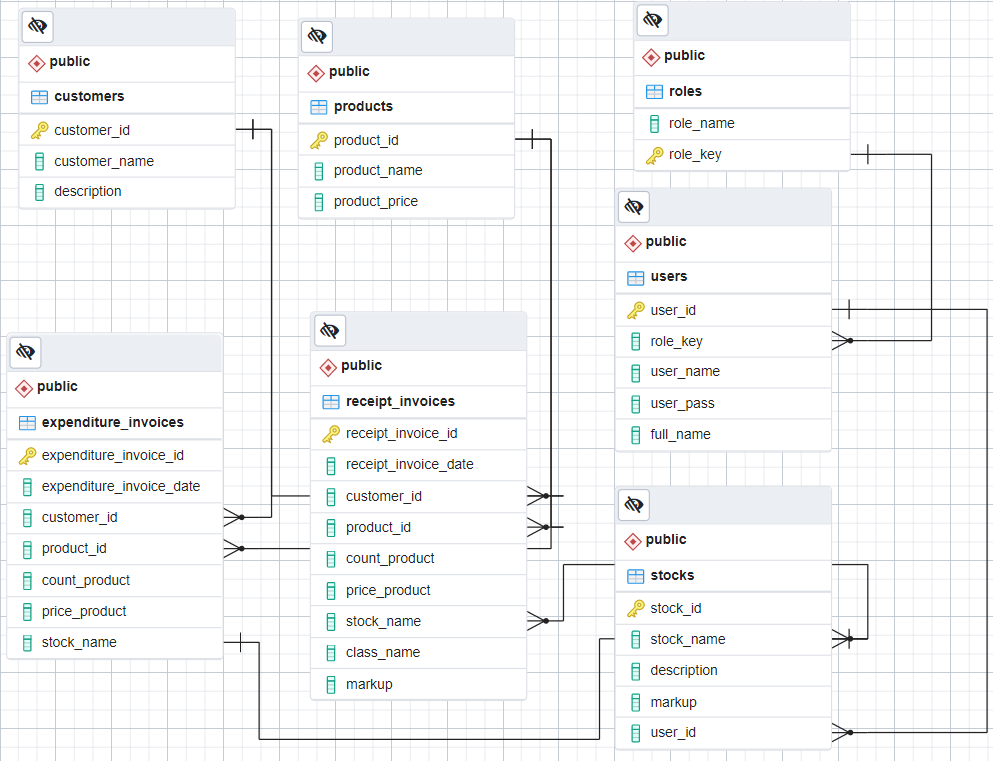


Рисунок 2.1 – Диаграмма базы данных

Для работы с программным обеспечением были написаны шесть моделей:

* *Customer*;
* *ExpenditureInvoice*;
* *Product*;
* *ReceiptInvoice*;
* *Role*;
* *Stock*;
* *User*.

А также была создана папка *Decorator*, которая содержит семь классов:

* *AbstractProduct*;
* *Product*;
* *ProductInStockDecorator*;
* *ProductInStockWithAverageMarkup*;
* *ProductInStockWithBigMarkup*;
* *ProductInStockWithCustomMarkup*;
* *ProductInStockWithSmallMarkup*.

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*](https://ru.wikipedia.org/wiki/Data_Access_Object)[*Object*](https://ru.wikipedia.org/wiki/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 в *Connection Pool* [5].

Интерфейс *IServise* является главным компонентом иерархии *DAO* в разработанном ПО для описания общих методов, которые будут использоваться при взаимодействии с базой данных [5]. В интерфейсе *IServise* это методы поиска, удаление по ключу, обновление и т.д. Диаграмма этих классов представлена на рисунке 2.2.

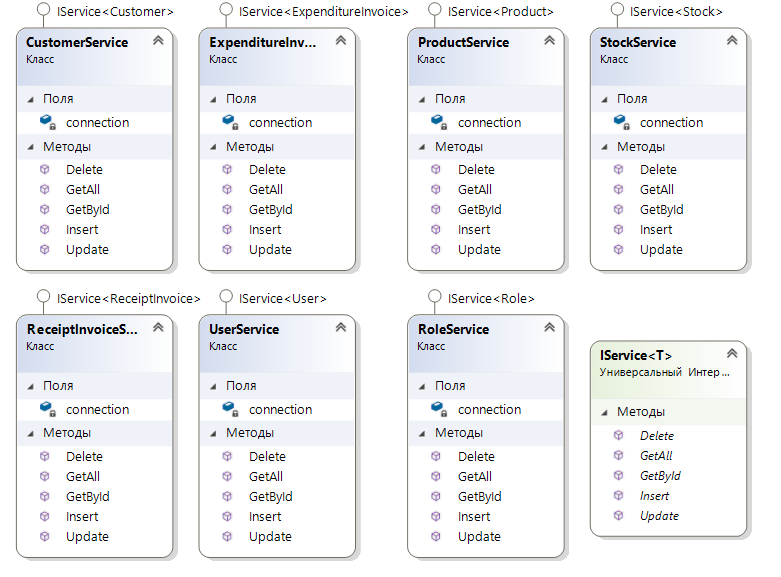


Рисунок 2.2 Диаграмма классов, наследуемых от *IServise*

## **2.3 Архитектура ADO.NET в программном комплексе по учету поступления товаров на склады предприятия**

Для хранения данных в разработанном программном обеспечении используется система управления базами данных *PostgreSQL*. Чтобы осуществлять связь между базой данных и приложением на *C*# необходим посредник. Таким посредником в разработанном программном обеспечении является технология *ADO*.*NET*.

*ADO*.*NET* предоставляет собой технологию работы с данными, которая является одной из основных компонентов платформы .*NET Framework*. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Функционал *ADO*.*NET* построен таким образом, чтобы предоставить разработчикам унифицированный интерфейс для работы с самыми различными СУБД.

Для взаимодействия с объектами базы данных в *ADO*.*NET* присутствует ограниченный круг объектов: *Connection*, *Command*, *DataReader*, *DataSet* и *DataAdapter*. Объект *Command* позволяет выполнять операции с данными из БД. С помощью объекта *Connection* происходит установка подключения к источнику данных. Объект *DataReader* считывает полученные в результате запроса данные. Объект *DataAdapter* является посредником между *DataSet* и источником данных. И Объект *DataSet* предназначен для хранения данных из БД и позволяет работать с ними независимо от базы данных [6 c.528].

Схематично архитектуру *ADO*.*NET* в разработанном ПО можно представить следующим образом (рисунок 2.3).

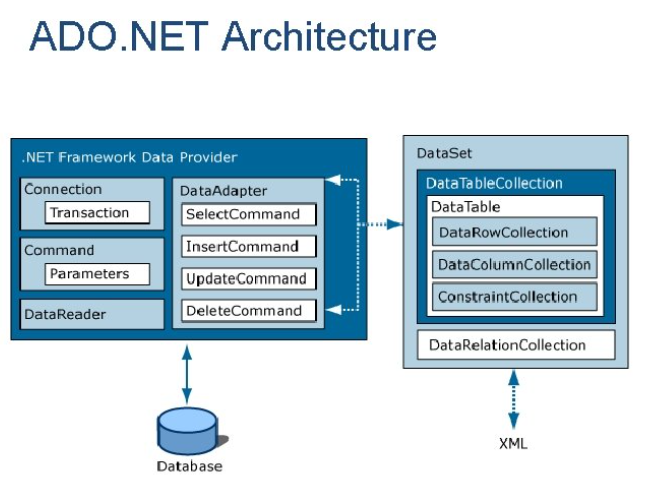


Рисунок 2.3 – Архитектура *ADO*.*NET* в разработанном ПО

## **2.4 Сущности и атрибуты сущностей приложения по учету поступления товаров на склады предприятия**

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

После изучения предметной области были выделены 7 основных сущностей:

*Roles, Users, Stocks, Customers, Expenditure*\_*Invoices, Receipt*\_*Invoices, Products.*

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.

Атрибут представляет свойство, которое описывает некоторую характеристику, существо.

Сущность – это объект в базе данных, в котором хранятся данные. Сущность может представлять собой нечто вещественное (предмет, место) или абстрактное (операция, отдел). В физической модели сущность называется таблицей.

Сущность *Roles* имеет следующие поля:

* *role*\_*name* (*character varying* (40)), которое обозначает название роли;
* *role*\_*key* (*character varying* (15)), которое обозначает ключевую роль.

Сущность *Users* имеет следующие поля:

* *user*\_*id* (*integer*), которое обозначает номер пользователя;
* *role*\_*key* (*character varying* (15)), которое обозначает ключевую роль;
* *user*\_*name* (*character varying* (40)), которое обозначает имя пользователя;
* *user*\_*pass* (*character varying* (40)), обозначающее пароль пользователя;
* *full\_name (character varying (50))*, которое обозначает полное имя.

Сущность *Stocks* имеет следующие поля:

* *stock*\_*id* (*integer*), которое обозначает номер склада;
* *stock*\_*name* (*character varying* (40)), которое обозначает название склада;
* *description* (*character varying* (50)), которое обозначает описание склада;
* *markup* (*real*), которое обозначает надбавку;
* *user*\_*id* (*integer*), которое обозначает номер пользователя.

Сущность Customers имеет следующие поля:

* *customer*\_*id* (*integer*), которое обозначает номер клиента;
* *customer*\_*name* (*character varying* (40)), которое обозначает имя клиента;
* *description* (*character varying* (50)), обозначающее описание клиента
* e*xpenditure*\_*Invoices\_id* (*integer*), номер расходной накладной;

Сущность *Expenditure*\_*Invoices* имеет следующие поля:

* *customer*\_*id* (*integer*), которое обозначает номер клиента;
* *stock*\_*name* (*varchar*(40)), которое обозначает название склада;
* *product*\_*id* (*integer*), которое обозначает номер продукта;
* *count*\_*product* (*real*), которое обозначает количество продуктов;
* *price*\_*product* (*real*), которое обозначает цену товара.

Сущность *Receipt*\_*Invoices* имеет следующие поля:

* *receipt*\_*invoice*\_*id* (*integer*), обозначающее номер приходной накладной;
* *customer\_id (integer)*, которое обозначает номер клиента;
* *receipt*\_*invoice*\_*date* (*date*), обозначающее дату приходной накладной
* *stock*\_*name* (*varchar*(40)), которое обозначает название склада;
* *product*\_*id* (*integer*), которое обозначает номер продукта;
* *count*\_*product* (*real*), которое обозначает количество продуктов;
* *price*\_*product* (*real*), которое обозначает цену товара.
* *class*\_*name* (*varchar*(40)), которое обозначает название класса;
* *markup* (*real*), которое обозначает надбавку.

Сущность *Products* имеет следующие поля:

* *product*\_*id* (*integer*), которое обозначает номер продукта;
* *product*\_*name* (*character varying* (40)),
* *product*\_*price* (*real*), которое обозначает цену продукта.

**2.5 Обоснование соответствия разработанной базы данных требованиям нормализации**

Целью нормализации реляционной базы данных является устранение недостатков структуры базы данных, приводящих к избыточности, которая, в свою очередь, потенциально приводит к различным аномалиям и нарушениям целостности данных.

Данные таблицы соответствуют требованиям нормализации реляционных БД. Таблицы находятся в первой нормальной форме, т. к. все их поля являются простыми (неделимыми), нет повторяющихся полей и каждый столбец таблиц хранит одно единственное значение и не является ни списком, ни множеством значений. Таблицы находятся во второй нормальной форме, т.к. они находятся в первой нормальной форме, имеют простые первичные ключи и каждое не ключевое поле функционально зависит от первичного ключа.

Таблицы находится 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 *int* и является автоинкрементным. Все первичные ключи в сущностях спроектированной структуры являются простыми и однозначно идентифицируют запись.

**2.6 Паттерн «Декоратор»**

Паттерн «Декоратор» предоставляет гибкую альтернативу подклассу для динамического расширения функциональности. Идея паттерна декоратора состоит в том, чтобы обернуть существующий класс, добавить к нему другие функции, а затем предоставить тот же интерфейс внешнему миру. Он используется для расширения или изменения функциональности во время выполнения.

Преимущества паттерна декоратора:

* динамически добавляет функциональность к существующим объектам;
* альтернатива подклассу;
* гибкий дизайн;
* поддерживает принцип открытости и закрытости.

На рисунке 2.4 изображена диаграмма классов которая показывает дизайн шаблона декоратора.

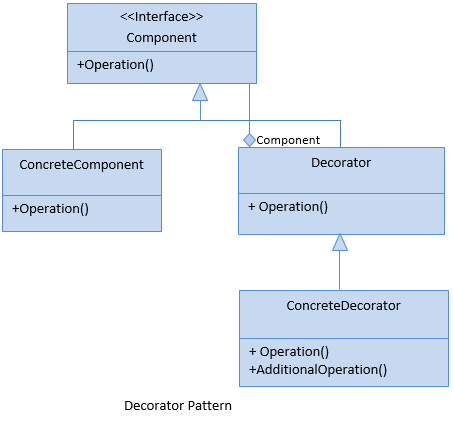


Рисунок 2.4 – Дизайн шаблона «Декоратор»

*Component*определяет интерфейс фактического объекта, который требует динамического добавления функциональных возможностей к *Concrete* *Components*.

*ConcreteComponent* – фактический объект, в который функции могут быть добавлены динамически.

*Decorator* определяет интерфейс для всех динамических функций, которые могут быть добавлены в *ConcreteComponent*.

*ConcreteDecorator*–все функции, которые можно добавить в *Concrete Component*. Каждая необходимая функциональность будет представлять собой один класс *ConcreteDecorator*.

Декорированный класс – это базовый класс конкретных классов-декораторов, наследующий от базового класса компонентов. Таким образом, декораторы можно использовать вместо конкретных элементов.

Диаграмма классов паттерна «Декоратор» расположена на рисунке 2.5.

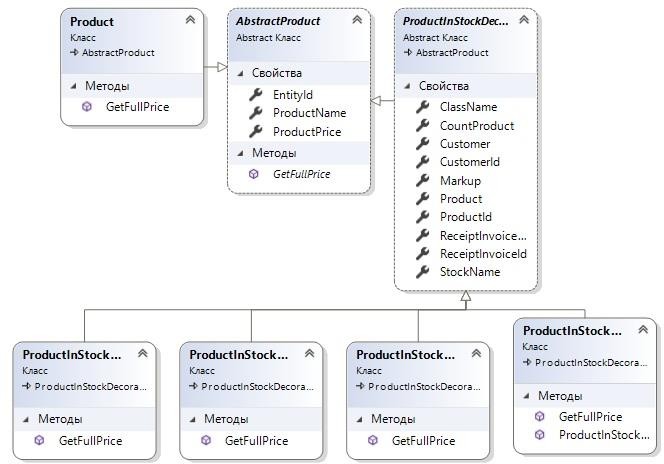


Рисунок 2.5 – Диаграмма классов паттерна «Декоратор»

Декоратор переадресует внешние вызовы декорируемому объекту сопровождая их наложением дополнительных обязанностей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Особенности платформы *WPF*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: https://metanit.com/sharp/wpf/1.php – Дата доступа: 12.04.2021
2. Рихтер Дж. *CLR* *VIA* *C#.* Программирование на платформе *Microsoft*.
3. .NET Framework 4.5 на языке *C#* / Рихтер Дж. 4-е изд., перераб. и доп. – Питер: 2019 – 896 с.
4. СУБД *PostgreSQL*: Свободная энциклопедия. – Электрон.данные. – Режим доступа: <http://bourabai.kz/dbt/servers/postgresql.htm>– Дата доступа: 12.04.2021
5. *Data* *Access* *Object*: Свободная энциклопедия. – Электрон.данные. – Режим доступа: https://habr.com/ru/post/262243/ – Дата доступа: 12.04.2021
6. 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.