

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет инженерно-экономический
Кафедра экономической информатики
Дисциплина «Программирование сетевых приложений»

«К ЗАЩИТЕ ДОПУСТИТЬ»

Руководитель курсового проекта

Ассистент кафедры ЭИ

_____. А.П. Лыщик

_____.2022

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**«ПОДСИСТЕМА УЧЕТА И РЕГИСТРАЦИИ ПОСТУПЛЕНИЯ
ТОВАРОВ НА СКЛАД ЛОГИСТИЧЕСКОЙ КОМПАНИИ»**

БГУИР КР 1-40 05 01-08 024 ПЗ

Выполнил студент группы 073601

Примакович Людмила Васильевна

(подпись студента)

Курсовая работа представлена на
проверку _____.2022

(подпись студента))

Минск 2022

Содержание

Введение	8
1 Анализ и моделирование предметной области программного средства	10
1.1 Описание работы склада	10
1.2 Разработка функциональной модели регистрации и учета поступления товаров на склад	11
1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований	16
1.4 Разработка информационной модели учета и регистрации поставок	18
1.5 UML-модели представления программного средства и их описание	20
2 Проектирование и конструирование программного средства	24
2.1 Постановка задачи	24
2.2 Архитектурные решения	24
2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства	27
2.4 Проектирование пользовательского интерфейса	30
2.5 Обоснование выбора компонентов и технологий для реализации программного средства	33
3 Тестирование и проверка работоспособности программного средства	36
4 Инструкция по разворачиванию приложения и сквозной пример, начиная от авторизации, демонстрируя реализацию всех вариантов использования	41
Заключение	52
Список использованных источников	53
Приложение А (обязательное) Отчет о проверке на заимствования в системе «Антиплагиат»	54
Приложение Б (обязательное) Листинг кода алгоритмов, реализующих бизнес- логику	55
Приложение В (обязательное) Листинг скрипта генерации базы данных	66
Ведомость документов курсового проекта	71

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

БД (база данных)	– представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины
Информационная система	– система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые), которые обеспечивают и распространяют информацию
Нормальная форма	– свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных
Среда выполнения	– вычислительное окружение, необходимое для выполнения компьютерной программы и доступное во время выполнения компьютерной программы
СУБД (система управления базами данных)	– совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных
ТЗ (техническое задание)	– документ, содержащий требования заказчика к объекту разработки, определяющий порядок и условия её проведения
<i>API (application programming interface)</i>	– описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой
<i>HTML (HyperText Markup Language)</i>	– стандартизированный язык разметки документов во Всемирной паутине
<i>HTTP (HyperText Transfer Protocol)</i>	– протокол прикладного уровня передачи данных изначально – в виде гипертекстовых документов в формате <i>html</i> , в настоящий момент используется для передачи произвольных данных.
<i>IDE (Integrated development environment)</i>	– комплекс программных средств, используемый программистами для разработки программного обеспечения
<i>IDEF0</i>	– методология функционального моделирования (англ. <i>function modeling</i>) и графическая нотация, предназначенная для формализации и описания бизнес-процессов

<i>Java</i>	– строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией <i>Sun Microsystems</i>
<i>SQL (structured query language)</i>	– язык структурированных запросов, декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных
<i>MySQL Server</i>	– свободная реляционная система управления базами данных
<i>MySQL Workbench CE 8.0</i>	– инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для MySQL.
<i>Sybase ASE (Adaptive Server Enterprise)</i>	– реляционная система управления базами данных компании <i>SAP</i> , одна из СУБД, использующая в качестве основного процедурного <i>SQL</i> -расширения язык <i>Transact-SQL</i>
<i>UML (Unified Modeling Language)</i>	– язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур
<i>URL (Uniform Resource Locator)</i>	– система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса
<i>Usability</i>	– способность продукта быть понимаемым, изучаемым, используемым и привлекательным для пользователя в заданных условиях

ВВЕДЕНИЕ

Сегодня информационные технологии оказывают большое влияние на протекание процессов ведения бизнеса. Успех деятельности любого предприятия зависит от оснащённости компьютерами, телефонами, современным ПО и т.д. Эти устройства позволяют общаться на больших расстояниях, а также производить обмен файлами.

Очень легко отслеживать и вести учет записей и бизнес-процессов на любом предприятии благодаря современным технологиям, которые позволяют автоматизировать процесс обмена информацией и ускорить процесс ее обработки. Это актуально в век высоких технологий, ведь от скорости работы с информацией сегодня зависит успех и эффективность работы предприятия, а человеку было бы очень трудоемко и долго фиксировать все необходимые данные и сортировать их вручную.

К примеру, на склады логистических компаний постоянно поступают различного рода товары. Отслеживать все соответствия привозимых товаров требованиям склада проблематично. Также существует проблема надлежащего хранения информации о поставках и поставщиках, корректного заполнения сопроводительных документов. Автоматизация процессов приемки и регистрации поступлений на склад значительно упрощает задачу.

Для анализа поставщиков необходимы большие объемы информации об их более ранних поставках. Вся эта информация должна быть хорошо структурирована и отфильтрована. Многочисленные таблички Excel с данными о поставках, поставщиках, продукции, отказах, доходах по складам и т.д. при большом количестве поставщиков и поставок не справляются с поставленной задачей. Система управления взаимодействия поставщиков и работников склада (кладовщиков) необходима на предприятии, чтобы ускорить процесс приемки и сделать его более надежным.

Автоматизация складского учёта влияет на скорость и качество выполнения основных складских процессов, приводит к совершенствованию систем управления и регулирования материальных и информационных потоков на складе. Это достигается путём внедрения современного программного обеспечения и компьютерного оборудования на предприятии.

Разработка автоматизированной системы учета и регистрации поступлений товаров на склад дает толчок к развитию новых форм взаимодействия поставщиков и работников склада, упрощает и ускоряет их взаимодействие, что ведет к осуществлению указанных процессов на более высоком уровне, а также позволяет легко выявлять недобросовестных и

надежных поставщиков, что, в свою очередь, оказывает немаловажное влияние на разработку и осуществление стратегии предприятия.

Объектом исследования являются процессы приемки товара на склад и регистрации поступления товарно-материальных ценностей (ТМЦ).

Повышение качества и эффективности управления приемкой ТМЦ путем автоматизации процессов учета поступивших товаров и регистрации новых поставок внутри системы и управления информацией о поставщиках, продукции и кладовщиках является целью этого курсового проекта.

Для того, чтобы достичь данной цели, необходимо выполнить ряд задач:

- изучить принципы осуществления приемки товаров на склад и ведения учетной документации;
- разработать достаточный функциональный набор, необходимый для эффективной работы программного средства;
- составить схемы алгоритмов, реализующих основной функционал программного средства;
- выполнить проектирование программного средства, используя UML-диаграммы и методологию IDEF0;
- спроектировать базу данных хранимой информации;
- реализовать серверную часть приложения, которая будет реализовывать бизнес логику, и будет выполнять работу с базой данных;
- реализовать клиентскую часть приложения, с удобным интерфейсом пользователя;
- протестировать полученное программное средство и убедиться, что оно работает корректно и эффективно.

Курсовой проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет %. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников». Скриншот проверки приведен в приложении А.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

1.1 Описание работы склада

Успешная работа предприятия складывается из суммарного воздействия различных факторов и грамотного выполнения ключевых функций. При этом стоит отметить, что правильный учёт товара можно отнести к одному из основных условий стабильной работы компании. Без учёта ТМЦ, размещённых на складе, трудно обеспечить их сохранность. Перед тем как доверить кладовщику материальные запасы, с ним, как правило, заключается договор. В нем описываются виды работ, которые выполняет сотрудник, и степень ответственности в случае потери или повреждения продукции, принимаемой на склад и хранящейся на складе. Грамотно организованный процесс учёта поступающих и хранящихся на складе материалов — это очень важный и нужный сегмент деятельности организации [1].

Совокупность работ, выполняемых на различных складах, примерно одинакова. Это объясняется тем, что в разных логистических процессах склады выполняют схожие функции: приемка товаров от поставщиков; временное размещение и хранение материальных запасов; преобразование материальных потоков (транспортировка, складирование); предоставление информации о движении товаров. Однако организация работы на конкретном складе имеет ряд особенностей, которые зависят как от размера склада, так и от его предназначения. Приёмка товаров на склад является важной составной частью технологического процесса. Она осуществляется материально ответственными лицами в соответствии с внутренними правилами приемки.

Приёмка материалов на складах организации заключается в проверке соответствия поступивших ТМЦ данным, указанным в товарно-транспортной накладной (ТТН) поставщика, спецификации, описи, упаковочных ярлыков и др., а также условиям договора. Отсутствие некоторых из данных документов не приостанавливает приёмки: составляется акт о фактическом наличии. В этих случаях в акте делается отметка об отсутствии документов [2].

В общем виде процесс работы склада можно изобразить наглядно (см. рисунок 1.1). На нем видно, что одними из основных проблем организации склада является необходимость грамотного снабжения запасами, т.е. сокращение времени ожидания отгрузки, и надлежащего контроля поставок, т.е. сверка с заявленным количеством и качеством.



Рисунок 1.1 – Логический процесс работы склада

В связи с этим процедура регистрации поступлений на склад и процесс ведения складского учета все чаще и чаще подвергается различного рода автоматизации.

1.2 Разработка функциональной модели регистрации и учета поступления товаров на склад

Основным процессом предметной области курсового проекта является регистрация и учет поступления товара. Данный процесс имеет множество особенностей и включает в себя формальности, требует выполнения всех инструкций и требований, а также заслуживает отдельного внимания. А значит, существует необходимость изложить все пункты и действия всех сторон предметной области.

Методология IDEF0 используется для создания функциональной модели, которая представляет собой структурированное изображение функций производственной системы или среды, а также информации и объектов, которые связывают эти функции. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов [3].

Результат анализа предметной области можно представить с помощью функциональной модели процесса регистрации и учета поступления товаров на склад. На рисунке 1.2 представлена контекстная диаграмма верхнего уровня.

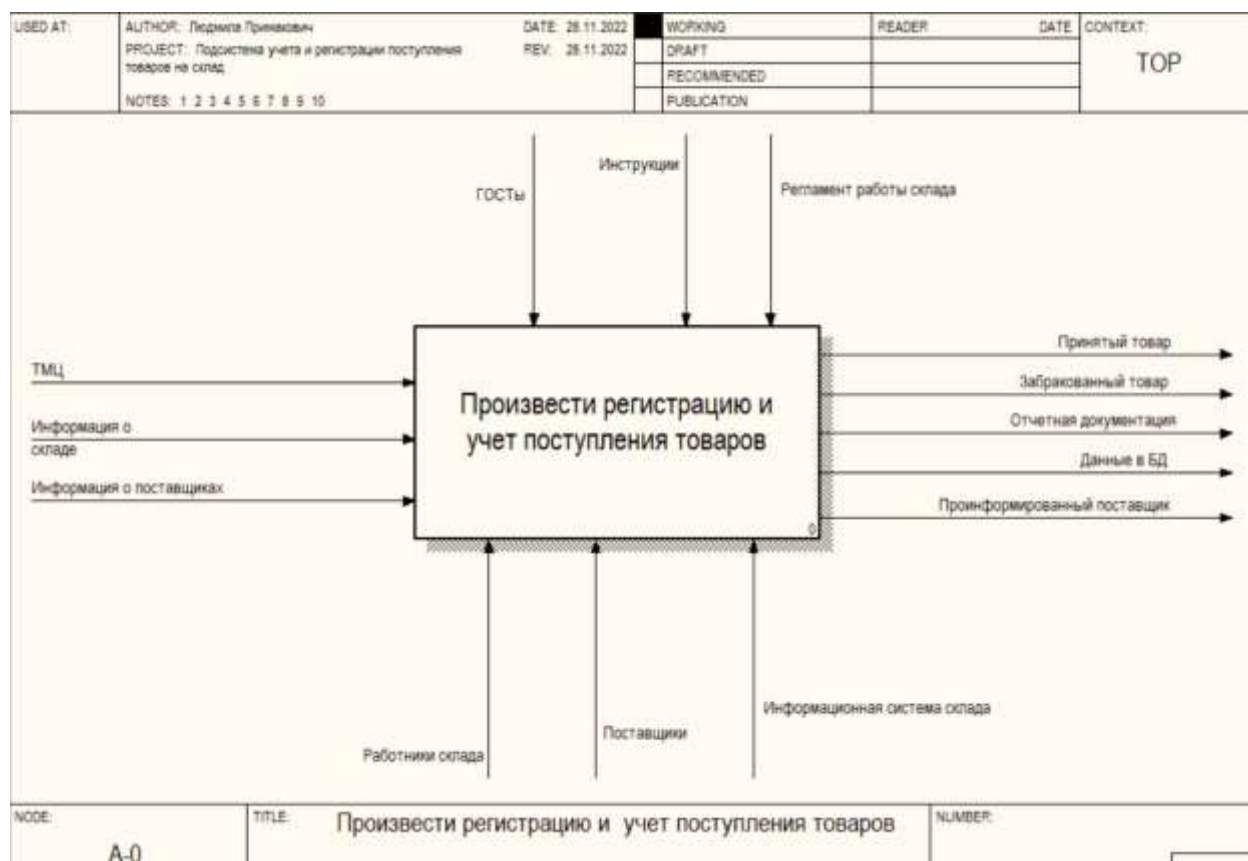


Рисунок 1.2 – Контекстная диаграмма

Затем представлена декомпозиция приведенной диаграммы, которая состоит из трех блоков (см. рисунок 1.3):

1. Оформить ТТН.
2. Принять товар.
3. Хранить товар на складе.

Декомпозиция – это разделение одной крупной цели на задачи для успешного достижения этой самой цели; простыми словами, декомпозицию применяют для продуктивного распределения времени и ресурсов и чтобы не испытывать страх перед огромной задачей[4].

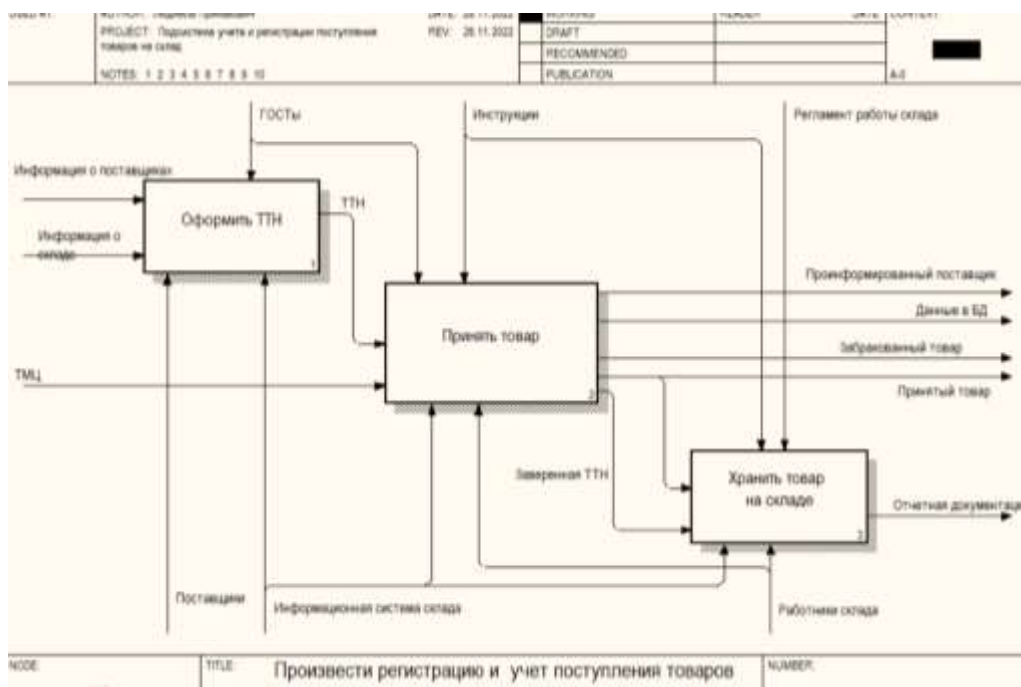


Рисунок 1.3 – Декомпозиция контекстной диаграммы

Процесс «Оформить ТТН» разбивается на три функциональных блока (см. рисунок 1.4):

1. Оформить данные о поставщике.
2. Внести данные о товаре.
3. Выбрать адрес поставки.

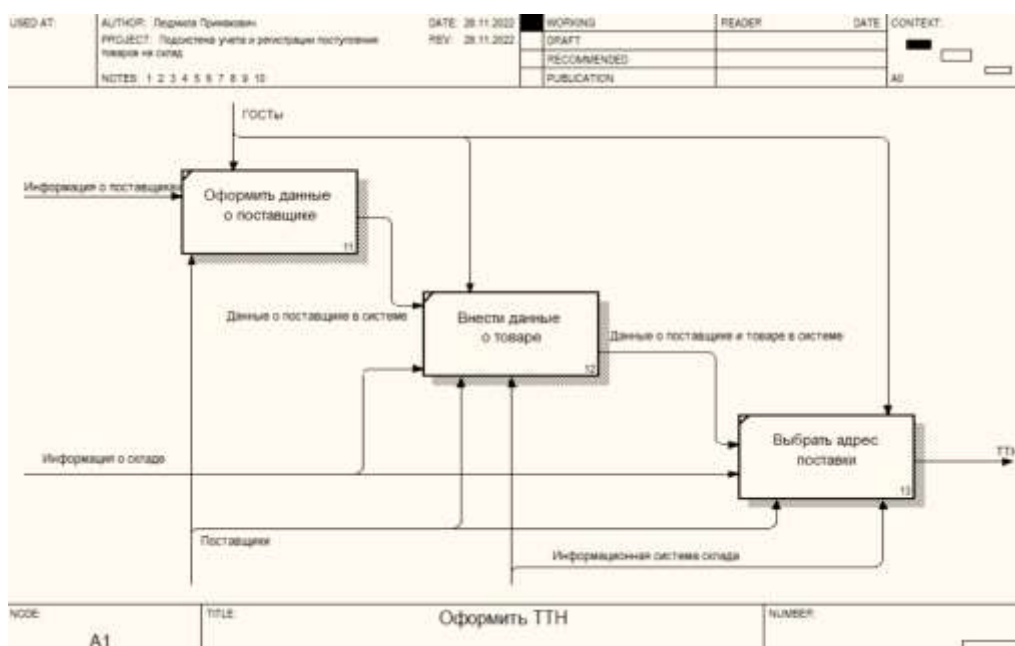


Рисунок 1.4 – Декомпозиция блока «Оформить ТТН»

Этап «Принять товар» разбит на шесть функциональных блоков (см. рисунок 1.5):

1. Проверить ТТН.
2. Проверить ТМЦ.
3. Сообщить поставщику результаты проверки.
4. Заверить ТТН.
5. Передать ТМЦ на хранение.
6. Внести данные в БД.

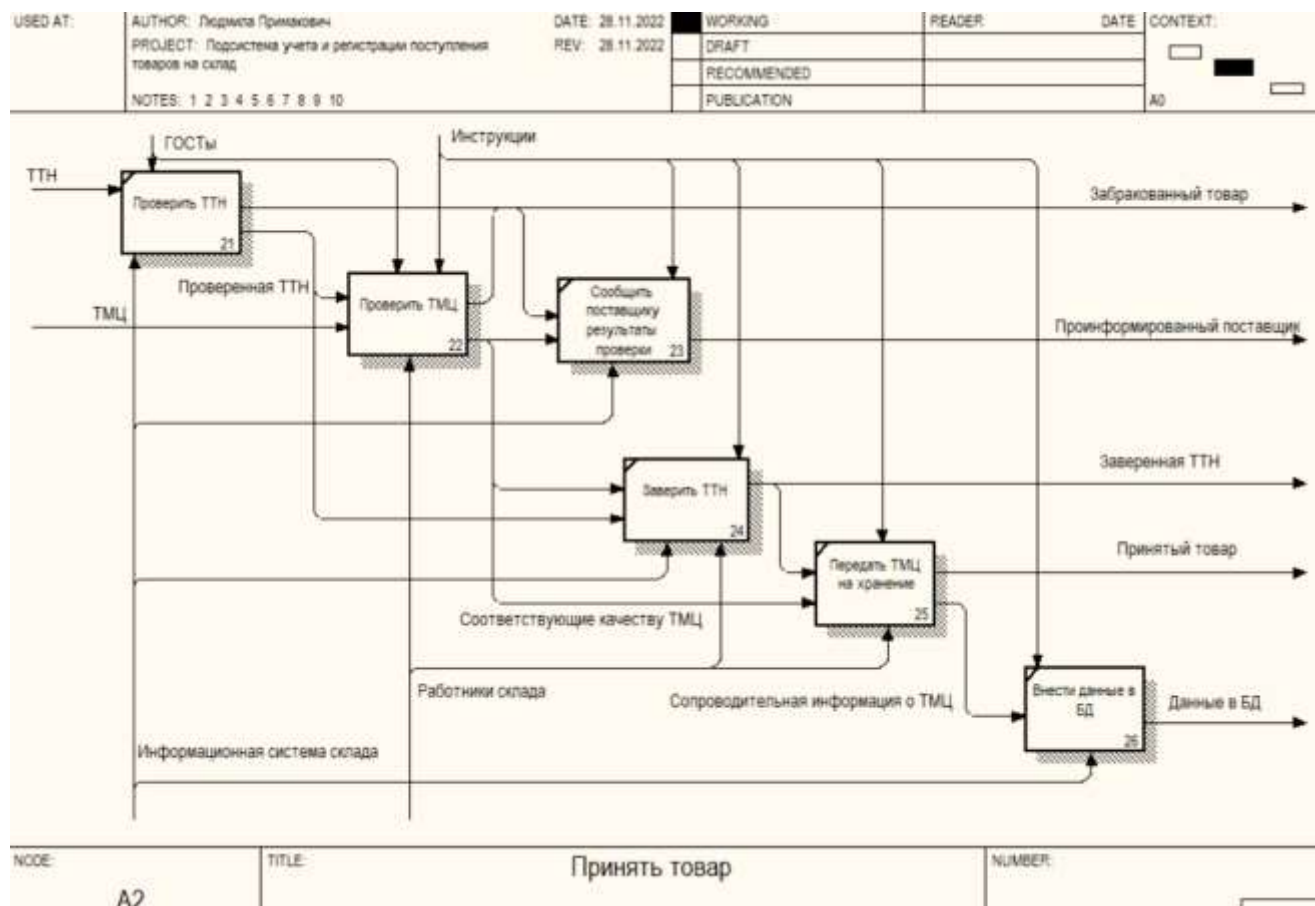


Рисунок 1.5 – Декомпозиция блока «Принять товар»

Данная декомпозиция наиболее полно описывает основную логику рассматриваемой предметной области курсового проекта и является наиболее развернутой декомпозицией всей функциональной модели. Этот факт дает нам право говорить, что именно декомпозиция блока «Принять товар» будет определять опорную логику разрабатываемого впоследствии программного проекта.

Декомпозиция блока «Проверить ТМЦ» данной диаграммы включает в себя два блока (см. рисунок 1.6):

1. Проверить соответствие качеству.
2. Проверить соответствие срокам.

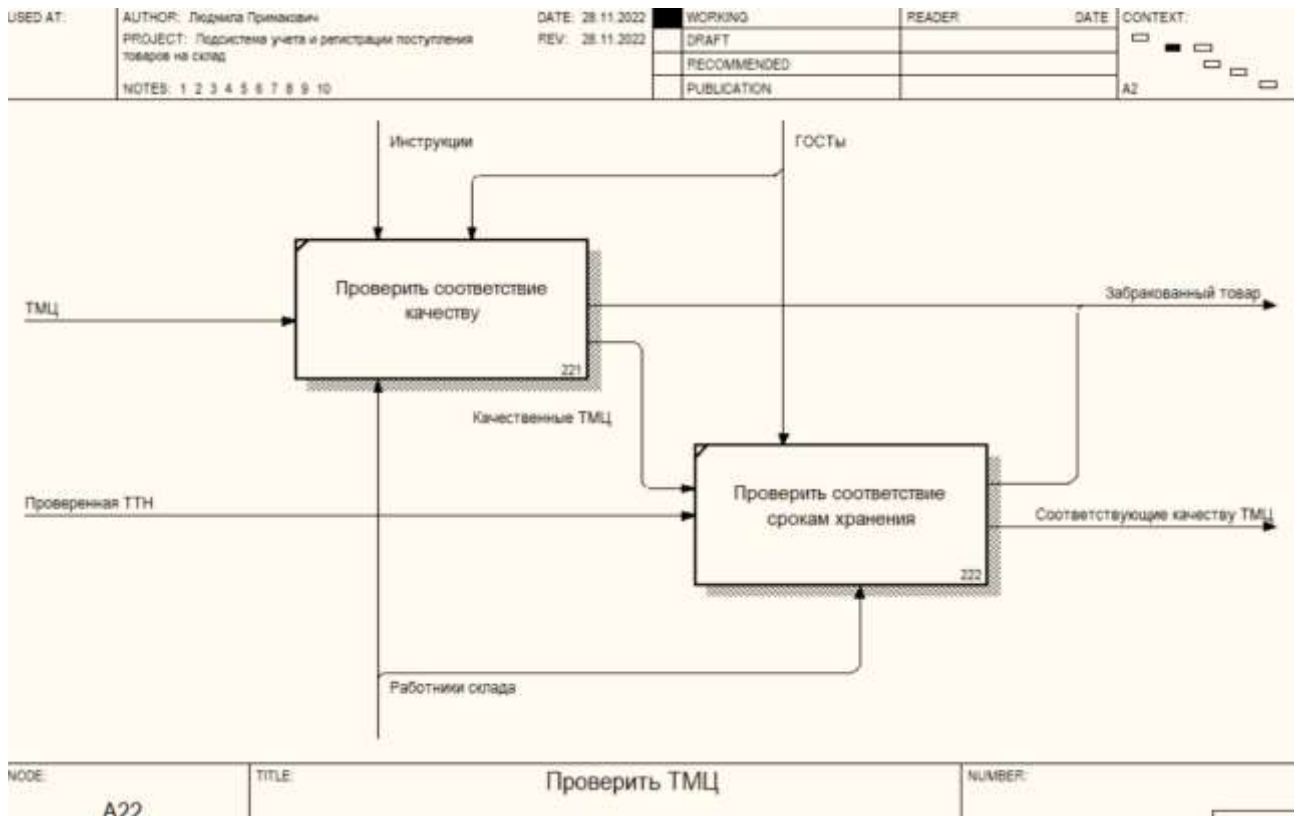


Рисунок 1.6 – Декомпозиция блока «Проверить ТМЦ»

Возвращаясь к декомпозиции контекстной диаграммы, блок «Хранить товар на складе» разбивается на четыре функциональных блока (смотри рисунок 1.7):

1. Разместить товар на складе.
2. Перемещать товар между корпусами.
3. Произвести инвентаризацию.
4. Обновить данные в БД.

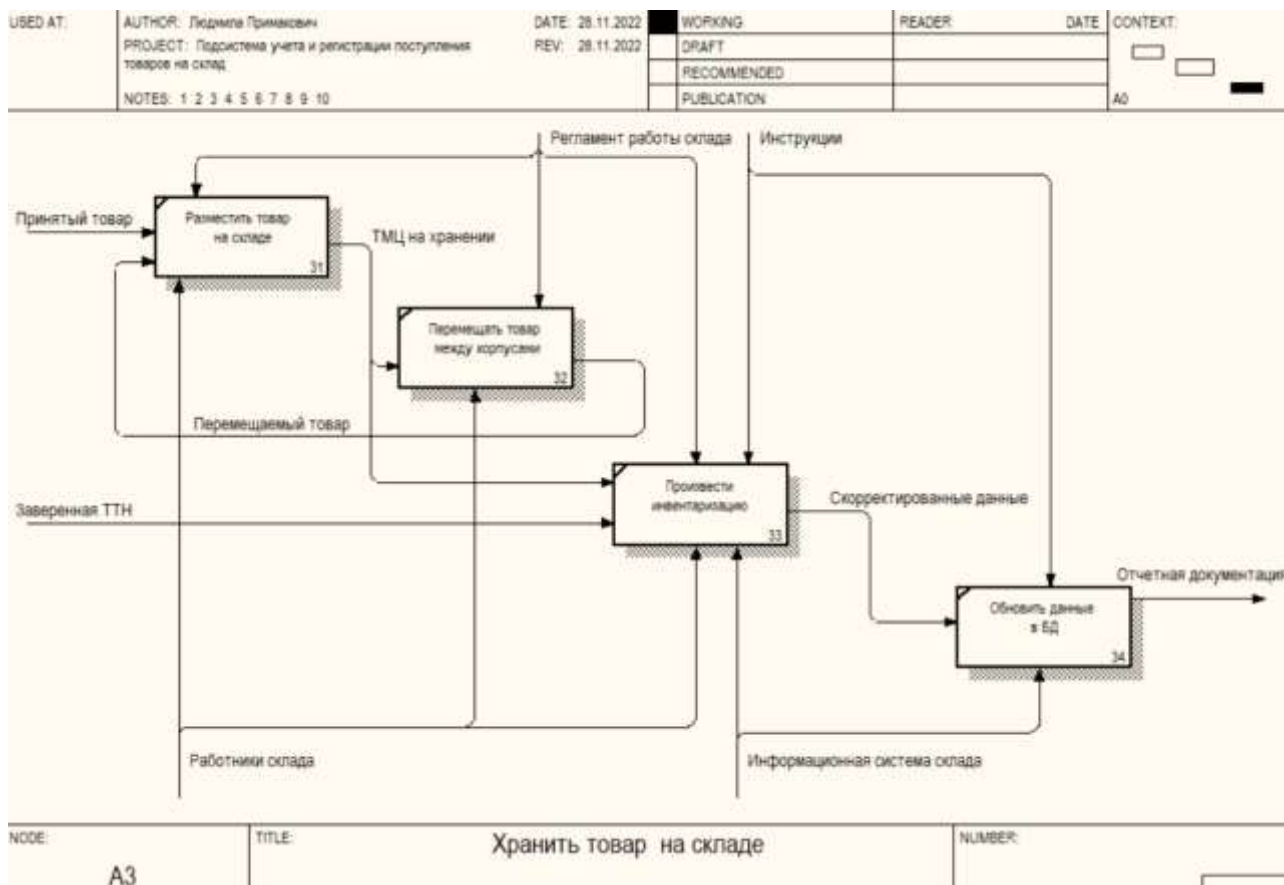


Рисунок 1.7 – Декомпозиция блока «Хранить товар на складе»

Подводя итог проделанной работе по описанию функциональной модели данного курсового проекта, можно отметить, что для организации процесса регистрации и учета поступлений товаров на склад логистической компании необходимо учитывать соотношение принятых и забракованных товаров от каждого конкретного поставщика, для того, чтобы в дальнейшем продолжать сотрудничество лишь с надежными поставщиками, тем самым максимизируя прибыль от деятельности склада и повышая эффективность его работы.

При помощи функционального проектирования были продемонстрированы все этапы бизнес-процессов предметной области.

1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Для разработки программного средства необходимо понять общий принцип осуществления регистрации прибытия товаров на склад, а также предусмотреть ведение учетной документации всех поступлений.

Поскольку каждый поставщик поставляет свой товар, причем это происходит неоднократно, необходимо спроектировать базу данных (БД), которая осуществляла бы хранение информации о каждой поставке в электронном виде. Это позволит упростить ведение учетной деятельности, а также позволит легко производить анализ поставщиков посредством их фильтрации по определенным критериям. Для корректной работы необходимо предусмотреть возможность добавления, редактирования и обновления информации в БД. Для хранения информации будет использован *MYSQL Server*. Подключение к нему будет осуществляться при авторизации, и, в зависимости от авторизованной роли, пользователь будет получать определенный набор возможностей для выполнения своей работы.

Для осуществления своей деятельности работники склада должны быть действительными сотрудниками, поэтому их личные данные вносятся администратором. Он же получает возможность удаления их, например, вследствие увольнения работника., а также имеет доступ ко всей базе данных, может производить анализ поставщиков и инвентаризацию, т.е. обновление полей ТТН в соответствии с датой.

Работники должны принимать товарно-транспортные накладные – это их уникальная функция. В результате приемки работник может как принять поступивший товар, так и отклонить, например, при несоответствии сроков годности или наличии повреждений.

Поставщик самостоятельно регистрируется в системе и имеет возможность оформить ТТН. Программа после заполнения полей формы требует оплату для заведения личного счета на каждую поставку.

Диаграмма вариантов использования представлена на рисунке 1.8.

Диаграмма вариантов использования (сценариев поведения, прецедентов) является исходным концептуальным представлением системы в процессе ее проектирования и разработки.

Суть данной диаграммы заключается в том, что проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования (use cases). Поэтому эту диаграмму ещё называют usecase-диаграммой. Каждый use case определяет набор действий, которые система совершает при взаимодействии с актерами. При этом на самой диаграмме никаким образом не отображается то, как именно будет реализован определенный набор действий.

- товарно-транспортные накладные (ТТН);
- платежи.

Поставщик при регистрации в системе записывает личные данные о своей компании, которые добавляются в сущность «Поставщики». При оформлении ТТН данные этого поставщика автоматически добавляются в сущность «ТТН» и создается его личный счет для оплаты услуг складирования.

Сведения о работниках склада, принимающих товары, должны вноситься администратором. Для этого вводятся сущности «Администратор» и «Работники». Первая таблица хранит в себе id, логин и пароль, а во второй храниться ещё и личная информация о работнике и склады, на которых он работает и которые хранятся в таблице «Склады».

При входе в систему каждый пользователь в обязательном порядке проходит аутентификацию, вводит личные данные, которые целесообразно где-то хранить. Для этой цели создана сущность «Ключи». В результате данная таблица связывается с тремя дочерними таблицами: «Администратор», «Работники» и «Поставщики» – связью один-к-одному.

Таблица «Платежи» необходима для учета и расчета прибыли. Она содержит поля id личного счета, баланс на счету и связана с таблицей «Поставщик» связью один-к-одному.

В таблице «Склады» храниться информация о складах, которыми располагает компания, а в таблице «Продукция» содержится информация о типах продукции, которую на этих складах можно принимать.

Информационная модель подсистемы учета и регистрации поступления товаров на склад приведена на рисунке 1.9.

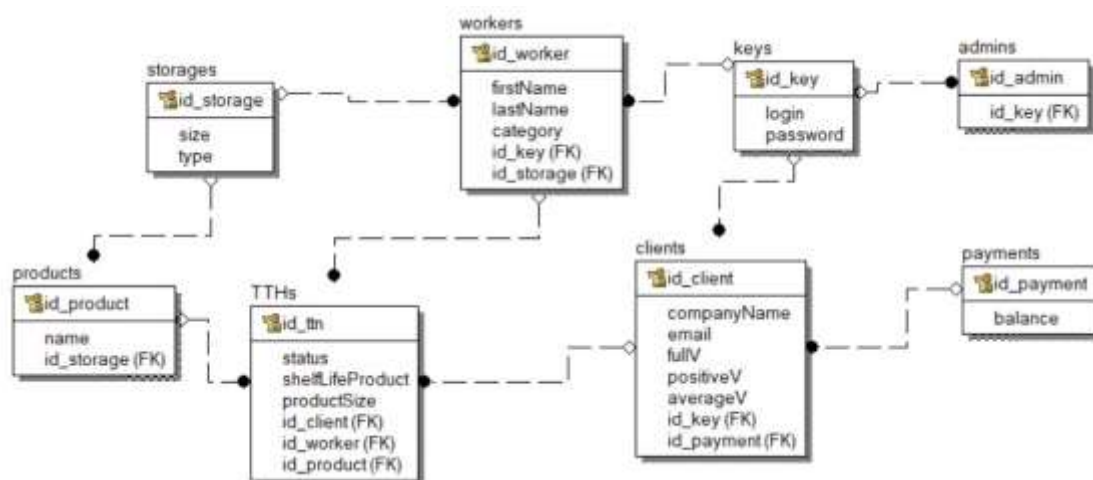


Рисунок 1.9 - Информационная модель подсистемы учета и регистрации поступления товаров на склад

В ходе разработки представленной модели и преобразования отношений в БД была проведена нормализация, следствием которой является приведение структуры БД к третьей нормальной форме. Это означает, что в данной базе данных отсутствуют кортежи с несколькими значениями атрибутов, каждый неключевой атрибут функционально полно зависит от потенциального ключа, отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

1.5 UML-модели представления программного средства и их описание

Язык UML предназначен для описания моделей, причем для работы с этим языком используются специальные редакторы диаграмм, такие как Rational Rose или online-редакторы. На UML можно содержательно описывать классы, объекты и компоненты в различных предметных областях, часто сильно отличающихся друг от друга [6].

Моделирование необходимо для понимания системы, причем единственной модели никогда не бывает достаточно. Следовательно, приходится разрабатывать большое количество взаимосвязанных моделей. Рассмотрим некоторые основные модели программного средства: диаграмму состояний, диаграмму последовательности и диаграмму развертывания системы.

Диаграмма состояний показывает, каким образом объект меняет одно состояние на другое. Подобного рода диаграммы служат для моделирования динамических аспектов системы. Данная диаграмма полезна при моделировании жизненного цикла объекта.

От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта реактивного, то есть того, чье поведение характеризуется реакцией на внешние события.

Пример этой диаграммы представлен на рисунке 1.10.

На данной диаграмме продемонстрирована деятельность администратора в системе: в каком состоянии может пребывать объект и какие части программного средства при этом будут задействованы.

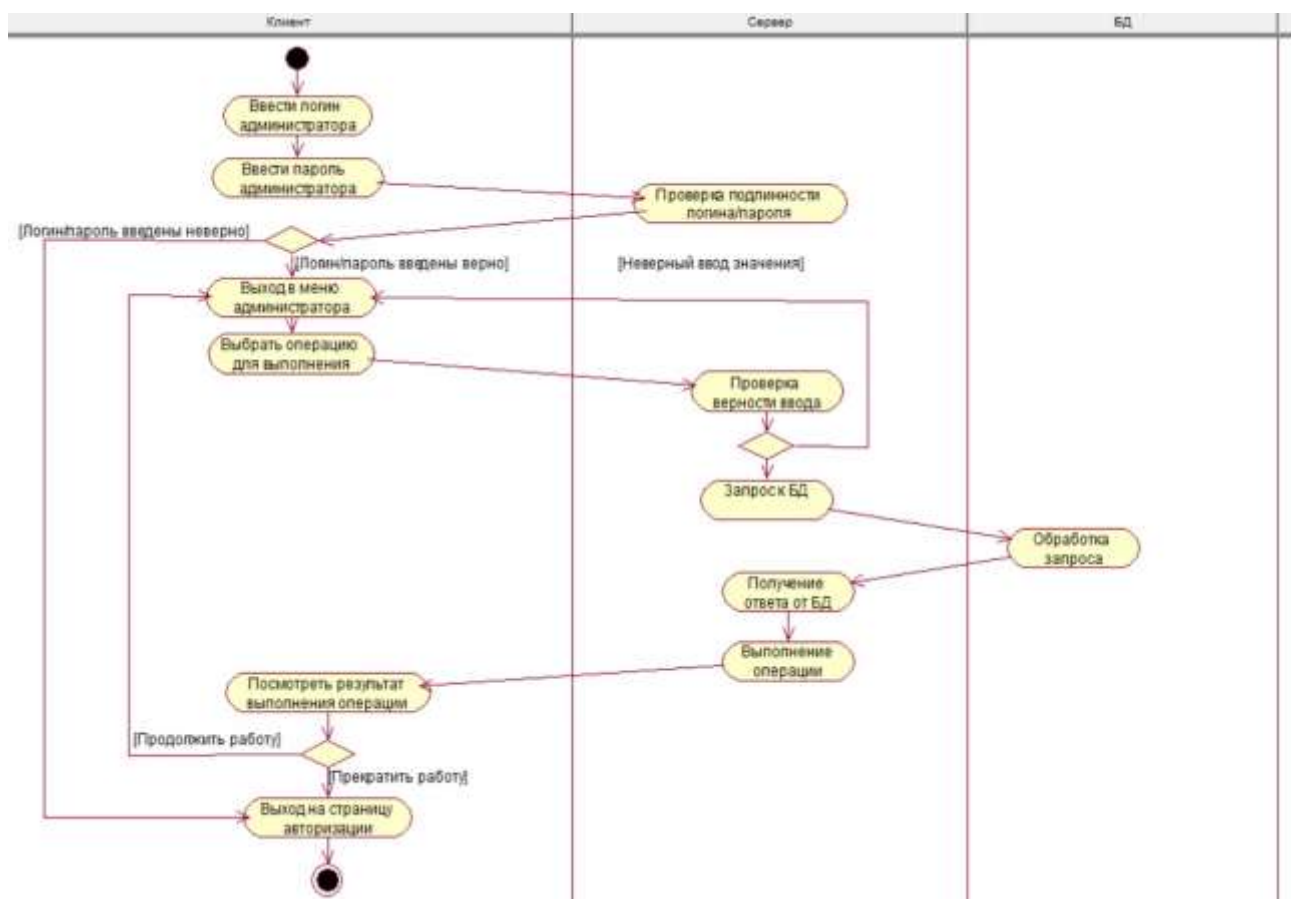


Рисунок 1.10 – Диаграмма состояний для деятельности администратора в системе

На рисунке 1.11 представлена диаграмма последовательности, на которой отображен процесс оформления товарно-транспортной накладной (ТТН). Диаграмма последовательности – одна из подвидов диаграмм взаимодействия, предназначена для моделирования взаимодействия различных объектов системы во времени, сопровождающегося обменом сообщениями. В качестве объектов на данной диаграмме могут выступать пользователи, инициирующие взаимодействие, классы, имеющие поведение, или программные компоненты.

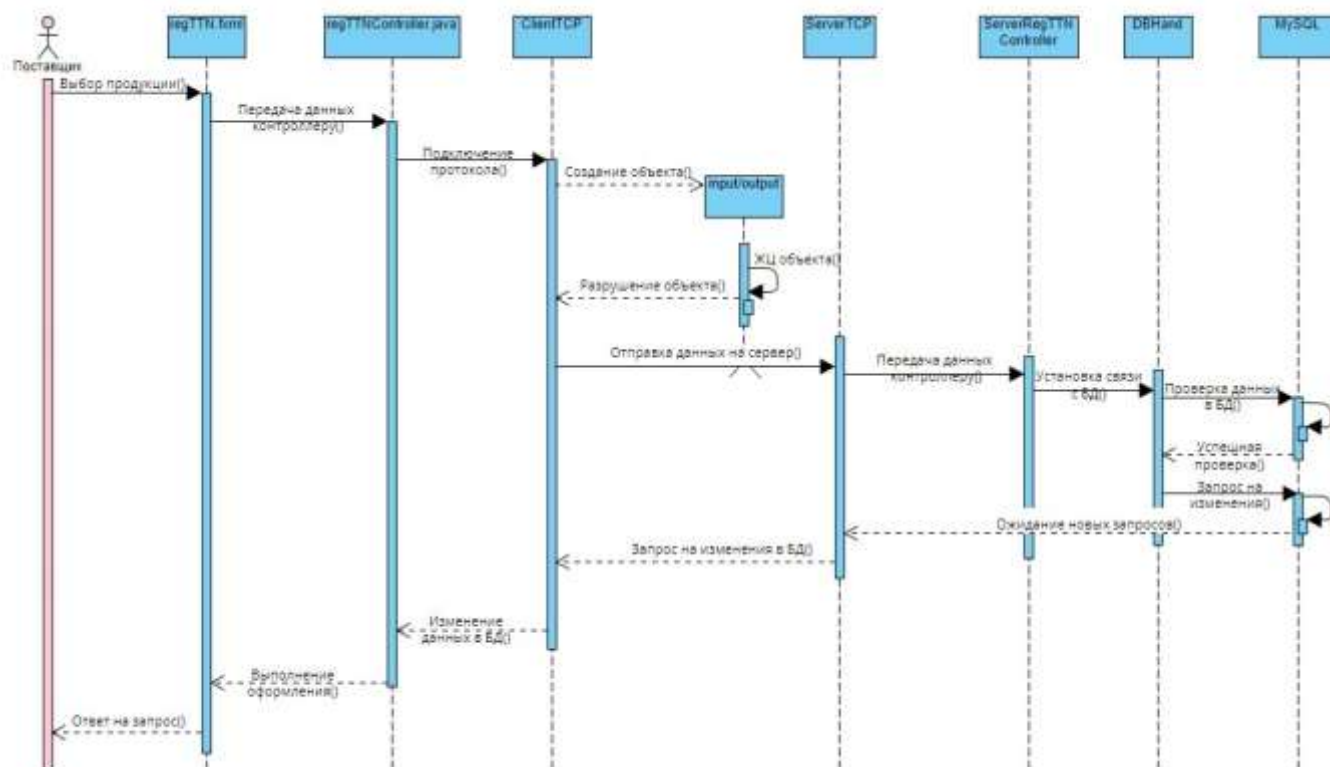


Рисунок 1.11 – Диаграмма последовательности процесса оформления ТТН

На приведенной диаграмме поставщик выбирает тип продукции для оформления ТТН и вводит необходимые данные в поля формы графического приложения, которые считывает контроллер и передает по TCP-протоколу на сервер. Сервер производит обработку данных и подключается к БД.

Там происходят соответствующие изменения и добавления записей, после чего отправляется ответ на сервер. Как в случае успеха, так и в случае неудачи сервер отправляет клиенту соответствующее сообщение, которое поставщик увидит на своем экране приложения.

Диаграмма развёртывания показывает топологию системы и распределение компонентов системы по ее узлам, а также соединения, которые представляют собой маршруты передачи информации между аппаратными узлами, которые на диаграмме обозначаются кубиками.

Диаграмма развертывания представлена на рисунке 1.12.

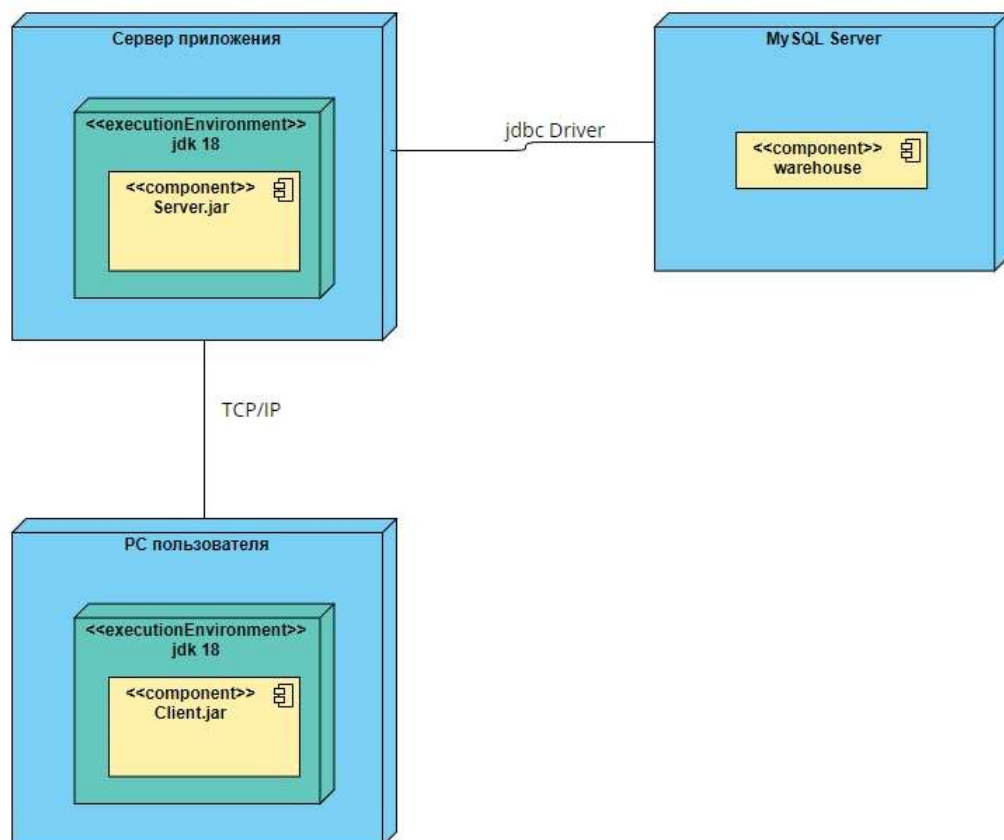


Рисунок 1.12 – Диаграмма развертывания системы

В качестве узлов выступают ПК пользователя, сервер и БД.

Для запуска разрабатываемого приложения необходимо наличие исполняемой среды JDK18 на компьютерах пользователей ПК. Также необходимо наличие MySQL Server для возможности доступа к БД.

Связь узлов ПК с сервером осуществляется по протоколу TCP/IP, а связь сервера с БД осуществляется при использовании драйвера jdbc.

Для успешной работы на устройствах должна быть установлена система Windows 8 и выше.

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Постановка задачи

На склады логистических компаний постоянно поступают различного рода товары. Отслеживать все соответствия привозимых товаров требованиям склада проблематично. Также существует проблема надлежащего хранения информации о поставках и поставщиках, корректного заполнения сопроводительных документов.

Следовательно, возникает необходимость в разработке программного приложения, которое позволило бы оперативно собирать и обобщать отчетные данные о приеме и регистрации товаров. Для складской деятельности немаловажна оценка или анализ поставщиков.

Разрабатываемое приложение должно содержать в себе удобный для восприятия и использования интерфейс и функционал, позволяющий осуществить всю описанную ранее деятельность: оформление и заверение накладных, блокировка недобросовестных поставщиков, управление данными персонала, а также редактирование внесенных данных и добавление новых. Также данное приложение должно иметь надежную систему хранения данных, осуществлять клиент-серверное взаимодействие, доступ в систему должен защищаться и разграничиваться системой ввода логинов и паролей.

Помимо прочего разрабатываемое приложение должно осуществлять полезное представление данных. Например, вывод данных обо всех накладных или поиск наилучшего поставщика и под. Также в данном приложении должны осуществляться проверки на корректность и полноту вводимых данных, так как нельзя исключать невнимательность или некоторый злой умысел пользователя, вносящего данные в систему.

В результате разработки пользователи получают готовый к использованию хорошо оптимизированный продукт, который будет выполнять все необходимые функции, хранить данные о поставках и позволит анализировать поставщиков на основе их поставок.

2.2 Архитектурные решения

При моделировании программных средств наиболее часто используются диаграммы классов, на которых изображаются классы и интерфейсы, а также связи (отношения) между ними. В данном курсовом проекте были

реализованы классы Admin, Worker, Client, TTN, Storage, Product, Payment, Request. Это классы, которые используются для хранения информации о сущностях системы. Первые три класса хранят в себе поля login и password, необходимые для корректной авторизации и последующей работы в системе. Помимо этого, в классе Worker храниться информация о работнике, как то фамилия, имя, разряд, склад, на котором он работает. В классе же Client храниться информация о названии компании-поставщика, а также email этой компании и id расчетного счета (информация о платежах храниться в payment). Кроме того, в этом классе храниться показатель надежности поставщика: отношение объема успешных поставок ко всему объему поставок от данного поставщика. В классе Storage находится информация о типе и размере склада, а в классе Product – информация о типе продукции и складе, на котором она может находиться. Центральным классом является класс TTN: в нем храниться информация о приходе товара и о его качестве. Все описанные классы хранятся в пакете StorOrg, что логично, ведь все они описывают структурную организацию склада. Диаграмма классов приведена на рисунке 2.1.

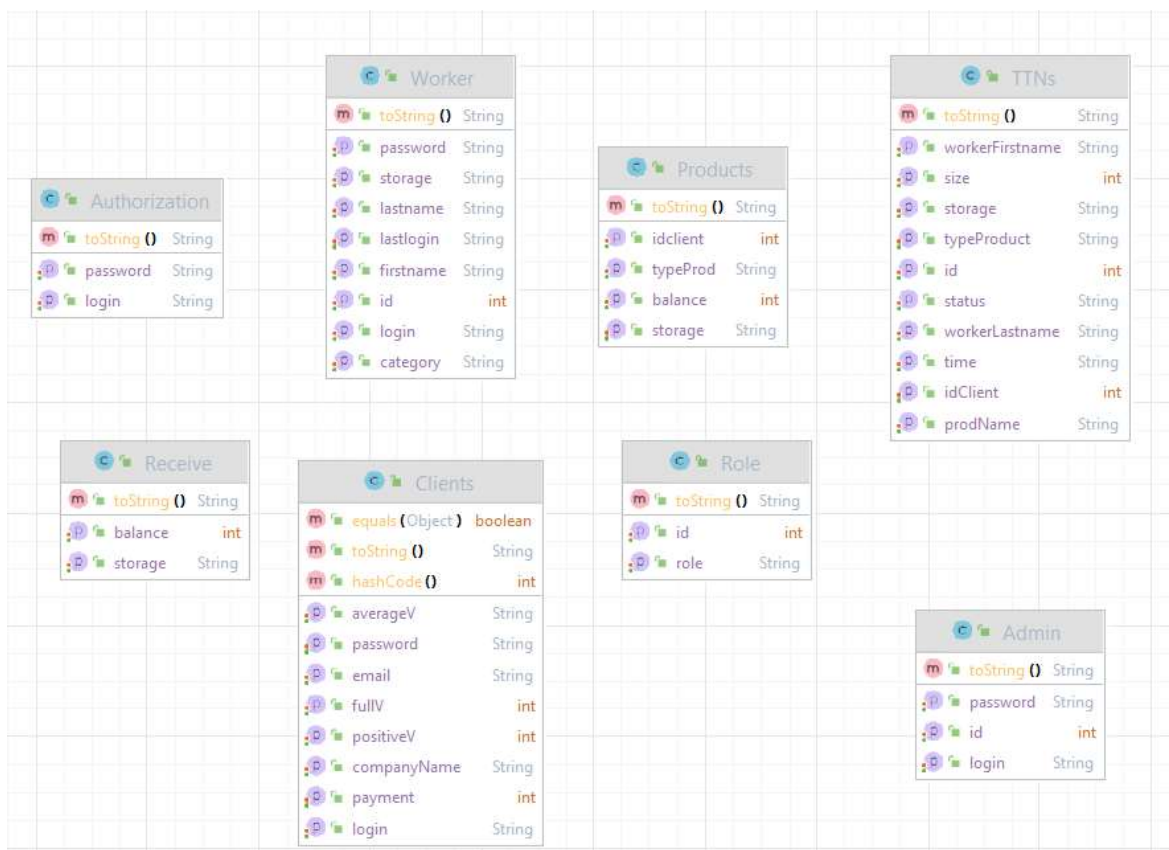


Рисунок 2.1 – Диаграмма классов пакета StorOrg

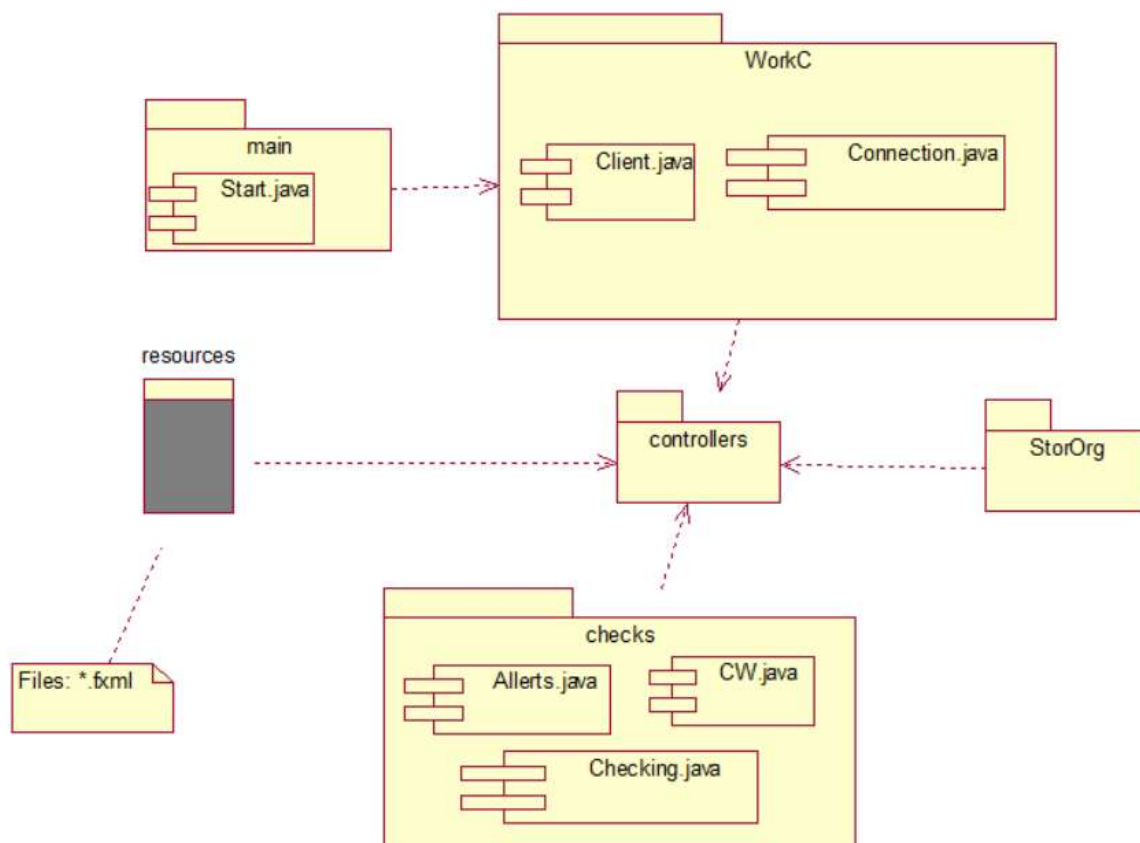


Рисунок 2.3 – Диаграмма компонентов клиентской стороны системы

2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства

Рассмотрим алгоритмы реализации некоторых методов серверной части программного продукта, которые соответствуют вариантам использования.

После запуска приложения открывается окно авторизации пользователей. После авторизации пользователь получает доступ к ресурсам системы в соответствии со своей ролью: если это роль администратора, то пользователь получает доступ практически ко всем ресурсам БД, если поставщик – к регистрации накладной, просмотру заявок и продукции, если работник склада – к регистрации накладных и списку заявок. Таким образом, обобщенный алгоритм функционирования системы можно представить в виде, изображенном на рисунках 2.4-2.5. Развернуто этот алгоритм представлен на чертеже 2.

Перед началом работы пользователь в обязательном порядке проходит авторизацию для определения занимаемой роли и соответствующего этой роли функционала. Для удачной авторизации пользователь должен верно указать свои логин и пароль. После ввода программой производится

подключение к базе данных, в которой проверяется таблица ключей. Если пользователь, с приведенным напором ключей зарегистрирован в системе, программа вернет код роли, а в случае несовпадения будет выведено окно ошибки с сообщением о неверном введении данных, при этом будет предложено ввести данные ещё раз.

Алгоритм описанной функции авторизации приведен на рисунке 2.5.

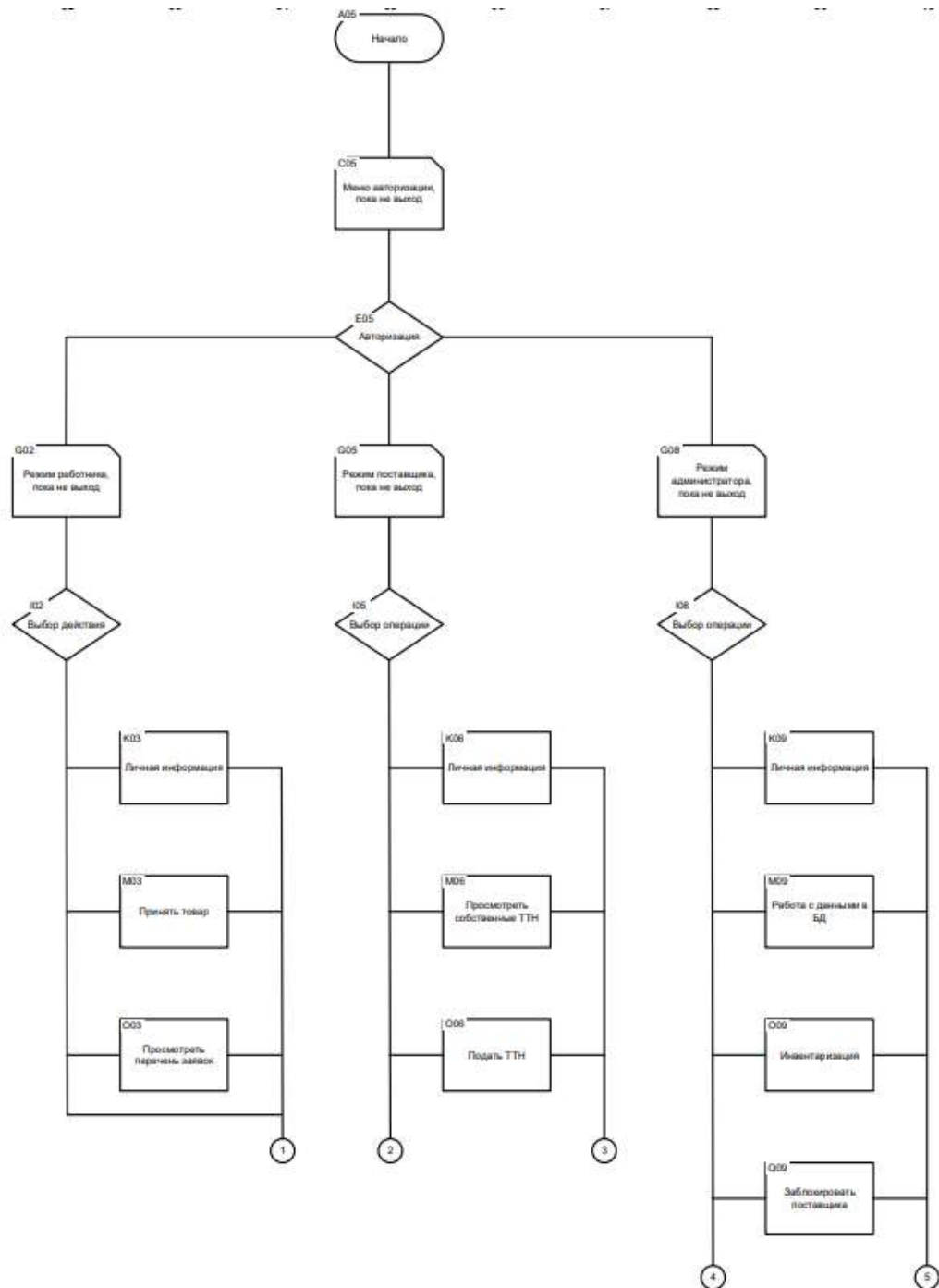


Рисунок 2.4 – Обобщенный алгоритм работы системы

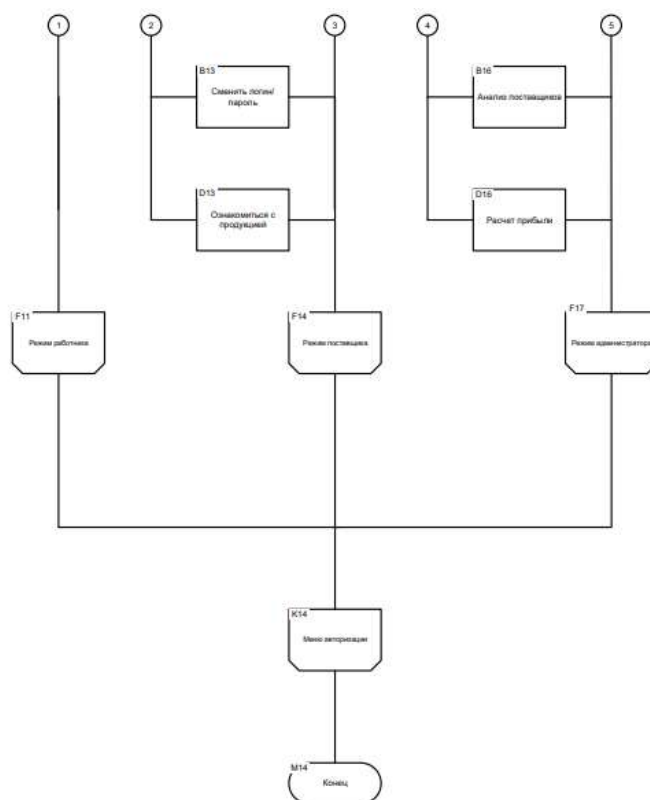


Рисунок 2.5 – Обобщенный алгоритм работы системы (окончание)

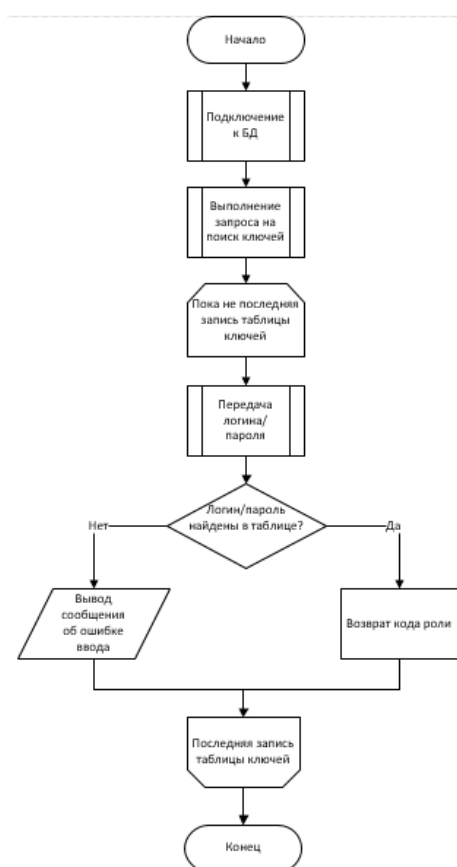


Рисунок 2.6 – Алгоритм функции авторизации

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс представляет собой совокупность средств для взаимодействия пользователя и компьютера, которая основана на представлении всех возможных вариантов использования, доступных пользователю, в виде графических компонентов экрана. Причем, пользователь может произвольно выбрать необходимых вариант при помощи клавиатуры или мыши, чего не было в интерфейсе командной строки. Одно из центральных требований к хорошему графическому интерфейсу ПС – его «предсказуемость», т.е. система должна работать предсказуемо, пользователь должен заранее интуитивно понимать, какой из вариантов развития событий последует за его командой.

Приложение имеет интерфейс, разработанный средствами JavaFX. Главное окно администратора содержит вкладки, предоставляющие доступ как к собственным донным, так и к различным категориям других сущностей, принимающих участие в бизнес-процессе – личные данные, анализ поставщиков, заблокировать поставщика, расчет прибыли и др. Визуальное представление этого окна приведено на рисунке 2.7. При нажатии на каждую из кнопок окна открывается новое окно, содержание которого соответствует описанному на кнопке действию.

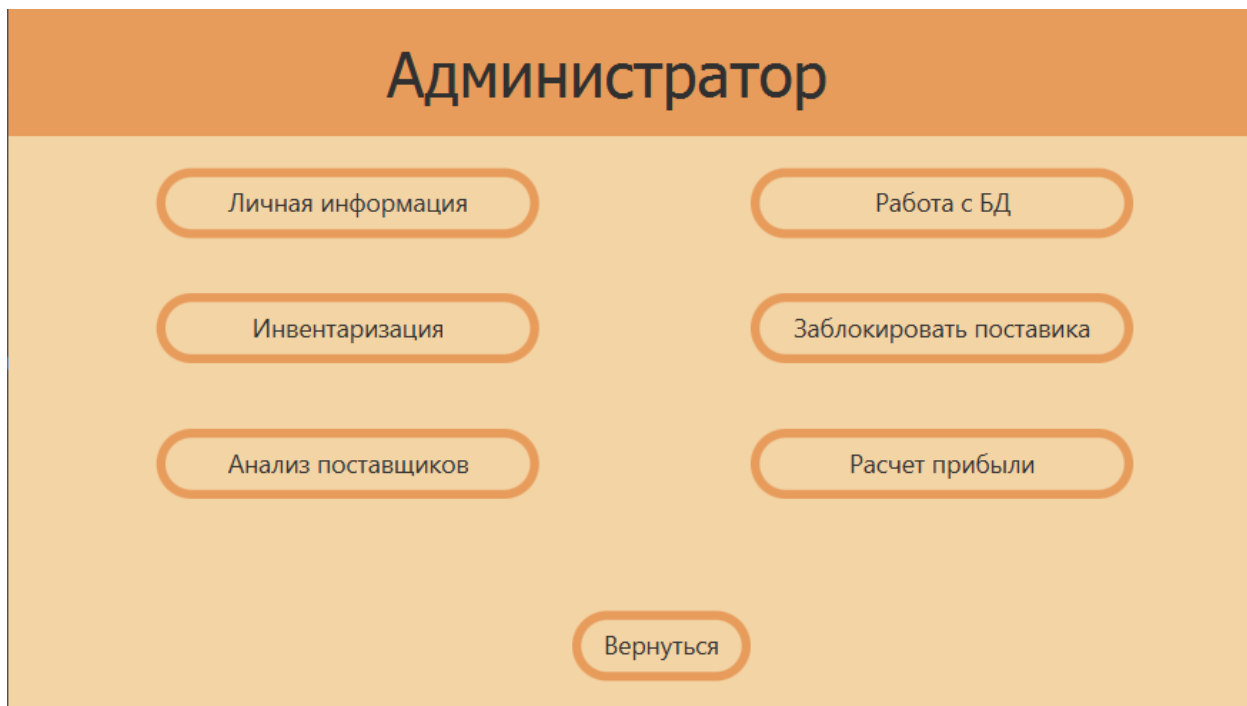


Рисунок 2.7 –Меню администратора

Отдельно стоит отметить кнопку «Работа с БД», при нажатии на которую откроется ещё одно окно с различного рода командами, доступными только администратору: добавление данных в базу, удаление их и модификация. Вид этого окна приведен на рисунке 2.8:

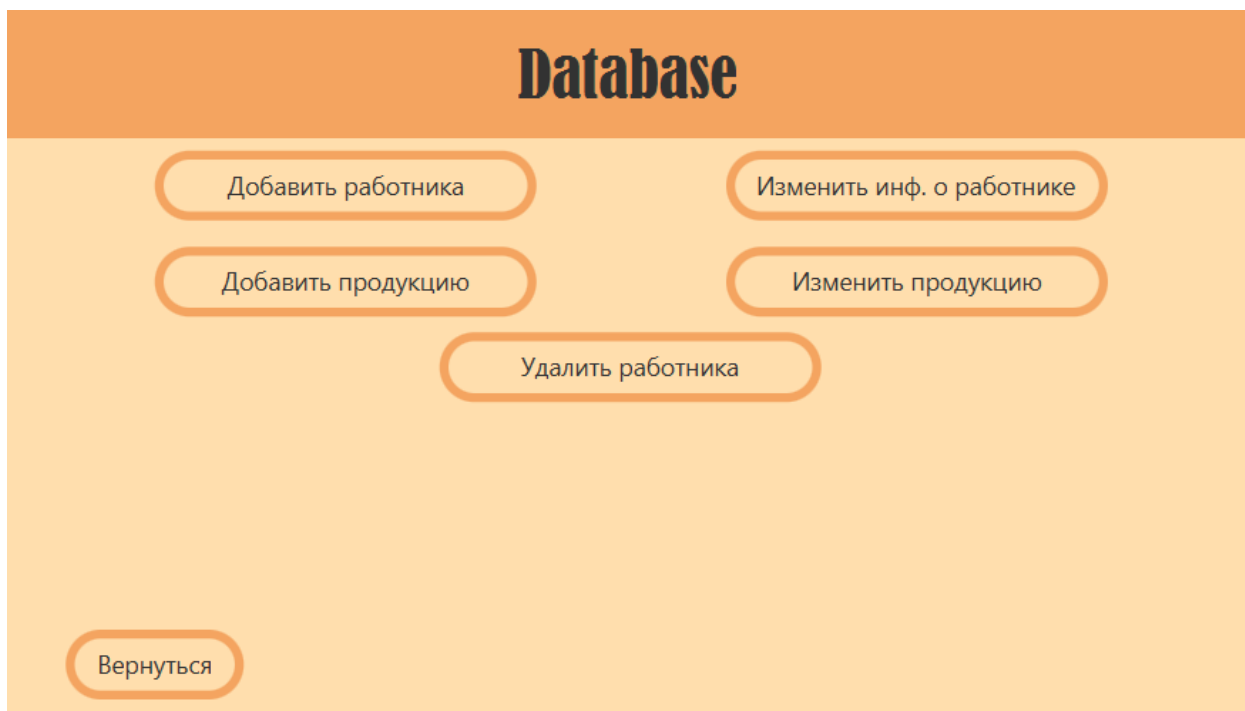


Рисунок 2.8 – Окно работы с БД

При нажатиях на кнопки возврата происходит выход в главное меню регистрации. После такого выхода появляется окно аутентификации (проверка логина и пароля пользователя системы). Данное окно (см. рисунок 2.9) служит для идентификации и регистрации пользователей программы. Идентификация необходима для разграничения доступа к отдельным командам и частям системы. Таким образом в программе, помимо администратора, могут авторизоваться ещё два типа пользователей: поставщик и работник. Их графические интерфейсы приведены на рисунках 2.10 и 2.11 соответственно. Регистрация же предусмотрена непосредственно для поставщиков, так как заранее зная их данные администратор не имеет никакой возможности, а значит предоставить их каждый поставщик должен самостоятельно.

The screenshot shows a web interface with an orange header containing the title 'Программа складского учета'. Below the header, on a light orange background, are two input fields for 'Логин' (Login) and 'Пароль' (Password). At the bottom, there are two buttons: 'Вход' (Login) and 'Регистрация' (Registration).

Программа складского учета

Логин

Пароль

[Вход](#) [Регистрация](#)

Рисунок 2.9 – Окно аутентификации и регистрации

The screenshot shows a web interface with an orange header containing the title 'Поставщик'. Below the header, on a light orange background, are several buttons arranged in a grid: 'Личная информация' (Personal information), 'Мои накладные' (My invoices), 'Оформить ТТН' (Issue TTN), 'Сменить логин/пароль' (Change login/password), 'Ознакомиться с продукцией' (Get acquainted with the product), and 'Назад' (Back).

Поставщик

[Личная информация](#) [Мои накладные](#)

[Оформить ТТН](#)

[Сменить логин/пароль](#) [Ознакомиться с продукцией](#)

[Назад](#)

Рисунок 2.10 – Окно поставщика

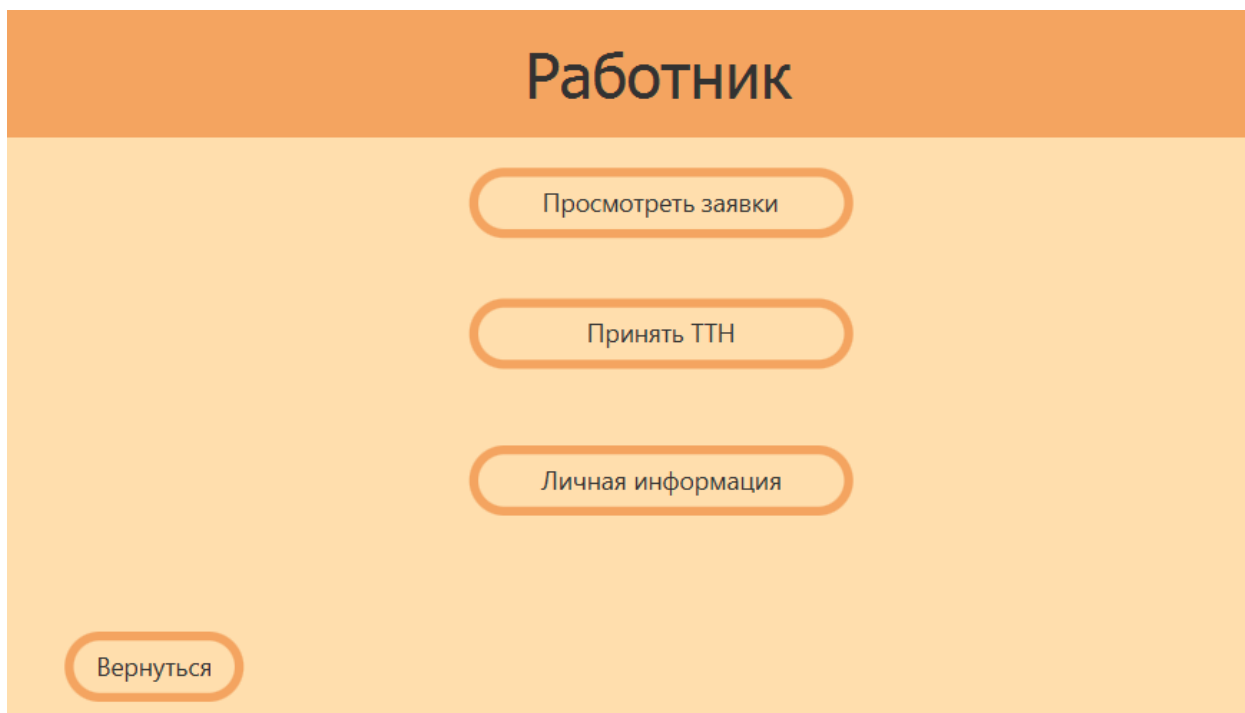


Рисунок 2.11 – Окно работника

Все окна и кнопки на них организованы однотипно, область каждого окна разделяется цветом на заголовок и основную часть. Все это облегчает восприятие информации и упрощает ориентацию в системе для новых пользователей или для пользователей, сменивших роль (например, с работника на администратора).

2.5 Обоснование выбора компонентов и технологий для реализации программного средства

В представленном курсовом проекте необходимо спроектировать программное приложение с архитектурой «клиент-сервер» на объектно-ориентированном языке Java. Указанное приложение должно позволять вести учет и регистрировать поступающую на склады продукцию.

Язык программирования Java отличается надежностью, высоким уровнем защиты и быстротой. На этом языке на сегодняшний день пишется большое количество различного рода программных продуктов, что говорит о его популярности и эффективности.

Термин «клиент-серверная архитектура» – сборное понятие, состоящее из двух взаимодополняющих компонентов: сервера и клиента. Такая архитектура предусматривает разделение процессов предоставления услуг и

отправки запросов на них на разных компьютерах в сети, каждый из которых выполняет свои задачи независимо [7].

Клиент – локальный компьютер на стороне виртуального пользователя, который выполняет отправку запроса к серверу для возможности предоставления данных или выполнения определенной группы системных действий. Сервер – компьютер или специальное системное оборудование, которое предназначается для разрешения определенного круга задач по процессу выполнения программных кодов. Особенности такой модели заключаются в том, что пользователь отправляет определенный запрос на сервер, где тот системно обрабатывается и конечный результат отсылается клиенту. Примечательно, что клиент и сервер могут являться просто двумя разными программами на одном устройстве, каждая из которых выполняет свою особую роль, при этом взаимодействуя с другой по определенному протоколу соединения. Схема взаимодействия клиент-сервер приведена на рисунке 2.12.



Рисунок 2.12 – Взаимодействие клиент-сервер

Соединение между клиентом и сервером осуществляется по стеку протоколов TCP/IP. Данный стек протоколов является более надежным в сравнении с его аналогом UDP/IP за счет того, что в TCP сначала устанавливается соединение, а затем происходит передача данных [8]. Также TCP имеет преимущество в удобстве передачи данных: их не надо вручную преобразовывать к виду дейтаграммы. Связь с БД осуществляется с использованием драйвера jdbc, причем эту связь осуществляет только серверная часть приложения. Клиенту, чтобы получить интересующую его информацию, необходимо сначала отправить запрос к серверу, а затем получить от него ответ. Такой подход исключает возможность

несанкционированного доступа к закрытой информации и её модифицирования.

В данном ПС в возможности сервера входит одновременное обслуживание сразу нескольких клиентов (поддержка многопоточности), клиент и сервер являются программами, взаимодействующими в рамках одного устройства. Сервер ожидает подключения и запускает новый поток для каждого клиента. По мере работы с клиентом сервер должен производить работу с базой данных.

С помощью *SQL*-запросов будет добавляться, обновляться, удаляться и выбираться вся необходимая информация из БД. Для разрабатываемого приложения наилучшим образом подойдет СУБД MySQL Workbench. Она предоставляет удобный и интуитивно понятный пользовательский интерфейс и эффективно интегрируется с кодом, написанном на Java.

Экземпляры клиентских приложений будут размещены на машинах сотрудников организации, на которых предварительно должен быть установлен *JDK 1.8* и выше. Доступ клиентских приложений к базе данных осуществляется через запросы к серверу, который, в свою очередь, связывается с БД при помощи драйвера *jdbc data provider for MySQL Server*, входящего в состав *JDK*.

Система предусматривает один способ ввода информации – вручную через пользовательский интерфейс приложения и один способ вывода – на экран машины сотрудника. Для пользователей приложения с клиентской стороны реализуется графический интерфейс, написанный с использованием стандартной библиотеки пользовательского интерфейса *JavaFX*. Эта библиотека предоставляет весь необходимый инструментарий для разработки и реализации удобных и визуально приятных форм *GUI*.

В качестве основной среды разработки в проекте используется *IntelliJ IDEA*, которая является бесплатной для пользования студентами. Для создания *UML*-диаграмм в проекте используется средство *Rational Rose* и онлайн-сервисы, которые позволяют осуществлять создание диаграмм развертывания, последовательности и компонентов с помощью простого моделирования и не требуют дополнительной инсталляции на устройство. Для выполнения *UML*-моделирования в стандарте *IDEF0* используется CASE-средство *CA AllFusion Process Modeler r7 (BPwin)*. Для информационного моделирования применяется средство *CA AllFusion ERwin Data Modeler r7 (ERwin)*.

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

Тестирование – это процесс выполнения программы, целью которого является выявление ошибок [9].

Основная функциональность описана в разделе 1.3 и представляется вариантами использования. Чтобы убедиться в корректности работы программного средства и его устойчивости к ошибкам пользователя было проведено тестирование, которое отражает возможные исключительные ситуации.

Прежде всего была рассмотрена возможность некорректной авторизации: неверное введение логина или пароля и вход без введения данных.

Программа реагирует на каждую из этих ситуаций, и пользователь видит на экране окно с описанием возникшей ошибки. Иллюстрация работы программы в данных ситуациях приведена на рисунках 3.1-3.2.

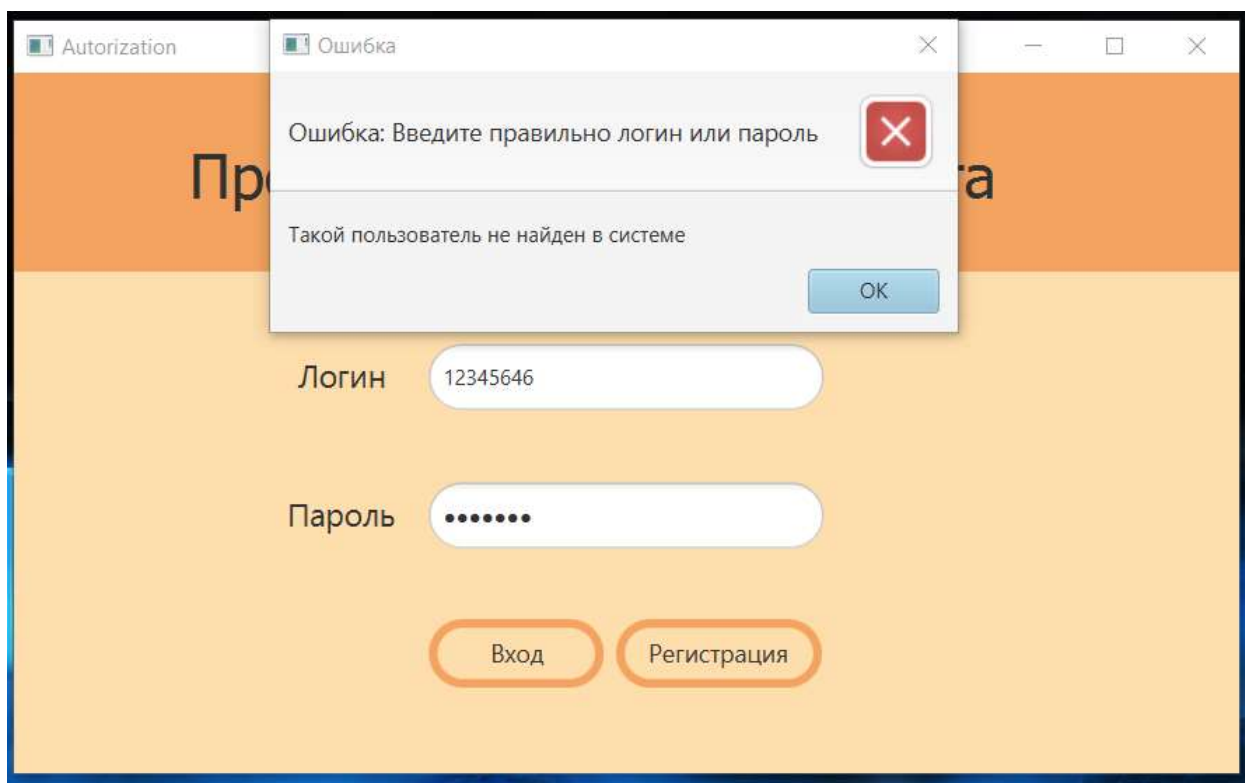


Рисунок 3.1 – Неверный ввод данных

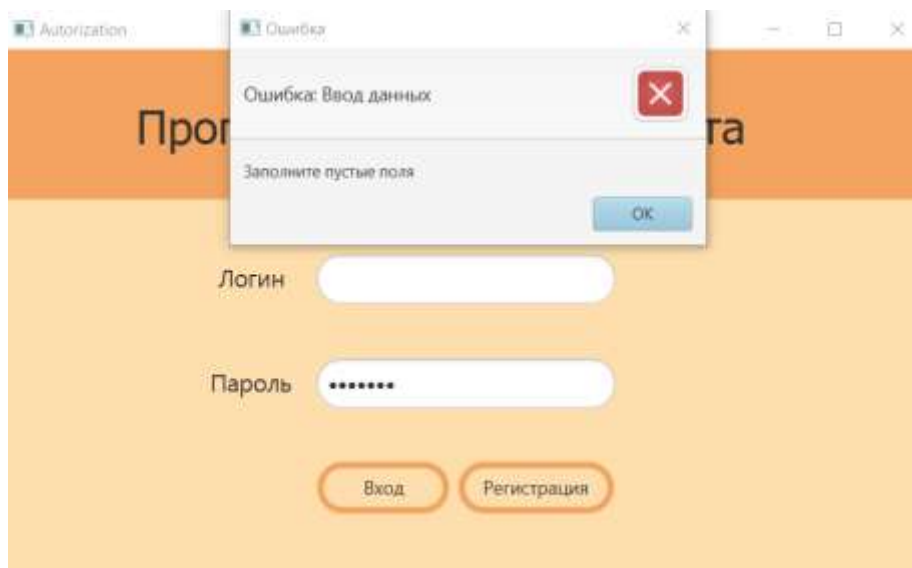


Рисунок 3.2 – Попытка входа без внесения данных

Следующая исключительная ситуация может возникнуть при заполнении полей различных форм. Например, при изменении данных о некотором работнике администратор может не ввести его фамилию. Реакция программы на такую ситуацию приведена на рисунке 3.3.

Также при регистрации пользователя может возникнуть ситуация, когда логин нового и уже существующего в программе пользователя совпадают. Чтобы исключить конфликтов комбинации логинов/паролей программа не позволяет двум пользователям иметь одинаковые логины. При попытке регистрации второго пользователя с уже существующим в программе логином на экран выводится окно ошибки. То же происходит при попытке внесения администратором повторного логина для разных работников. Описанные ситуации представлены на рисунках 3.4-3.5.

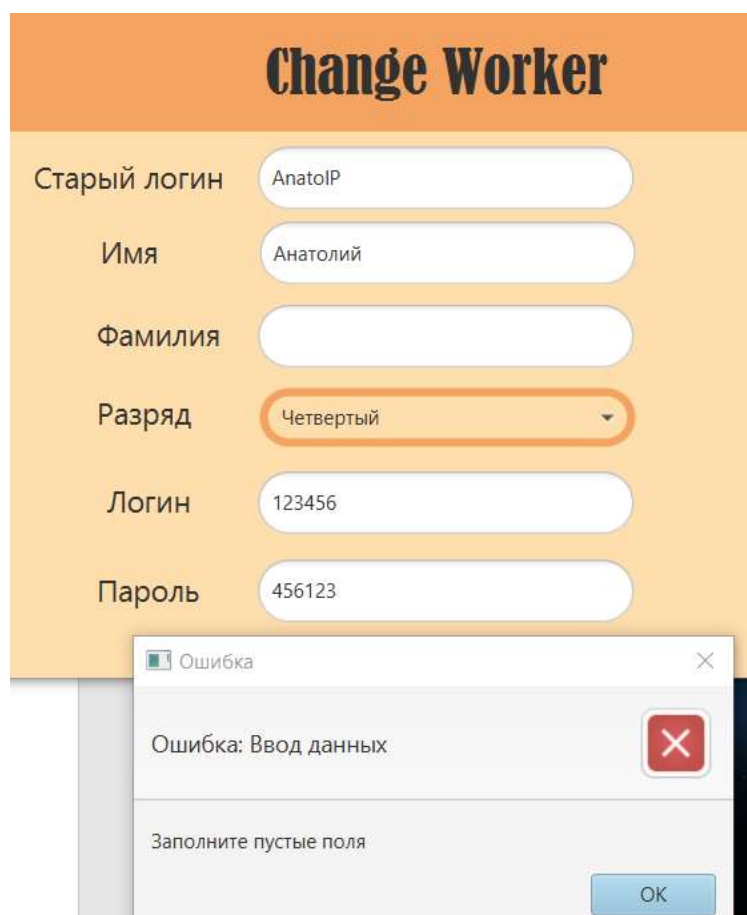


Рисунок 3.3 – Незаполненные поля формы

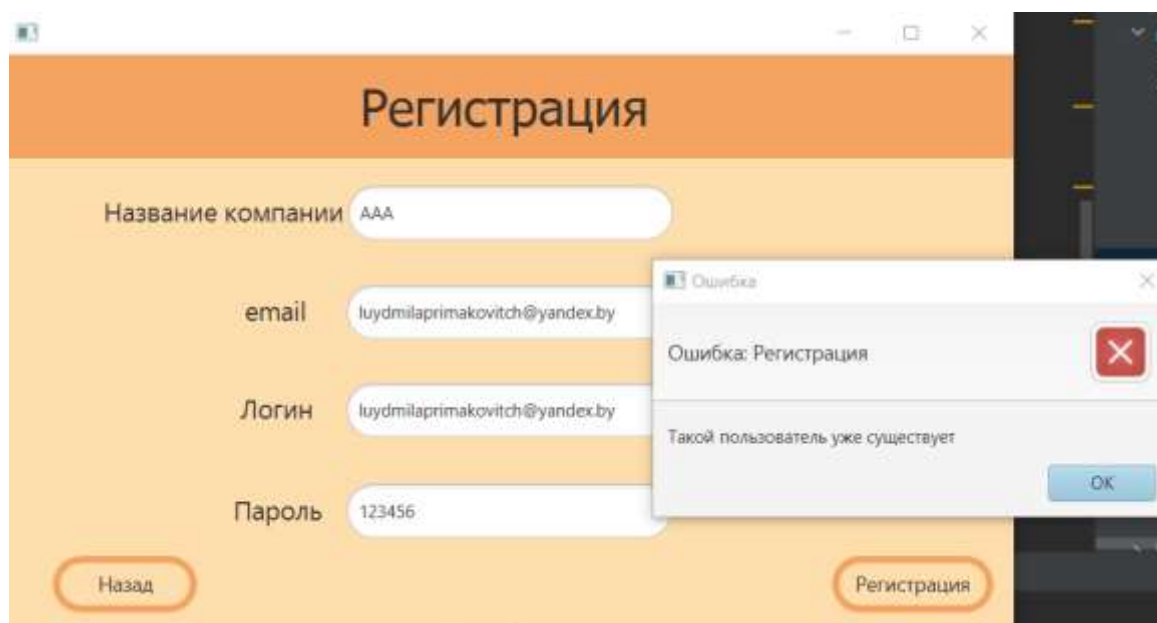


Рисунок 3.4 – Регистрация поставщика при повторении логина

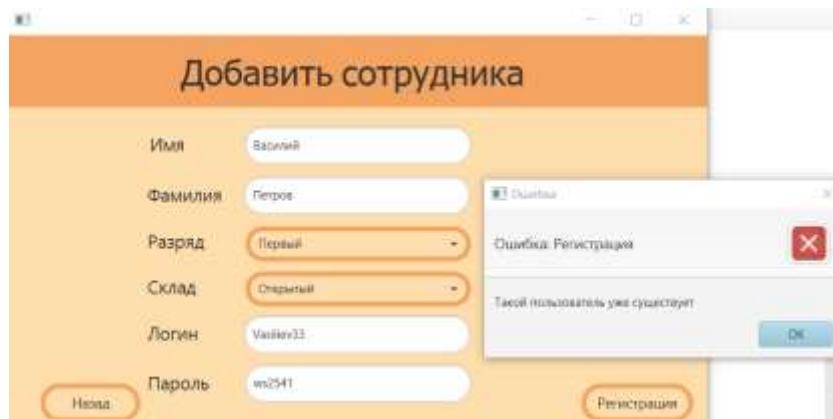


Рисунок 3.5 – Регистрация сотрудника при повторении логина

Также ошибки могут возникать при регистрации накладных поставщиками, которые могут неверно указать склад для продукции или попытаться поставить больше товара, чем склад может принять в настоящее время. В каждом из этих случаев регистрация накладной блокируется, а пользователю сообщается о некорректности введения данных. Реакция программы продемонстрирована на рисунках 3.6-3.7.

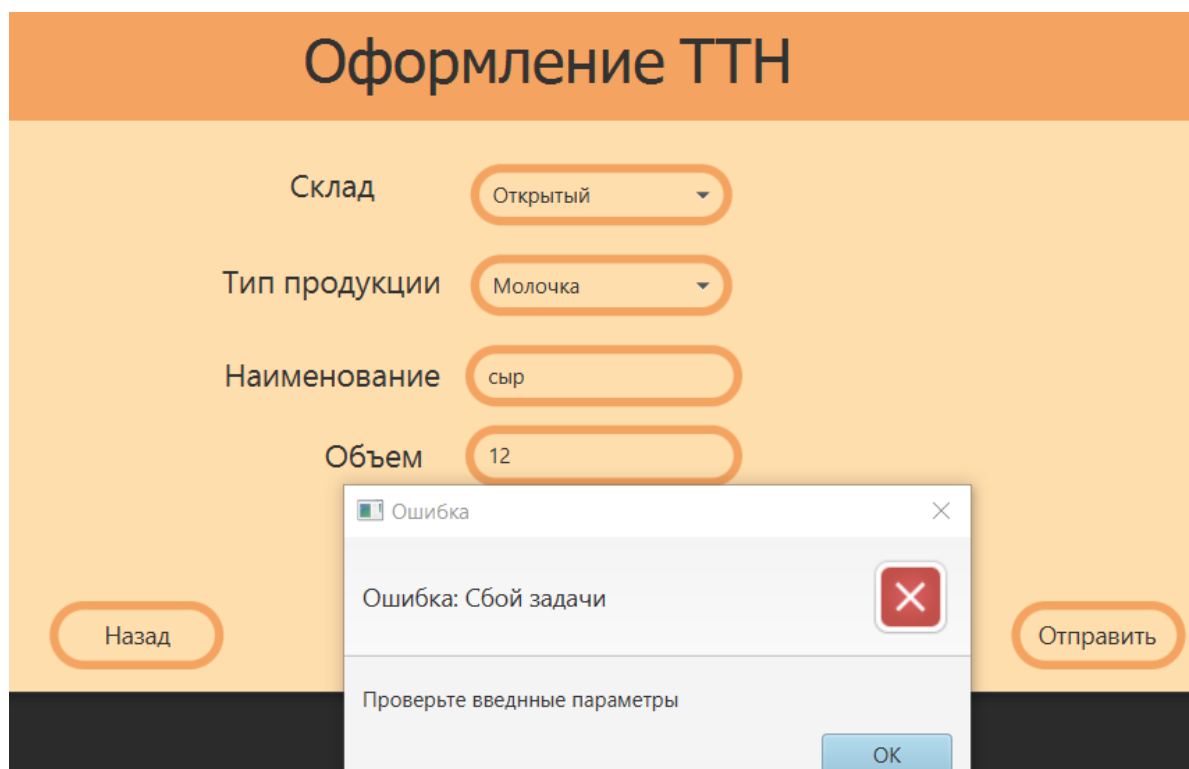


Рисунок 3.6 – Неверно выбран склад

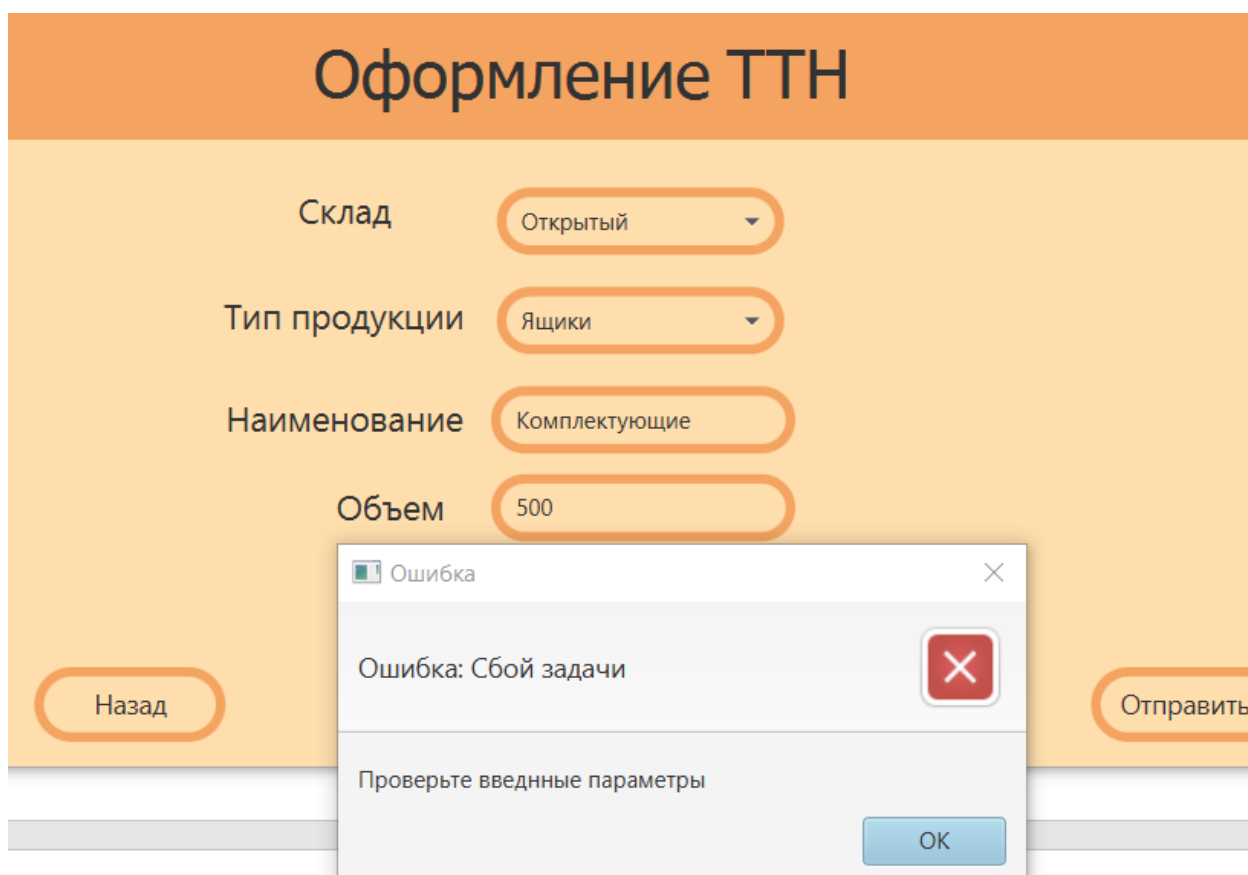


Рисунок 3.7 – Введен слишком большой объем

После вывода данных диалоговых окон с сообщениями об ошибках программа дает возможность скорректировать введенные данные и попробовать отправить заново. При успешном проведении операций выводится окно сообщения об успехе, что помогает пользователю не теряться и быть уверенным, что его данные были считаны и программа не зависла.

Исходя из результатов тестирования можно сказать, что разработанное программное средство удовлетворяет функциональным требованиям и что все его функции выполняются корректно.

4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ И СКВОЗНОЙ ПРИМЕР, НАЧИНАЯ ОТ АВТОРИЗАЦИИ, ДЕМОНИСТРИРУЯ РЕАЛИЗАЦИЮ ВСЕХ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Для начала работы необходимо запустить сначала серверную часть приложения. Сделать это можно из IDE, выбрав соответствующий файл для запуска, который называется WorkS.Start и запустив его. После запуска в консоли появится сообщение о начале работы сервера (см. рисунок 4.1).

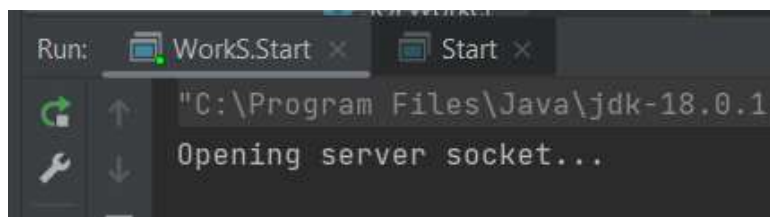


Рисунок 4.1 – Запуск сервера

По мере выполнения команд на серверной консоли будет выводиться информация об успешно и неудачно проведенных командах, о подключаемых пользователях и т.д. после начала работы серверной части аналогичным образом запускаем клиентскую часть. После запуска появляется окно авторизации, в котором требуется ввести логин и пароль (для авторизованных пользователей) или зарегистрироваться (для неавторизованных поставщиков). Для входа нет необходимости указывать роли, т.к. сервер сам определит их в соответствии с введенными данными авторизации. Вид формы авторизации представлен на рисунке 4.2.

Если данные введены корректно, то появится меню, соответствующее роли, для дальнейшей работы. Рассмотрим сначала деятельность администратора в системе. Одним из вариантов его деятельности может быть просмотр собственной личной информации (см. рисунок 4.3). Форма «Меню администратора» показана была ранее на рисунке 2.7.

Программа складского учета

Логин: luydmilaprimakovitch@yandex.by

Пароль:

Вход Регистрация

Рисунок 4.2 – Авторизация

Personal Information

Логин	Пароль
luydmilaprimakovitch@yandex.by	123456

Логин

Пароль

Назад Изменить

Рисунок 4.3 – Просмотр личной информации администратором

Администратор также может управлять данными в базе данных. Например, ему предоставлена возможность добавить нового работника или изменить информацию о старом. Также администратор может удалить данные о некотором работнике, а также добавить и изменить данные о продукции. Без

произведения этих действий администратора работники и поставщики не смогут выполнять свою деятельность. Работа программы для некоторых из указанных случаев продемонстрирована на рисунках 4.4-4.6.

The screenshot shows a web form titled "Добавить сотрудника" (Add Employee) on an orange background. The form contains the following fields and controls:

- Имя** (Name): Text input with "Василий" (Vasily).
- Фамилия** (Surname): Text input with "Куrolесов" (Kurolesov).
- Разряд** (Rank): Dropdown menu with "Третий" (Third) selected.
- Склад** (Warehouse): Dropdown menu with "Крытый" (Covered) selected.
- Логин** (Login): Text input with "VasKur".
- Пароль** (Password): Text input with "VKL22".
- Назад** (Back): Button at the bottom left.
- Регистрация** (Registration): Button at the bottom right.

Рисунок 4.4 – Добавление работника

The screenshot shows a web form titled "Change Worker" on an orange background. The form contains the following fields and controls:

- Старый логин** (Old login): Text input with "anat".
- Имя** (Name): Text input with "Анатолий" (Anatoly).
- Фамилия** (Surname): Text input with "Петров" (Petrov).
- Разряд** (Rank): Dropdown menu with "Второй" (Second) selected.
- Логин** (Login): Text input with "AnPetrov".
- Пароль** (Password): Text input with "Ap123".
- Изменить** (Change): Button at the bottom right.

Рисунок 4.5 – Изменение данных о сотруднике

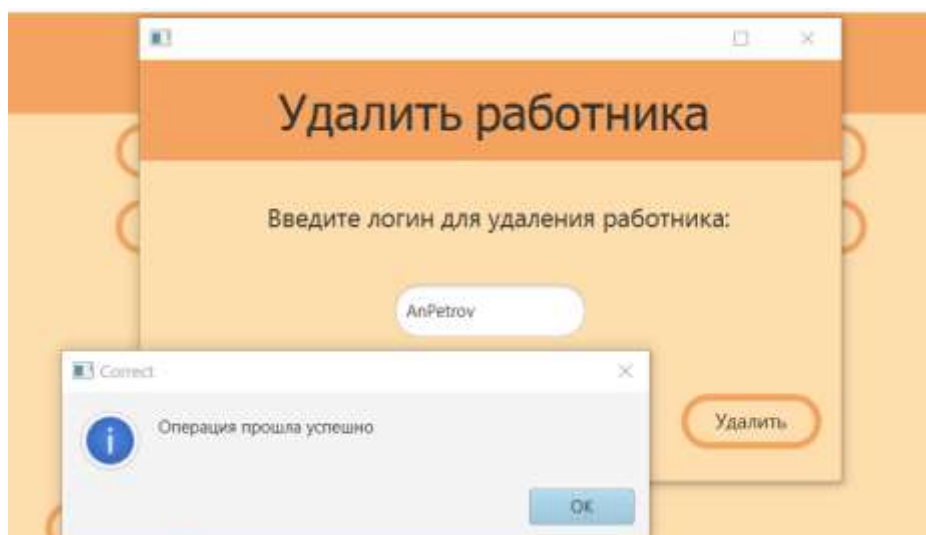


Рисунок 4.6 – Удаление работника

Без деятельности других ролей в системе производить иные действия администратор не может: для анализа поставщиков и блокировки их нужны личные данные поставщиков, а для расчета прибыли и инвентаризации – оформленные и заверенные накладные. Эти варианты использования будут рассмотрены позже.

Рассмотрим регистрацию поставщика. Примечательно, что зарегистрироваться в программе может только поставщик, т.к. работников регистрирует сам администратор. Для регистрации поставщик должен указать некоторые свои личные данные. Пример регистрации приведен на рисунке 4.7.

The image shows a web application interface with an orange header. The main content area has a light orange background. At the top, there is a title 'Регистрация' (Registration). Below the title, there are four input fields with labels to their left: 'Название компании' (Company name) with the value 'BestTree', 'email' with the value 'BTcompanY@mail.ru', 'Логин' (Login) with the value 'Ivanov1972', and 'Пароль' (Password) with the value 'bestTree10'. At the bottom left, there is a button labeled 'Назад' (Back). At the bottom right, there is a button labeled 'Регистрация' (Registration).

Рисунок 4.7 – Регистрация поставщика

После удачной регистрации поставщику сразу открывается окно поставщика (см. рисунок 2.10). Поставщик, как и администратор может просмотреть свою личную информацию. Также он может ознакомиться со списком располагаемой на складе продукции (см. рисунок 4.8).

The screenshot shows a window titled "Продукция" (Products) with a light orange background. Inside, there is a table with two columns: "Склад" (Warehouse) and "Тип продукции" (Product type). The table lists six rows of data, followed by three empty rows. Below the table, there are three buttons: "Назад" (Back) on the left, "Склад" (Warehouse) in the center, and "Найти" (Find) on the right.

Склад	Тип продукции
Закрытый	Зерно
Крытый	Фрукты/овощи
Открытый	Ящики
Рефрижераторный	Молочка
Стеллажный	Древесина
Стеллажный	Иное

Назад Склад Найти

Рисунок 4.8 – Список продукции, которую можно поставлять

Основная задача поставщика – оформить накладную. Данный вариант надежно защищен различного рода проверками на некорректности ввода, о которых говорилось в главе 3. При оформлении ТТН поставщику необходимо быть внимательным, выбирать только доступные типы продукции и соответствующие им склады, а также вводить реальные, а не запредельные объемы в метрах кубических с расчетом на то, что все склады компании рассчитаны на 5000 кубометров. Пример корректного заполнения накладной приведен на рисунке 4.9.

Оформление ТТН

Склад

Тип продукции

Наименование

Объем

Рисунок 4.9 – Оформление накладной

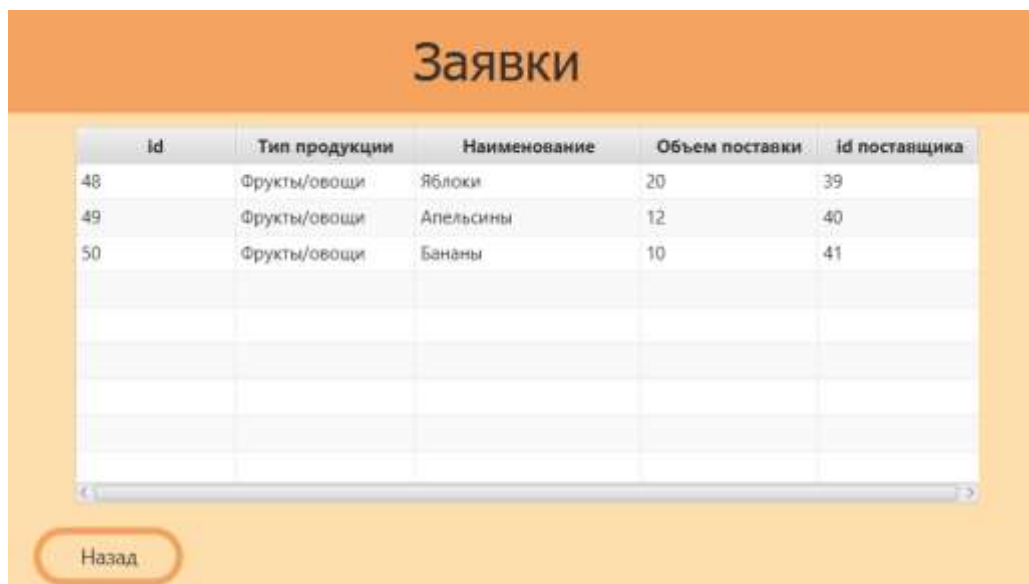
После оформления поставщик может ознакомиться с накладными, которые оформлялись от его логина за все время его деятельности в программе (см. рисунок 4.10). Также поставщик может сменить логин/пароль.

Накладные

id	Тип продукции	Наименование	Объем поставки	Статус
46	Зерно	Рапс	12	принято
47	Молочка	Сыр	23	заявка
51	Зерно	Пшеница	20	заявка
52	Молочка	Йогурт	12	ушел

Рисунок 4.10 – Просмотр собственных накладных.

Окно работы пользователя было продемонстрировано на рисунке 2.11. Роли работника доступен просмотр заявок, т.е. тех ТТН, которые связаны с его id и у которых стоит статус «заявка». Визуальное представление этого варианта использования приведено на рисунке 4.11.

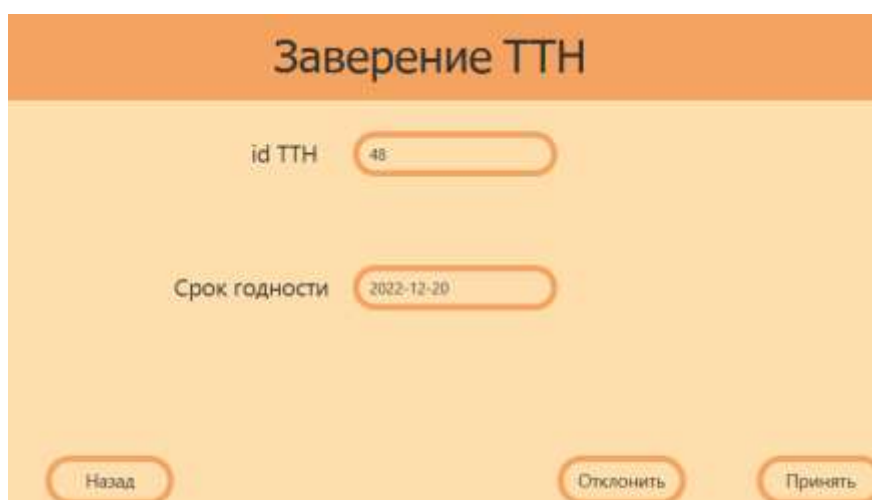


id	Тип продукции	Наименование	Объем поставки	id поставщика
48	Фрукты/овощи	Яблоки	20	39
49	Фрукты/овощи	Апельсины	12	40
50	Фрукты/овощи	Бананы	10	41

Назад

Рисунок 4.11 – Просмотр заявок на поставку

Помимо этого, работник может принять поступившие товары. Для этого он должен заверить накладную. Просмотрев список заявок, работник выбирает заявку с определенным id, и в окне заверения накладных ввести его. Затем указывается срок годности и выбирается одна из кнопок: «Принять» или «Отклонить». Работа с этим окном показана на рисунке 4.12.



Заверение ТТН

id ТТН 48

Срок годности 2022-12-20

Назад Отклонить Принять

Рисунок 4.12 – Заверение ТТН

Также пользователь может проверить свою личную информацию. Это целесообразно, т.к. данные он сам о себе не вносит, за него это делает администратор, а значит в некоторых моментах могут быть несоответствия. Также таким образом работник может посмотреть, на каком складе он работает. Пример представлен на рисунке 4.13.

Personal Information				
Логин	Имя	Фамилия	Разряд	Склад
VasKur	Василий	Куролесов	Третий	Крытый

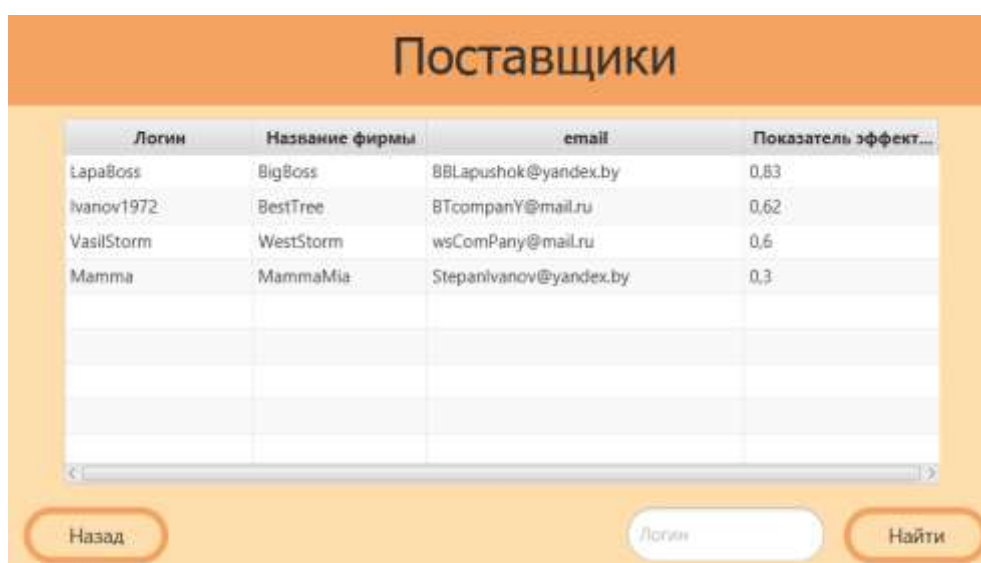
Рисунок 4.13 – Личная информация работника

После многих шагов работы работников и поставщиков в базе данных появляется достаточное количество информации, чтобы вести аналитическую деятельность администратора. Самое важное в организации работы склада – грамотно оценить поставщиков. Программа предоставляет такую возможность по нескольким направлениям (см. рисунок 4.14).



Рисунок 4.14 – Окно анализа поставщиков

Например, администратор захочет получить информацию о лучшем поставщике. Тогда список действующих поставщиков отсортируется по показателю эффективности в порядке убывания, а значит лучший поставщик окажется на самом верху (см. рисунок 4.15). показателем эффективности в данном программном средстве считаем отношение принятого объема поставок к общему заявленному объему. Другие варианты визуально выглядят аналогично, меняется только критерий сортировки: по возрастанию критерия эффективности для поиска наихудшего поставщика и по объему принятого объема по убыванию для сортировки по объемам поставки.



Логин	Название фирмы	email	Показатель эффект...
LapaBoss	BigBoss	8BLapushok@yandex.by	0,83
Ivanov1972	BestTree	BTcompanY@mail.ru	0,62
VasilStorm	WestStorm	wsComPany@mail.ru	0,6
Mamma	MammaMia	StepanIvanov@yandex.by	0,3

Назад Логин Найти

Рисунок 4.15 – Поиск наилучшего поставщика (первый в таблице)

После проведения анализа поставщиков администратор может принять решение заблокировать некоторого поставщика. Для этого достаточно выбрать кнопку блокировки поставщика и ввести его логин. Пример такого удаления приведен на рисунке 4.16.

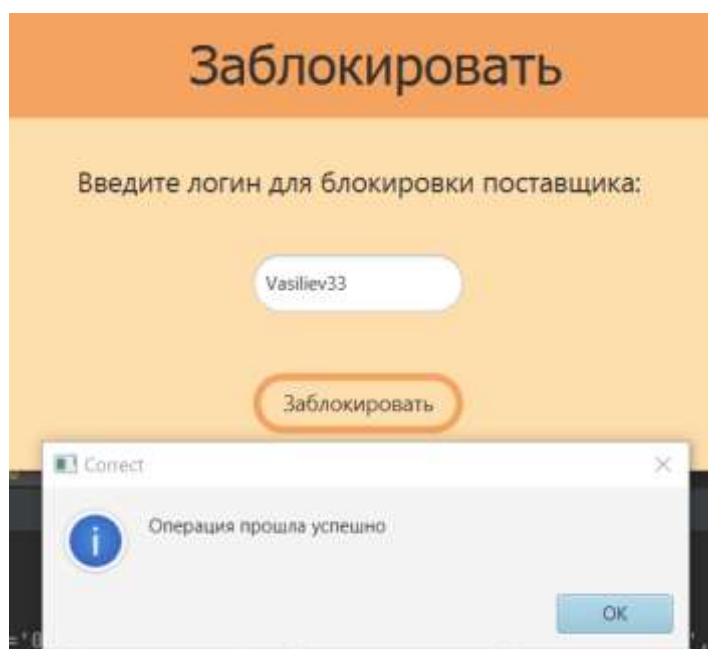


Рисунок 4.16 – Блокировка поставщика по логину

Также регистрация каждой накладной имеет свою стоимость для поставщиков (20 рублей), а значит у администратора появляется возможность рассчитать прибыль по каждому складу. Это позволит наглядно определить, на какой склад чаще всего регистрируют поставки, и, следовательно, какой из складов самый прибыльный. Демонстрация результатов такого расчета приведена на рисунке 4.17.



Рисунок 4.17 – Диаграмма прибыльности складов

При нажатии на кнопку «Сохранить» текущий расчет прибыли будет сохранен в файл в виде, показанном на рисунке 4.18.

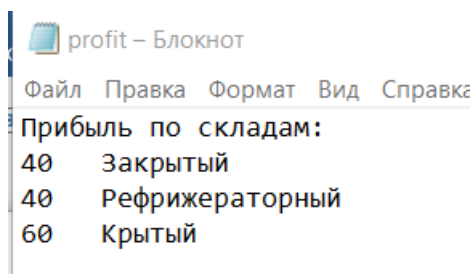


Рисунок 4.18 – Вывод данных о прибыльности складов в файл

При проведении инвентаризации статус накладных, срок годности которых истек, меняется в базе данных на «уехал» (см. рисунок 4.20). Также после проведения инвентаризации освобождается место на складе, которое было занято принятыми товарами.

Накладные				
id	Наименование	Объем поставки	Статус	Срок годности
46	Рапс	12	принято	2022-12-20
47	Сыр	23	заявка	
48	Яблоки	20	принято	2022-12-30
49	Апельсины	12	ушел	2022-12-07
50	Бананы	10	ушел	2022-12-06
51	Пшеница	20	принято	2022-12-12
52	Йогурт	12	ушел	2022-12-10
53	молоко	10	заявка	

Назад

Рисунок 4.19 – Данные о ТТН после инвентаризации

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта было разработано программное приложение, позволяющие автоматизировать процесс приемки товаров и ведение учета поступлений на склад. В нем предусмотрена система администрирования, а также проверка учетных данных при авторизации пользователей и разграничение ролей доступа.

Приложение удовлетворяет основным характеристикам, которые были заявлены. Оно удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области.

При разработке данного программного продукта была учтена логика пользователя, и интерфейс данной программы сделан удобным и понятным.

Достигнута основная цель курсового проекта: повышение качества и эффективности управления приемкой ТМЦ путем автоматизации процессов учета поступивших товаров и регистрации новых поставок внутри системы и управления информацией о поставщиках, продукции и кладовщиках.

Использование данной программы в условиях работы реальной логистической компании сделает работу автоматизированной, более быстрой и упрощенной. В дальнейшем данная программа может редактироваться и совершенствоваться в соответствии с требованиями предметной области. Подводя итог, можно сказать, что цели и задачи, поставленные перед данной работой, успешно достигнуты и выполнены.

Выполнено проектирование и разработка программного средства учета и регистрации поступления товаров на склад логистической компании: выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*-диаграмм; разработана информационная модель системы, представленная в виде схемы базы данных; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование системы, показавшее ее соответствие функциональным требованиям, поставленным в задании на разработку.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Как организовать складской учет в компании учета [Электронный ресурс]. — Режим доступа: <https://assistentus.ru/vedenie-biznesa/skladskojuchet/>.
- [2] Автоматизация складского учета на малом предприятии [Электронный ресурс]. — Электронный данные. — Режим доступа: <https://elar.rsvpu.ru/handle/123456789/29095>.
- [3] Нотация IDEF0 [Электронный ресурс]. — Электронный данные. — Режим доступа: <https://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/bpmodeling/idef0>.
- [4] Декомпозиция: как упростить себе жизнь и достичь результатов [Электронный ресурс]. — Электронный данные. — Режим доступа: <https://lpgenerator.ru/blog/dekompoziciya-chto-eto-takoe-prostymi-slovami/>.
- [5] Фаулер, М. UML. Основы. — 3-е изд. — СПб.: Символ-плюс, 2006. — 192 с.
- [6] Яковлева А.О., Язык UML. Диаграммы UML. — Методическое пособие. — Минск, 2006. — 86 с.
- [7] Архитектура «Клиент-Сервер» [Электронный ресурс]. — Электронный данные. — Режим доступа: <https://itelon.ru/blog/arkhitektura-klient-server/>.
- [8] Протокол TCP/IP [Электронный ресурс]. — Электронный данные. — Режим доступа: <https://www.ibm.com/docs/ru/aix/7.1?topic=management-transmission-control-protocolinternet-protocol>.
- [9] Рекс, Б. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование / Б. Рекс. — СПб : Лори, 2016

ПРИЛОЖЕНИЕ А
(обязательное)
Отчет о проверке на заимствования в системе «Антиплагиат»

Оригинальность 96,9%	Заимствования 3,1%	Цитирования 0%	Самоцитирования 0%
<div>ПОЛНЫЙ ОТЧЕТ КРАТКИЙ ОТЧЕТ ИСТОРИЯ ОТЧЕТОВ</div> <div>РАСПЕЧАТАТЬ ВЫГРУЗИТЬ СОЗДАТЬ ССЫЛКУ</div>			

Свойства документа

Структура документа

Текстовые метрики NEW

Имя исходного файла	ПЗ.pdf
Авторы документа	<div>Не указано</div> <div>Не указано</div>
Название документа	ПЗ
Тип документа	Не указано

Рисунок А.1 – Проверка в система «Антиплагиат»

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг кода алгоритмов, реализующих бизнес-логику

Работа серверной части:

```
package WorkS;

public class Start {
    public static final int PORT_WORK = 9006;

    public static void main(String[] args) {
        Server server = new Server(PORT_WORK);
        new Thread(server).start();
    }
}

package WorkS;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server implements Runnable {
    protected int serverPort = 9006;
    protected ServerSocket serverSocket = null;
    protected boolean isStopped = false;

    public Server(int port){
        this.serverPort = port;
    }

    @Override
    public void run(){
        openServerSocket();
        while(! isStopped()){
            Socket clientSocket = null;
            try {
                clientSocket = this.serverSocket.accept();
            } catch (IOException e) {
                if(isStopped()) {
                    System.out.println("Server Stopped.") ;
                    return;
                }
                throw new RuntimeException("Error accepting client
connection", e);
            }
            new Thread(new Work(clientSocket)).start();
            System.out.println("Клиент подключен.");
        }
        System.out.println("Server Stopped.") ;
    }
}
```

Продолжение приложения Б

```
private synchronized boolean isStopped() {
    return this.isStopped;
}

public synchronized void stop() {
    System.out.println("Stopping Server");
    this.isStopped = true;
    try {
        this.serverSocket.close();
    } catch (IOException e) {
        throw new RuntimeException("Error closing server", e);
    }
}

private void openServerSocket() {
    System.out.println("Opening server socket...");
    try {
        this.serverSocket = new ServerSocket(this.serverPort);
    } catch (IOException e) {
        throw new RuntimeException("Cannot open port " +
this.serverPort, e);
    }
}

package WorkS;

import DB.SQLFactory;
import StorOrg.*;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.sql.SQLException;
import java.util.ArrayList;

public class Work implements Runnable {
    protected Socket clientSocket = null;
    ObjectInputStream sois;
    ObjectOutputStream soos;

    public Work(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            sois = new ObjectInputStream(clientSocket.getInputStream());
            soos = new ObjectOutputStream(clientSocket.getOutputStream());
            while (true) {
```

Продолжение приложения Б

```
System.out.println("Получение команды от клиента...");
String choice = sois.readObject().toString();
System.out.println(choice);
System.out.println("Команда получена");
switch (choice) {
    case "adminInf" -> {
        System.out.println("Запрос к БД на получение
информации об администраторе: " + clientSocket.getInetAddress().toString());
        SQLFactory sqlFactory = new SQLFactory();
        ArrayList<Admin> infList =
sqlFactory.getAdmin().get();
        System.out.println(infList.toString());
        soos.writeObject(infList);
    }

    case "registrationWorker" -> {
        System.out.println("Запрос к БД на проверку
пользователя(таблица workers), клиент: " +
clientSocket.getInetAddress().toString());
        Worker worker = (Worker) sois.readObject();
        System.out.println(worker.toString());

        SQLFactory sqlFactory = new SQLFactory();

        if (sqlFactory.getWorkers().insert(worker)) {
            soos.writeObject("OK");
        } else {
            soos.writeObject("Incorrect Data");
        }
    }

    case "registrationClient" -> {
        System.out.println("Запрос к БД на проверку
пользователя(таблица clients), клиент: " +
clientSocket.getInetAddress().toString());
        Clients client = (Clients) sois.readObject();
        System.out.println(client.toString());

        SQLFactory sqlFactory = new SQLFactory();
        Role r = sqlFactory.getClients().insert(client);
        System.out.println(r.toString());

        if (r.getId() != 0 && r.getRole() != "") {
            soos.writeObject("OK");
            soos.writeObject(r);
        } else {
            soos.writeObject("This user is already
existed");
        }
    }

    case "workerInf" -> {
        System.out.println("Запрос к БД на проверку
работника (таблица workers), клиент: " +
clientSocket.getInetAddress().toString());
```

Продолжение приложения Б

```

        Role r = (Role) sois.readObject();
        System.out.println(r.toString());

        SQLFactory sqlFactory = new SQLFactory();

        ArrayList<Worker> worker =
sqlFactory.getWorkers().getWorker(r);
        System.out.println(worker.toString());
        soos.writeObject(worker);
    }

    case "changeWorker" -> {
        System.out.println("Запрос к БД на изменение
пользователя (таблица workers), клиент: " +
clientSocket.getInetAddress().toString());
        Worker worker = (Worker) sois.readObject();
        System.out.println(worker.toString());

        SQLFactory sqlFactory = new SQLFactory();

        if
(sqlFactory.getWorkers().changeWorker(worker)) {
            soos.writeObject("OK");
        } else {
            soos.writeObject("Incorrect Data");
        }
    }

    case "delWorker" -> {
        System.out.println("Выполняется удаление
работника...");

        Worker worker = (Worker) sois.readObject();
        System.out.println(worker.toString());

        SQLFactory sqlFactory = new SQLFactory();

        if
(sqlFactory.getWorkers().deleteWorker(worker)) {
            soos.writeObject("OK");
        } else {
            soos.writeObject("Ошибка при удалении
студента");
        }
    }

    case "authorization" -> {
        System.out.println("Выполняется авторизация
пользователя....");

        Authorization auth = (Authorization)
sois.readObject();

        System.out.println(auth.toString());

        SQLFactory sqlFactory = new SQLFactory();
    }

```

Продолжение приложения Б

```
        Role r = sqlFactory.getRole().getRole(auth);
        System.out.println(r.toString());
        if (r.getId() != 0 && r.getRole() != "") {
            soos.writeObject("OK");
            soos.writeObject(r);
        } else
            soos.writeObject("There is no data!");
    }

    case "addProd"->{
        System.out.println("Выполняется      добавление
продукции...");

        Products product = (Products) sois.readObject();
        System.out.println(product.toString());

        SQLFactory sqlFactory = new SQLFactory();

        if
(sqlFactory.getProducts().insertProducts(product)) {
            soos.writeObject("OK");
        } else {
            soos.writeObject("Ошибка      при      записи
продукции.");
        }
    }
}

} catch (IOException | ClassNotFoundException | SQLException e)
{
    System.out.println("Client disconnected." + e.getMessage());
}
}

package DB;

import java.sql.SQLException;

public abstract class AbstractFactory {
    public abstract SQLWorker getWorkers() throws SQLException,
ClassNotFoundException;
    public abstract SQLClients getClients() throws SQLException,
ClassNotFoundException;
    public abstract SQLAuthorization getRole() throws SQLException,
ClassNotFoundException;
    public abstract SQLProducts getProducts() throws SQLException,
ClassNotFoundException;
    public abstract SQLTTNs getTTNs() throws SQLException,
ClassNotFoundException;
    public abstract SQLAdmin getAdmin() throws SQLException,
ClassNotFoundException;
```


Продолжение приложения Б

```
    public abstract SQLReceive getReceive() throws SQLException,
ClassNotFoundException;
}
package DB;

import java.sql.SQLException;

public class SQLFactory extends AbstractFactory {
    public SQLWorker getWorkers() throws SQLException, ClassNotFoundException
    {
        return SQLWorker.getInstance();
    }

    public SQLTTNs getTTNs() throws SQLException, ClassNotFoundException {
        return SQLTTNs.getInstance();
    }
    public SQLClients getClients() throws SQLException, ClassNotFoundException
    {
        return SQLClients.getInstance();
    }

    public SQLAuthorization getRole() throws SQLException,
ClassNotFoundException {
        return SQLAuthorization.getInstance();
    }

    public SQLAdmin getAdmin() throws SQLException, ClassNotFoundException {
        return SQLAdmin.getInstance();
    }

    @Override
    public SQLReceive getReceive() throws SQLException, ClassNotFoundException
    {
        return SQLReceive.getInstance();
    }

    public SQLProducts getProducts() throws SQLException,
ClassNotFoundException {
        return SQLProducts.getInstance();
    }
}
```

Работа клиентской части:

```
package main;

import WorkC.Client;
import WorkC.Connection;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;
```

Продолжение приложения Б

```
import java.io.IOException;

public class Start extends Application{
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Start.class.getResource("/main.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 700, 400);
        stage.setTitle("Authorization");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) throws IOException {
        Connection.client = new Client("127.0.0.2", "9006");
        launch();
    }
}

package WorkC;

import java.io.*;
import java.net.Socket;

public class Client {
    private Socket clientSocket;
    private ObjectOutputStream outputStream;
    private ObjectInputStream inputStream;

    private String message;

    public Client(String ipAddress, String port){
        try {
            clientSocket = new Socket(ipAddress, Integer.parseInt(port));
            outputStream = new
ObjectOutputStream(clientSocket.getOutputStream());
            inputStream = new ObjectInputStream(clientSocket.getInputStream());
        } catch (IOException e) {
            System.out.println("Server not found: " + e.getMessage());
            System.exit(0);
        }
    }

    public void sendMessage(String message){
        try {
            outputStream.writeObject(message);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void sendObject(Object object){
        try {
```

Продолжение приложения Б

```
        outputStream.writeObject(object);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String readMessage() throws IOException {
    try {
        message = (String) inStream.readObject();
    } catch (ClassNotFoundException | IOException e) {
        e.printStackTrace();
    }

    return message;
}

public Object readObject(){
    Object object = new Object();
    try {
        object = inStream.readObject();
    } catch (ClassNotFoundException | IOException e) {

        e.printStackTrace();
    }
    return object;
}

public void close() {
    try {
        clientSocket.close();
        inStream.close();
        outputStream.close();
    } catch (EOFException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

package WorkC;

public class Connection {
    public static Client client;
    public static int id = 0;
    public static String role = "";
}

package StorOrg;

import java.io.Serializable;

public class Worker implements Serializable {
```

Продолжение приложения Б

```
int id;
String lastlogin;
String firstname;
String lastname;
String category;
String storage;
String login;
String password;

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

public String getStorage() {
    return storage;
}

public void setStorage(String storage) {
    this.storage = storage;
}

public String getLogin() {
    return login;
}
```

Продолжение приложения Б

```
public void setLogin(String login) {
    this.login = login;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getLastlogin() {
    return lastlogin;
}

public void setLastlogin(String lastlogin) {
    this.lastlogin = lastlogin;
}

@Override
public String toString() {
    return "Worker{" +
        "id=" + id +
        ", lastlogin='" + lastlogin + '\'' +
        ", firstname='" + firstname + '\'' +
        ", lastname='" + lastname + '\'' +
        ", category='" + category + '\'' +
        ", storage='" + storage + '\'' +
        ", login='" + login + '\'' +
        ", password='" + password + '\'' +
        '}';
}
}

package StorOrg;

import java.io.Serializable;

public class Role implements Serializable {
    private int id;
    private String role;

    public Role() {
        this.id = 0;
        this.role = "";
    }

    public Role(int id, String role) {
        this.id = id;
        this.role = role;
    }
}
```

Продолжение приложения Б

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}

@Override
public String toString() {
    return "Role{" +
        "id=" + id +
        ", role='" + role + '\'' +
        '}';
}
}

package StorOrg;

import java.io.Serializable;

public class Authorization implements Serializable {
    private String login;
    private String password;

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "Authorization{" +
            "login='" + login + '\'' +
            ", password='" + password + '\'' +
            '}';
    }
}
```

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг скрипта генерации базы данных

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO
_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----
-- Schema mydb
-- -----
-- Schema warehouse
-- -----

-- Schema warehouse
-- -----
CREATE SCHEMA IF NOT EXISTS `warehouse` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `warehouse` ;
-- -----
-- Create table keys
-- -----
CREATE TABLE IF NOT EXISTS `warehouse`.`keys` (
  `id_keys` INT NOT NULL AUTO_INCREMENT,
  `login` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_keys`),
  UNIQUE INDEX `login_UNIQUE` (`login` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 79
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----
-- Create table payments
-- -----
CREATE TABLE IF NOT EXISTS `warehouse`.`payments` (
  `idpayment` INT NOT NULL AUTO_INCREMENT,
  `payment` INT NOT NULL,
  `balance` INT UNSIGNED NULL DEFAULT '0',
  PRIMARY KEY (`idpayment`),
  UNIQUE INDEX `payment_UNIQUE` (`payment` ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 75
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----
-- Create table admins
-- -----
```

Продолжение приложения В

```
CREATE TABLE IF NOT EXISTS `warehouse`.`admins` (  
  `idadmins` INT NOT NULL AUTO_INCREMENT,  
  `id_keys` INT NOT NULL,  
  PRIMARY KEY (`idadmins`),  
  UNIQUE INDEX `id_keys_UNIQUE` (`id_keys` ASC) VISIBLE,  
  CONSTRAINT `fk_admins_keys`  
    FOREIGN KEY (`id_keys`)  
    REFERENCES `warehouse`.`keys` (`id_keys`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE)  
ENGINE = InnoDB  
AUTO_INCREMENT = 21  
DEFAULT CHARACTER SET = utf8mb4  
COLLATE = utf8mb4_0900_ai_ci;  
-----  
-- Create table clients  
-----  
CREATE TABLE IF NOT EXISTS `warehouse`.`clients` (  
  `idclient` INT NOT NULL AUTO_INCREMENT,  
  `companyName` VARCHAR(45) NOT NULL,  
  `email` VARCHAR(45) NOT NULL,  
  `id_keys` INT NOT NULL,  
  `idpayment` INT NULL DEFAULT NULL,  
  `averageV` VARCHAR(45) NULL DEFAULT NULL,  
  `fullV` INT UNSIGNED NULL DEFAULT '0',  
  `positiveV` INT UNSIGNED NULL DEFAULT '0',  
  PRIMARY KEY (`idclient`),  
  UNIQUE INDEX `id_keys_UNIQUE` (`id_keys` ASC) VISIBLE,  
  UNIQUE INDEX `idpayment_UNIQUE` (`idpayment` ASC) VISIBLE,  
  INDEX `fk_clients_keys_idx` (`id_keys` ASC) VISIBLE,  
  CONSTRAINT `fk_clients_keys`  
    FOREIGN KEY (`id_keys`)  
    REFERENCES `warehouse`.`keys` (`id_keys`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
  CONSTRAINT `fk_clients_payments`  
    FOREIGN KEY (`idpayment`)  
    REFERENCES `warehouse`.`payments` (`idpayment`)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE)  
ENGINE = InnoDB  
AUTO_INCREMENT = 42  
DEFAULT CHARACTER SET = utf8mb4  
COLLATE = utf8mb4_0900_ai_ci;  
-----  
-- Create table storages  
-----  
CREATE TABLE IF NOT EXISTS `warehouse`.`storages` (  
  `idstorage` INT NOT NULL AUTO_INCREMENT,  
  `type` VARCHAR(45) NOT NULL,  
  `size` INT UNSIGNED NOT NULL DEFAULT '5000',  
  PRIMARY KEY (`idstorage`),  
  UNIQUE INDEX `storage_UNIQUE` (`type` ASC) VISIBLE)
```


Продолжение приложения В

```
ENGINE = InnoDB
AUTO_INCREMENT = 22
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----
-- Create table products
-- -----
CREATE TABLE IF NOT EXISTS `warehouse`.`products` (
  `idproduct` INT NOT NULL AUTO_INCREMENT,
  `typeName` VARCHAR(45) NOT NULL,
  `idstorage` INT NOT NULL,
  PRIMARY KEY (`idproduct`),
  INDEX `fk_products_storages_idx` (`idstorage` ASC) VISIBLE,
  CONSTRAINT `fk_products_storages`
    FOREIGN KEY (`idstorage`)
      REFERENCES `warehouse`.`storages` (`idstorage`))
ENGINE = InnoDB
AUTO_INCREMENT = 19
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----
-- Create table ttns
-- -----
CREATE TABLE IF NOT EXISTS `warehouse`.`ttns` (
  `idttn` INT NOT NULL AUTO_INCREMENT,
  `productName` VARCHAR(45) NOT NULL,
  `status` VARCHAR(45) NOT NULL,
  `productSize` INT NULL DEFAULT NULL,
  `shelfLifeProd` DATE NULL DEFAULT NULL,
  `idproduct` INT NOT NULL,
  `idworker` INT NOT NULL,
  `idclient` INT NOT NULL,
  PRIMARY KEY (`idttn`),
  INDEX `fk_groups_products_idx` (`idproduct` ASC) VISIBLE,
  INDEX `fk_groups_workers_idx` (`idworker` ASC) VISIBLE,
  INDEX `fk_ttns_clients_idx` (`idclient` ASC) VISIBLE,
  CONSTRAINT `fk_ttbs_workers`
    FOREIGN KEY (`idworker`)
      REFERENCES `warehouse`.`workers` (`idworker`),
  CONSTRAINT `fk_ttns_clients`
    FOREIGN KEY (`idclient`)
      REFERENCES `warehouse`.`clients` (`idclient`)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT `fk_ttns_products`
    FOREIGN KEY (`idproduct`)
      REFERENCES `warehouse`.`products` (`idproduct`))
ENGINE = InnoDB AUTO_INCREMENT = 54
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----
-- Create table workers
-- -----
```

Продолжение приложения В

```
CREATE TABLE IF NOT EXISTS `warehouse`.`workers` (  
  `idworker` INT NOT NULL AUTO_INCREMENT,  
  `firstname` VARCHAR(45) NOT NULL,  
  `lastname` VARCHAR(45) NOT NULL,  
  `category` VARCHAR(45) NOT NULL,  
  `id_keys` INT NOT NULL,  
  `idstorage` INT NOT NULL,  
  PRIMARY KEY (`idworker`),  
  UNIQUE INDEX `id_keys_UNIQUE` (`id_keys` ASC) VISIBLE,  
  INDEX `fk_workers_keys_idx` (`id_keys` ASC) VISIBLE,  
  INDEX `fk_workers_storages_idx` (`idstorage` ASC) VISIBLE,  
  CONSTRAINT `fk_workers_keys`  
    FOREIGN KEY (`id_keys`)  
      REFERENCES `warehouse`.`keys` (`id_keys`)  
      ON DELETE CASCADE  
      ON UPDATE CASCADE,  
  CONSTRAINT `fk_workers_storages`  
    FOREIGN KEY (`idstorage`)  
      REFERENCES `warehouse`.`storages` (`idstorage`))  
ENGINE = InnoDB  
AUTO_INCREMENT = 21  
DEFAULT CHARACTER SET = utf8mb4  
COLLATE = utf8mb4_0900_ai_ci;  
  
-- -----  
-- Procedures  
-- -----  
  
DELIMITER $$  
USE `warehouse`$$  
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_ttns`(idk int, prodT  
varchar(45), stor varchar(45),  
nameP varchar(45), balance int, V int)  
BEGIN  
  CALL get_idprod(stor, prodT, @idprod);  
  CALL get_idworker(stor, @idworker);  
  call get_idclient(idk,@idc);  
  call update_storage(stor, V);  
  
  insert into `warehouse`.`ttns`(`productName`, `productSize`,  
`status`, `idclient`, `idproduct`, `idworker`)  
values (nameP, V, 'заявка', @idc, @idprod, @idworker);  
select last_insert_id() into @idttn;  
  
  call get_fullV(@idc, @fv);  
  UPDATE `warehouse`.`clients`  
  SET `fullV` = (@fv+V)  
  where `idclient` = @idc;  
  
  set @idpay = 0;  
  CALL get_idpay(@idc, @idpay);  
  case  
    when @idpay != 0  
      then CALL update_pay(@idpay, balance);
```

Продолжение приложения В

```
ELSE
    CALL insert_payment(3+V%5+2*V+@idc%13, balance, @idpayment);
UPDATE `warehouse`.`clients`
SET `idpayment` = @idpayment
    where `idclient` = @idc;
end case;
END$$

DELIMITER ;

DELIMITER $$
USE `warehouse`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `find_login`(uslogin
VARCHAR(45), uspass VARCHAR(45), OUT id_user INT, OUT usrole VARCHAR(10))
BEGIN
    SET id_user = 0;
    SET usrole = "";
    SELECT id_keys INTO id_user FROM `keys`
    WHERE `login` = uslogin AND `password` = uspass;

    SELECT COALESCE(ur, "") into usrole
    FROM ( select "client" as ur from `clients` where id_keys = id_user
    union
    select "worker" as ur from `workers` where id_keys = id_user
    union
    select "admin" as ur from `admins` where id_keys = id_user
    ) as T;
END$$

DELIMITER ;

DELIMITER $$
USE `warehouse`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `update_pay`(idp int, b int)
BEGIN
    set @newpay=0;
    select `balance` into @newpay from payments where `idpayment` = idp;
    UPDATE `warehouse`.`payments`
    SET `balance` = (@newpay+b)
        where `idpayment` = idp;
END$$

DELIMITER ;
```

ВЕДОМОСТЬ ДОКУМЕНТОВ КУРСОВОГО ПРОЕКТА