

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

Факультет инженерно-экономический
Кафедра экономической информатики
Дисциплина «Основы алгоритмизации и программирования»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсовой работы
Ассистент кафедры ЭИ
_____._____.2024 В.А.Литвинова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:
**«РАЗРАБОТКА ПРОГРАММЫ УЧЁТА
ГРУЗОВЫХ ПЕРЕВОЗОК»**

БГУИР КР 6-05-0611-01 026 ПЗ

Выполнил студент группы 378103
Летко Артём Семёнович

(подпись студента)
Курсовая работа представлена на
проверку _____._____.2024

(подпись студента)

Минск 2024

РЕФЕРАТ

БГУИР КР 6-05-0611-01 026 ПЗ

РАЗРАБОТКА ПРОГРАММЫ УЧЁТА ГРУЗОВЫХ ПЕРЕВОЗОК:
курсовая работа /А.С Летко. – Минск : БГУИР, 2024, – п.з. – 64 с., чертежей
(плакатов) – 5 л. формата А4.

Пояснительная записка 64 с., 33 рис., 2 табл., 7 источников, 2 приложения.

Ключевые слова: программное обеспечение, учет грузовых перевозок, автоматизация.

Целью данного курсового проекта является разработка программного продукта для автоматизации процесса учета грузовых перевозок.

Объект исследования – процесс учета грузовых перевозок и проведения учета грузовой перевозки.

Предмет исследования – методы и средства организации учета грузовых перевозок.

Методология проведения работы: в процессе разработки системы использованы современные подходы к обработке данных, функциональный анализ процессов, моделирование системы с помощью UML-диаграмм.

Результаты работы: были изучены основные бизнес-процессы в области учета перевозок. В ходе исследования было разработано консольное приложение, предназначенное для управления грузами и формирования учета перевозок. Созданное приложение отличается удобным и интуитивно понятным интерфейсом, что позволяет не только вести учет грузов, но и оптимизировать процессы работы.

Программный продукт разработан на языке C с применением *MS Visual Studio 2022*.

Область применения результатов: с помощью разработанного программного средства можно получить отчет о всех сделках. Разработанное программное средство полностью отвечает всем функциональным требованиям, необходимым при учёте реализации готовой продукции.

СОДЕРЖАНИЕ

Введение	4
1 Анализ и моделирование учета грузовых перевозок	6
1.1 Описание предметной области учета грузовых перевозок	6
1.2 Классификация информационных систем в грузовых перевозках.....	8
1.3 Анализ процесса проведения учета груза в нотации IDEF0	9
1.4 Представление процесса приобретения ювелирной продукции в рамках BPMN-модели.....	18
1.5 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований	19
1.6 UML-модели представления программного средства и их описание	24
2 Проектирование и конструирование автоматизированной системы учета грузовых перевозок	28
2.1 Постановка задач.....	28
2.2 Разработка модульной структуры программы	30
2.3 Выбор способа организации данных.....	31
2.4 Разработка перечня пользовательских функций программы	33
2.5 Разработка схем алгоритмов работы программы	37
3 Тестирование и проверка работоспособности программного средства	42
4 Инструкция по развертыванию приложения и сквозной тестовый пример	44
4.1 Авторизация	44
4.2 Модуль администратора	45
4.3 Модуль пользователя	48
Заключение	51
Список использованных источников	53
Приложение А	54
Приложение Б.....	55

ВВЕДЕНИЕ

В настоящее время всё большее значение приобретают грузовые перевозки. Сейчас любой желающий может заказать доставку любого груза. Всё что используют люди постоянно приходится перевозить из одного места в другое, что обуславливает актуальность выбранной темы курсовой работы.

Объектом исследования является процесс учета грузовых перевозок. Предметом исследования является совокупность научной литературы, практических опытов, юридических документов, алгоритмов вычисления способов повышения эффективности объекта исследований.

Учет грузовых перевозок — это процесс фиксации и систематического учета всех перевозок, связанных с реализацией грузовых перевозок.

Предметом исследования является совокупность научной литературы, юридических документов, практических опытов, алгоритмов вычисления способов повышения эффективности объекта исследований.

Целью данной курсовой работы является оптимизация процесса учета грузовых перевозок посредством разработки консольного приложения. Система предоставления учета грузовых перевозок будет написана на языке программирования Си, позволяет добавлять информацию грузовых перевозок, видеть общий объем доставленных грузов, знать текущие грузовые перевозки и много другое.

Для достижения поставленной цели были определены следующие задачи:

- исследовать процесс учета грузоперевозок;
- произвести анализ учета грузоперевозок;
- разработать приложение учета грузоперевозок.

Документ включает четыре раздела, каждый из которых посвящен различным аспектам управления учета грузовых перевозок. Первый раздел фокусируется на методологии IDEF0, используемой для анализа проведения учета грузовой перевозки. Здесь объясняются роли администратора и пользователя, а также подробно рассматривается информационная модель. В этом разделе представлены диаграммы UML и описывается структура файлов, необходимая для разработки системы учета грузовых перевозок.

Во втором разделе рассматривается проектирование и разработка автоматизированной системы управления учета грузовых перевозок. Определяются цели проектирования, предлагается модульная структура программы, разрабатываются пользовательские функции, и приводится руководство пользователя. В этом разделе также представлены блок-схемы алгоритмов, которые помогают понять логику работы системы.

Третий раздел посвящен тестированию и валидации разработанной программы. Здесь описываются методы тестирования, приводятся различные тестовые сценарии, рассматриваются результаты испытаний, а также обсуждаются корректирующие действия для устранения выявленных ошибок. Эти тесты необходимы, чтобы убедиться в том, что система учета грузовых перевозок работает корректно.

Четвертый раздел служит руководством по развертыванию грузовых перевозок, включая практический пример использования. Здесь подробно описывается процесс установки и настройки приложения, а также предоставляются примеры для тестирования различных модулей, таких как авторизация, модуль администратора и модуль пользователя.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ УЧЕТА ГРУЗОВЫХ ПЕРЕВОЗОК

1.1 Описание предметной области учета грузовых перевозок

Грузовая перевозка – это процесс перемещения товаров и грузов из одного места в другое с использованием различных видов транспорта. Этот процесс играет критическую роль в мировой экономике, обеспечивая поставку товаров от производителя к потребителю, а также перемещение различных видов грузов от одного места к другому в рамках логистических цепочек.

Ключевые аспекты грузовых аспектов:

1 Логистика: логистика в грузовых перевозках включает в себя планирование, координацию и контроль всего процесса перемещения товаров. Это включает определение оптимальных маршрутов, выбор наиболее эффективных транспортных средств и распределение груза по ним. Оптимизация логистических процессов помогает сократить время доставки, уменьшить затраты на транспортировку и повысить уровень обслуживания клиентов.

2 Транспортные средства: различные виды транспорта имеют свои преимущества и ограничения. Например, грузовики часто используются для малых и средних расстояний, в то время как железнодорожный транспорт может быть более эффективным для длинных расстояний. Мультимодальные перевозки, когда используется комбинация нескольких видов транспорта, становятся все более распространенными для обеспечения оптимальной доставки грузов.

3 Тарификация и ценообразование: стоимость грузовых перевозок зависит от множества факторов, включая расстояние, вес, тип груза, срочность доставки, транспортный маршрут и т.д. Ценообразование также подвержено изменениям в зависимости от рыночных условий, ставок топлива, инфляции и других экономических факторов.

4 Безопасность и страхование: грузовые перевозки подвержены различным рискам, таким как кража, повреждение или утрата груза. Поэтому безопасность играет критическую роль во всех этапах перевозки. Страхование грузов помогает защитить груз от потерь и повреждений, предоставляя компенсацию в случае несчастного случая.

5 Технологии и инновации: современные технологии играют все более важную роль в грузовых перевозках, позволяя улучшить эффективность и

безопасность процесса. Это включает в себя системы управления транспортом и складами, автоматизированные системы отслеживания грузов, дроны для доставки и т.д.

6 Законодательство и регулирование: грузовые перевозки подвержены различным законам и регулированиям, как на национальном, так и на международном уровне. Это включает в себя правила дорожного движения, таможенные законы, стандарты безопасности и другие нормативные акты.

7 Управление запасами: одним из ключевых аспектов грузовых перевозок является управление запасами. Это включает в себя определение оптимального уровня запасов товаров на складе, чтобы удовлетворить потребности клиентов и минимизировать издержки, связанные с хранением товаров.

8 Экологические аспекты: в последние годы вопросы экологии стали все более важными в грузовых перевозках. Ответственные перевозчики стремятся минимизировать воздействие своей деятельности на окружающую среду, в том числе сокращать выбросы загрязняющих веществ и использовать более экологичные виды топлива и транспортные средства.

9 Международные грузовые перевозки: мировая торговля и экономическая глобализация привели к росту международных грузовых перевозок. Это включает в себя не только транспорт товаров через границы, но и выполнение различных таможенных процедур, соблюдение международных норм и стандартов, а также учет различных культурных и юридических особенностей разных стран.

10 Службы отслеживания грузов: для обеспечения прозрачности и контроля над грузовыми перевозками используются различные службы отслеживания грузов. Они позволяют как перевозчикам, так и заказчикам, отслеживать местоположение и статус груза в реальном времени.

11 Обратные логистические процессы: грузовые перевозки не заканчиваются с момента доставки товаров клиенту. Обратные логистические процессы включают в себя управление возвратами, переработку товаров, утилизацию и обратную доставку груза в случае необходимости.

12 Аналитика и оптимизация: сбор и анализ данных о грузовых перевозках позволяет выявлять тенденции, идентифицировать узкие места и оптимизировать логистические процессы. Это может включать в себя прогнозирование спроса, оптимизацию маршрутов, управление запасами и другие аспекты.

Грузовые перевозки – это сложный и многогранный процесс, который требует внимательного планирования, координации и управления со стороны всех участников цепочки поставок.

1.2 Классификация информационных систем в грузовых перевозках

Информационные системы грузовых перевозок классифицируются в зависимости от их функциональности, области применения и уровня интеграции. Вот основные типы классификации информационных систем грузовых перевозок:

1 По области применения:

- операционные информационные системы (ОИС): ОИС предназначены для управления повседневными операциями и процессами в грузовых перевозках. Это включает в себя операции, такие как планирование маршрутов, отслеживание грузов, управление складом и т.д. ОИС обычно ориентированы на поддержку оперативных задач и предоставляют пользователю инструменты для быстрого выполнения повседневных задач.

- управленческие информационные системы (УИС): УИС предназначены для поддержки принятия стратегических решений и анализа данных в грузовых перевозках. Они предоставляют управленцам и руководителям информацию о производительности, эффективности операций и трендах в отрасли. УИС помогают формировать стратегии развития, оптимизировать бизнес-процессы и принимать обоснованные решения на основе данных.

2 По уровню интеграции:

- стандартные (автономные) системы: эти системы функционируют самостоятельно и не взаимодействуют с другими информационными системами. Они могут быть развернуты внутри организации и предоставляют функциональность для управления конкретными аспектами грузовых перевозок. Примерами могут быть отдельные системы управления транспортом (TMS) или складскими системами управления (WMS).

- интегрированные системы: эти системы объединяют в себе несколько функциональных областей и обеспечивают обмен данными между ними. Например, интегрированная информационная система грузовых перевозок может включать модули для управления заказами, складом, транспортом и финансами, обеспечивая единое управление всеми аспектами бизнеса. Интегрированные системы позволяют лучше координировать процессы и улучшить эффективность бизнеса в целом.

3 По типу технологий:

– локальные информационные системы: эти системы развертываются и используются в пределах организации или предприятия. Они могут быть разработаны и поддерживаться самой организацией или приобретаться у сторонних поставщиков. Локальные системы обычно предоставляют более полный контроль и настраиваемость, но требуют значительных инвестиций в инфраструктуру и поддержку.

– облачные информационные системы: эти системы хранятся и обрабатываются на удаленных серверах и доступны по запросу через интернет. Они обычно предоставляются в виде сервиса (SaaS) и могут быть более гибкими и масштабируемыми, чем локальные системы. Облачные системы уменьшают нагрузку на внутреннюю ИТ-инфраструктуру и могут быть более доступными для малых и средних предприятий.

4 По функциональности:

– системы управления транспортом (TMS): TMS предназначены для управления транспортными процессами, включая планирование маршрутов, назначение транспортных средств, отслеживание грузов и обработку заказов. Они обеспечивают эффективное управление транспортными операциями и помогают сократить затраты и повысить обслуживание клиентов.

– системы управления складом (WMS): WMS специализируются на управлении складскими операциями, такими как прием, хранение, отбор и отгрузка товаров. Они обеспечивают контроль над запасами и помогают улучшить эффективность складских процессов. Системы управления заказами (OMS): OMS предназначены для управления заказами и процессом их выполнения. Они обеспечивают контроль над процессом заказов, отслеживание статуса заказов и автоматизацию выполнения заказов.

1.3 Анализ процесса проведения учета груза в нотации IDEF0

Моделирование процесса начинается с построения контекстной диаграммы, используемой в целях обозначения основной функции системы и её границ. В случае с темой учета грузовых перевозок основной функцией можно назвать «Провести учёт груза» (рис. 1.1). Входными данными являются груз, бланк ТТН, топливная накладная. Выходными данными в данной модели являются чек и лист доставки. К механизмам относятся бухгалтер, аналитик данных, грузчик и водитель-экспедитор. Управление процесса происходит при использовании правила хранения товара, трудовой кодекс и правила перевозки.

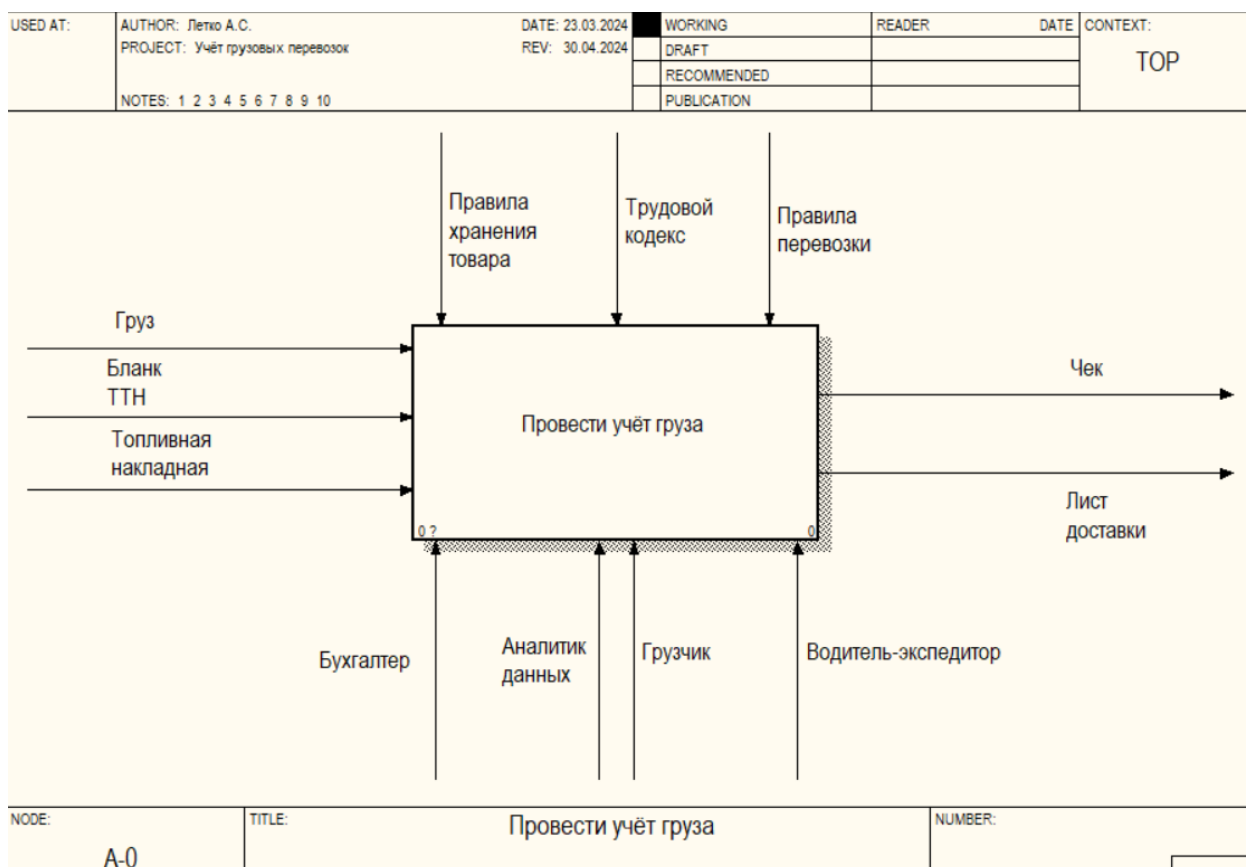


Рисунок 1.1 – Контекстная диаграмма процесса «Провести учёт груза»

Таким образом, контекстная диаграмма описывает внешние свойства системы, позволяет описать её назначение и границы системы.

Декомпозиция контекстной диаграммы (рис. 1.2) содержит в себе следующие функции: заполнить ТТН, произвести погрузку, перевезти груз, проверить груз на соответствие, подписать лист доставки. Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

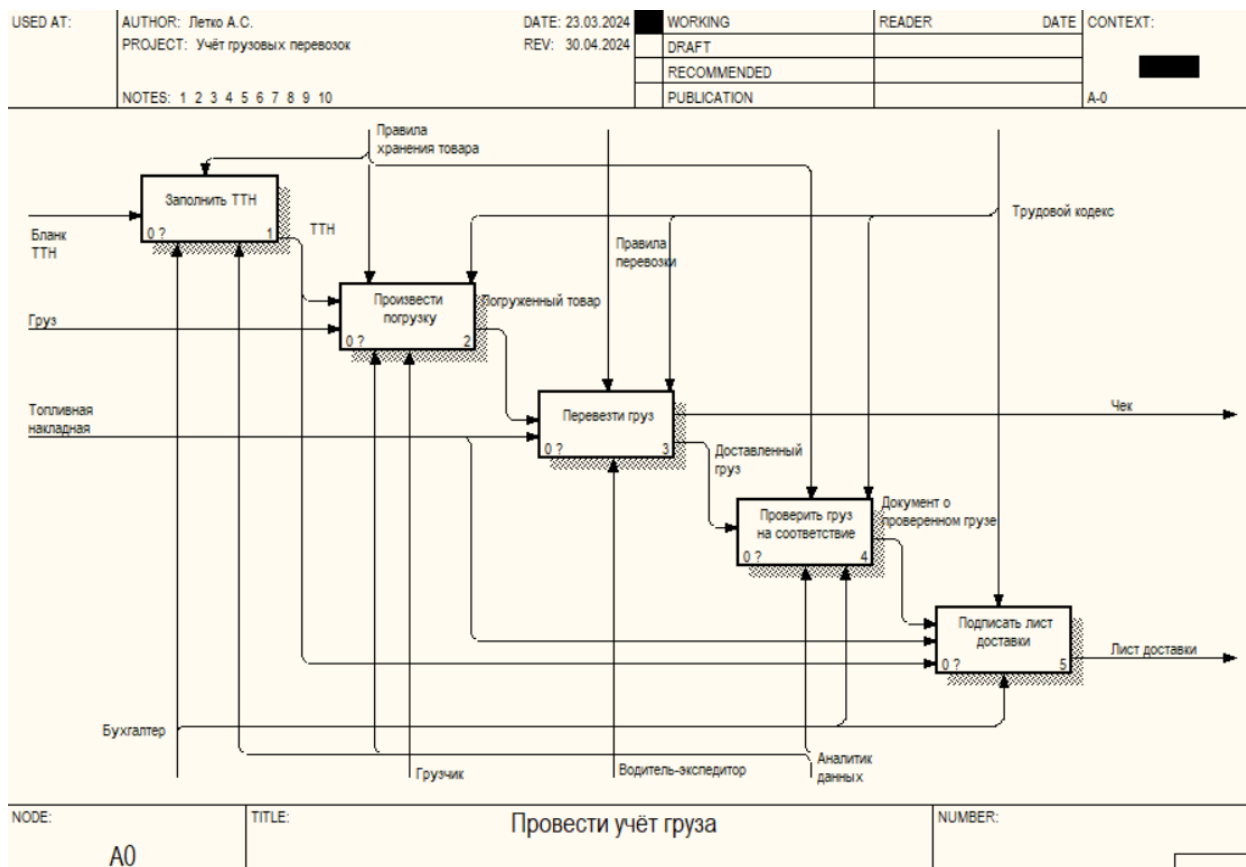


Рисунок 1.2 – Диаграмма декомпозиции процесса «Провести учёт груза»

На рис. 1.3 представлена декомпозиция процесса «Заполнить ТТН», здесь осуществляется проверка данных, заполнение и подписание бланка ТТН, который служит основанием для расчётов с перевозчиками за оказанные транспортные услуги, и с её помощью осуществляется учёт выполненной транспортной работы. Полученная информация служит основой для дальнейших этапов грузовых перевозок.

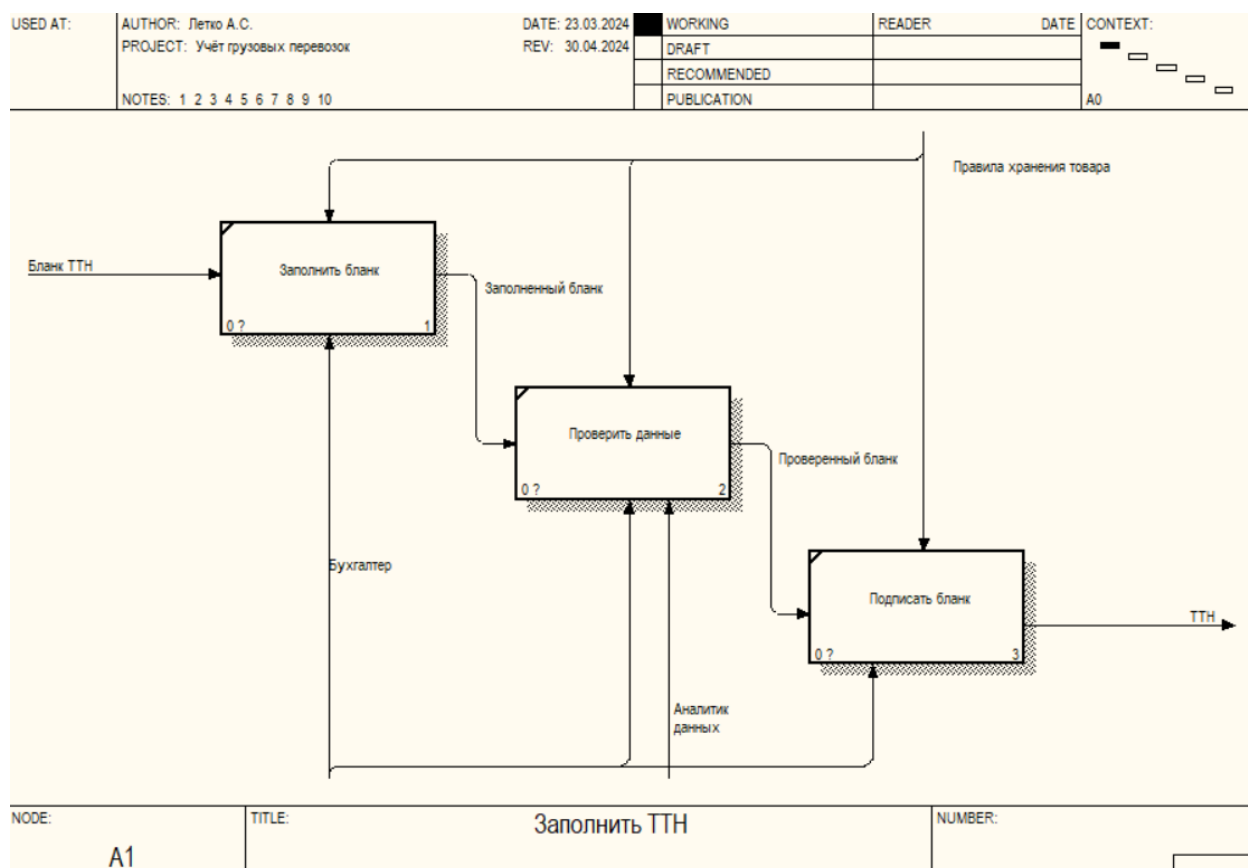


Рисунок 1.3 – Диаграмма декомпозиции процесса «Заполнить ТТН»

На рис. 1.4 представлена декомпозиция процесса «Произвести погрузку», на данной декомпозиции совершается подготовка груза к погрузке, подготовка транспорта для груза и погрузка груза. Данные процессы необходимы для дальнейших действий, где груз является важным компонентом грузовой перевозки.

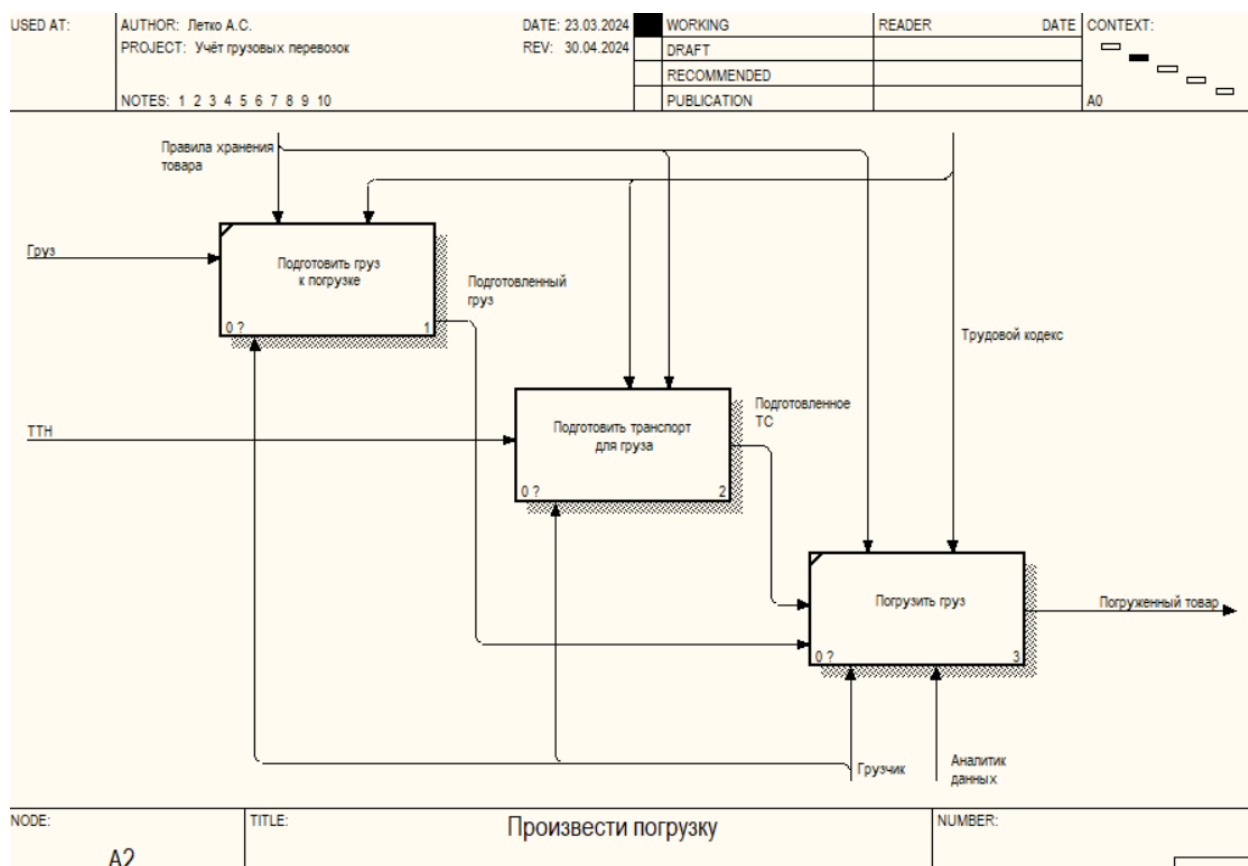


Рисунок 1.4 – Диаграмма декомпозиции процесса «Произвести погрузку»

На рис. 1.5 представлена декомпозиция процесса «Перевезти груз», на этом этапе проводится построение маршрута, проверка транспортного средства, заправка транспортного средства и доставка груза. В результате построенного маршрута и определённого груза мы можем подготовить определённое транспортное средство для преодоления различных препятствий.

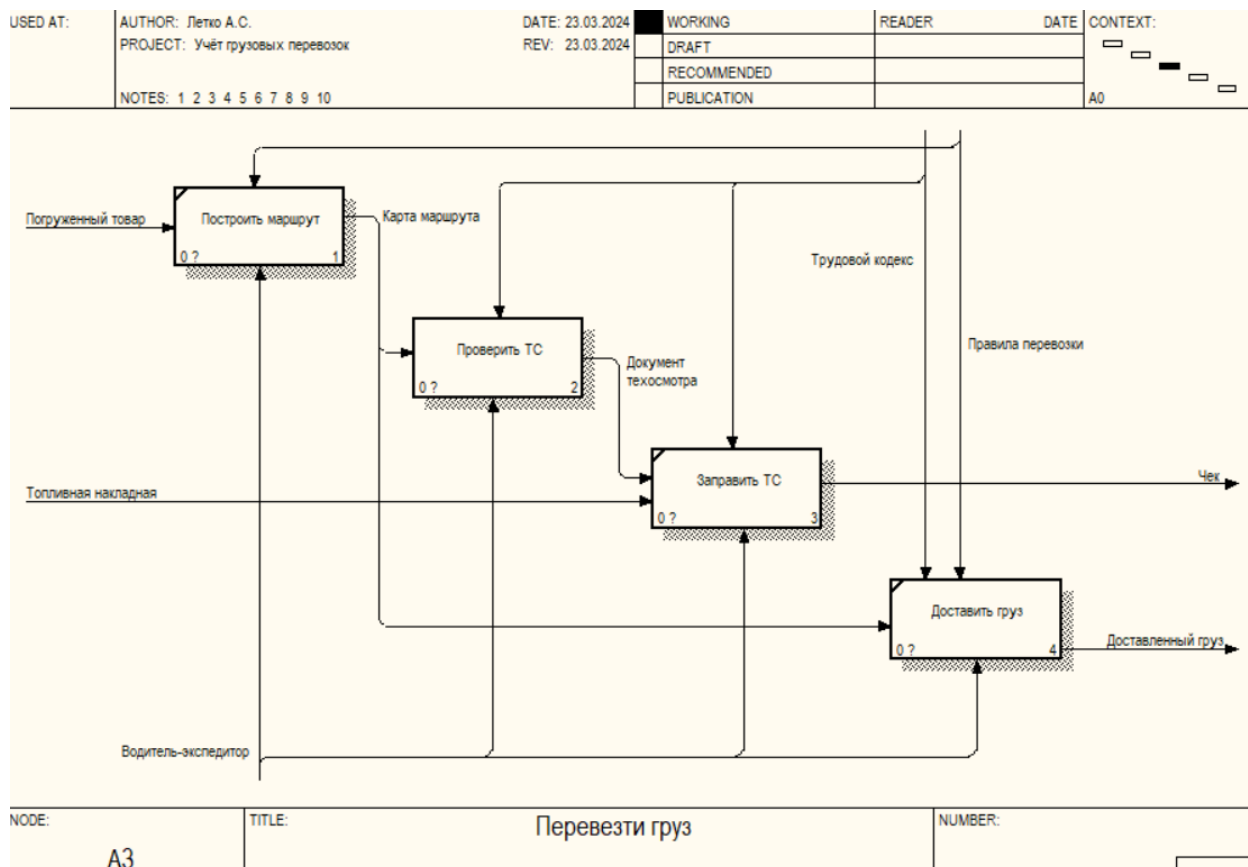


Рисунок 1.5 – Диаграмма декомпозиции процесса «Перевезти груз»

На рис. 1.6 представлена декомпозиция процесса «Проверить груз на соответствие», на этом этапе осуществляется осмотр груза визуально, сверка груза с документами и составление документа о проверенном грузе. Результаты этого анализа позволяют выявить любые отклонения или несоответствия предполагаемого груза. Это важный этап, поскольку на основе полученных результатов можно принимать решения о дальнейших действиях.

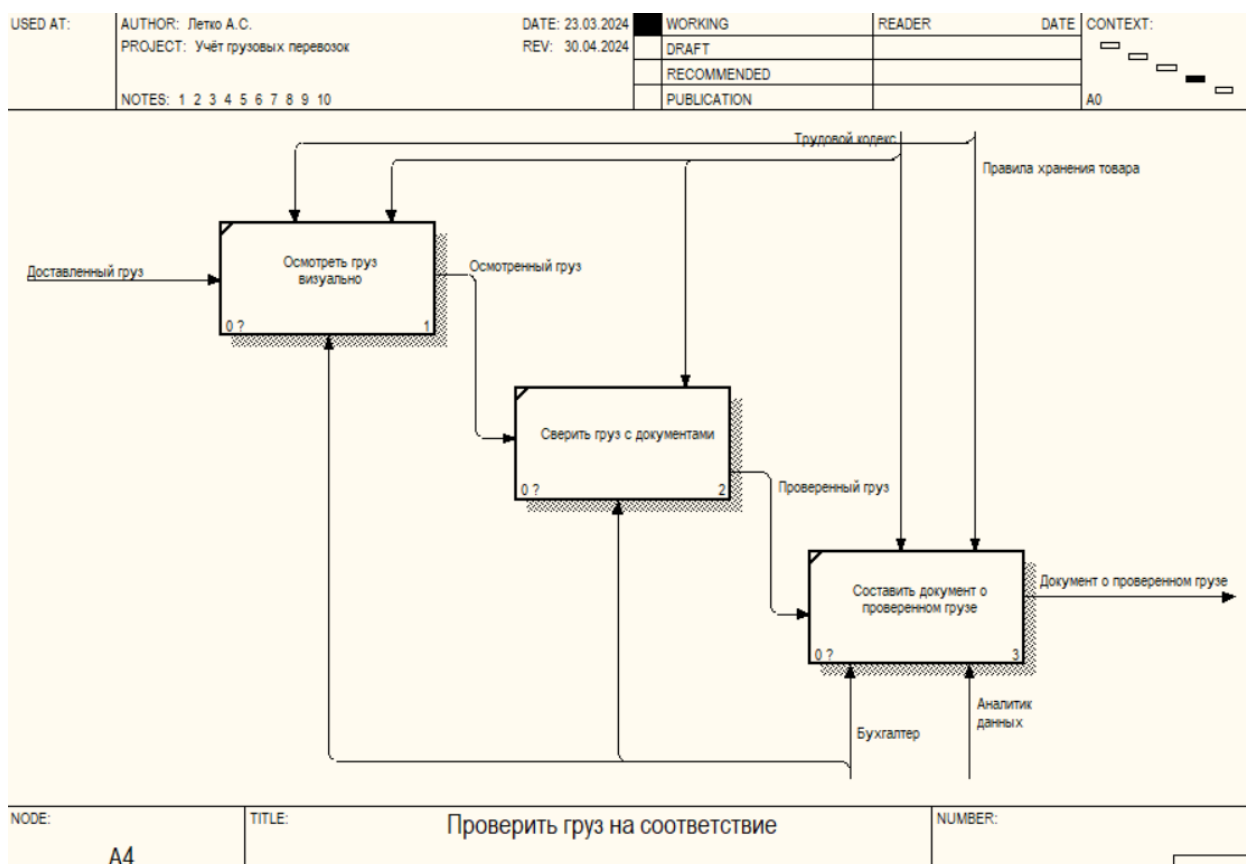


Рисунок 1.6 – Диаграмма декомпозиции процесса «Проверить груз на соответствие»

На рис. 1.7 представлена декомпозиция процесса «Подписать лист доставки», на данном уровне осуществляется проверка накладных, подготовка листа и его подпись. Это важные этапы в документообороте, которые помогают обеспечить правильность и законность перемещения товаров.

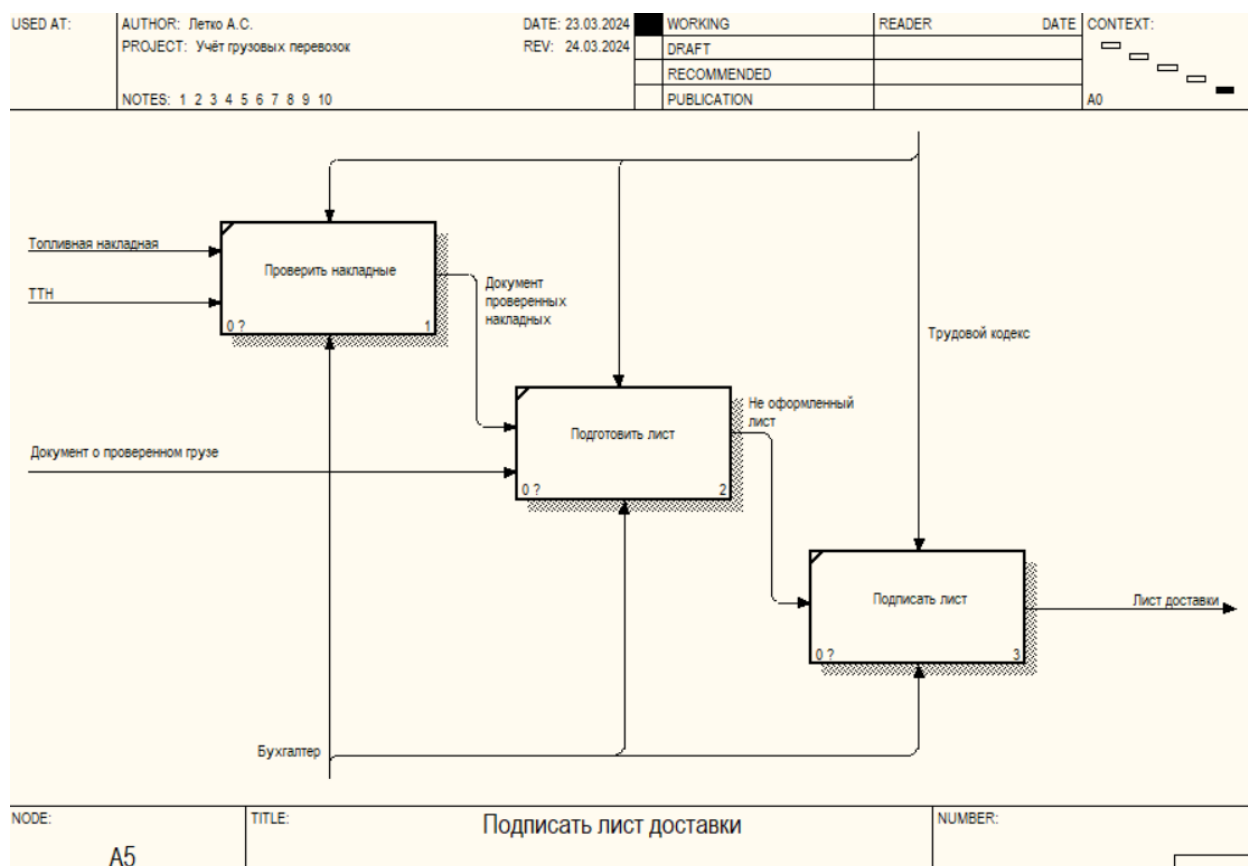


Рисунок 1.7 – Диаграмма декомпозиции процесса «Подписать лист доставки»

На рис. 1.8 представлена декомпозиция процесса «Составить отчетность», в которой осуществляется получение списка доступных ТС для груза, выбор ТС и отгон ТС к месту погрузки. Результатом выбора подходящего ТС важно для эффективной и безопасной транспортировки грузов.

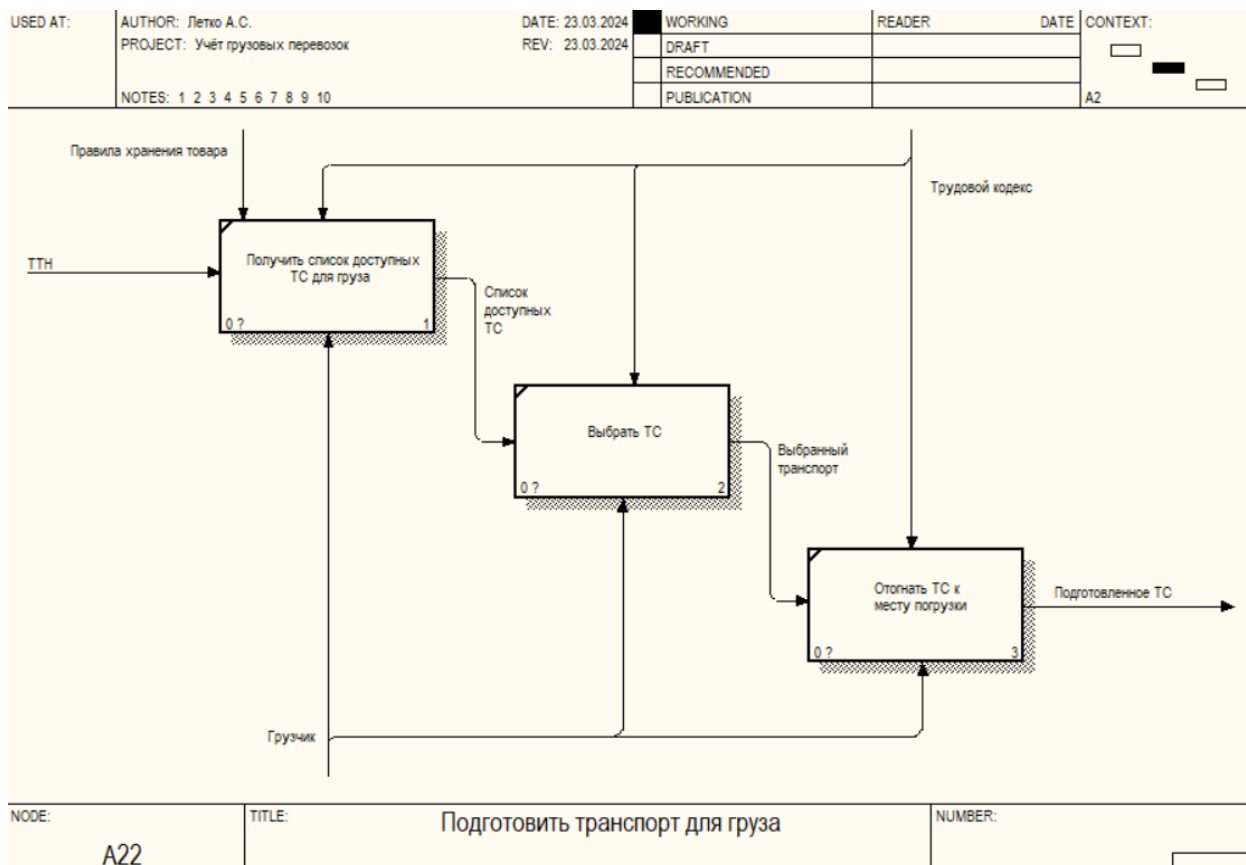


Рисунок 1.8 – Диаграмма декомпозиции процесса «Подготовить транспорт для груза»

На рис. 1.9 представлена декомпозиция процесса «Проверить ТС», в которой осуществляется проверка давления в шинах, осмотр двигателя и проверка уровня топлива. Проверка давления в шинах можно выявить неисправность и проверить соответствие давления рекомендациям производителя. Осмотр двигателя может обнаружить следы износа, повреждения или загрязнения, а также проверить уровень состояния масла, фильтров и системы охлаждения. При проверке уровня топлива можно проверить работу датчика уровня топлива и оценить точность показаний на приборной панели.

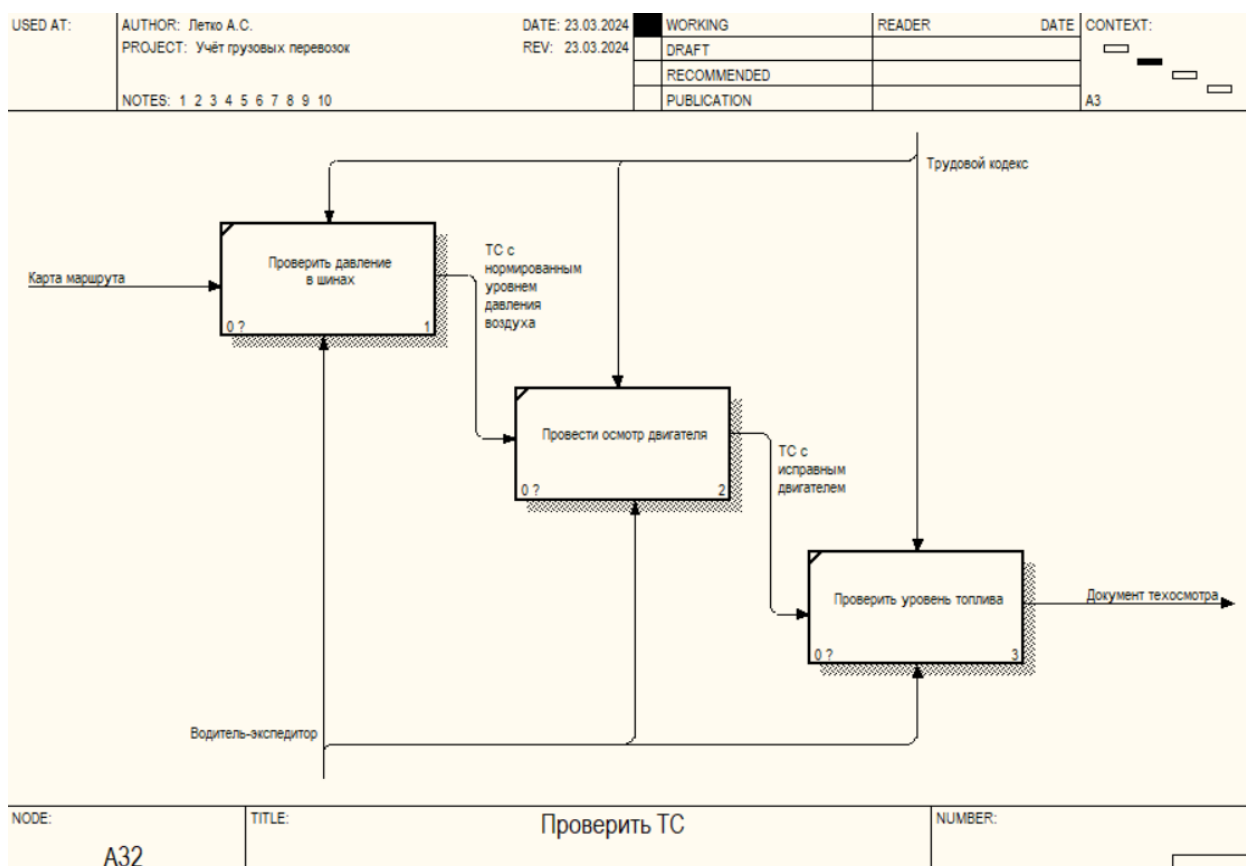


Рисунок 1.9 – Диаграмма декомпозиции процесса «Проверить ТС»

Таким образом, использование метода моделирования IDEF0 обеспечивает комплексный подход к управлению грузовыми перевозками, что способствует повышению уровня качества, удовлетворенности заказчиков и конкурентоспособности предприятия в отрасли.

1.4 Представление процесса приобретения ювелирной продукции в рамках BPMN-модели

Процесс приобретения груза начинается с требований клиента. Клиент оформляет заявку на груз организации.

После, организация получает заявку и производит анализ на корректность запроса, а также производит проверку на доступность груза. Если заявка будет не успешной, то клиенту заказ отменяется, иначе клиент осведомляется об успешном заказе и теперь он должен произвести оплату счёта. Если счёт будет оплачен неуспешно, то клиент должен будет повторить оплату ещё раз, иначе происходит погрузка груза, вывоз груза на указанную в заявке точку расположения.

Использование BPMN-моделей в процессе приобретения груза позволяет организациям структурировать и визуализировать каждый этап процесса, идентифицировать возможные узкие места и оптимизировать процесс для повышения эффективности и улучшения качества обслуживания клиентов.

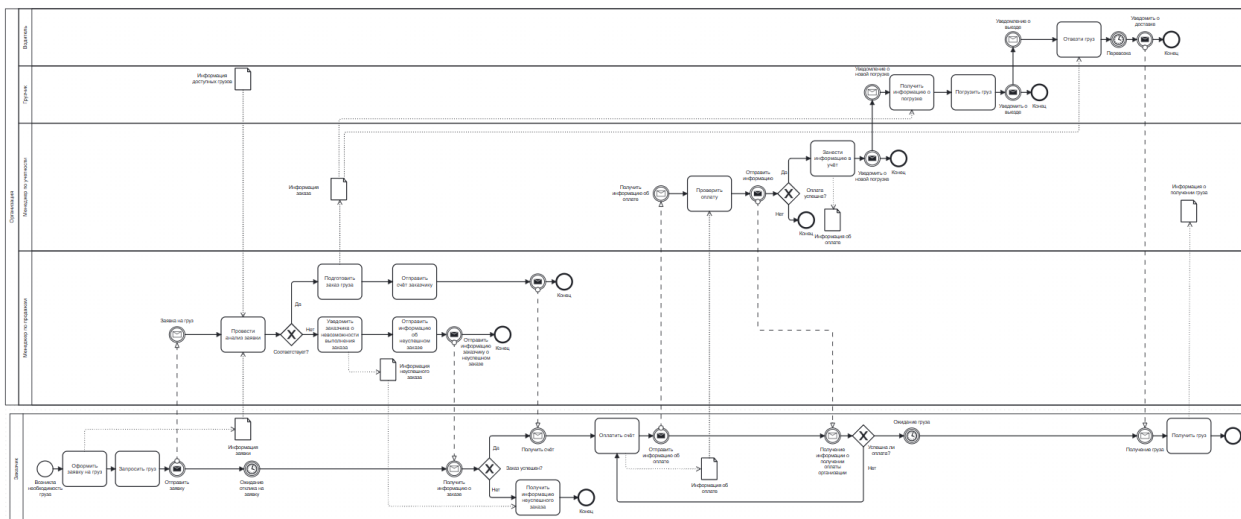


Рисунок 1.10 – BPMN – модель учета грузовых перевозок

1.5 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Для выполнения курсовой работы требуется создать программное приложение, соответствующее определенным требованиям. Приложение должно включать в себя возможность разделения пользователей на роли пользователя и администратора с различными уровнями доступа.

Администратор должен иметь возможность управлять учетными записями пользователей, включая добавление, редактирование и удаление. Также администратор должен иметь возможность работать с данными, включая их редактирование и обработку.

Пользователь, в свою очередь, должен иметь возможность просматривать данные, вводить информацию о товаре, а также просматривать информацию об остатках и стоимости товара.

Приложение должно поддерживать функции поиска и сортировки данных по различным параметрам, запись информации в файл, а также обработку исключительных ситуаций, таких как некорректный ввод данных пользователем или отсутствие файла с записями для чтения.

Также необходимо предусмотреть навигацию, запрос на подтверждение выполнения определенных действий, таких как удаление записей, и обратную связь с пользователем.

Для реализации поставленных задач разработаем информационную модель системы. Для этого выделим следующие базовые сущности:

Администратор: обладает возможностью управления учетными записями пользователей и работой с данными.

Пользователь: имеет доступ к просмотру данных, вводу информации о перевозке груза.

Данная модель позволит эффективно организовать систему и обеспечить реализацию всех необходимых функций, учитывая различные роли пользователей. Разделение пользователей на роли позволит лучше контролировать доступ к функционалу приложения и обеспечит безопасность данных. Администратор сможет эффективно управлять учетными записями пользователей, а также иметь контроль над данными и операциями в системе. Пользователи, в свою очередь, будут иметь доступ только к тем функциям, которые необходимы для выполнения своих задач, что повысит удобство использования приложения и сократит вероятность ошибок. Такая модель также способствует упрощению обслуживания и поддержки приложения, так как каждая роль имеет четко определенные функциональные возможности, что облегчит процесс обучения новых пользователей и сопровождения системы в будущем.

В таблице 1.1 перечислены актёры, функции и варианты использования.

Таблица 1.1 Функции и варианты использования

Актер	Функции	Варианты использования
Администратор	Управление учетными записями пользователей	Добавление новых пользователей в систему
		Редактирование информации о существующих пользователях
		Удаление учетных записей при необходимости
		Совершение сортировок пользователей
		Блокировка учётных записей

Продолжение таблицы 1.1

Актер	Функции	Варианты использования
	Управление данными	Просмотр, редактирование и удаление данных
		Выполнение операций по обработке данных, таких как импорт и экспорт
		Настройка уровней доступа
	Управление доступом	для различных пользовательских ролей
		Контроль за безопасностью данных
	Управление системой	Мониторинг работы приложения и выявление возможных проблем
		Решение технических вопросов и поддержка пользователей
Пользователь	Просмотр информации о товарах	Просмотр актуального ассортимента товаров
		Оценка остатков товаров на складе
	Внесение информации о товарах	Добавление информации о новых поступлениях товаров в базу данных
		Обновление информации о товарах в случае изменений в ассортименте
	Поиск и фильтрация товаров	Поиск конкретного товара по его наименованию
		Фильтрация товаров по ценовому диапазону
	Создание отчетов	Подготовка отчетов для руководства о текущей ситуации и трендах в продажах
	Просмотр данных	Просмотр информации о товарах, остатках и стоимости
		Получение актуальной информации о продукции

Продолжение таблицы 1.1

Актер	Функции	Варианты использования
	Поиск и сортировка данных	Использование функций поиска и фильтрации для нахождения нужной информации
		Сортировка данных по различным параметрам
	Взаимодействие с системой	Возможность просмотра своего профиля
	Взаимодействие с администратором	Обратная связь с администратором по вопросам работы приложения и предложениями по его улучшению

Для наглядного представления всех возможностей разработанного программного продукта используется диаграмма вариантов использования (use case-диаграмма) в нотации UML, представленная на рисунке 1.11.

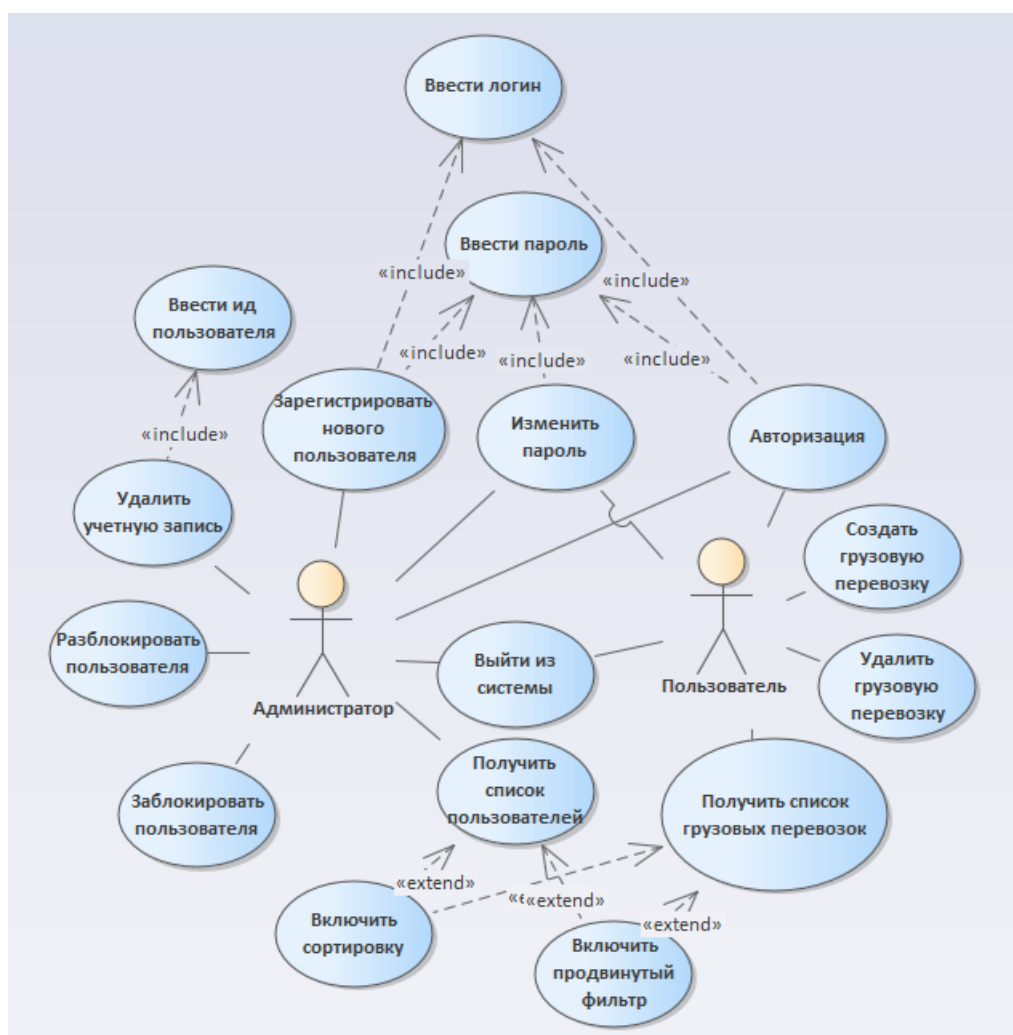


Рисунок 1.11 – Диаграмма вариантов использования системы учета грузовых перевозок

Данная модель позволит эффективно организовать систему и обеспечить реализацию всех необходимых функций, учитывая различные роли пользователей. Разделение пользователей на роли позволит лучше контролировать доступ к функционалу приложения и обеспечит безопасность данных. Администратор сможет эффективно управлять учетными записями пользователей, а также иметь контроль над данными и операциями в системе. Пользователи, в свою очередь, будут иметь доступ только к тем функциям, которые необходимы для выполнения своих задач, что повысит удобство использования приложения и сократит вероятность ошибок. Такая модель также способствует упрощению обслуживания и поддержки приложения, так как каждая роль имеет четко определенные функциональные возможности, что облегчит процесс обучения новых пользователей и сопровождения системы в будущем.

1.6 UML-модели представления программного средства и их описание

1.6.1 Диаграмма последовательности

Для анализа взаимодействия различных компонентов программного обеспечения часто используется диаграмма последовательности. На диаграмме последовательности программного обеспечения, предназначенного для учета грузовых перевозок (рисунок 1.12), отображены этапы взаимодействия между его компонентами.

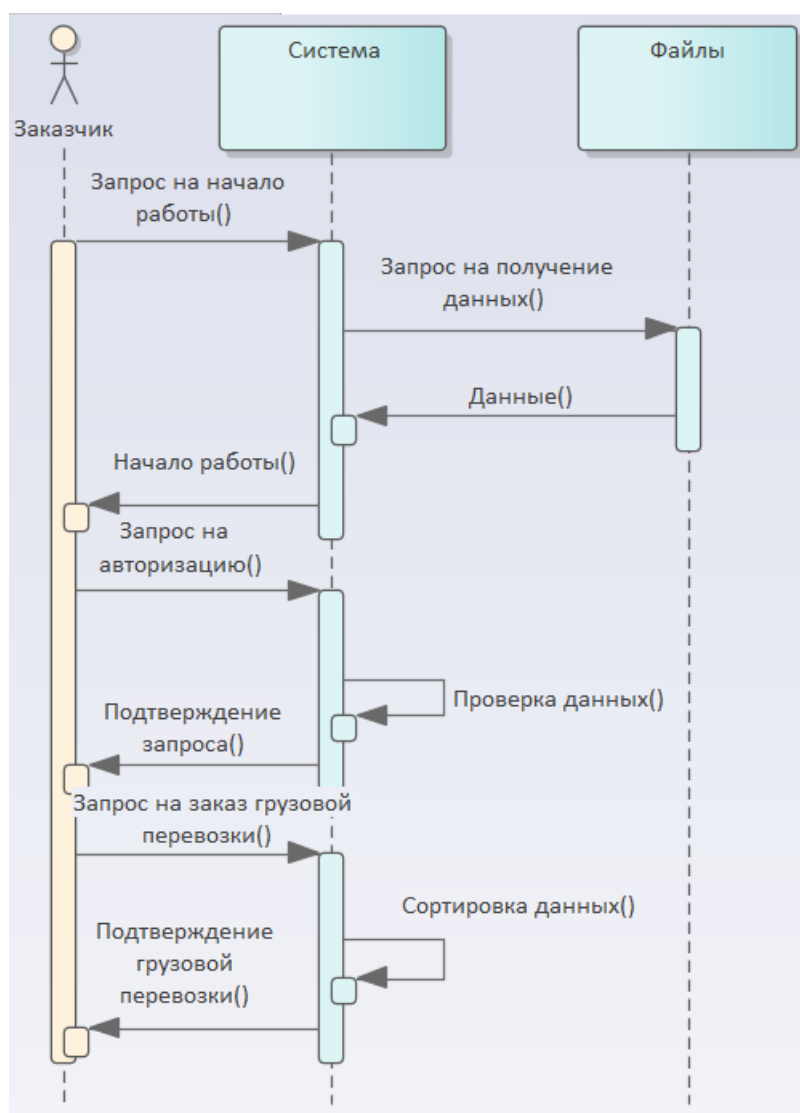


Рисунок 1.12 – Диаграмма последовательности работы приложения

Процесс работы диаграммы последовательности для учета грузовых перевозок начинается с запроса груза заказчиком. Затем логистическая компания запрашивает ТС для доставки на нём груза клиенту. После происходит перевозка груза заказчику.

1.6.2 Диаграмма состояния

Для того, чтобы детально изучить работу программного средства необходимо понимать, в каких состояниях может находиться программное средство. Для этого служит диаграмма состояний. Диаграмму состояний системы учёта продаж ювелирных изделий можно увидеть на рисунке 1.13.



Рисунок 1.13 – Диаграмма состояний учета

В этой диаграмме состояния:

1 Груз поступил на склад: начальное состояние, в котором груз прибывает на склад для последующей обработки и отгрузки.

2 Груз подготовлен к отгрузке: груз был обработан на складе и готов к отправке. В это состояние он переходит после подготовки к отгрузке.

3 Груз находится в пути: груз был отправлен и находится в процессе доставки клиенту.

4 Груз доставлен на склад клиента: груз доставлен на склад клиента, но еще не принят им.

5 Груз принят клиентом: груз был принят клиентом и учет его завершен.

Эта диаграмма состояния иллюстрирует основные этапы и переходы в процессе учета грузовых перевозок от момента поступления на склад до их доставки клиенту.

1.6.3 Диаграмма компонентов

Диаграммы компонентов предоставляют способ наглядно представить структуру программной системы, разделив ее на различные компоненты и показывая связи и зависимости между ними. Эти компоненты могут включать файлы, библиотеки, модули, исполняемые файлы, пакеты и другие подобные объекты. Диаграммы компонентов особенно полезны для демонстрации детальной структуры системы и иллюстрации того, как различные компоненты взаимодействуют между собой через интерфейсы или схемы. Такой подход позволяет лучше понять организацию кода и архитектуру программы на низком уровне абстракции. На рисунке 1.14 приведена диаграмма компонентов разрабатываемой системы:

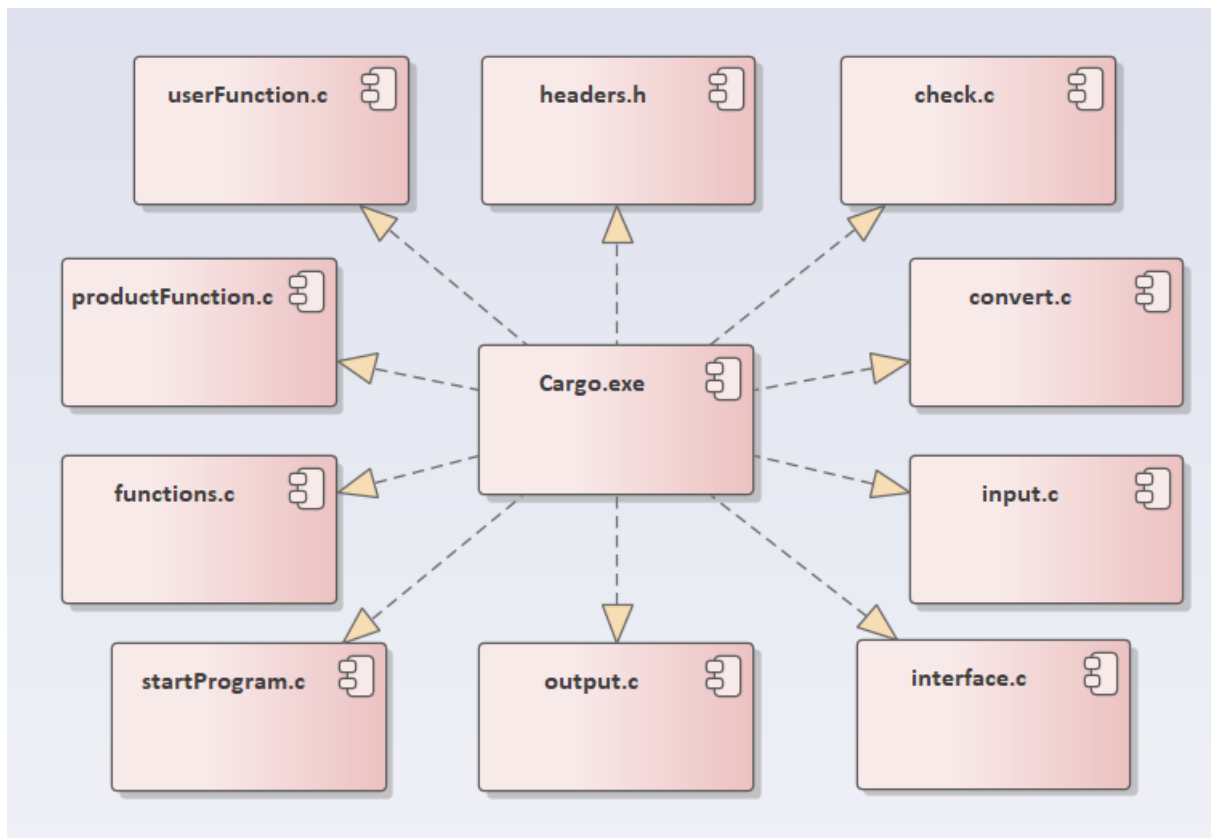


Рисунок 1.15 – Диаграмма компонентов

На диаграмме есть два основных компонента:

- Cargo.exe, который представляет исполняемый файл – верхний уровень системы;
- check.c, convert.c, input.c, interface.c, output.c, startProgram.c, functions.c, productFunction.c, userFunction.c, headers.h, которые представляют компоненты кода или модули, которые составляют программу. Файлы check.c, convert.c, input.c, interface.c, output.c, startProgram.c, functions.c, productFunction.c и userFunction.c содержат исходный код программы, а headers.h содержит объявления функций и структур данных, используемых в других файлах.

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УЧЕТА ГРУЗОВЫХ ПЕРЕВОЗОК

На этапе проектирования и конструирования программы учета грузовых перевозок осуществляется создание ее структуры и архитектуры с учетом требований и спецификаций. Этот этап включает в себя определение структур данных для хранения информации о грузовых перевозках, выбор алгоритмов для обработки данных, разработку модулей и плана реализации программы.

Программисты и архитекторы программного обеспечения работают над созданием архитектуры и дизайна программы, учитывая как функциональные, так и нефункциональные требования к ней. Это включает в себя определение структур данных, которые будут использоваться для хранения информации о грузовых перевозках и пользователей, выбор подходящих алгоритмов для обработки данных, а также проектирование пользовательского интерфейса для удобства взаимодействия с программой.

Важной частью проектирования и конструирования программы является определение модульной структуры, то есть разделение программы на отдельные компоненты или модули, каждый из которых выполняет определенную функцию. Например, модули для добавления, просмотра и удаления грузовых перевозок могут быть реализованы отдельно для обеспечения лучшей структурированности и управляемости программы.

Целью проектирования и конструирования программы учета грузовых перевозок является создание качественного и эффективного программного обеспечения, которое удовлетворяет потребности в учете грузовых перевозок. Для этого необходимо правильно спроектировать архитектуру программы, выбрать подходящие технологии и инструменты разработки, а также провести тестирование и оптимизацию программы перед ее внедрением.

Исходя из поставленной задачи "разработать систему учета продаж в ювелирном магазине", ожидается, что система будет обладать всем необходимым функционалом, современным интерфейсом, безопасностью и быстродейственностью. Можно выделить следующие требования к системе учета продаж.

2.1 Постановка задач

Исходя из поставленной задачи "разработать систему учета грузовых перевозок", ожидается, что система будет обладать всем необходимым

функционалом, современным интерфейсом, безопасностью и быстродейственностью. Можно выделить следующие требования к системе учета грузовых перевозок.

2.1.1. Технические проблемы, которые должна решать система:

- управление данными;
- безопасность данных;
- производительность;
- затрата ресурсов времени;
- человеческий фактор.

2.1.2. Задачи, решаемые системой:

- учет грузовых перевозок;
- представление данных пользователю по запросу;
- обеспечение сортировки, поиска данных;
- регистрация грузовых перевозок;
- генерация отчетов;
- возможность авторизации.

2.1.3. Требования к продукту. Разработанная программная поддержка должна быть реализована в виде консольного-приложения на языке Си. Используемые фреймворки, библиотеки и языки:

- системные библиотеки;
- стандартные библиотеки языка Си;
- фреймворк для тестирования (CUnit);
- утилиты для сборки и управления проектом (Make, CMake);
- документация (Doxygen);

2.1.4. Требования к дополнительным технологиям. Должны быть использованы следующие инструменты:

- «CMake», версия 3.8.1 (утилита для сборки проекта);
- «Valgrind», версия 3.10.0 (инструмент для анализа памяти);
- «Doxygen», версия 1.8.0 (система для генерации документации);
- «passport», версия 0.4.1 (прослойка для авторизации);
- «libxml2», версия 2.0.1 (библиотека для работы с XML);
- «GDB», версия 9.4.0 (инструмент для отладки кода);
- «Check», версия 0.13.1 (фреймворк для модульного тестирования);

2.2 Разработка модульной структуры программы

Процесс разработки модульной структуры программы для учета грузовых перевозок представляет собой разбиение кода на отдельные независимые компоненты, называемые модулями. Каждый модуль выполняет конкретную функцию или решает определенную задачу, что облегчает разработку, тестирование и поддержку программы. Основные принципы разработки модульной структуры включают выделение функциональных блоков, минимизацию связей между модулями и повторное использование кода.

В данном проекте программа организована с использованием файлов с расширениями `.c` и `.h`, которые выступают в роли отдельных модулей, специализированных на выполнении определенных задач. Файлы `.c` содержат исходный код программы на языке Си, включая реализацию функций, методов и других элементов программы. Эти файлы являются основным местом для написания кода, который компилируется в исполняемый файл программы. Файлы `.h` содержат только объявления функций, переменных и других элементов, которые будут использоваться в программе. В заголовочных файлах определяется интерфейс каждого модуля, включая предоставляемые функции и их использование. Обычно эти заголовочные файлы включаются в другие файлы с помощью директивы `#include`, что обеспечивает доступность функций и типов данных компилятору для корректной работы программы. Рассмотрим основные модули программы:

- модуль `interface.c` отвечает за отображение всех интерфейсов. Он обеспечивает удобный и интуитивно понятный интерфейс для взаимодействия пользователя с программой;
- модуль `check.c` содержит универсальные функции для проверки корректности введенных пользователем типа данных;
- модуль `convert.c` содержит универсальные функции, которые конвертируют некоторые типы данных в другие;
- модуль `input.c` содержит универсальные функции, которые отвечают за ввод и клавиатуры и с файла;
- модуль `output.c` содержит универсальные функции, которые отвечают за вывод в консоль, либо в файл
- модуль `productFunction` содержит универсальные функции, которые отвечают за работу для работы с грузовыми перевозкам;
- модуль `userFunction` содержит универсальные функции, которые отвечают за работу для работы с пользователями.

Каждый модуль выполняет свою уникальную функцию и взаимодействует с другими модулями для обеспечения полноценного функционирования программы. Модульная структура облегчает разработку, тестирование и поддержку программы, делая ее код более структурированным и понятным.

2.3 Выбор способа организации данных

При выборе метода организации данных важно определить типы информации, которые будут использоваться в приложении. Это могут быть числовые значения различных типов, текстовые строки, даты, идентификаторы и другие. В данном проекте была выбрана структура данных для организации информации.

Структура данных представляет собой более сложную концепцию, чем простое объединение переменных разных типов под одним именем. В структуре каждая переменная называется членом и может быть различных типов, включая базовые типы (например `int`, `double`, `char` и `float`) и пользовательские типы данных, в том числе другие структуры. Она обеспечивает гибкий инструмент для создания комплексных сущностей, объединяющих необходимые данные для решения задач программы.

В данном проекте для организации данных были внедрены следующие структуры, описанные в таблице 2.1.

Таблица 2.1 Описание структур

Структуры	Описание	Поля
User	Содержит информацию о пользователе.	id – идентификатор privilege – роль в системе blocked – блокировка char* login – логин char* solid – соль char* hash – хэш
Product	Содержит информацию о грузе.	id – идентификатор codeIdentification – (ИН) идентификационный номер number – количество грузов weight – вес груза name – название груза

Продолжение таблицы 2.1

Структуры	Описание	Поля
ProductTransportation	Содержит информацию о грузовой перевозке.	id – идентификатор numberProducts – количество продуктов numberStops – количество остановок priceGeneral – общая цена fuleConsumed – количество затраченного топлива distanceTraveled – пройденное расстояние typeTransport – тип транспорта countryOrigin – страна отправки countryDelivery – страна доставки townOrigin – город отправки townDelivery – город доставки dateOrigin – дата отправки dateDelivery – дата доставки timeOrigin – время отправки timeDelivery – время доставки products – массив грузов
Transportations	Содержит информацию о грузовых перевозках.	id – идентификатор number – количество грузовых перевозок

В данном проекте каждая из этих структур предназначена для хранения конкретного типа данных, необходимого для работы приложения.

В проекте, было принято решение использовать бинарные файлы для хранения данных. Бинарные файлы представляют собой компактный и эффективный способ хранения информации, обеспечивая быструю запись и считывание данных. Они отлично подходят для работы со структурированными данными, такими как информация о финансовых операциях, поскольку позволяют сохранять информацию в оптимальном формате, минимизируя размер файла и обеспечивая удобный доступ к данным.

Выбор бинарных файлов в качестве метода организации данных обусловлен их преимуществами, такими как высокая производительность при обработке данных, надежность и простота использования. Этот подход

позволит эффективно управлять финансовой информацией пользователей и обеспечить быстрый доступ к данным в любой момент времени.

2.4 Разработка перечня пользовательских функций программы

Этот раздел посвящен определению и описанию функциональности, которую приложение будет предоставлять пользователям. Важно учитывать различные потребности пользователей и обеспечивать удобный и интуитивно понятный интерфейс для взаимодействия с программой. Вот основные пользовательские функции, которые будут реализованы в нашем приложении:

– Запуск системы: при запуске системы пользователя встречает стартовый интерфейс на котором он может выбрать авторизацию или выход из системы.

```
void basicMenu() {
    int programIsWork = 1;
    while (programIsWork) {
        clearConsole();
        int choice = inputFirstNatural(2, BASICMENU);
        switch (choice) {
            case 1:
                authorizationMenu();
                break;
            case 2:
                programIsWork = 0;
                clearConsole();
                printf_s("Вы успешно вышли из системы!");
                break;
        }
    }
    return;
}
```

– Изменение пароля: пользователь может изменить пароль своего аккаунта.

```
void resetPassword(struct User currentUser) {
    clearConsole();
    printf_s("Введите новый пароль:");
    char* newPassword = getPressedCharacters(LENGTHMAXPASSWORD);
    if (strlen(newPassword) < LENGTHMINPASSWORD)
        printf_s("\nПароль слишком короткий!\n");
    else {
        printf_s("\nПодтвердить новый пароль[Enter], иначе[любая другая клавиша].\n");
        struct User* arrayUsers = NULL;
        int countUsers = 0;
        if (getPressedCharacter() == 13 && inputFileUsers(&arrayUsers, &countUsers) && arrayUsers) {
            for (int _user = 0; _user < countUsers; _user++)
                if (arrayUsers[_user].id == currentUser.id) {
                    if (arrayUsers[_user].login)
                        free(arrayUsers[_user].login);
                    if (arrayUsers[_user].solid)
                        free(arrayUsers[_user].solid);
                    if (arrayUsers[_user].hash)
                        free(arrayUsers[_user].hash);
                    arrayUsers[_user].login = _strdup(currentUser.login);
                    arrayUsers[_user].hash = toHash(newPassword, (arrayUsers[_user].solid =
generateSolid()));
                    arrayUsers[_user].id = currentUser.id;
                }
        }
    }
}
```

```

        arrayUsers[_user].privilege = currentUser.privilege;
        arrayUsers[_user].blocked = currentUser.blocked;
        break;
    }
    if (saveFileUsers(&arrayUsers, &countUsers))
        printf_s("Пароль был изменён!\n");
    else
        printf_s("Пароль не был изменён!\n");
}
else
    printf_s("Пароль не был изменён!\n");
if (arrayUsers)
    free(arrayUsers);
}
pressAnyButton();
if (newPassword)
    free(newPassword);
}

```

— Авторизация: происходит вход в аккаунт пользователем

```

void authorizationMenu() {
    int programIsWork = 1;
    char* inputLogin = NULL, *inputPassword = NULL;
    while (programIsWork) {
        clearConsole();
        int choice = inputFirstNatural(4, AUTHORIZATIONMENU);
        switch (choice) {
            case 1:
                clearConsole();
                printf_s("Введите логин:");
                if (inputLogin)
                    free(inputLogin);
                inputLogin = getPressedCharacters(LENGTHMAXLOGIN);
                break;
            case 2:
                clearConsole();
                printf_s("Введите пароль:");
                if (inputPassword)
                    free(inputPassword);
                inputPassword = getPressedCharacters(LENGTHMAXPASSWORD);
                break;
            case 3: {
                int lengthLogin = 0, lengthPassword = 0;
                if (inputLogin)
                    lengthLogin = (int)strlen(inputLogin);
                if (inputPassword)
                    lengthPassword = (int)strlen(inputPassword);
                if (!lengthLogin && !lengthPassword)
                    printf_s("\nВведите логин и пароль!\n");
                else if (!lengthLogin)
                    printf_s("\nВведите логин!\n");
                else if (!lengthPassword)
                    printf_s("\nВведите пароль!\n");
                else {
                    struct User* arrayUsers = NULL;
                    int countUsers = 0;
                    if (inputFileUsers(&arrayUsers, &countUsers)) {
                        int indexFindedUser = -1;
                        for (int _user = 0; _user < countUsers; _user++)
                            if (!strcmp(arrayUsers[_user].login, inputLogin) &&
                                !strcmp(arrayUsers[_user].hash, toHash(inputPassword, arrayUsers[_user].solid)))
                                indexFindedUser = _user;
                        if (indexFindedUser != -1) {
                            if (arrayUsers[indexFindedUser].blocked)

```

```

        printf("\nПользователь заблокирован!\n");
    else {
        printf("\nУспешный вход!\n");
        pressAnyButton();
        if (arrayUsers[indexFindedUser].privilege)
            adminMenu(arrayUsers[indexFindedUser]);
        else
            userMenu(arrayUsers[indexFindedUser]);
        return;
    }
}
else {
    printf("\nНеверный логин или пароль!\n");
}
}
if (arrayUsers)
    free(arrayUsers);
}
pressAnyButton();
}
break;
case 4:
    programIsWork = 0;
    break;
}
}
if (inputLogin)
    free(inputLogin);
if (inputPassword)
    free(inputPassword);
}

```

– Регистрация пользователя: происходит регистрация пользователя

```

void registerUser() {
    int programIsWork = 1, admin = 0, blocked = 0;
    char* newLogin = NULL, * newPassword = NULL;
    while (programIsWork) {
        clearConsole();
        int choice = inputFirstNatural(6, REGISTERUSERMENU);
        switch (choice) {
            case 1:
                clearConsole();
                printf_s("Введите новый логин:");
                if (newLogin)
                    free(newLogin);
                newLogin = getPressedCharacters(LENGTHMAXLOGIN);
                break;
            case 2:
                clearConsole();
                printf_s("Введите новый пароль:");
                if (newPassword)
                    free(newPassword);
                newPassword = getPressedCharacters(LENGTHMAXPASSWORD);
                break;
            case 3:
                clearConsole();
                printf_s("Нажмите '1', чтобы дать администратора пользователю, иначе нажмите любую
другую клавишу.");
                if (getPressedCharacter() == '1')
                    admin = 1;
                else
                    admin = 0;
                break;
            case 4:

```

```

clearConsole();
printf_s("Нажмите '1', чтобы дать блокировку пользователю, иначе нажмите любую
другую клавишу.");
if (getPressedCharacter() == '1')
    blocked = 1;
else
    blocked = 0;
break;
case 5: {
    int lengthNewLogin = 0, lengthNewPassword = 0;
    if (newLogin)
        lengthNewLogin = (int)strlen(newLogin);
    if (newPassword)
        lengthNewPassword = (int)strlen(newPassword);
    if (!lengthNewLogin && !lengthNewPassword)
        printf_s("\nВведите логин и пароль!\n");
    else if (!lengthNewLogin)
        printf_s("\nВведите логин!\n");
    else if (!lengthNewPassword)
        printf_s("\nВведите пароль!\n");
    else if (lengthNewLogin < LENGTHMINLOGIN)
        printf_s("\nЛогин слишком короткий!\n");
    else if (lengthNewPassword < LENGTHMINPASSWORD)
        printf_s("\nПароль слишком короткий!\n");
    else {
        struct User* arrayUsers = NULL;
        int countUsers = 0, loginFree = 1;
        if (inputFileUsers(&arrayUsers, &countUsers)) {
            for (int _user = 0; _user < countUsers; _user++)
                if (!strcmp(arrayUsers[_user].login, newLogin)) {
                    printf_s("\nЛогин уже занят!\n");
                    loginFree = 0;
                    break;
                }
        }
        if (loginFree) {
            struct User newUser;
            newUser.blocked = blocked;
            newUser.privilege = admin;
            newUser.login = newLogin;
            newUser.solid = generateSolid();
            newUser.hash = toHash(newPassword, newUser.solid);
            if (addUserFileUsers(&newUser)) {
                printf_s("\nПользователь успешно зарегистрирован!\n");
                programIsWork = 0;
            }
            free(newUser.solid);
            free(newUser.hash);
            newUser.login = NULL;
        }
        if (arrayUsers)
            free(arrayUsers);
    }
    pressAnyButton();
}
break;
case 6:
    programIsWork = 0;
    break;
}
if (newLogin)
    free(newLogin);
if (newPassword)

```

```
        free(newPassword);  
    return;  
}
```

Каждая из этих функций направлена на облегчение управления реализацией готовой продукции и улучшение пользовательского опыта при работе с программой. Они будут разработаны с учетом принципов удобства использования и эффективности, чтобы удовлетворить потребности пользователей и оптимизировать процесс управления реализацией продукции.

2.5 Разработка схем алгоритмов работы программы

В процессе разработки алгоритмов важно учитывать как требования к программе, так и потребности пользователей. Это поможет создать приложение, которое будет соответствовать ожиданиям пользователей и обеспечивать им максимальное удовлетворение от использования. Алгоритмы должны быть эффективными и надежными, чтобы гарантировать стабильную работу программы в различных сценариях использования.

2.5.1 Алгоритм функции startProgram

На рисунке 2.1 изображена схема алгоритма работы программы на которой изображены основные этапы функционирования. Работы программы начинается с выбора: авторизация или выйти из системы. Роль пользователя определяется на основе данных из файла. Затем, если пользователь совершил успешный вход в систему, то отображается соответствующее меню в зависимости от роли пользователя. Иначе, если он выбрал выход из системы, то программа завершится.

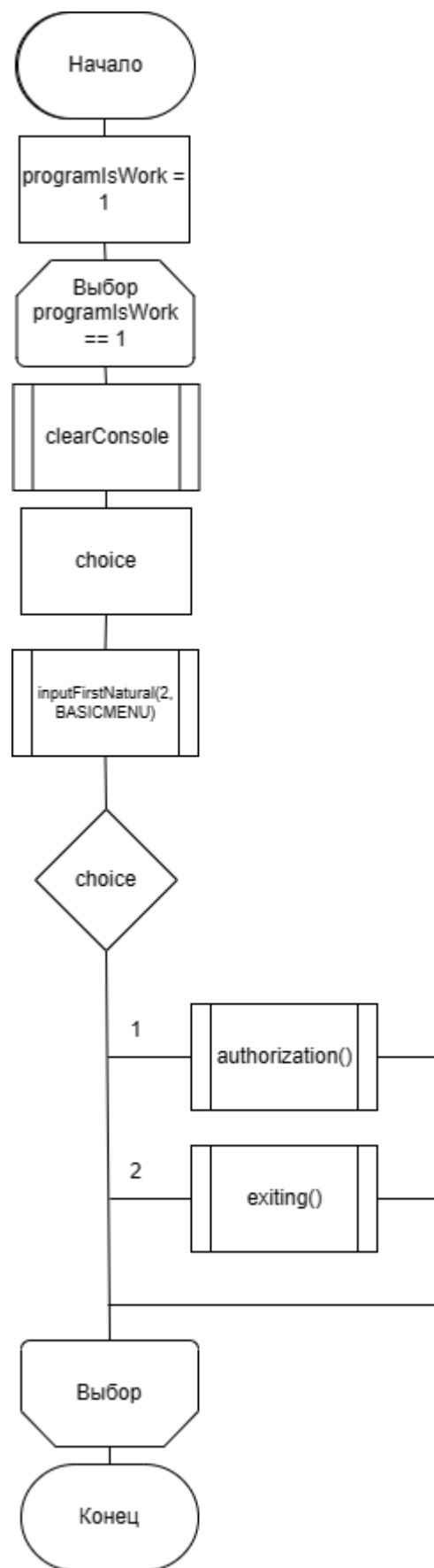


Рисунок 2.1 – Схема алгоритма работы системы

2.5.2 Алгоритм функции меню администратора

На рисунке 2.2 представлена схема алгоритма работы функции меню пользования у администратора. Начинается все с отображения меню на экране, далее у администратора есть выбор для дальнейшего действия. После того как выбор был сделан, выполняется соответствующая функция.

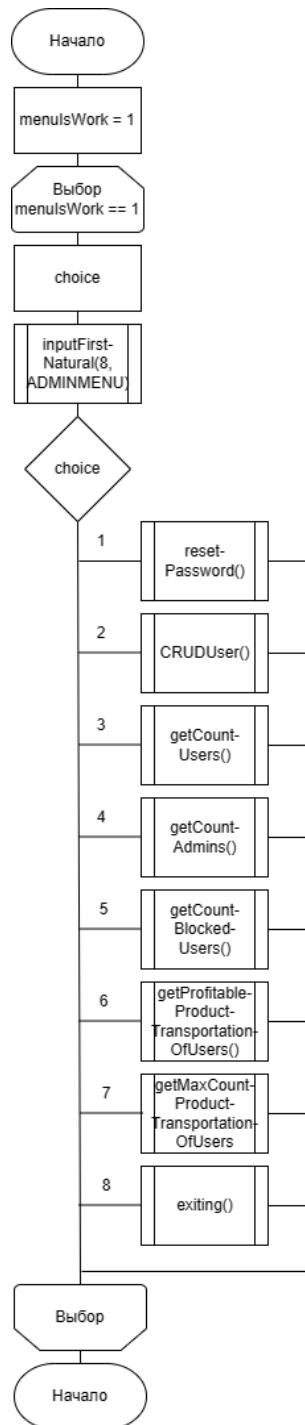


Рисунок 2.2 – Схема алгоритма работы меню администратора

2.5.3 Алгоритм функции изменения профиля у пользователя

На рисунке 2.3 продемонстрирована схема алгоритма работы функции изменения пароля у пользователя. Все начинается с ввода нового пароля, если пароль корректен, то дальше через начинается процесс нахождения данных пользователя с текущим логином. После новый пароль сохраняется.

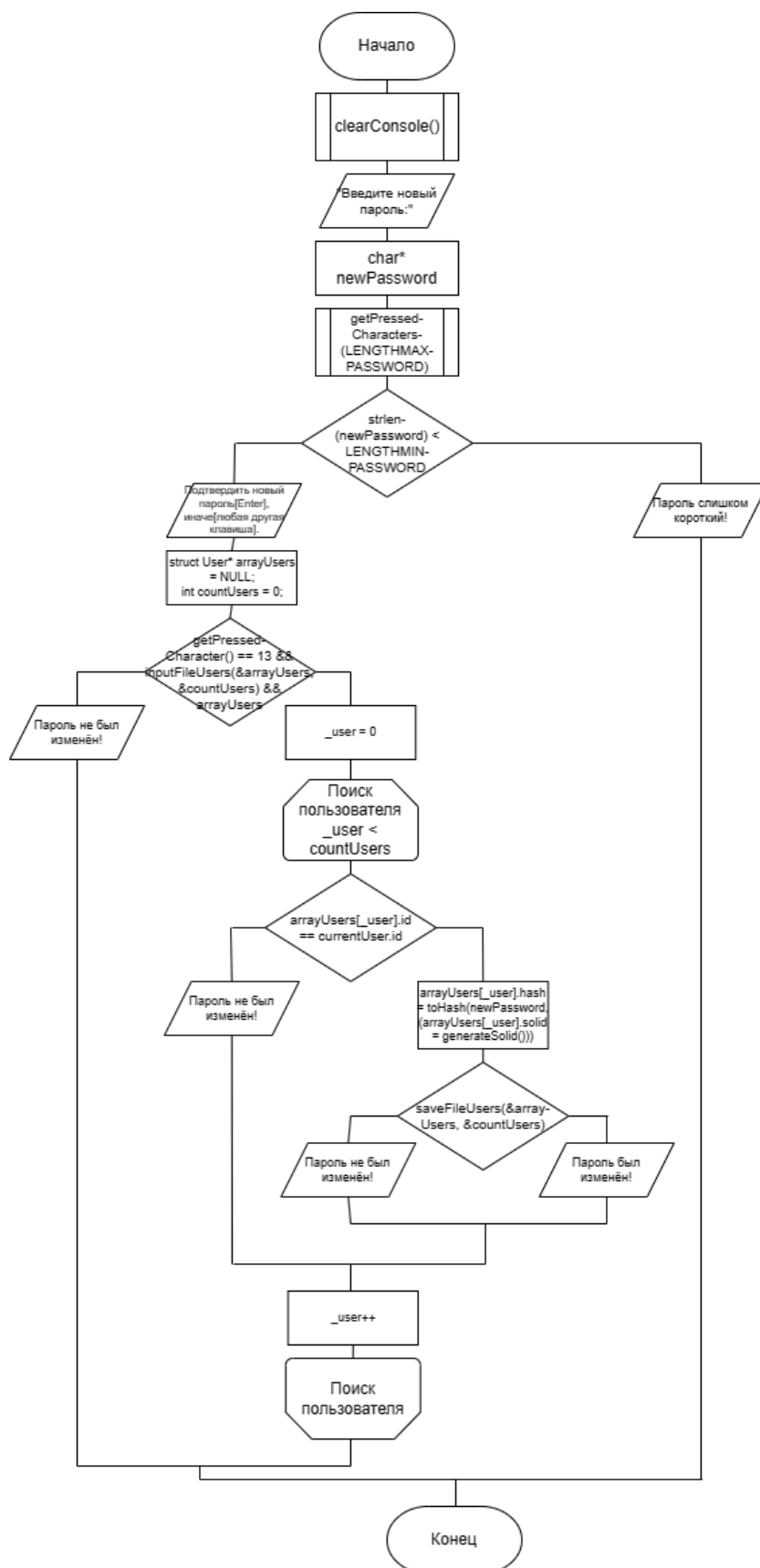


Рисунок 2.3 – Схема алгоритма работы функции изменения пароля у пользователя

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

В процессе разработки программного обеспечения важно уделить особое внимание проверке и тестированию его функциональности. Это включает в себя не только проверку правильной работы основных функций, но и обработку возможных ошибочных ситуаций, которые могут возникнуть при взаимодействии пользователя с программой. Ниже представлены основные аспекты проверки и тестирования, которые проводятся в ходе разработки программного обеспечения:

- некорректный ввод: проверка на наличие символов, не соответствующих ожидаемому формату ввода;
- неверный логин или пароль: проверка правильности введенных учетных данных пользователя;
- проверка существования файлов: при обращении к файлам для чтения или записи проверяется их наличие.
- проверка на корректность ввода чисел и предотвращение ввода символов, не являющихся цифрами;

В программе осуществляется проверка на правильность ввода идентификатора пользователя. Если введенный идентификатор пользователя не существует, программа выводит соответствующее сообщение об ошибке, как показано на рисунке 3.1



Рисунок 3.1 – Ввод несуществующего идентификатора пользователя

В случае авторизации, если не верно введены данные программа также выдает сообщение об ошибке, которое отображено на рисунке 3.2. Эти механизмы обеспечивают защиту программы от несанкционированного доступа и улучшают общий пользовательский опыт.




Рисунок 3.2 – Ввод неправильного логина или пароля пользователя

В программном решении также предусмотрена проверка на случай, если файл, необходимый для работы приложения, окажется поврежденным или удаленным. В случае обнаружения подобной ситуации при попытке открыть файл, программа выдаст соответствующее сообщение об ошибке, на рисунке 3.3, которое позволит пользователю осознать произошедшее. Эта функция обеспечивает повышенную надежность работы приложения и предотвращает возможные сбои из-за отсутствия необходимых файлов данных.

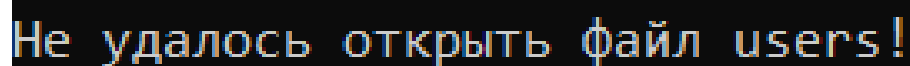
A black rectangular box with the text "Не удалось открыть файл users!" in a yellow, monospaced font with a blue outline.

Рисунок 3.3 – Файл не обнаружен или поврежден

В ситуации когда администратор производит регистрацию нового пользователя и пишет в поле логина уже существующий логин, то тоже присутствует проверка, на рисунке 3.4 можно увидеть соответствующей ситуации ответ.


A black rectangular box with the text "Логин уже занят!" in a yellow, monospaced font with a blue outline.

Рисунок 3.4 – Попытка занять чужой логин

Эти механизмы проверки помогают гарантировать корректность ввода данных пользователем, предотвращая возможные ошибки и улучшая удобство использования программы. Они обеспечивают защиту программы от некорректного ввода данных, что способствует повышению ее надежности и стабильности. Кроме того, эти проверки существенно улучшают пользовательский опыт, делая взаимодействие с программой более интуитивно понятным и безопасным.

4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ И СКВОЗНОЙ ТЕСТОВЫЙ ПРИМЕР

В этом разделе содержится подробное описание работы программы, включая ключевые функциональные возможности, особенности пользовательского интерфейса и основные шаги взаимодействия с программой.

4.1 Авторизация

Чтобы обеспечить безопасный доступ к данным и функциям программы, реализован механизм авторизации, позволяющий пользователям входить в систему с использованием своих учетных данных. Процесс начинается с того, что пользователь вводит свой логин, который проверяется на соответствие зарегистрированным учетным записям в базе данных. Если пользователь с введенным логином найден, система переходит к следующему этапу аутентификации. Отображение окна ввода логина и пароля отображено на картинке 4.1.

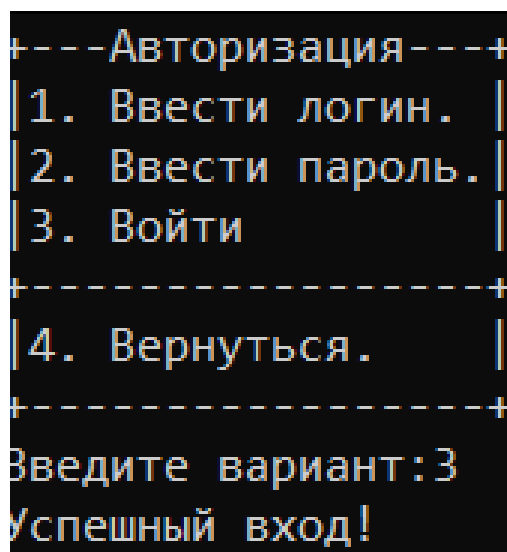


Рисунок 4.1 – Отображение окна авторизации

После успешного ввода логина пользователю предлагается ввести пароль. Введенный пароль шифруется и затем сравнивается с зашифрованным паролем, хранящимся в файле для данного пользователя. Если пароль или

логин введен неверно, система отображает сообщение о неправильном вводе данных, как отображено на рисунке 4.2.

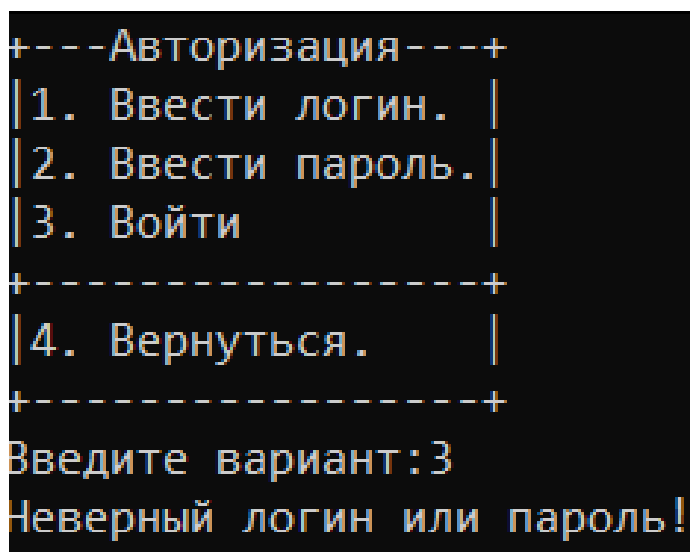


Рисунок 4.2 – Отображение окна, неправильности введенных данных

Если авторизация прошла успешно, пользователь попадает в окно с функционалом, соответствующим его роли. Процесс авторизации включает взаимодействие пользователя с интерфейсом программы, обработку введенных данных, проверку пользователя в базе данных, а также анализ правильности введенного пароля и проверку на наличие блокировки учетной записи.

4.2 Модуль администратора

Когда процедура аутентификации с использованием учетных данных администратора проходит успешно, пользователь перенаправляется в административный интерфейс, который отображается в отдельном окне программы, как показано на рисунке 4.3.

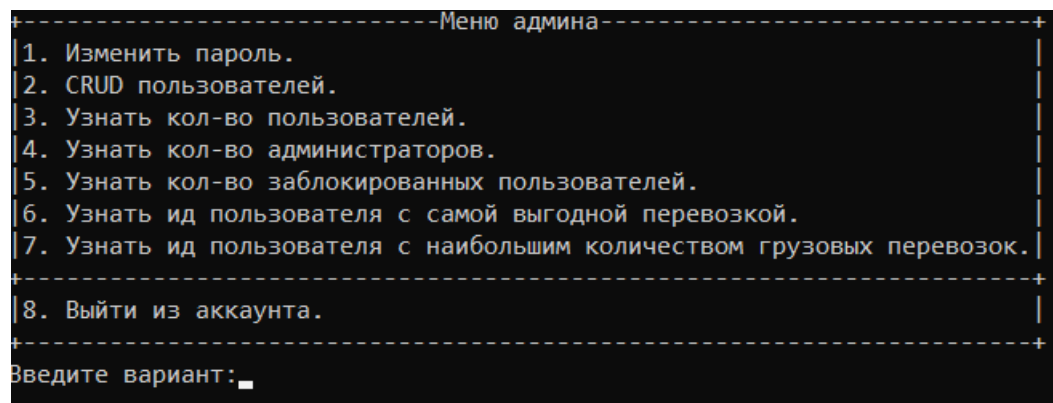


Рисунок 4.3 – Отображение окна администратора

При выборе опции "CRUD пользователей" администратор перенаправляется в окно, где происходит работа с данными пользователей. Интерфейс функций для работы с данными пользователей представлен на рисунке 4.4.

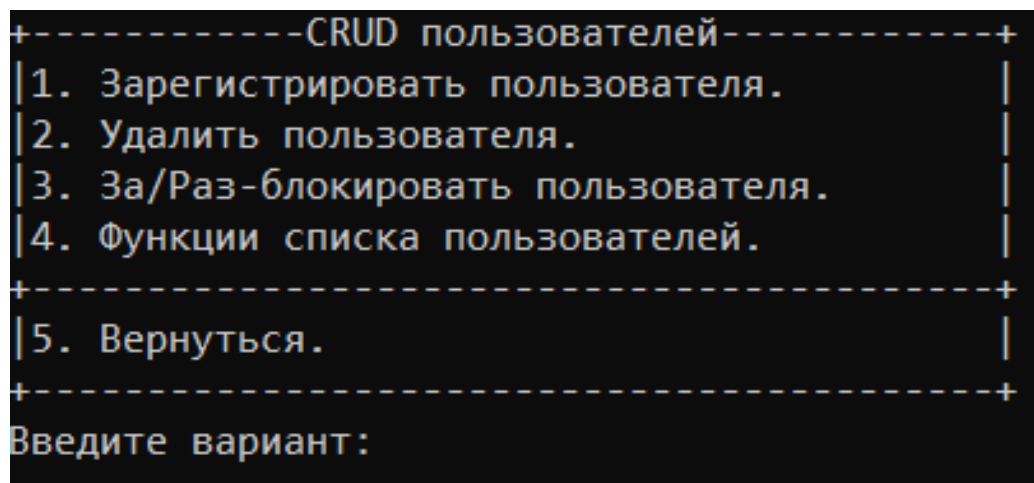


Рисунок 4.4 – Отображение окна просмотра всех товаров

Далее при выборе опции "Функции списка пользователей" происходит перенаправление в следующее окно, в котором можно подключить фильтр с сортировкой и получить список пользователей, данное окно представлено на рисунке 4.5.

```

+-Функции списка пользователей-+
|1. Управление фильтром.      |
|2. Управление сортировкой.   |
|3. Получить список.         |
+-----+
|4. Вернуться.                |
+-----+

```

Рисунок 4.5 – Отображение окна функций списка пользователей

При выборе опции "Получить список" выскочит таблица с данными всех пользователей, данная таблица представлена на рисунке 4.6.

№	ид	логин	хэш	соль	Администратор	Блокировка
1	0	admin	Yt15	Hf76fd	Да	Нет
2	1	user1	VU44SUL9	GU66W	Нет	Нет
3	2	user2	fiX6r7h1	qiF8v3	Нет	Нет
4	3	user3	yE6cfS5n	jE8k	Нет	Нет
5	4	timofei	SgDNdVHrZL mWDg	ZyR	Нет	Нет
6	5	yarik	5369331071	139	Нет	Да
7	6	maksim	R0GX8IR0GX 8I	F0W	Нет	Нет
8	7	artyom	iQ54wL67bX 02	iZ60	Нет	Нет
9	8	eswvgesrbe r	d5Y118e3X7 77x5Z532x5 Y498e3A230 v8P1	b1T894	Нет	Нет
10	9	23487g87w4 8gohow84hg owh43ghwo3	01653w4325 5r52711j59 759150540r	86313n	Нет	Нет

Рисунок 4.6 – Отображение таблицы данных пользователей

4.3 Модуль пользователя

После успешной аутентификации с использованием учетных данных пользователя, система автоматически перенаправляет его в пользовательский интерфейс, который отображается в отдельном окне программы, изображенном на рисунке 4.7.

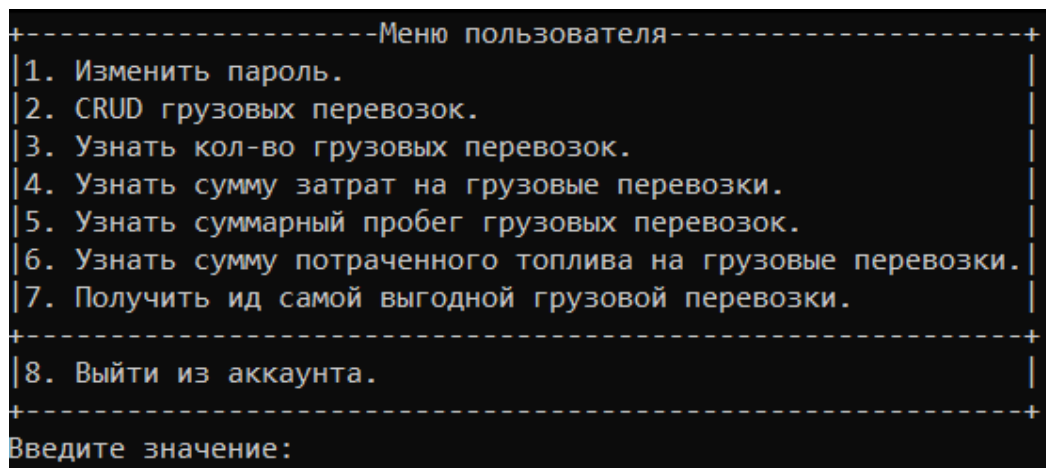


Рисунок 4.7 – Отображение окна пользователя

В данном окне пользователь имеет доступ к обширной информации о своем профиле, включая уникальный идентификатор, логин, номер телефона, регион и адрес. Кроме того, представлен широкий спектр функционала, доступного пользователю:

- Изменить пароль: позволяет изменить пароль текущего аккаунта;
- Узнать количество грузовых перевозок.
- Узнать сумму затрат на грузовые перевозки.
- Узнать суммарный пробег грузовых перевозок.
- Узнать сумму потраченного топлива на грузовые перевозки.
- Получить идентификатор самой выгодной грузовой перевозки.
- CRUD грузовых перевозок: позволяет работу с грузовыми перевозками.
- Выйти из аккаунта: возвращает пользователя в окно авторизации.

Это окно обеспечивает удобный и интуитивно понятный интерфейс для взаимодействия пользователя с программой, позволяя ему легко управлять своим профилем и осуществлять различные операции в системе.

При выборе опции "CRUD Грузовых перевозок" пользователь будет перенаправлен в окно, где будут доступны ему функции для работы с грузовыми перевозками. Данное окно представлено на рисунке 4.8.

```
+-----CRUD грузовых перевозок-----+
| 1. Создать грузовую перевозку.      |
| 2. Удалить грузовую перевозку.      |
| 3. Получить список грузовых перевозок. |
+-----+
| 4. Вернуться.                      |
+-----+
Введите значение: _
```

Рисунок 4.8 – Отображение окна CRUD грузовых перевозок

При выборе опции "Создать грузовую перевозку" пользователь будет перенаправлен в окно, где отображается список для заполнения полной формы грузовой перевозки. Данное окно представлено на рисунке 4.9.

```
+-----Новая грузовая перевозка-----+
| 1. Ввести страну отправки.          |
| 2. Ввести страну доставки.          |
| 3. Ввести город отправки.           |
| 4. Ввести город доставки.           |
| 5. Ввести адрес отправки.           |
| 6. Ввести адрес доставки.           |
| 7. Ввести дату отправки.            |
| 8. Ввести дату доставки.            |
| 9. Ввести время отправки.           |
| 10. Ввести время доставки.          |
| 11. Ввести тип ТС.                  |
| 12. Ввести кол-во затраченного топлива. |
| 13. Ввести пройденное расстояние.    |
| 14. Ввести кол-во остановок.         |
| 15. Ввести общую стоимость перевозки. |
| 16. Ввести кол-во различных грузов.  |
+-----+
| 17. Отменить новую перевозку.        |
| 18. Подтвердить.                    |
+-----+
Введите значение: _
```

Рисунок 4.9 – Отображение окна создания грузовой перевозки

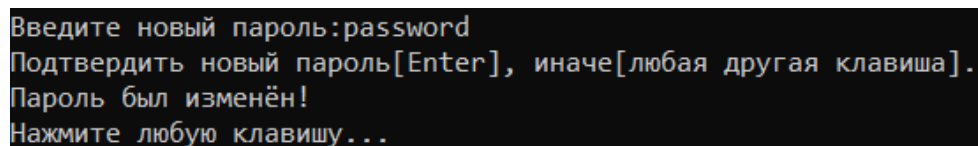
Пользователь также имеет возможность изменять пароль своего аккаунта. Для этого он должен выбрать опцию "Изменить пароль" из меню. После выбора этой опции пользователь перенаправляется в окно, где у него запрашивают новый пароль, как показано на рисунке 4.10.



Введите новый пароль:password

Рисунок 4.10 – Отображение окна изменения пароля

После успешного ввода пароля отобразится выбор на подтверждение нового пароля. Окно успешного создания нового пароля представлено на рисунке 4.11.



Введите новый пароль:password
Подтвердить новый пароль[Enter], иначе[любая другая клавиша].
Пароль был изменён!
Нажмите любую клавишу...

Рисунок 4.11 – Отображение окна изменения пароля

В целом, руководство пользователя создано таким образом, чтобы обеспечить простоту и эффективность использования программы для всех категорий пользователей.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была достигнута цель оптимизации процесса учета грузовых перевозок посредством разработки консольного приложения. Изучен процесс учета грузовых перевозок, что привело к решению различных проблем. В ходе работы была разработана программа учета грузовых перевозок. Учтены все требования, поставленные для конкретной специфики предметной области: возможность эффективной регистрации перевозок. Разработанная система способна сохранять всю необходимую информацию на неограниченный срок, обеспечивая повторное использование, стабильность, безопасность данных, и, кроме того, не нарушает авторские права и патенты.

В первом разделе проведен тщательный анализ учета грузовых перевозок. В этом разделе рассмотрены ключевые аспекты, такие как особенности грузовых перевозок, анализ бухгалтерской отчетности, а также создание информационной модели, отражающей специфику этой области. Было добавлено применение методологии IDEF0, чтобы глубже проанализировать процесс проведения учета грузовых перевозок. Разработана математическая модель учета грузовых перевозок, учтено влияние конкурентов на отрасль. Кроме того, были созданы диаграммы UML для визуализации структуры системы, блок-схемы алгоритмов работы программы, сформулированы технические задачи и предоставлено руководство пользователя.

Во втором разделе описано проектирование и разработка автоматизированной системы управления учетом грузовой перевозки. Этот этап дал возможность более подробно понять и определить оптимальные решения в сфере учета перевозок. Раздел включает цели проектирования, модульную структуру программы, пользовательские функции, а также блок-схемы алгоритмов, иллюстрирующие логику системы.

Третий раздел посвящен разработке программного обеспечения, включающей архитектуру веб-приложения, используемые технологии и фреймворки, обеспечивающие эффективное функционирование программы. Здесь рассматриваются выбор технологических стеков, структура кода и методы обеспечения безопасности системы. Особое внимание уделяется надежности, масштабируемости и производительности, чтобы система могла справляться с высокими нагрузками и обеспечивать безопасную работу.

В четвертом разделе дается характеристика программного продукта после его разработки и развертывания. Также проведен анализ целесообразности инвестиций в разработку, с учетом предполагаемой

рентабельности проекта. В результате оценки эффективности разработки продемонстрирована высокая рентабельность, подтверждающая обоснованность вложений. Была успешно достигнута основная цель настоящей работы – улучшение процесса грузовых перевозок и оптимизация операций, связанных с этим процессом. Решение данной задачи включало в себя тщательный анализ особенностей работы грузовых перевозок, разработку программного обеспечения для учета перевозок, а также создание эффективной системы управления данными и операциями.

В результате проведенного исследования и анализа процесса учета грузовых перевозок были выявлены ключевые аспекты, влияющие на эффективность бизнеса. Программное обеспечение разработано с учетом этих особенностей, обеспечивая возможность надежного учета перевозок, анализа инвентаря и управления бизнес-процессами.

Создание программной поддержки привело к существенному снижению трудозатрат как для пользователей, использующих систему для учета грузовых перевозок, так и для администраторов, занимающегося оперативными процессами. Уникальные функциональности программы, такие как точный расчет и долгосрочное сохранение всей необходимой информации, сделали не только эффективным инструментом учета, но и устойчивым решением для долгосрочного использования.

Таким образом, данная работа успешно достигла поставленных целей, предложив полноценное программное решение для учета грузовых перевозок, способствуя улучшению операционной эффективности и снижению трудозатрат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Иванов, А. А., Петров, Б. Б. (2020). "Управление грузовыми перевозками." [Электронный ресурс]. – Режим доступа: <http://example.com/cargo-sales-management> (Дата обращения: 30.04.2024).

[2] "Методология IDEF0: основы и примеры применения." [Электронный ресурс]. Режим доступа: <http://example.com/idef0-methodology> (Дата обращения: 01.05.2024).

[3] "Методическое пособие по тестированию программного обеспечения." [Электронный ресурс]. Режим доступа: <http://example.com/testing-guide> (Дата обращения 30.04.2024).

[4] "Алгоритмы и структуры данных в приложениях для бизнеса." [Электронный ресурс]. Режим доступа: <http://example.com/business-algorithms> (Дата обращения: 01.05.2024).

[5] Сидорова, В. В. (2018). "Автоматизированные системы управления продажами." Минск: Издательство "Учебник", 300 с

[6] Кузнецов, Г. Г., Смирнов, Д. Д. (2019). "Проектирование UML-диаграмм." [Электронный ресурс]. Режим доступа: <http://example.com/uml-diagrams-design> (Дата обращения 29.04.2024).

[7] Антиплагиат [Электронный ресурс]. Режим доступа: <https://antiplagiat.ru/>.

ПРИЛОЖЕНИЕ А

(обязательное)

Отчет о проверке на заимствования в системе «Антиплагиат»

Рисунок А.1 – Отчет о проверке на заимствования в системе
«Антиплагиат»

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг кода алгоритмов, реализующих основную бизнес-логику приложения

```
void createTransportationMenu(int currentId) {
    int menuIsWork = 1;
    struct ProductTransportation productTransportation;
    productTransportation.countryOrigin = NULL,
    productTransportation.countryDelivery = NULL,
    productTransportation.townOrigin = NULL,
    productTransportation.townDelivery = NULL,
    productTransportation.dateOrigin = NULL,
    productTransportation.dateDelivery = NULL,
    productTransportation.timeOrigin = NULL,
    productTransportation.timeDelivery = NULL,
    productTransportation.addressOrigin = NULL,
    productTransportation.addressDelivery = NULL,
    productTransportation.typeTransport = NULL;
    while (menuIsWork) {
        clearConsole();
        int choice = inputFirstNatural(18, CREATEPRODUCTTRANSPORTATIONMENU);
        switch (choice) {
            case 1: {
                clearConsole();
                printf("Введите страну отправки:");
                char* countryOrigin = inputWord(MAXLENGTHCREATEPRODUCT);
                if (countryOrigin && isCapitalizedWord(countryOrigin)) {
                    if (productTransportation.countryOrigin)
                        free(productTransportation.countryOrigin);
                    productTransportation.countryOrigin = countryOrigin;
                }
                else {
                    printf("\nВведена некорректная страна!\n");
                    pressAnyButton();
                    if (productTransportation.countryOrigin)
                        free(productTransportation.countryOrigin);
                }
            }
            break;
            case 2: {
                clearConsole();
                printf("Введите страну отправки:");
                char* countryDelivery = inputWord(MAXLENGTHCREATEPRODUCT);
                if (countryDelivery && isCapitalizedWord(countryDelivery)) {
                    if (productTransportation.countryDelivery)
                        free(productTransportation.countryDelivery);
                    productTransportation.countryDelivery = countryDelivery;
                }
                else {
                    printf("\nВведена некорректная страна!\n");
                }
            }
        }
    }
}
```

```

        pressAnyButton();
        if (countryDelivery)
            free(countryDelivery);
    }
}
break;
case 3:
{
    clearConsole();
    printf("Введите город отправки:");
    char* townOrigin = inputWord(MAXLENGTHCREATEPRODUCT);
    if (townOrigin && isCapitalizedWord(townOrigin)) {
        if (productTransportation.townOrigin)
            free(productTransportation.townOrigin);
        productTransportation.townOrigin = townOrigin;
    }
    else {
        printf("\nВведен некорретный город!\n");
        pressAnyButton();
        if (townOrigin)
            free(townOrigin);
    }
}
break;
case 4:
{
    clearConsole();
    printf("Введите город доставки:");
    char* townDelivery = inputWord(MAXLENGTHCREATEPRODUCT);
    if (townDelivery && isCapitalizedWord(townDelivery)) {
        if (productTransportation.townDelivery)
            free(productTransportation.townDelivery);
        productTransportation.townDelivery = townDelivery;
    }
    else {
        printf("\nВведен некорретный город!\n");
        pressAnyButton();
        if (townDelivery)
            free(townDelivery);
    }
}
break;
case 5:
    clearConsole();
    printf("Введите адрес отправки:");
    if (productTransportation.addressOrigin)
        free(productTransportation.addressOrigin);
    productTransportation.addressOrigin
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
break;
case 6:
    clearConsole();
    printf("Введите адрес доставки:");

```



```

        if (productTransportation.addressDelivery)
            free(productTransportation.addressDelivery);
        productTransportation.addressDelivery
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        break;
    case 7:
        clearConsole();
        printf("Введите дату отправки:");
        char* dateOrigin = getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isDate(dateOrigin)) {
            if (productTransportation.dateOrigin)
                free(productTransportation.dateOrigin);
            productTransportation.dateOrigin = dateOrigin;
        }
        else if (dateOrigin) {
            printf("\nВведена некорретная дата!\n");
            pressAnyButton();
            free(dateOrigin);
        }
        break;
    case 8:
        clearConsole();
        printf("Введите дату доставки:");
        char* dateDelivery
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isDate(dateDelivery)) {
            if (productTransportation.dateDelivery)
                free(productTransportation.dateDelivery);
            productTransportation.dateDelivery = dateDelivery;
        }
        else if (dateDelivery) {
            printf("\nВведена некорретная дата!\n");
            pressAnyButton();
            free(dateDelivery);
        }
        break;
    case 9:
        clearConsole();
        printf("Введите время отправки:");
        char* timeOrigin = getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isTime(timeOrigin)) {
            if (productTransportation.timeOrigin)
                free(productTransportation.timeOrigin);
            productTransportation.timeOrigin = timeOrigin;
        }
        else if (timeOrigin) {
            printf("\nВведено некорретное время!\n");
            pressAnyButton();
            free(timeOrigin);
        }
        break;
    case 10:
        clearConsole();

```

```

        printf("Введите время доставки:");
        char* timeDelivery =
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isTime(timeDelivery)) {
            if (productTransportation.timeDelivery)
                free(productTransportation.timeDelivery);
            productTransportation.timeDelivery = timeDelivery;
        }
        else if (timeDelivery) {
            printf("\nВведено некорретное время!\n");
            pressAnyButton();
            free(timeDelivery);
        }
        break;
    case 11:
        clearConsole();
        printf("Введите тип ТС:");
        if (productTransportation.typeTransport)
            free(productTransportation.typeTransport);
        productTransportation.typeTransport =
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        break;
    case 12:
        clearConsole();
        printf("Введите кол-во затраченного топлива:");
        char* fuelConsumed =
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isFloat(fuelConsumed)) {
            productTransportation.fuelConsumed =
convertPtrCharToFloat(fuelConsumed);
        }
        else {
            printf("\nВведена некорретная запись!\n");
            pressAnyButton();
        }
        if (fuelConsumed)
            free(fuelConsumed);
        break;
    case 13:
        clearConsole();
        printf("Введите пройденное расстояние:");
        char* distanceTraveled =
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isFloat(distanceTraveled)) {
            productTransportation.distanceTraveled =
convertPtrCharToFloat(distanceTraveled);
        }
        else {
            printf("\nВведена некорретная запись!\n");
            pressAnyButton();
        }
        if (distanceTraveled)
            free(distanceTraveled);

```

```

        break;
    case 14:
        clearConsole();
        printf("Введите кол-во остановок:");
        productTransportation.numberStops = inputInteger(POSITIVE);
        break;
    case 15:
        clearConsole();
        printf("Введите общие затраты на перевозку:");
        char* priceGeneral =
getPressedCharacters(MAXLENGTHCREATEPRODUCT);
        if (isFloat(priceGeneral)) {
            productTransportation.priceGeneral =
convertPtrCharToFloat(priceGeneral);
        }
        else {
            printf("\nВведена некорретная запись!\n");
            pressAnyButton();
        }
        if (priceGeneral)
            free(priceGeneral);
        break;
    case 16:
        clearConsole();
        printf("Введите кол-во рализчных грузов:");
        productTransportation.numberProducts = inputInteger(POSITIVE);
        if (!productTransportation.numberProducts) {
            printf("\nВведена некорретная запись!\n");
            pressAnyButton();
        }
        else {
            productTransportation.products = (struct Product*)malloc(sizeof(struct
Product) * productTransportation.numberProducts);
            int currentCountProducts = 0;
            if (productTransportation.products) {
                int creatingProductIs = 1;
                for (int _product = 0; _product <
productTransportation.numberProducts; _product++) {
                    struct Product* currentProductPtr =
&productTransportation.products[_product];
                    currentProductPtr->codeIdentification = 0;
                    currentProductPtr->number = 0;
                    currentProductPtr->weight = 0;
                    currentProductPtr->name = NULL;
                }
                while (creatingProductIs && currentCountProducts !=
productTransportation.numberProducts) {
                    int choice = inputFirstNatural(8,
CREATEPRODUCTSMENU);
                    struct Product* currentProductPtr =
currentProductPtr->id = currentCountProducts + 1;
                    switch (choice) {

```

	case 1:	clearConsole();	
		printf("Введите название груза:");	
getPressedCharacters(MAXLENGTHCREATEPRODUCT);		currentProductPtr->name	=
		break;	
	case 2:	clearConsole();	
номер груза:");		printf("Введите идентификационный	
inputInteger(POSITIVE);		currentProductPtr->codeIdentification	=
		break;	
	case 3:	clearConsole();	
		printf("Введите вес груза:");	
getPressedCharacters(MAXLENGTHCREATEPRODUCT);		char* weightProduct	=
		if (isFloat(weightProduct)) {	
convertPtrCharToFloat(weightProduct);		currentProductPtr->weight	=
		free(weightProduct);	
		}	
запись!\n");		else if (weightProduct) {	
		printf("\nВведена некорретная	
		pressAnyButton();	
		free(weightProduct);	
		}	
		break;	
	case 4:	clearConsole();	
		printf("Введите кол-во грузов:");	
inputInteger(POSITIVE);		currentProductPtr->number	=
		break;	
	case 5:	creatingProductIs = 0;	
currentCountProducts;		productTransportation.numberProducts	=
		break;	
	case 6:	creatingProductIs = 0;	
		productTransportation.numberProducts = 0;	
		free(productTransportation.products);	
		break;	
	case 7:	if (currentProductPtr->name == NULL)	
>codeIdentification == 0)		printf("\nВведите название!\n");	
индефикацияционный номер!\n");		else if (currentProductPtr-	
		printf("\nВведите	

```

else if (currentProductPtr->weight == 0)
    printf("\nВведите вес груза!\n");
else if (!currentProductPtr->number)
    printf("\nВведите          кол-во
грузов!\n");

else {
    printf("\nГруз          успешно
создан!(#%d груз - #%d)\n", ++currentCountProducts, productTransportation.numberProducts);
}
pressAnyButton();
break;
}
}
else {
    printf("\nНе          удалось          выделить          память          в
CreateProductTransportationMenu!\n");
    pressAnyButton();
}
}
break;
case 17:
    menuIsWork = 0;
    break;
case 18:
    if (!productTransportation.countryOrigin)
        printf("\nВведите страну отправки!\n");
    else if (!productTransportation.countryDelivery)
        printf("\nВведите страну доставки!\n");
    else if (!productTransportation.townOrigin)
        printf("\nВведите город отправки!\n");
    else if (!productTransportation.townDelivery)
        printf("\nВведите город доставки!\n");
    else if (!productTransportation.addressOrigin)
        printf("\nВведите адрес отправки!\n");
    else if (!productTransportation.addressDelivery)
        printf("\nВведите адрес доставки!\n");
    else if (!productTransportation.dateOrigin)
        printf("\nВведите дату отправки!\n");
    else if (!productTransportation.dateDelivery)
        printf("\nВведите дату доставки!\n");
    else if (!productTransportation.timeOrigin)
        printf("\nВведите время отправки!\n");
    else if (!productTransportation.timeDelivery)
        printf("\nВведите время доставки!\n");
    else if (productTransportation.distanceTraveled == 0)
        printf("\nВведите пройденное расстояние!\n");
    else if (productTransportation.fuelConsumed == 0)
        printf("\nВведите затраченное топливо!\n");
    else if (productTransportation.numberProducts == 0)
        printf("\nВведите количество продуктов!\n");
    else if (!productTransportation.typeTransport)
        printf("\nВведите тип транспорта!\n");

```

```

else if (productTransportation.priceGeneral == 0)
    printf("\nВведите стоимость перевозки!\n");
else {
    struct Transportations* transportations = NULL;
    int countTransportations = 0;
    if (inputFileTransportations(&transportations, &countTransportations))
    {
        int addedPT = 0;
        for (int _transportations = 0; _transportations <
countTransportations; _transportations++) {
            if (transportations[_transportations].id == currentId) {
                struct ProductTransportation*
newArrayProductTransportation = (struct ProductTransportation*)malloc(sizeof(struct ProductTransportation) *
(transportations[_transportations].number + 1));
                if (newArrayProductTransportation) {
                    for (int _PT = 0; _PT <
transportations[_transportations].number; _PT++) {
                        newArrayProductTransportation[_PT]
                        =
transportations[_transportations].arrayProductTransportation[_PT];
                    }
                    int freeId = 0, idFinded = 0;
                    while (!idFinded) {
                        freeId++;
                        if
(!transportations[_transportations].number)
                            idFinded = 1;
                        for (int
_productTransportation = 0; _productTransportation <
transportations[_transportations].number;
_productTransportation++)
                            if
(transportations[_transportations].arrayProductTransportation[_productTransportation].id == freeId) {
                                break;
                            }
                        else if
(_productTransportation + 1 == transportations[_transportations].number)
                            idFinded
= 1;
                    }
                    productTransportation.id = freeId;
                    newArrayProductTransportation[(transportations[_transportations].number)++] = productTransportation;
                    addedPT = 1;
                    free(transportations[_transportations].arrayProductTransportation);
                    transportations[_transportations].arrayProductTransportation = newArrayProductTransportation;
                }
                else {
                    //problem
                }
                break;
            }
        }
    }
}

```

```

    }
    if (!addedPT) {
        struct Transportations* newTransportations = (struct
Transportations*)malloc(sizeof(struct Transportations) * (countTransportations + 1));
        if (newTransportations) {
            for (int _transportations = 0; _transportations
< countTransportations; _transportations++) {

                newTransportations[_transportations] = transportations[_transportations];
                }
                /*int freeId = 0, idFinded = 0;
                while (!idFinded) {
                    freeId++;
                    if (!countTransportations)
                        idFinded = 1;
                    for (int _transportations = 0;
_transportations < countTransportations; _transportations++)
                        if
(newTransportations[_transportations].id == freeId) {

                            break;
                        }
                        else if (_transportations + 1
== countTransportations)

                            idFinded = 1;
                }*/

                newTransportations[countTransportations].arrayProductTransportation = (struct
ProductTransportation*)malloc(sizeof(struct ProductTransportation));
                if
(newTransportations[countTransportations].arrayProductTransportation) {

                    newTransportations[countTransportations].id = currentId;

                    newTransportations[countTransportations].number = 1;

                    productTransportation.id = 1;

                    newTransportations[countTransportations++].arrayProductTransportation = &productTransportation;
                    free(transportations);
                    transportations =

newTransportations;

                }
                else {
                    printf("\nHe удалось выделить
память в createProductTransportation!\n");

                    break;
                }
            }
            else {
                printf("\nHe удалось выделить память в
createProductTransportation!\n");

                break;
            }
        }
    }
}

```

```

        &countTransportations)) {
            if
                (saveFileTransportations(&transportations,
printf("\nГрузовая перевозка успешно создана!\n");
menuIsWork = 0;
            }
        }
    }
    pressAnyButton();
    break;
}
}
}

```