Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УЧЕТА ДЕЯТЕЛЬНОСТИ КОМПАНИИ ПО МЕЖДУНАРОДНЫМ ПЕРЕВОЗКАМ.

БГУИР КП 1-40 05 01-12 021 ПЗ

Студент гр. 824402 Г. А.

Руководитель ассистент каф. Е. А.

Минск 2020

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc41542220)

[1 Обзор автомотизируемой предметной области, программных аналогов, методов и алгоритм решения поставленной задачи 7](#_Toc41542221)

[1.1 Обзор автоматизируемой предметной области 7](#_Toc41542223)

[1.2 Обзор программных аналогов 9](#_Toc41542224)

[1.2.1 Maxoptra 10](#_Toc41542225)

[1.2.2 NovaTrans 11](#_Toc41542226)

[1.3 Обзор методов и алгоритмов решения поставленной задачи 12](#_Toc41542227)

[2 Функциональное моделирование на основе стандарта IDEF0 14](#_Toc41542228)

[2.1 Функциональные модели 14](#_Toc41542230)

[3. Схема данных автоматизированной системы учета деятельности компании по международным перевозкам 18](#_Toc41542231)

[4 Разработка и описание диаграммы классов приложения 21](#_Toc41542232)

[5 Разработка и описание диаграммы вариантов использования приложения 32](#_Toc41542233)

[6 Схема алгоритма работы всей программы и алгоритма работы нескольких основных методов 33](#_Toc41542234)

[7 Описание запуска приложения, его использования и результаты работы программы 34](#_Toc41542235)

[Заключение 46](#_Toc41542236)

[Список использованных источников 47](#_Toc41542237)

[Приложение А (обязательное) Листинг кода (к пункту 4) 48](#_Toc41542238)

[Приложение Б (обязательное) Диаграмма классов приложение (к пункту 4) 61](#_Toc41542240)

[Приложение В (обязательное) Диаграмма вариантов использования (к пункту 5) 62](#_Toc41542241)

[Приложение Г](#_Toc41542244) [[(обязательное) Схема алгоритмов (к пункту 6) 63](#_Toc41542244)](#_Toc41542245)

# ВВЕДЕНИЕ

В современном мире существует множество бизнесов и ни один из них не обходится без автоматизированных систем, направленных на повышение эффективности работы и развития отраслей. Современные технологии также не обошли стороной такую сферу бизнеса, как международные перевозки. Так как на мировой арене очень востребованы услуги компаний по международным перевозкам, их существует большое количество. Такие компании предоставляют перевозку каких-либо грузов и продукции.

Внедрение информационных технологий в данную сферу позволяет повысить эффективность предоставляемых услуг, а именно повысить качество этой услуги, сократить время доставки, увеличить объем выполняемой работы, сократить затраты, повысить доход, рационально распределять средства и рабочее время, собирать статистику и анализировать ее, устранять уязвимые зоны деятельности и т.д.

Услуги по перевозке грузов требуют крайне скоординированных действий, ведь речь идёт о сроках, объёмах, качестве перевозки, что выливается в главный аспект – прибыль. Организация работы и определяет величину этого параметра и для организации успешных сделок необходимо иметь автоматизированные программные средства, помогающие рабочему персоналу доставлять грузы с наименьшим количеством проблем.

В большинстве своём, благодаря таким компаниям, люди во всём мире могут получать абсолютно любую продукцию, а некоторые компании предоставляют услуги по перевозке целых домов по суше, например, если эта постройка исторически важна, но на её месте необходима какая-либо инфраструктура. На сегодняшний день любой человек в любом уголке мира может купить и насладиться фруктами, которые физически не могут произрастать в его стране или в данном регионе.

В настоящее время, людям сложно было бы представить мир, в котором невозможны грузоперевозки, в том числе и на дальние расстояния, без этого технологическое развитие могло бы оказаться под вопросом. Большинство продукции в Беларусь привозятся из-за границы.

Курсовой проект несет образовательный смысл, так как в данный момент существует множество универсальных программных средств, направленных на международные перевозки. Но крупные компании производят программное обеспечение под свои специфические бизнес процессы и задачи.

Для достижения поставленной цели необходимо решить следующие задачи:

* осуществить анализ предметной области;
* составить функциональную модель взаимодействия компании грузоперевозчика и клиента;
* создать систему хранения данных;
* проанализировать логическую и физическую модель представления данных;
* разработать простой и удобный интерфейс приложения.

Написание курсового проекта будет выполняться с помощью современного компилятора с использованием языка программирования С++. Интерфейс программы будет выполнен на русском языке и будет представлять собой структурированную текстовую информацию в консольном окне.

Если подытожить, то мы познакомимся с основными бизнес процессами, разберемся в организации и структуре небольшой компании по международным перевозкам, получим опыт в построении архитектуры бизнес приложений и написании их.

# 1 ОБЗОР АВТОМАТИЗИРУЕМОЙ ПРЕДМЕТНОЙ ОБЛАСТИ,

# ПРОГРАММНЫХ АНАЛОГОВ, МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

## 1.1 Обзор автоматизируемой предметной области

Предметной областью данного курсового проекта является компания по международным грузоперевозкам, которая представляет собой взаимосвязанную систему хранения объектов транспортировки, транспорт компании, работников и клиентов, обработка данных, предоставления услуг.

Деятельность данной компании определяется:

* организация учета работников;
* организация учета клиентов;
* организация учета транспорта;
* обслуживание клиента.

Учет работников представляет собой регистрация по средству формы либо добавления администратором в систему хранения данных, так же администратор имеет право удалить работника из базы данных;

Учет транспорта заключается в хранении данных о каждой единице транспорта и управления этими данными.

Учет и обслуживание клиентов заключается в хранении данных о клиентах и о их объектах транспортировки, изменения данных, предоставлении о статусе их грузах.

Самые распространенные, доступные и универсальные - это перевозки грузов автомобильным транспортом. Мировая автомобильная промышленность выпускает автомобили самого разного назначения, которые могут перевезти практически любой груз, любого веса и габаритов, лишь бы транспортные развязки позволили это сделать.   
 Несомненным достоинством автомобиля является высокая скорость доставки груза, причем не требуется никаких промежуточных перевалочных баз. В процессе доставки груза имеется возможность постоянного контроля его состояния, груз доставляется прямо до дверей заказчика [1].

Как и все в этом мире, автомобильный вариант имеет свои достоинства и недостатки. Вот его положительные стороны:

* не требуется перегрузок во время доставки;
* присутствие водителя и, если нужно, охраны обеспечивает высокую степень сохранности груза;
* высокая мобильность автомобиля и быстрота доставки груза;
* отсутствие необходимости накапливания груза на промежуточном складе, т.к. у грузовиков большой разброс грузоподъемности, что позволяет доставлять грузы ритмично.

Причем только автомобиль дает полную автономность при доставке грузов. К несомненным недостаткам автомобильных грузоперевозок следует отнести:

* невыгодность использования автопоездов для перевозки больших объемов грузов;
* прямая зависимость качества перевозки от состояния дорог;
* неоправданно высокий тариф при перевозках на большие расстояния;
* невозможность перевозить сверхгабаритные грузы.

Таким образом, необходимо отметить тот факт, что на сегодняшний день функционирование любой страны не может обойтись без процесса транспортировки груза, так как экономика государства главным образом зависит от сотрудничества с другими странами.

Чтобы клиент смог совершить заказ, у него должен быть зарегистрирован личный кабинет, информация о котором хранится в базе данных. Каждый личный кабинет должен содержать следующую информацию о клиенте:

* возраст;
* ФИО клиента.

У каждого клиента есть свой уникальный логин и пароль, с помощью которого он может войти в систему и совершать необходимые действия.

При создании заказа, клиент должен предоставить соответствующую информацию:

* вес груза;
* габариты;
* дату отправки и доставки;
* место отправки и доставки.

В базе данных также должны хранится и данные о сотрудниках. Таким образом сотрудники должны предоставить следующие личные данные:

* ФИО сотрудника;
* возраст;

В транспортной компании сотрудник должен выполнять следующие задачи:

* подтверждать заказы, так как после его совершения, для решения некоторых вопросов сотрудник будет обязан связаться с заказчиком для уточнения деталей или изменения условий доставки, в силу того, что не всегда возможна доставка или перевозка в назначенные заказчиком сроки;
* отправляет заказ и информацию о нём в отдел грузоперевозки, дабы груз был подготовлен к отправке и последующей доставке клиенту;
* по прибытию груза отмечает в базе этот факт, иными словами подтверждает доставку и отправляет отчёт на обработку.

Клиенту должна быть предоставлена возможность выполнения следующих услуг:

* возможность совершить заказ;
* отследить готовность доставки;
* удалять заказ (только заявка, всё по согласованию с персоналом).

Все пункты, описанные выше ускоряют работу компании, ведь множество действий может быть совершено дистанционно и быстро, сокращая труд сотрудников и экономя время заказчику, что является основополагающим фактором в деле грузоперевозок.

## 1.2 Обзор программных аналогов

В данном пункте будут рассмотрены наиболее популярные автоматизированные информационные системы учёта грузоперевозок. К таким можно отнести: Maxoptra, NovaTrans и др. Каждая система должна выполнять следующие функции:

* создание заказа;
* регистрация новых клиентов;
* формирование отчётов;
* и другие возможности с учетом необходимых компании грузоперевозок услуг.

Как правило такие программы разделены на модули (АРМы – автоматизируемые рабочие места), которые в свою очередь включают некоторые функции. К основным АРМам можно отнести следующие:

* комплектование;
* архивация;
* администратор;
* личный кабинет (клиент).

Такое разделение значительно облегчает работу в приложении и помогает пользователю лучше ориентироваться в нем.

## 1.2.1 Maxoptra

Maxoptra – система планирования, оптимизации и контроля маршрутов движения транспорта и выездных сотрудников (см. рис. 1.1).

Функционал данной системы включает в себя несколько модулей:

* построение маршрутов;
* коммуникации с клиентами;
* контроль транспорта;
* транспортное планирование;
* сотрудники и клиенты;
* отчеты;
* администрирование и настройка.

Каждый модуль включает в себя огромное количество действий, позволяющие выполнять самые разнообразные задачи начиная от регистрации клиента, заканчивая составлением необходимой отчетности.

Также есть возможность подключения дополнительных модулей:

* электронный обмен документами;
* мобильное приложение;
* блок повышения лояльности;
* и другие [2].

Эти модули могут быть полезны некоторым крупным грузоперевозчикам, где объем данных значительно больше, нежели в небольших.

Такие функции ускоряют и оптимизируют рабочий процесс, так как тратится меньше времени на сбор всей информации о водителях и отчётах.

Данная система разработана в Российской Федерации.

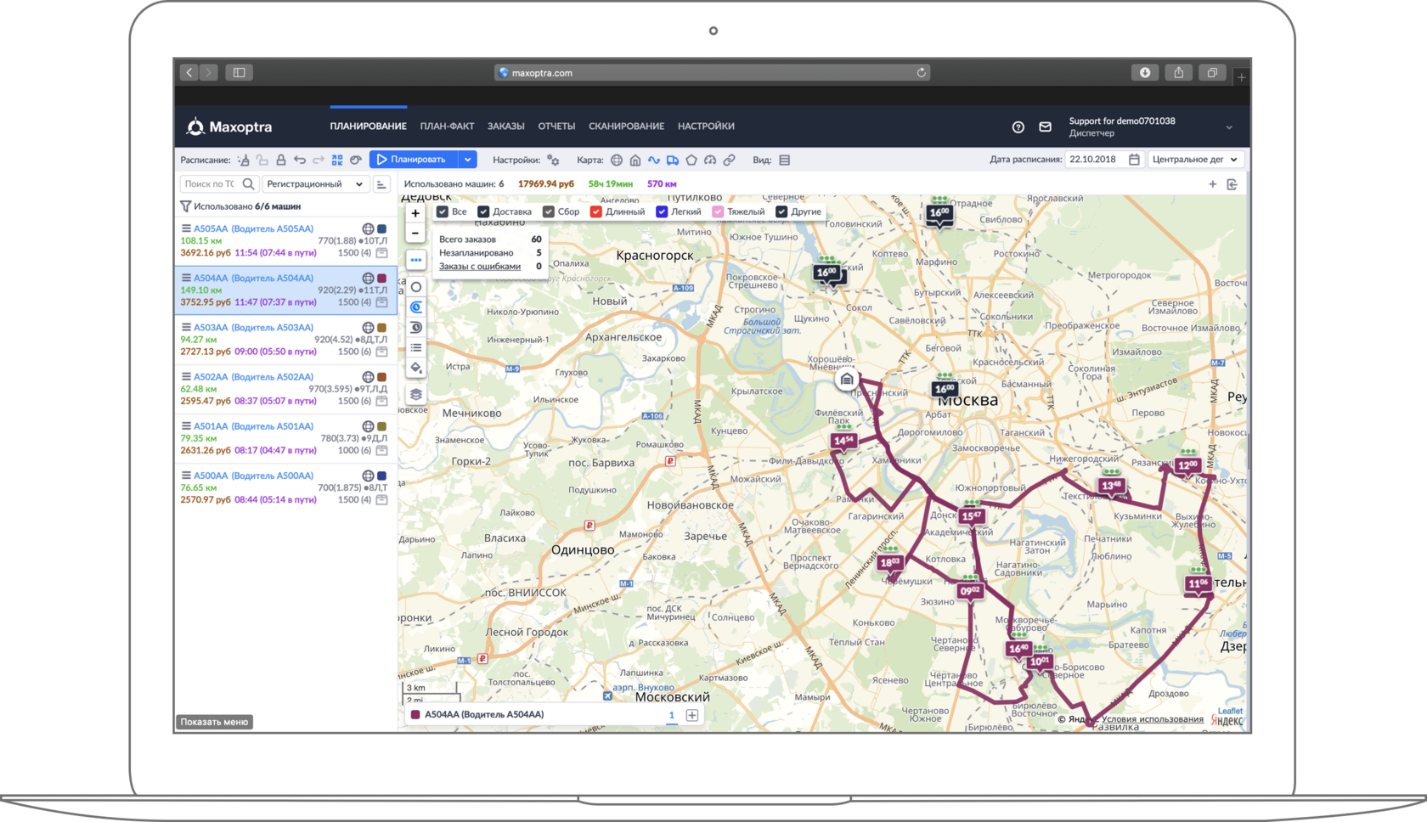


Рисунок 1.1 – интерфейс программы Maxoptra.

## 1.2.2 NovaTrans

NovaTrans – онлайн-сервис, в котором сочетаются как улучшения для работы всех сотрудников, так и мощный ресурс в лице сети Интернет (рис. 1.2).

Система NovaTrans предоставляет следующие возможности:

* формирование электронного заказа;
* получение доступа к удаленным базам данных;
* отслеживание состояния заявок;
* сбор статистики;
* поиск заказов;
* финансовый учёт [3].

Как и в рассматриваемой в пункте 1.2.1 системе, данная система разделена на следующие модули (АРМы):

* электронный обмен документами;
* мобильное приложение;
* блок повышения лояльности;
* и другие.

Каждый модуль ограничивается своим набором действий.

Данная система также разработана в Российской Федерации, однако может использоваться и на международном рынке с силу локализации и некоторых доработок.

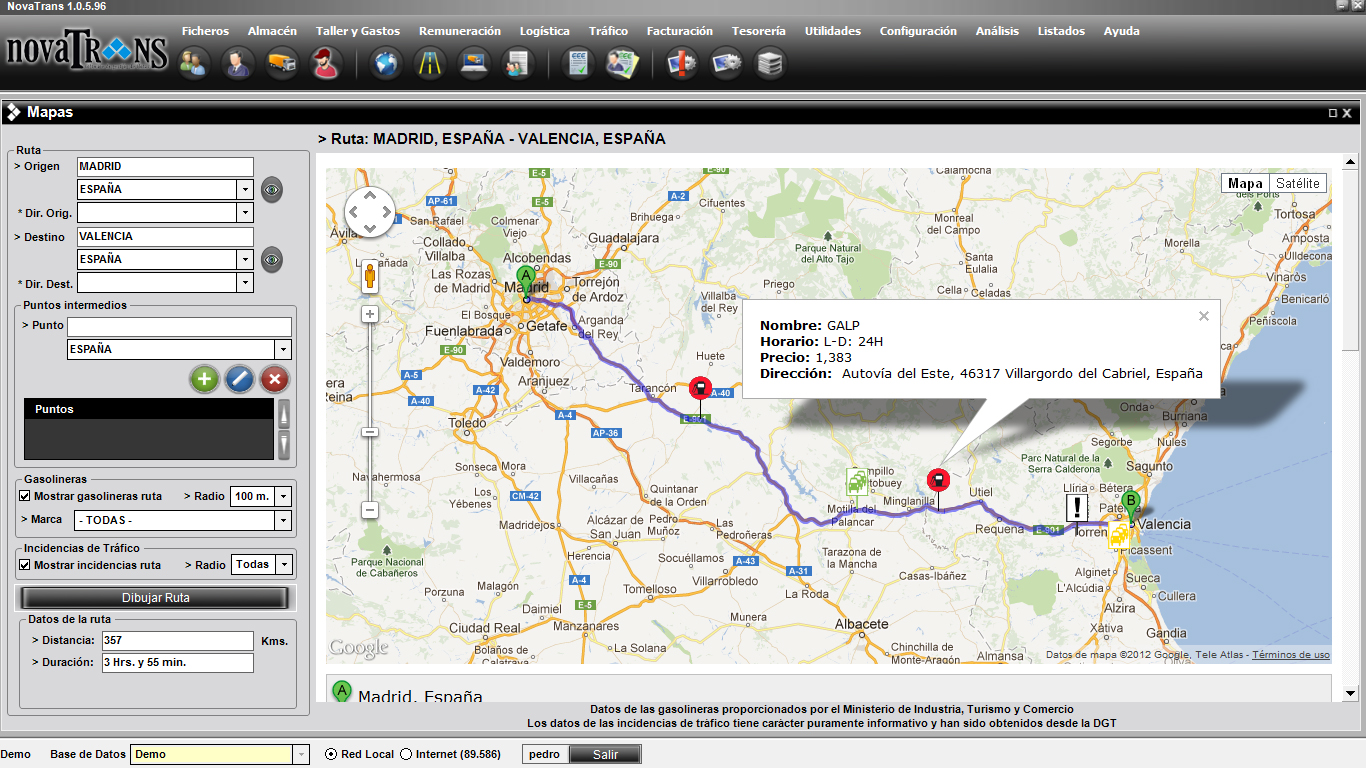


Рисунок 1.2 – интерфейс программы NovaTrans.

## 1.3 Обзор методов и алгоритмов решения поставленной задачи

В рамках курсового проекта необходимо разработать автоматизированную систему работы транспортной компании. В зависимости от перечня предоставляемых услуг пользователю в системе должны быть предусмотрены ограничения на выполняемые действия сотрудниками и клиентами. Так, например, сотрудник должен принимать, подтверждать и корректировать заказы, чего не может делать клиент. Однако, последнему предоставлена возможность совершения заказа и получения информации о компании.

В ходе разработки автоматизированной системы нужно реализовать следующие методы:

* авторизация клиента, которая включает регистрацию в системе и вход пользователя в свою учетную запись;
* редактирование личных данных;
* просмотр необходимой информации как клиентом, так и сотрудником;
* принятие сотрудником поступивших заказов;
* поиск и фильтрация заказов по заданным критериям клиентом и сотрудником;
* сортировка списка заказов сотрудником по указанным критериям;
* добавление и удаление машин;
* подтверждение выполнения заказа сотрудником.

Реализация функции регистрации заключается в создании своего личного кабинета, ограниченного определенным набором задач, которые может выполнять пользователь в зависимости от уровня доступа. Для того, чтобы кабинет был создан необходимо ввести данные, при этом, если кабинет с таким логином уже существует, будет выдана ошибка.

Функция входа в учетную запись ограничивается проверкой на правильность введенных данных. В случае, если запись (логин и пароль) хранящаяся в файле не была найдена, будет выдана ошибка о том, что были введены неверные данные.

Функция редактирования личных данных предполагает внесение пользователем изменений в своем личном кабинете. После изменения выдается сообщение об успешном изменении.

Функция просмотра ограничивается простым выводом информации из файла. Причем, при разработке данной функции необходимо поставить задачу создания простого и удобного пользовательского интерфейса.

Функция приема заказов заключается в просмотре и изменении статуса сотрудником.

Функции поиска и фильтрации ограничиваются простым выводом информации, заданной определенным критерием, на экран в читабельной форме.

Функция сортировки информации по заданным критериям предполагает сортировку по свободным водителям, машинам и сортировка по статусу заказа.

Функция добавления и удаления машин будет реализована у администратора.

Также перед нами поставлена задача сделать максимально удобный и приятный глазам интерфейс, чтобы он был простым в использовании, а главное читабельным. Для этого необходимо сделать так, чтобы пользователь смог работать в программе при помощи нажатия 1-2 клавиш. Чтобы программа была приятна глазу, можно изменить стандартную черно-белую гамму на более привлекательную.

# 2 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ

# СТАНДАРТА IDEF0

## 2.1 Функциональные модели

В разделе, после анализа предметной области, была разработана и реализована функциональная модель процесса ремонта и последующей продажи автомобилей с двумя уровнями декомпозиции при помощи онлайн-сервиса «draw.io», с использованием методологии описания функциональных моделей «IDEF0».

После рассмотрения предметной области, как она должна работать в реальной жизни, можно составить функциональную модель процесса доставки грузов. На рисунке 2.1 представлена контекстная диаграмма верхнего уровня.

Указаны необходимые средства для выполнения данной задачи:

а) Имеющиеся на входе ресурсы:

* заказ на доставку;
* документы на груз;
* груз.

б) Средства контроля и управления:

* законодательство Республики Беларусь;
* внутренние правила компании;
* правила международных перевозок;
* данные и условия заказчика.

в) Механизм выполнения составляют:

* менеджер;
* водитель;
* бухгалтер.

г) Выходные ресурсы:

* доставленный груз;
* бухгалтерский отчёт.

Пункты, с а) по в) описывают все процессы и ресурсы, необходимые для достижения конечных целей, указанных в пункте г).

В итоге был описан процесс получения заявки на доставку груза и последующей его перевозке.

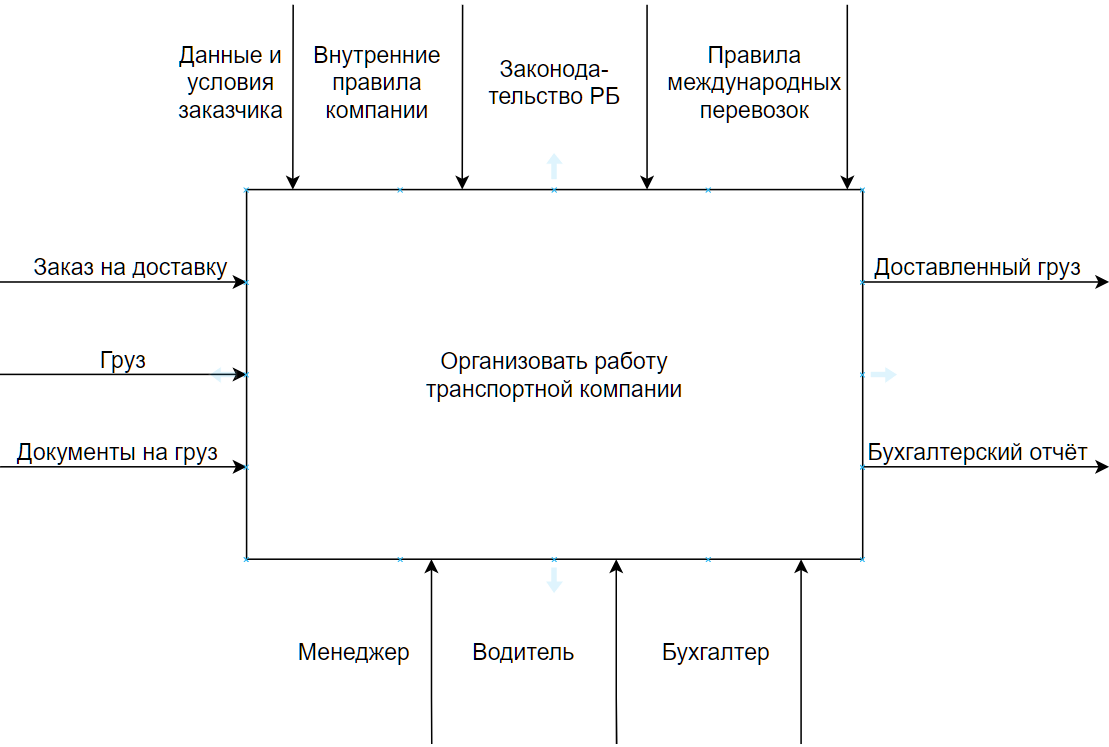


Рисунок 2.1 – Контекстная диаграмма верхнего уровня.

Далее рассмотрим схему с двумя уровнями декомпозиции, что позволит лучше и правильнее понять бизнес-модель рассматриваемой области. Декомпозиция контекстной диаграммы верхнего уровня (см. рис. 2.2).

Представленная декомпозиция состоит из четырёх блоков:

* корректировка и уточнение данных;
* подбор свободного и подходящего персонала;
* доставка груза;
* подтверждение доставки и составление отчёта.

В первый блок «Корректировка и уточнение данных» подходит ресурс «Заказ на доставку», впоследствии он будет обработан менеджером, ведь для корректной доставки необходимо, чтобы все данные, такие как время доставки и отправки, габариты груза и др. сходилось, и клиент получил заказ вовремя. Управляется этот процесс и предоставляются рамки данными и условиями заказчика. На выходе получается полный, сформированный и корректный заказ.

Во второй блок приходит готовый заказ и сам груз, который необходимо доставить. Возникают ситуации, а они достаточно частые, что груз в одну машину не помещается и необходимо выделять несколько водителей. В этом блоке и происходит распределение персонала для доставки груза. Управление происходит внутренними правилами компании, а распределением занимается менеджер.

Переход в третий блок происходит с поступлением в него груза и персонала, который должен его доставить, а также документы на этот груз. Это всё необходимо для доставки в точку назначения, что и является выходным ресурсом. Работа, осуществляемая водителем под управлением законодательства, правил компании и правил международных перевозок.

В последний блок о подтверждении доставки и составления отчёта поступают документы на груз и сам груз в точке назначения, после чего он будет разгружен. Бухгалтер и менеджер подтверждают доставку и заполняют бухгалтерский отчёт об расходах и прибыли компании. Управление осуществляется правилами международных перевозок, данными и условиями заказчика.

На выходе из схемы получается доставленный груз и бухгалтерский отчёт.

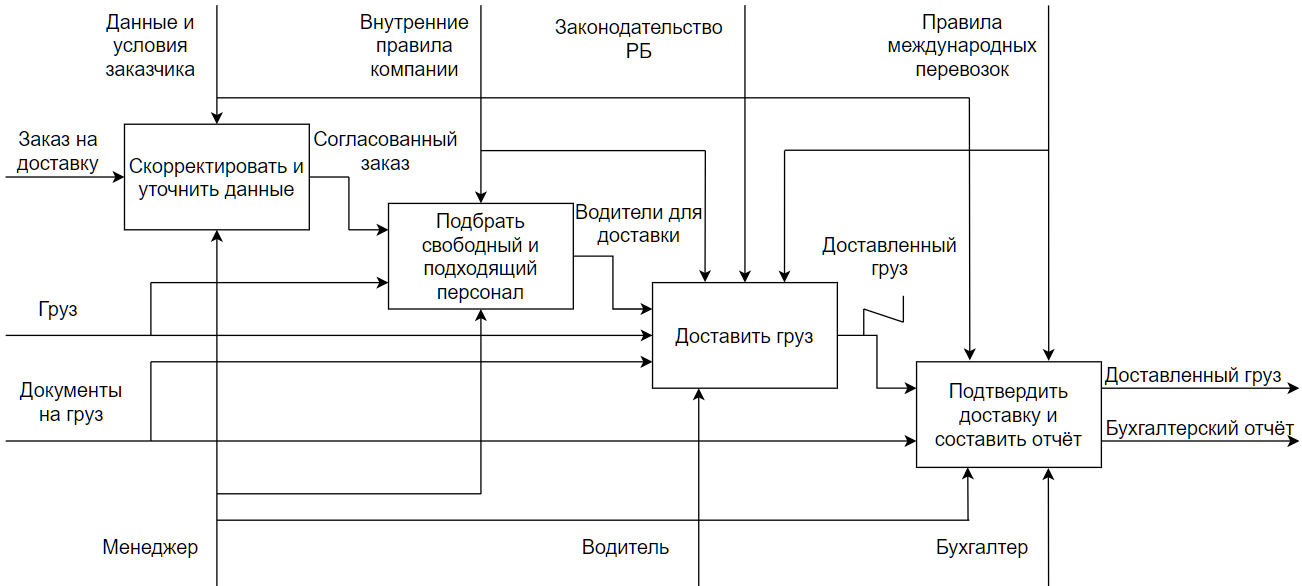


Рисунок 2.2 – Декомпозиция контекстной диаграммы верхнего уровня.

Далее рассмотрим декомпозицию блока «Подбор свободного и подходящего персонала» (см. рис. 2.3). На диаграмме наглядно представлены действия по осуществлению работы в этом блоке, исполняющим звеном служит менеджер, под руководством правил компании. Обрабатывается груз и готовый заказ, на выход приходит груз и персонал, который доставит заказ.

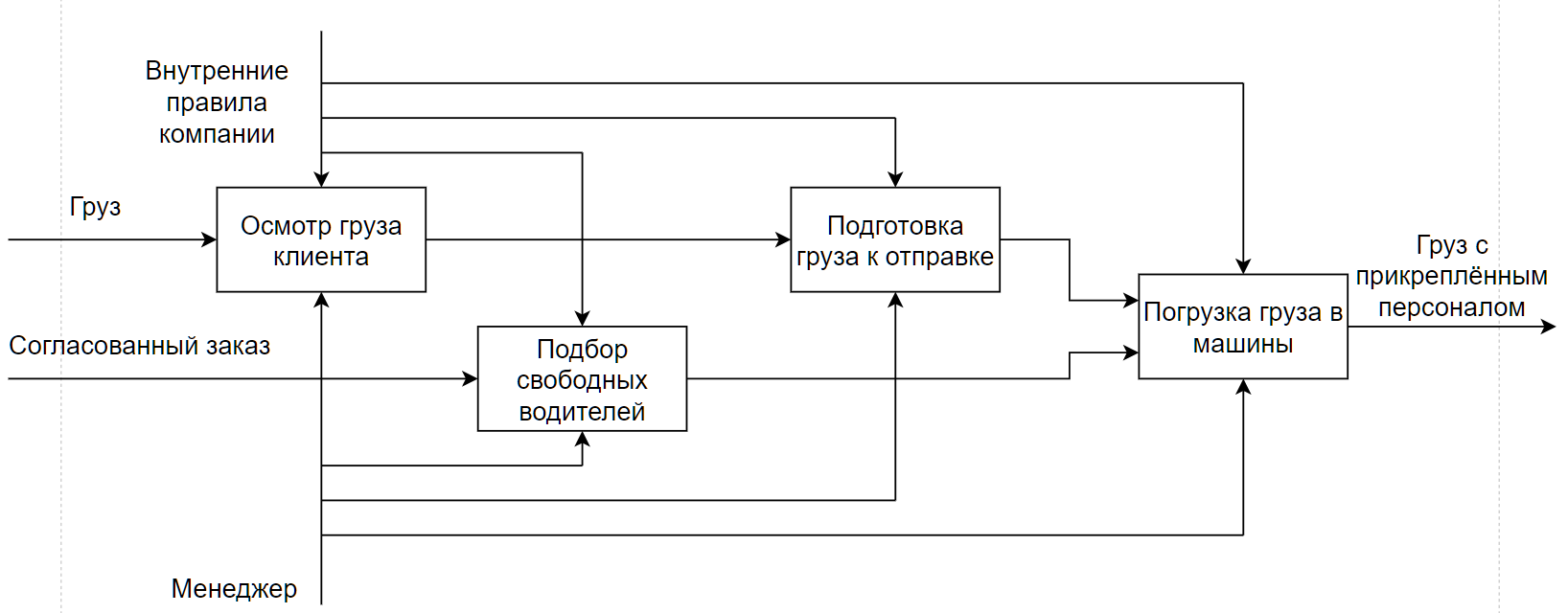


Рисунок 2.3 – Декомпозиция диаграммы нижнего уровня.

По завершении построения диаграмм стало понятно, как именно организована работа в данной сфере деятельности, основную работу производит непосредственно человек, доставляющий сам заказ.

# 3. СХЕМА ДАННЫХ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УЧЕТА ДЕЯТЕЛЬНОСТИ КОМПАНИИ ПО МЕЖДУНАРОДНЫМ ПЕРЕВОЗКАМ

Данные, написанной программой, должны храниться в файлах. Файлы бывают двух видов: бинарные и текстовые. В данной работе использовались файлы текстового формата.++++++++++++++++++++++++++++++++++++++  
 Информация о администраторах хранится в текстовом файле с названием «admin.txt». Данный файл включает личную информацию об администраторе и информацию необходимую для авторизации, а именно:

* имя;
* фамилия;
* возраст;
* статус, из которого следует должность;
* логин;
* пароль в зашифрованном виде.

Информация о машинах храниться в текстовом файле с названием «car.txt». Данный файл включает в себя основную информацию о машине:

* производитель;
* модель;
* колёсная база;
* регистрационный номер.

Информация о грузах клиентов хранится в файле с названием «cargo.txt». Данный фал включает в себя основную информацию о грузе и пункте назначения:

* вес груза;
* место выгрузки;
* дата выгрузки;
* место загрузки;
* дата загрузки;
* размер груза.

Информация о клиентах хранится в текстовом файле с названием «client.txt». Данный файл включает личную информацию об клиенте и информацию необходимую для авторизации, а именно:

* имя;
* фамилия;
* возраст;
* логин;
* пароль в зашифрованном виде;
* количество грузов.

Информация о водителях хранится в текстовом файле с названием «driver.txt». Данный файл включает личную информацию об водителе, информацию о опыте вождения, и информацию необходимую для авторизации, а именно:

* имя;
* фамилия;
* логин;
* пароль в зашифрованном виде;
* опыт вождения;
* дата получения прав;

Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать читать, открывать файлы. Файл – именованный набор байтов, который может быть сохранен на некотором накопителе. Другими словами, под файлом понимается некоторая последовательность байтов, которая имеет своё, уникальное имя, например, «FILE.TXT». В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение, например: file.txt и file.dat – разные файлы, хоть и имеют одинаковые названия. Существует такое понятие, как полное имя файлов – это полный адрес к директории файла с указанием имени файла, например,: «D:\docs\file.txt». Важно понимать эти базовые понятия, иначе сложно будет работать с файлами.   
 Для работы с файлами необходимо подключить заголовочный файл «fstream». В «fstream» определены несколько классов стандартной библиотеки C++ и подключены заголовочные файлы «ifstream» и «ofstream» для считывания и записи файлов соответственно.   
 Для считывания из файла используется объект ifstream, открывается файл при помощи функции open(). В качестве параметров передаются название файла, а также режимы открытия файлов, которые не являются обязательными. Далее при помощи >> происходит запись в файл книг и при помощи функции write запись в файл работников. Эти варианты необходимы каждый в своем случае, так как запись объектов типа string в файлы при помощи второго метода не рекомендуется. Чтобы записать данные в файл, необходимо создать объект класса ofstream, открыть при помощи той же функции open с тем же набором режимов открытия. Запись осуществляется либо при помощи <<, либо при помощи функции read(). В качестве параметров в функции read и write передается буфер и кол-во символов для считывания или записи [4].   
 Таким образом, использование файлов для хранения информации является рациональным решением при небольших проектах.

# 4 РАЗРАБОТКА И ОПИСАНИЕ ДИАГРАММЫ КЛАССОВ

# ПРИЛОЖЕНИЯ

Создания пользовательских типов данных является неотъемлемой частью объектно-ориентированного программирования. Такой подход использовался и при разработке данного проекта.

В приложении Б представлена диаграмма всех классов, используемых в данном проекте, так же в приложении А представлен листинг кода. В данном проекте были разработаны следующие классы:

* Person;
* Worker (наследуется от Person);
* Client (наследуется от Person);
* Car;
* Date;
* Place;
* Direction (обвёртка для Place и Date);
* Cargo;
* Company;
* SmartPointer<T> (вместе со структурой Status<T>).

Класс Person используется для создания администраторов, так же от него наследуются такие классы как Worker и Client (см. рис 4.1). Данные класс содержит следующие поля и методы:

* поля firstname (имя), surname (фамилия), age (возраст), password (пароль), status (уровень доступа), login (логин), autorisation (флаг авторизации);
* конструктор без параметров;
* конструктор с параметрами (firstname, surname, age, password, login, autorisation, status);
* деструктор;
* метод присваивания имени (setFirstname());
* метод присваивания фамилии (setSurname());
* метод присваивания возраста (setAge());
* метод присваивания пароля (setPassword());
* метод присваивания логина (setLogin());
* метод присваивания уровня доступа (setStatus());
* метод присваивания флага авторизации (setAutorisation());
* перегруженный оператор << для вывода информации о человеке;
* перегруженный оператор >> для ввода информации о человеке;
* метод возврата имени (getFirstname());
* метод возврата фамилии (getSurname());
* метод возврата возраста (getAge());
* метод возврата пароля (getPassword());
* метод возврата логина (getLogin());
* метод возврата уровня доступа (getStatus());
* метод возврата флага авторизации (getAutorisation());
* Метод для записи объекта класса Person в файл.

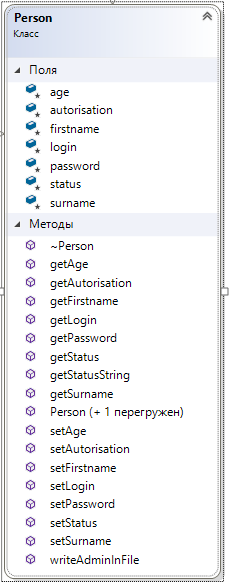


Рисунок 4.1 – Поля и методы класса Person.

Класс Worker используется для создания работников (см. рис 4.2).. Он также наследуется от класса Person и содержит следующие поля и методы:

* поля drivingExperience (опыт вождения), driverLicenseDate (дата получения прав), namber (номер водителя), statusDriverOnline (флаг), workerFree(флаг);
* конструктор без параметров;
* конструктор с параметрами (firstname, surname, age, drivingExperience, day, month, year, password, status, login, autorisation, namber, workerFree);
* деструктор;
* метод присваивания флага (setWorkerFree());
* метод присваивания номера водителя (setNumberDriver());
* метод присваивания флага (setStatusDriver());
* метод присваивания даты получения прав (setDriverLicenseDate()) путем передачи объекта Date;
* метод присваивания даты получения прав (setDriverLicenseDate()) путем передачи трех целочисленных переменных;
* метод присваивания опыта вождения (setDrivingExperience());
* метод возврата флага (getStatusDriver());
* метод возврата номера водителя (getNumberDriver());
* метод возврата флага (getWorkerFree());
* метод возврата даты получения прав (getDriverLicenseDate()) путем получения объекта Date;
* метод возврата опыта вождения (getDrivingExperience());
* перегруженный оператор << для вывода информации о работнике;
* перегруженный оператор >> для ввода информации о работнике;
* Метод для записи объекта класса Worker в файл (writeDriverInFile()).

Класс Client используется для создания клиентов (см. рис 4.1). Он также наследуется от класса Person и содержит следующие поля и методы:

* поле shipments (множество объектов Cargo хранящиеся в STL контейнере vector);
* конструктор без параметров;
* конструктор с параметрами (firstname, surname, age, password, login, autorisation, status);
* конструктор копирования;
* деструктор;
* перегруженный оператор << для вывода информации о клиенте;
* перегруженный оператор >> для ввода информации о клиенте;
* метод возврата STL контейнера в котом хранятся объекты Cargo;
* метод добавления объекта в STL контейнер;
* перегруженный метод объекта в STL контейнер;
* метод удаляющий объект из контейнера STL;
* метод позволяющий вывести все объекты Cargo в консоль;
* метод позволяющий отсортировать STL контейнер по весу;
* метод возврата размера STL контейнера;
* метод записи объекта Client в файл.

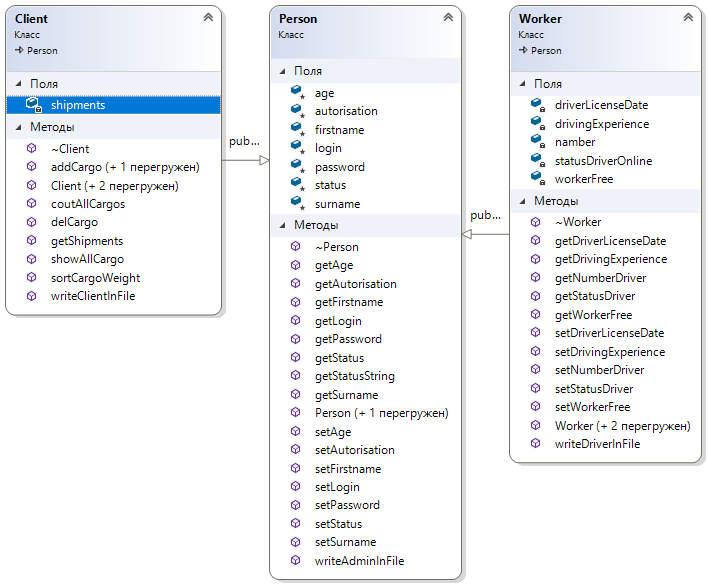


Рисунок 4.1 – Поля и методы классов Worker, Person and Client.

Класс Car используется для создания машин (см. рис 4.2) и содержит следующие поля и методы:

* поля wheelbase (колесная база), mark (производитель), model (модель), machineNumber (регистрационный номер), number (вспомогательная переменная);
* конструктор без параметров;
* конструктор с параметрами (wheelbase, mark, model, machineNumber, number, status);
* конструктор копирования;
* деструктор;
* метод присваивания колесной базы (setWheelbase());
* метод присваивания производителя (setMark());
* метод присваивания регистрационного номера (setMachineNumber());
* метод присваивания модели (setModel());
* метод возврата модели (getModel ());
* метод возврата колесной базы (getWheelbase());
* метод возврата производителя (getMark);
* метод возврата регистрационного номера (getMachineNumber());
* перегруженный оператор << для вывода информации о машине;
* перегруженный оператор >> для ввода информации о машине;
* метод записи объекта в файл (writeCarInFile()).

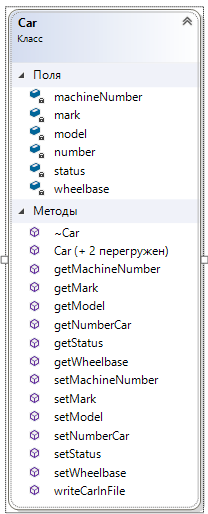


Рисунок 4.2 – Поля и методы класса Car.

Класс Place используется для группировки строк в адрес (см. рис 4.3). и содержит следующие поля и методы:

* поля country, city, street, house;
* конструктор без параметров;
* конструктор с параметрами (country, city, street, house);
* деструктор;
* метод присваивания страны (setCountry());
* метод присваивания города (setCity());
* метод присваивания улицы (setStreet());
* метод присваивания дома (setHouse());
* метод присваивания всех данных (setPlace());
* метод возврата страны (getCountry());
* метод возврата города (getCity());
* метод возврата улицы (getStreet());
* метод возврата дома (getHouse());
* перегруженный оператор << для вывода информации о месте;
* перегруженный оператор >> для ввода информации о месте.

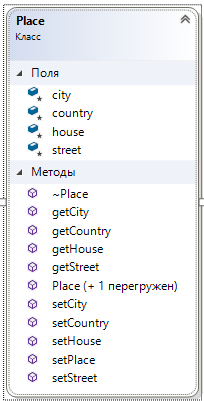


Рисунок 4.3 – Поля и методы класса Place.

Класс Date используется для группировки целочисленных переменных в дату (см. рис 4.4). и содержит следующие поля и методы:

* поля day, month, year;
* конструктор без параметров;
* конструктор копирования;
* конструктор с параметрами;
* метод присваивания всех данных (setDate());
* метод присваивания дня (setDay());
* метод присваивания месяца (setMonth());
* метод присваивания года (setYear())
* метод возврата дня (getDay());
* метод возврата месяца (getMonth());
* метод возврата года (getYear())
* перегруженный оператор << для вывода даты;
* перегруженный оператор >> для ввода даты.

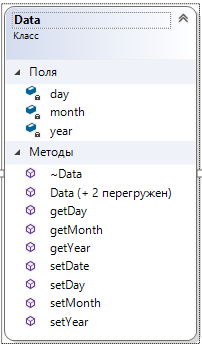


Рисунок 4.4 – Поля и методы класса Date.

Класс Cargo используется для хранения данных об грузе(см. рис 4.5). и содержит следующие поля и методы:

* поля weight (вес груза), overallDimensions (размер груза), orderFulfillmentDriver (водитель выполняющий работу), car, direction;
* конструктор без параметров;
* конструктор с параметрами;
* деструктор;
* метод присваивания веса груза (setWeight);
* метод присваивания размер груза (setOverallDimensions);
* метод присваивания машины (setCar);
* метод присваивания водителя выполняющего работу (setorderFulfillmentDriver);
* метод возврата веса груза ();
* метод возврата размер груза (getWeight);
* метод возврата машины (getCar);
* метод возврата водителя выполняющего работу (getOrderFulfillmentDriver);
* метод вывода объекта в консоль (outCargo);
* метод ввода информации об объекте из консоли (inCargo);
* перегруженный оператор << для вывода груза;
* метод вывода списка водителей по критерию занятости (getDriversTrueOrFalse);
* метод записи объекта в файл (writeCargoInFile).

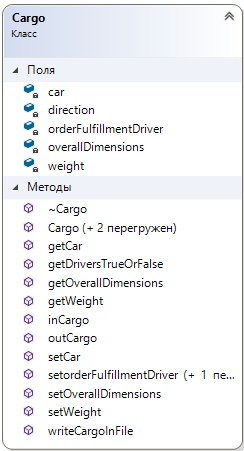


Рисунок 4.5 – Поля и методы класса Cargo.

Класс Direction используется в качестве обвертки классов Date и Place, которая позволяет хранить информацию об выгрузке и загрузки груза (см. рис 4.6). и содержит следующие поля и методы:

* поля placeOfShipment (место выгрузки), shippingDate (дата выгрузки), placeOfUnloading (место загрузки), dateOfUnloading (дата выгрузки);
* конструктор без параметров;
* конструктор с параметрами;
* конструктор копирования;
* деструктор;
* метод присваивания места выгрузки, так же его перегрузка (setPlaceOfShipment);
* метод присваивания даты выгрузки, так же его перегрузка (setShippingDate);
* метод присваивания места загрузки, так же его перегрузка (setPlaceOfUnloading);
* метод присваивания даты загрузки, так же его перегрузка (setDateOfUnloading);
* метод возврата места выгрузки (getPlaceOfShipment);
* метод возврата даты выгрузки (getShippingDate);
* метод возврата места загрузки (getPlaceOfUnloading);
* метод возврата даты загрузки (getDateOfUnloading);
* перегруженный оператор << для вывода все информации;
* перегруженный оператор >> для ввода все информации.

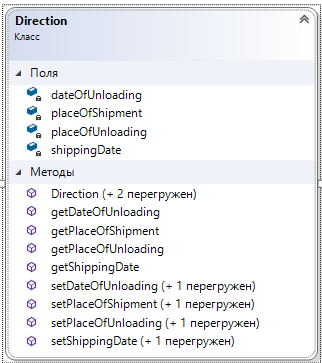


Рисунок 4.6 – Поля и методы класса Direction.

Класс Company используется в качестве объединяющего класса, который выполняет основной функционал приложения и хранит в себе всю информацию (см. рис 4.7). и содержит следующие поля и методы:

* поля clients (STL контейнер для хранения клиентов), admins (STL контейнер для хранения администраторов), cars (STL контейнер для хранения машин), drivers (STL контейнер для хранения водителей), statusAutorisation (флаг авторизации), who (флаг, обозначающий кто авторизован в системе);
* конструктор без параметров;
* деструктор;
* метод добавления клиента в вектор (addClient);
* метод добавления администратора (addAdmin);
* метод добавления водителя (addDriver);
* метод добавления машины (addCar);
* метод присваивания флага авторизации (setStatusAutorisation);
* метод возврата флага авторизации (getStatusAutorisation)
* метод авторизации ();
* метод регистрации ();
* метод выводящий всех клиентов в консоль (showAllClients);
* метод выводящий всех администраторов в консоль (showAllAdmins);
* метод выводящий всех водителей в консоль (showAllDrivers);
* метод выводящий всех машин в консоль (showALLCar);
* метод удаляющий водителя (delDriver);
* метод удаляющий клиента (delClient);
* метод удаляющий администратора (delAdmin);
* метод поиска клиента (findClientrAndShowAllInf);
* метод поиска администратора (findAdminAndShowAllInf);
* метод поиска водителя (findDriverAndShowAllInf);
* метод сортировки клинтов по возрасту (sortClientsAge);
* метод вывода информации о грузе для определенного водителя (showCargoForDriver);
* метод записи всех водителей в файл (writeAllDriversInFile);
* метод записи всех администраторов в файл (writeAllAdminsInFile);
* метод записи всех машин в файл (writeAllCarsInFile);
* метод записи всех клиентов в файл (writeAllClientsInFile);
* метод считывания всех водителей из файла (readAllDriversFromFile);
* метод считывания всех администраторов из файла (readAllAdminsFromFile);
* метод считывания всех машин из файла (readAllCarsFromFile);
* метод считывания всех клиентов из файла (readALLClientsFromFile);
* метод возврата размера вектора водителей (getSizeVectorDrivers);
* статический метод фильтрации водителей по критерию занятости (getDriversTrueOrFalse);
* метод организации функционала для клиента (menuClient);
* метод организации функционала для водителя (menuDriver).

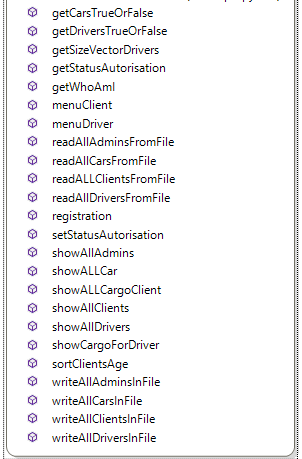
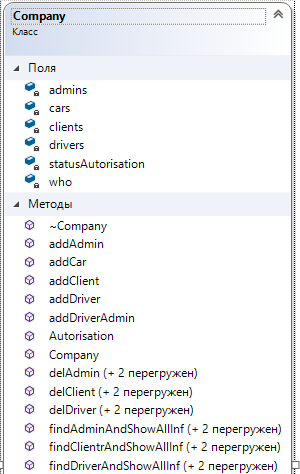


Рисунок 4.7 – Поля и методы класса Company.

В данной главе были рассмотрены все необходимые методы для работы данной программы. Разработанный функционал позволит без затруднений обрабатывать данные.

# 5 РАЗРАБОТКА И ОПИСАНИЕ ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ

Проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых прецедентов. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой [5].

В приложении В (рис. В1) представлена диаграмма вариантов использования данного приложения. Действующими лицами в ней являются: администратор, водитель, клиент. Каждому из них доступны различные функции.

Клиент имеет возможность выполнять следующие действия:

* посмотреть свои заказы;
* сделать заказ;
* удалить заказ;
* посмотреть свободных водителей;
* изменения личных данных.

Администратор может пользоваться следующими функциями:

* показать всех клиентов, водителей, администраторов;
* добавить водителя, администратора, клиента;
* удалить водителя, клиента, администратора;
* поиск водителя, администратора, клиента;
* добавить машину;
* сортировать клиентов по возрасту;
* показать все машины.

Водитель может использовать функции:

* просмотреть грузы;
* просмотреть машину;
* отметить груз.

# 6 СХЕМА АЛГОРИТМА РАБОТЫ ВСЕЙ ПРОГРАММЫ И АЛГОРИТМА РАБОТЫ НЕСКОЛЬКИХ ОСНОВНЫХ МЕТОДОВ

Понятие алгоритма является одним из основных понятий вычислительной математики и информатики.

Алгоритм — строго определенная последовательность действий для некоторого исполнителя, приводящая к поставленной цели или заданному результату за конечное число шагов.

Любой алгоритм составляется в расчете на конкретного исполнителя с учетом его возможностей. Исполнитель — субъект, способный исполнять некоторый набор команд. Совокупность команд, которые исполнитель может понять и выполнить, называется системой команд исполнителя.

Для выполнения алгоритма исполнителю недостаточно только самого алгоритма. Выполнить алгоритм — значит применить его к решению конкретной задачи, т. е. выполнить запланированные действия по отношению к определенным входным данным. Поэтому исполнителю необходимо иметь исходные (входные) данные — те, что задаются до начала алгоритма.

В результате выполнения алгоритма исполнитель должен получить искомый результат — **выходные данные**, которые исполнитель выдает как результат выполненной работы. В процессе работы исполнитель может создавать и использовать данные, не являющиеся выходными, — **промежуточные данные** [6].

В приложении Г (рис. Г1) представлена схема алгоритма работы программы. Сперва, при запуске программы должна выполниться авторизация пользователя или регистрация, где будет выбран уровень доступа, в зависимости от которого дальнейшие меню и функции будут разниться. В следующем меню можно выбрать желаемый метод, затем произойдёт его выполнение.

В приложении Г (рис. Г2-Г5) представлены блок-схемы функций работы файлов и проверки их существования. Все эти функции начинаются проверкой на наличие файла и, если файла нет, создают его. В дальнейшем функции выполняют группу задач, действия над которыми необходимы.

# 7 ОПИСАНИЕ ЗАПУСКА, ИСПОЛЬЗОВАНИЯ И РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Консольное приложение запускается при открытии файла «CourseProdject.exe». При запуске консольного приложения пользователю предоставляется меню с двумя пунктами: регистрация и авторизация (см. рис. 7.1). Перемещения по меню происходит по средству ввода соответсвуйщих чисел.

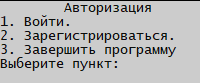


Рисунок 7.1 – Меню авторизации и регистрации.

При выборе регистрации или авторизации следует меню уровня доступа функциональной наполненности приложения, то есть пользователь может войти или зарегистрироваться в качестве: администратора, клиента или водителя (см. рис 7.2 и рис. 7.3).

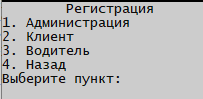


Рисунок 7.2 – Меню выбора уровня доступа при регистрации.

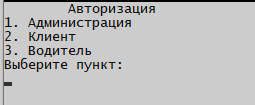


Рисунок 7.3 – Меню выбора уровня доступа при авторизации.

При входе в систему пользователю нужно ввести логин и пароль. Если введенные данные верны, то выведется сообщения об успешной авторизации (см. рис. 7.4), если же данные не верны, то будет выведено соответствующее сообщение (см. рис. 7.5). После попытки авторизации пользователем, через две секунды произойдет переход в соответствуйщее меню в зависимости от результат авторизации.+++++++++++++++++++++++++++++++++++++++++

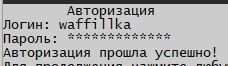


Рисунок 7.4 – Меню авторизации при вводе верных данных.



Рисунок 7.4 – Меню авторизации при вводе неверных данных.

При регистрации в качестве администратора и клиента пользователю требуется ввести данные, а именно: имя, фамилия, возраст, логин и пароль (см. рис. 7.5).++++++++++++++++++++++++++++++++++++++++++++++++++++

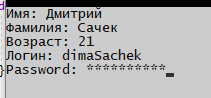


Рисунок 7.5 – Меню регистрации администратора или клиента.

При регистрации в качестве водителя пользователю требуется ввести данные, а именно: имя, фамилия, возраст, стаж вождения, дату получения водительских прав, логин и пароль (см. рис. 7.6). Так же пользователю предоставляется возможность регистрации со своей машиной либо без нее (см. рис. 7.7 и 7.8).

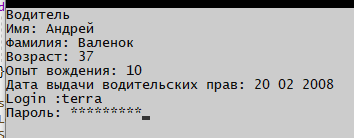


Рисунок 7.6 – Меню регистрации водителя без машины.

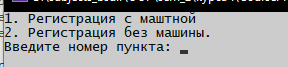


Рисунок 7.7 – Меню выбора регистрации водителя.

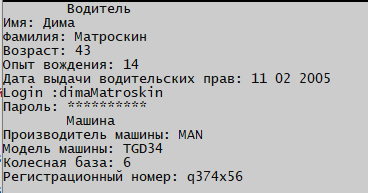


Рисунок 7.8 – Меню регистрации водителя с машиной.

При авторизации пользователем в качестве администратора предоставляется меню, в котором можно передвигаться по средству ввода чисел соответствующих пунктов (см. рис. 7.9).

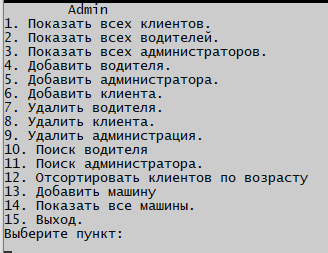


Рисунок 7.8 – Меню администратора.

Выбрав один из первых трех пунктов, администратор может посмотреть всех клиентов, водителей и администраторов соответственно (см. рис. 7.9, рис. 7.10, рис. 7.11).

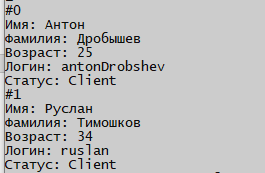


Рисунок 7.9 – Вывод всех клиентов в консоль.

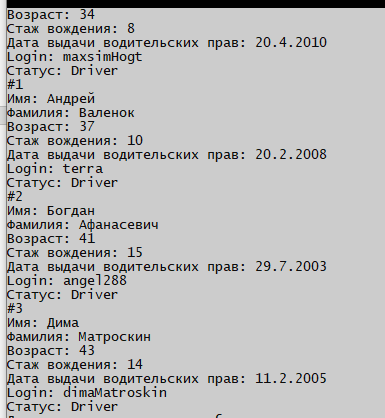


Рисунок 7.10 – Вывод всех водителей в консоль.

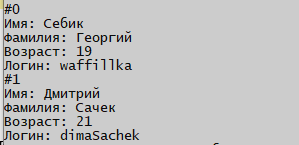


Рисунок 7.11 – Вывод всех администратор в консоль.

Выбрав один из пунктов с третьего по шестой и тринадцатый, администратор может добавить клиента, администратора, водителя, машину соответственно (см. рис. 7.12, рис. 7.13, рис. 7.14, рис. 7.15).

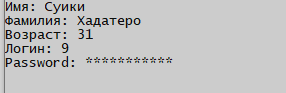


Рисунок 7.12 – Добавления клиента.

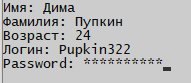


Рисунок 7.13 – Добавления администратора.

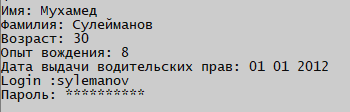


Рисунок 7.14 – Добавления водителя.

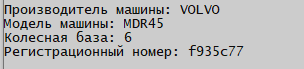


Рисунок 7.15 –Добавления машины.

Пункты с седьмого по девятый позволят удалить из базы данных водителя, клиента, администратора соответственно. +  
 Пункты десятый и одиннадцатый позволяют осуществлять поиск водителей и администраторов по логину либо по имени и фамилии соответственно (см. рис. 7.16,рис. 7.17, рис. 7.18, рис. 7.19, рис. 7.20). Если данные введены неверно, то выводится сообщения о том, что поиск не дал результата.



Рисунок 7.16 – Меню выбора критерия поиска.

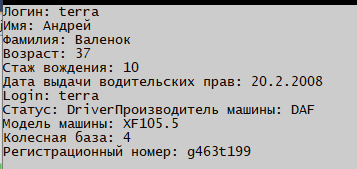


Рисунок 7.17 – Поиск водителя по логину.

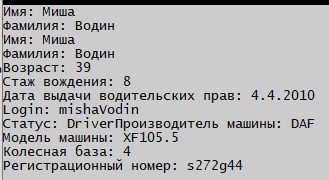


Рисунок 7.18 – Поиск водителя по имени и фамилии.

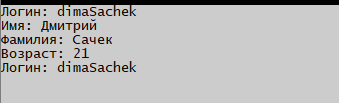


Рисунок 7.19 – Поиск администратора по логину.

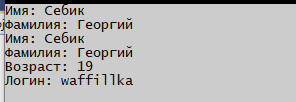


Рисунок 7.20 – Поиск администратора по имени и фамилии.

Пункт двенадцать позволяет отсортировать склиентов по возрасту. Пункт четырнадцать предоставляет возможность просмотреть все машины в базе данных (см. рис. 7.21).

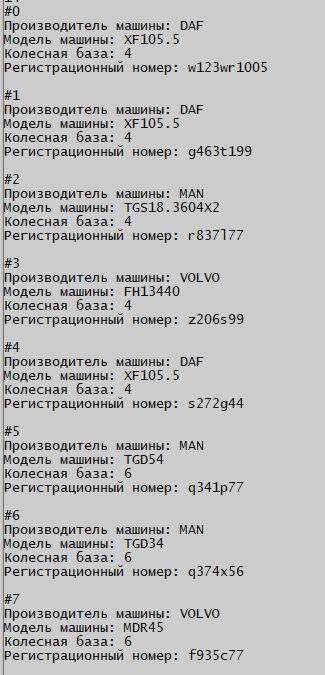


Рисунок 7.21 – Просмотр всех машин.

При авторизации пользователем в качестве клиента, будет представлено меню, в котором можно передвигаться по средству ввода чисел соответствующих пунктов (см. рис. 7.22).

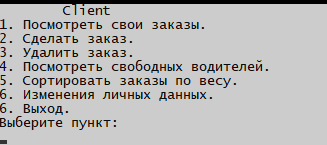


Рисунок 7.22 – Меню клиента.

Первый пункт позволяет посмотреть все сделанные заказы данного клиента (см. рис. 7.23). Так же клиент может сделать заказ вторым пунктом меню (см. рис. 7.24) и третьим пунктом удалить определенный заказ.

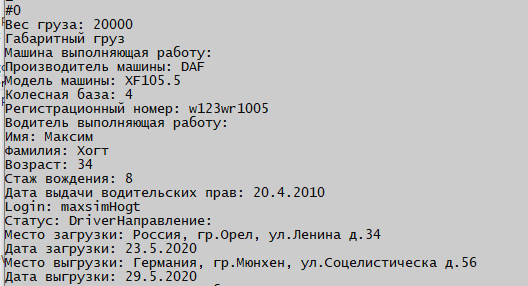


Рисунок 7.23 – Вывод всех заказов данного клиента.

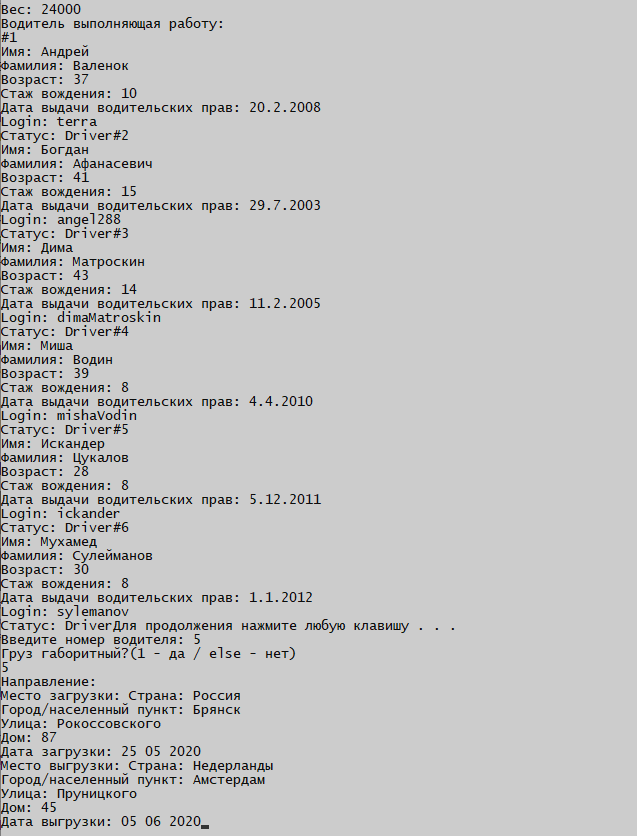


Рисунок 7.24 – Меню оформления заказа.

Также при выборе четвертого или пятого пунктов приложение выводит на в консоль фильтрованных водителей по критерию занятости (см. рис. 7.26) или сортирует грузы данного клиента по весу соответственно. Существует возможность изменения личных данных клиента, а именно: имя, фамилия, возраст, пароль (см. рис. 7.25).

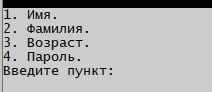


Рисунок 7.25 – Меню редактирования личных данных клиента.

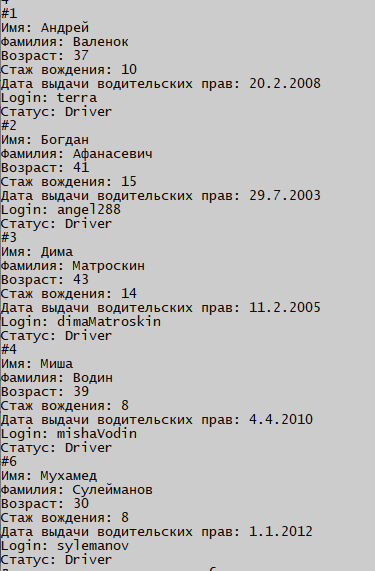


Рисунок 7.26 – Вывод свободных водителей в консоль.

При авторизации пользователем в качестве водителя, будет представлено меню, в котором можно передвигаться по средству ввода чисел соответствующих пунктов (см. рис. 7.27).

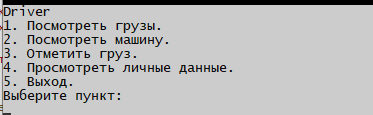


Рисунок 7.27 – Меню водителя.

Первый пункт меню позволяет вывести в консоль грузы, которые выполняет данный водитель, также второй пункт позволяет вывести в консоль машину за которой закреплен данный водитель (см. рис. 7.28). Что касательно четвертого пункта, который позволяет просмотреть личные данные водителя (см. рис. 7.29).

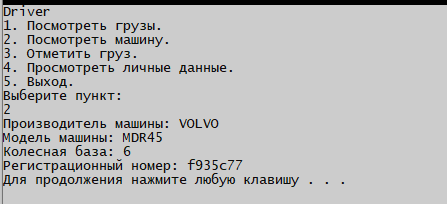


Рисунок 7.28 –Вывод в консоль закрепленной за водителем машины.

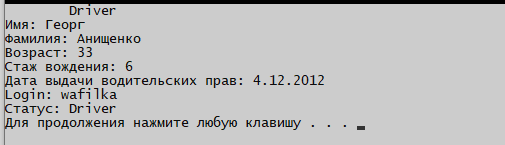


Рисунок 7.29 –Вывод в консоль закрепленной за водителем машины.

# ЗАКЛЮЧЕНИЕ

Цель курсовой работы заключалась в том, чтобы повысить качество, скорость и прибыльность компании за счёт автоматизации бизнес-процесса путём написания автоматизированной системы учёта перевозок. В результаты выполненной работы были достигнуты все задачи, поставленные в начале.

Приложение позволяет работать с информацией, а именно добавлять и удалять её, а также редактировать. Администратору доступны такие функции, как: просмотр данных об пользователях, добавление пользователей, удаление пользователей, поиск пользователей, добавление и редактирование автомобилей. Однако водитель и клиент ограничены по функционалу.

Разработанная программа удовлетворяет всем пунктам технического задания:

* программа реализована на языке программирования С++ с использованием консольного приложения;
* интерфейс выполнен в максимально минималистичном виде, однако читабелен и приятен глазу;
* производится полная шифровка паролей для ограничения доступа к ним третьим лицам;
* вся информация, полученная в результате работы программы, записывается в текстовые файлы. В оперативной памяти данные хранятся с использованием контейнеров библиотеки STL;
* в приложении соблюдены основные правила объектно-ориентированного метода программирования: полиморфизм, инкапсуляция, абстракция, наследование;
* программа способна работать на ОС Windows 7 и выше, интерфейс выполнен на русском языке.

Ни одна компания по грузоперевозкам не может обходиться без автоматизированных электронных систем. Данный продукт как никогда актуален в настоящее время, однако больше он подойдёт для небольшой компании, так как в нём отсутствуют многие критерии, необходимые крупным компаниям, такие как статистика и информация, отчётность по финансам.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Al-ba [Электронный ресурс]. – Режим доступа: [https://www.al-ba.ru](https://www.al-ba.ru/)  
[2] Maxoptra [Электронный ресурс]. – Режим доступа: [https://maxoptra.ru](https://maxoptra.ru/)

[3] NovaTrans [Электронный ресурс]. – Режим доступа: http://www.novatrans.lv

[4] CppStudio [Электронный ресурс]. – Режим доступа: http://cppstudio.com

[5] УчительПро [Электронный ресурс]. – Режим доступа: https://uchitel.pro

[6] Habr [Электронный ресурс]. – Режим доступа: <https://habr.com>

# ПРИЛОЖЕНИЕ А

# (обязательное) Листинг кода

**(к пункту 4)**

//библиотеки С++

#include<iostream>

#include<vector>

#include <fstream>

#include<algorithm>

#include<string>

#include<cctype>

#include "Client.h"

## #include "Menu.h"

//пространство имен

using namespace std;

using namespace menu;

//функция входа

void LogIn(Company& company, char n) {

string log, password;

cout << "\tАвторизация" << endl;

cout << "Логин: ";

cin >> log;

cout << "Пароль: ";

passwordEnterWithStars(password);

if (company.Autorisation(n, log, encryptPassword(password)))

cout << "Авторизация прошла успешно!" << endl;

else

cout << "Неверный логин или пароль!" << endl;

}

//функция авторизации

bool Company::Autorisation(char status, string login, string password) // 1 - админ / 2 - клиент / 3 - водитель

{

if (status == '1')//Admin

{

for (int i = 0; i < this->admins.size(); i++)

if (this->admins[i].getLogin() == login && this->admins[i].getPassword() == password)

{

this->who = 1;

return statusAutorisation = true;

}

}

else if(status == '2')//Client

{

for (int i = 0; i < this->clients.size(); i++)

if (this->clients[i].getLogin() == login && this->clients[i].getPassword() == password)

{

this->who = 2;

this->clients[i].setAutorisation(true);

return statusAutorisation = true;

}

}

else if (status == '3')//Driver

{

for (int i = 0; i < drivers.size(); i++)

if (drivers[i].getLogin() == login && drivers[i].getPassword() == password)

{

drivers[i].setAutorisation(true);

this->who = 3;

return statusAutorisation = true;

}

}

return false;

}

//функция регистрации

bool Company::registration(char status)

{

system("cls");

if (status == '1')

{

Person t;

cin >> t;

addAdmin(t);

}

else if (status == '2')

{

Client t;

cin >> t;

addClient(t);

}

else if (status == '3')

{

cout << "1. Регистрация с маштной\n2. Регистрация без машины.\nВведите номер пункта: ";

cin >> status;

system("cls");

if (status == '1') {

cout << "\tВодитель" << endl;

Worker driver;

driver.setStatusDriver(true);

driver.setStatus(3);

driver.setNumberDriver(drivers.size());

cin >> driver;

addDriver(driver);

cout << "\tМашина" << endl;

Car car;

cin >> car;

addCar(car);

}

else

{

cout << "Водитель" << endl;

Worker driver;

cin >> driver;

driver.setStatusDriver(true);

driver.setStatus(3);

driver.setNumberDriver(drivers.size());

addDriver(driver);

}

}

return false;

}

//считывания всех машин из файла

void Company::readAllCarsFromFile(char\* file\_name)

{

fstream f;

int wheelbase, number;

string mark, model, machineNumber;

bool status;

f.open(file\_name);

if (!f) {

cerr << "Невозможно открыть базовый файл\n";

exit(-1);

}

f.seekg(0);

while (!f.eof())

{

f >> wheelbase >> mark >> model >> machineNumber >> number >> status;

if (mark != "")

{

cars.push\_back(Car(wheelbase, mark, model, machineNumber, number, status));

}

}

f.close();

}

//считывания всех водителей из файла

void Company::readAllDriversFromFile(char\* file\_name)

{

fstream f;

string firstname, surname, password, login;

int age, status, drivingExperience, namber, day, month, year;

Data driverLicenseDate;

bool statusDriver, autorisation, workerFree;

f.open(file\_name);

if (!f) {

cerr << "Невозможно открыть базовый файл\n";

exit(-1);

}

f.seekg(0);

while (!f.eof())

{

f >> firstname >> surname >> age >> autorisation >> status >> login >> password >> namber >> drivingExperience >> day >> month >> year >> workerFree;

if (firstname != "")

{

drivers.push\_back(Worker(firstname, surname, age, drivingExperience, day, month, year, password, status, login, autorisation, namber, workerFree));

}

}

f.close();

}

//считывания всех администраторов из файла

void Company::readAllAdminsFromFile(char\* file\_name)

{

fstream f;

string firstname, surname, password, login;

int age, status;

bool autorisation;

f.open(file\_name);

if (!f) {

cerr << "Невозможно открыть базовый файл\n";

exit(-1);

}

f.seekg(0);

while (!f.eof())

{

f >> firstname >> surname >> age >> autorisation >> status >> login >> password;

if(firstname != "")

admins.push\_back(Person(firstname, surname, age, password, login, autorisation, status));

}

f.close();

}

//считывания всех клиентов и их заказы из файла

void Company::readALLClientsFromFile(char\* file\_client, char\* file\_cargo)

{

fstream client, cargo;

string firstname, surname, password, login, placeOfUnloadinCountry, placeOfUnloadinCity, placeOfUnloadinStreet, placeOfUnloadinHouse;

string placeOfShipmentCountry, placeOfShipmentCity, placeOfShipmentStreet, placeOfShipmentHouse;

int age, status, countCargo, orderFulfillmentDriver, car, dateOfUnloadingDay, dateOfUnloadingMonth, dateOfUnloadingYear, shippingDateDay, shippingDateMonth, shippingDateYear;

bool autorisation, overallDimensions;

float weight;

client.open(file\_client);

cargo.open(file\_cargo);

if (!client) {

cerr << "Невозможно открыть базовый файл\n";

exit(-1);

}

if (!cargo) {

cerr << "Невозможно открыть базовый файл\n";

exit(-1);

}

client.seekg(0);

cargo.seekg(0);

int n = 0;

while (!client.eof())

{

client >> firstname >> surname >> age >> autorisation >> status >> login >> password >> countCargo;

if (firstname != "")

{

clients.push\_back(Client(firstname, surname, age, password, login, autorisation, status));

for (int i = 0; i < countCargo; i++)

{

cargo >> weight >> overallDimensions >> orderFulfillmentDriver >> car >> dateOfUnloadingDay >> dateOfUnloadingMonth >> dateOfUnloadingYear >> shippingDateDay >> shippingDateMonth >> shippingDateYear >> placeOfShipmentCountry >> placeOfShipmentCity >> placeOfShipmentStreet >> placeOfShipmentHouse >> placeOfUnloadinCountry >> placeOfUnloadinCity >> placeOfUnloadinStreet >> placeOfUnloadinHouse;

clients[n].addCargo(Cargo(weight, overallDimensions, orderFulfillmentDriver, car, dateOfUnloadingDay, dateOfUnloadingMonth, dateOfUnloadingYear, shippingDateDay, shippingDateMonth, shippingDateYear,

placeOfShipmentCountry, placeOfShipmentCity, placeOfShipmentStreet, placeOfShipmentHouse, placeOfUnloadinCountry, placeOfUnloadinCity, placeOfUnloadinStreet, placeOfUnloadinHouse));

}

n++;

}

}

client.close();

cargo.close();

}

//запись всех клиентов и их заказы в файл

void Company::writeAllClientsInFile(char\* file\_client, char\* file\_cargo)

{

ofstream f;

f.open(file\_client, ios::trunc);

f.close();

f.open(file\_cargo, ios::trunc);

f.close();

for (int y = 0; y < clients.size(); y++)

{

clients[y].writeClientInFile(file\_client);

for (int i = 0; i < clients[y].getShipments().size(); i++)

{

clients[y].getShipments()[i].writeCargoInFile(file\_cargo);

}

}

}

//запись клиента и его заказы в файл

void Client::writeClientInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::app);

if (!f) { cerr << "Невозможно открыть базовый файл\n"; exit(-1); }

f.seekp(0, ios\_base::end);

f << endl << this->firstname << "\t" << this->surname << "\t" << this->age << "\t" << this->autorisation << "\t" << this->status << "\t" << this->login << "\t" << this->password << "\t" << this->shipments.size();

if (!f) { cerr << "Запись в файл невозможна\n"; exit(-1); }

f.close();

}

//запись всех администраторов в файл

void Company::writeAllAdminsInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::trunc);

f.close();

for (auto admin : admins)

admin.writeAdminInFile(file\_name);

}

//запись администратора в файл

void Person::writeAdminInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::app);

if (!f) { cerr << "Невозможно открыть базовый файл\n"; exit(-1); }

f.seekp(0, ios\_base::end);

f << endl << this->firstname << "\t" << this->surname << "\t" << this->age << "\t" << this->autorisation << "\t" << this->status << "\t" << this->login << "\t" << this->password;

if (!f) { cerr << "Запись в файл невозможна\n"; exit(-1); }

f.close();

}

//запись всех водителей в файл

void Company::writeAllDriversInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::trunc | ios::binary);

f.close();

for (auto& driver : drivers)

driver.writeDriverInFile(file\_name);

}

//запись водителя в файл

void Worker::writeDriverInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::app);

if (!f) { cerr << "Невозможно открыть базовый файл\n"; exit(-1); }

f.seekp(0, ios\_base::end);

f << endl << this->firstname << "\t" << this->surname << "\t" << this->age << "\t" << this->autorisation << "\t" << this->status << "\t" << this->login << "\t" << this->password << "\t" << this->namber << "\t" << this->drivingExperience << "\t" << this->driverLicenseDate.getDay() << "\t " << this->driverLicenseDate.getMonth() << "\t" << this->driverLicenseDate.getYear() << "\t" << this->workerFree;

if (!f) { cerr << "Запись в файл невозможна\n"; exit(-1); }

f.close();

}

//запись все машин в файл

void Company::writeAllCarsInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::trunc | ios::binary);

f.close();

for (Car w : cars)

w.writeCarInFile(file\_name);

}

// запись машины в файл

void Car::writeCarInFile(char\* file\_name)

{

ofstream f;

f.open(file\_name, ios::app);

if (!f) { cerr << "Невозможно открыть базовый файл\n"; exit(-1); }

f.seekp(0, ios\_base::end);

f << endl << this->wheelbase << "\t" << this->mark << "\t" << this->model << "\t" << this->machineNumber << "\t" << this->number << "\t" << this->status;

if (!f) { cerr << "Запись в файл невозможна\n"; exit(-1); }

f.close();

}

//функция поиска администратора

void Company::findAdminAndShowAllInf()

{

int n;

string surname, name, login;

do {

menuFindFIOLogin();

cin >> n;

} while (n < 1 || n > 2);

switch (n)

{

case 1:

system("cls");

cout << "Имя: ";

cin >> name;

cout << "Фамилия: ";

cin >> surname;

findAdminAndShowAllInf(name, surname);

break;

case 2:

system("cls");

cout << "Логин: ";

cin >> login;

findAdminAndShowAllInf(login);

break;

}

}

//функция поиска администратора по ФИО

void Company::findAdminAndShowAllInf(string name, string surname)

{

int i = 0;

for (i = 0; i < admins.size(); i++)

if (admins[i].getFirstname() == name && admins[i].getSurname() == surname)

{

cout << admins[i];

system("pause");

return;

}

cout << "Данного администратора нет или данные неверные!" << endl;

}

//функция поиска администратора по логину

void Company::findAdminAndShowAllInf(string login)

{

int i = 0;

for (i = 0; i < admins.size(); i++)

if (admins[i].getLogin() == login)

{

cout << admins[i];

system("pause");

return;

}

cout << "Данного администратора нет или данные неверные!" << endl;

}

//функция поиска клиента

void Company::findClientrAndShowAllInf()

{

int n;

string surname, name, login;

do {

menuFindFIOLogin();

cin >> n;

} while (n < 1 || n > 2);

switch (n)

{

case 1:

system("cls");

cout << "Имя: ";

cin >> name;

cout << "Фамилия: ";

cin >> surname;

findClientrAndShowAllInf(name, surname);

break;

case 2:

system("cls");

cout << "Логин: ";

cin >> login;

findClientrAndShowAllInf(login);

break;

}

}

//функция поиска клиента по ФИО

void Company::findClientrAndShowAllInf(string name, string surname)

{

int i = 0;

for (i = 0; i < clients.size(); i++)

if (clients[i].getFirstname() == name && clients[i].getSurname() == surname)

{

cout << clients[i];

system("pause");

return;

}

cout << "Данного клиента нет или данные неверные!" << endl;

}

//функция поиска клиента по логину

void Company::findClientrAndShowAllInf(string login)

{

int i = 0;

for (i = 0; i < clients.size(); i++)

if (clients[i].getLogin() == login)

{

cout << clients[i];

system("pause");

return;

}

cout << "Данного клиента нет или данные неверные!" << endl;

}

//функция удаления клиента

void Company::delClient()

{

int n;

string surname, name, login;

do {

menuFindFIOLogin();

cin >> n;

} while (n < 1 || n > 2);

switch (n)

{

case 1:

system("cls");

cout << "Имя: ";

cin >> name;

cout << "Фамилия: ";

cin >> surname;

delClient(name, surname);

break;

case 2:

system("cls");

cout << "Логин: ";

cin >> login;

delClient(login);

break;

}

system("pause");

}

//функция удаления клиента по ФИО

void Company::delClient(string name, string surname)

{

for (int i = 0; i < clients.size(); i++)

if (clients[i].getFirstname() == name && clients[i].getSurname() == surname)

{

cout << "Клиент " << clients[i].getFirstname() << "" << clients[i].getSurname() << "(" << clients[i].getLogin() << ") удален!" << endl;

clients.erase(clients.begin() + i);

system("pause");

return;

}

cout << "Данного водителя нет или данные неверные!" << endl;

}

//функция удаления клиента по логину

void Company::delClient(string login)

{

for (int i = 0; i < clients.size(); i++)

if (clients[i].getLogin() == login)

{

cout << "Клиент " << clients[i].getFirstname() << "" << clients[i].getSurname() << "(" << clients[i].getLogin() << ") удален!" << endl;

clients.erase(clients.begin() + i);

system("pause");

return;

}

cout << "Данного водителя нет или данные неверные!" << endl;

}

void Company::delAdmin()

{

int n;

string surname, name, login;

do {

menuFindFIOLogin();

cin >> n;

} while (n < 1 || n > 2);

switch (n)

{

case 1:

system("cls");

cout << "Имя: ";

cin >> name;

cout << "Фамилия: ";

cin >> surname;

delAdmin(name, surname);

break;

case 2:

system("cls");

cout << "Логин: ";

cin >> login;

delAdmin(login);

break;

}

system("pause");

}

void Company::delAdmin(string login)

{

for (int i = 0; i < admins.size(); i++)

if (admins[i].getLogin() == login)

{

cout << "Администратор " << admins[i].getFirstname() << "" << admins[i].getSurname() << "(" << admins[i].getLogin() << ") удален!" << endl;

admins.erase(admins.begin() + i);

system("pause");

return;

}

cout << "Данного водителя нет или данные неверные!" << endl;

}

void Company::delAdmin(string name, string surname)

{

for (int i = 0; i < admins.size(); i++)

if (admins[i].getFirstname() == name && admins[i].getSurname() == surname)

{

cout << "Администратор " << admins[i].getFirstname() << "" << admins[i].getSurname() << "(" << admins[i].getLogin() << ") удален!" << endl;

admins.erase(admins.begin() + i);

system("pause");

return;

}

cout << "Данного водителя нет или данные неверные!" << endl;

}

# ПРИЛОЖЕНИЕ Б (обязательное)

**Диаграмма классов**

**(к пункту 4)**

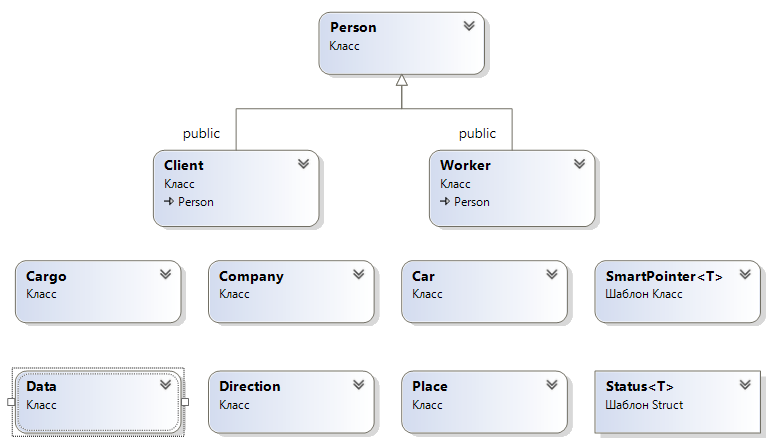
****

Рисунок Б1 – Диаграмма классов программы

# ПРИЛОЖЕНИЕ В

# (обязательное) Диаграмма вариантов использования (к пункту 5)

# 

Рисунок В.1 – Диаграмма вариантов использования

# ПРИЛОЖЕНИЕ Г

# (обязательное) Схема алгоритмов (к пункту 6)

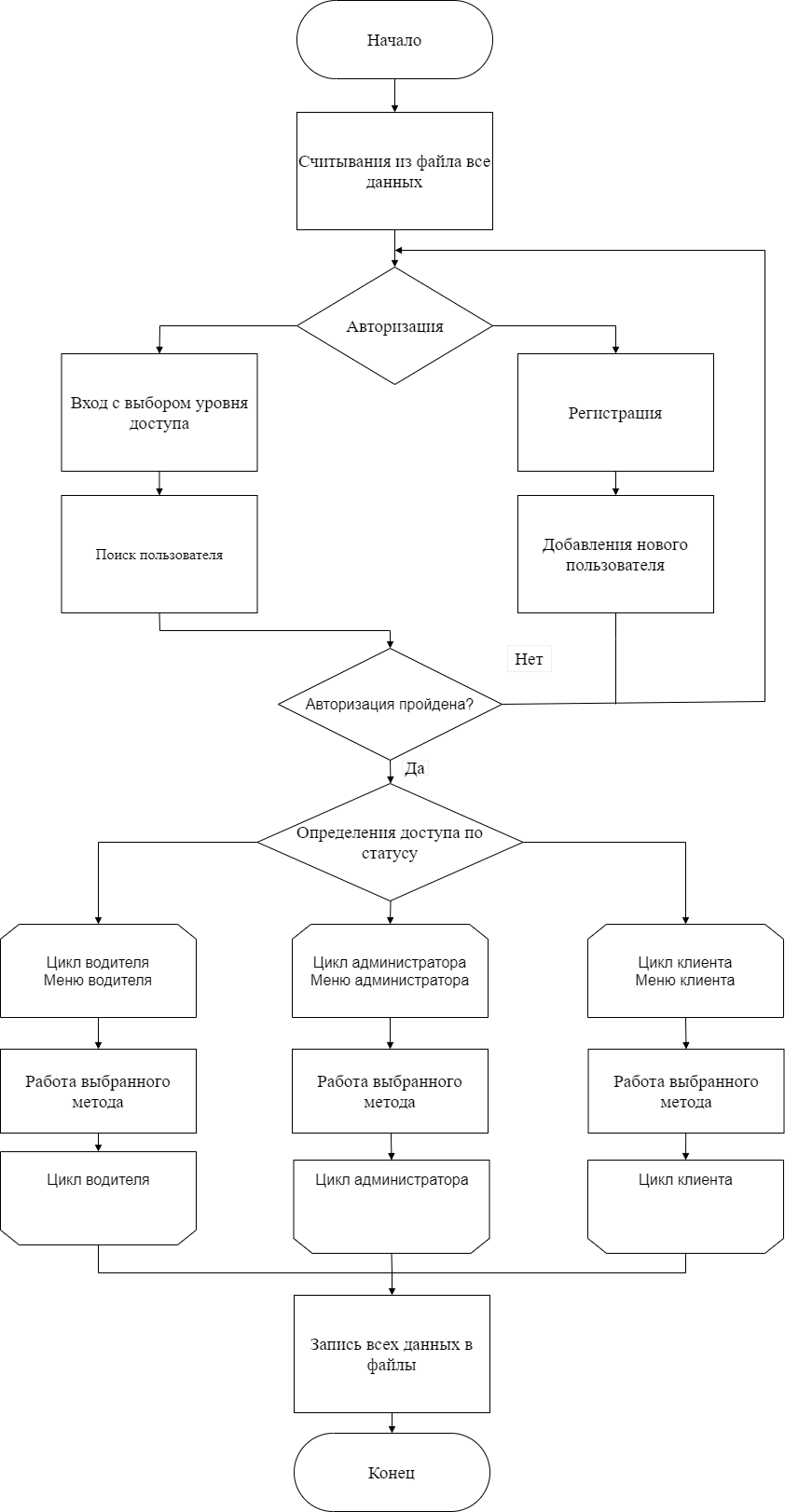


Рисунок Г1 – Блок-схема алгоритма работы программы

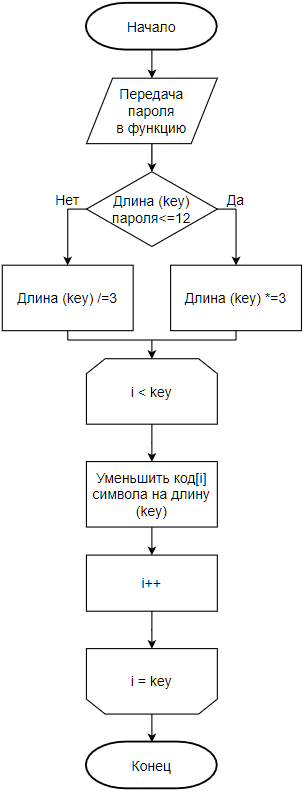


Рисунок Г2 – Блок-схема алгоритма шифрования пароля

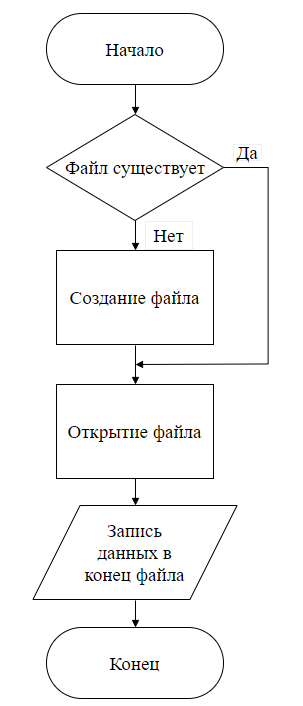


Рисунок Г3 – Блок-схема записи данных в файл

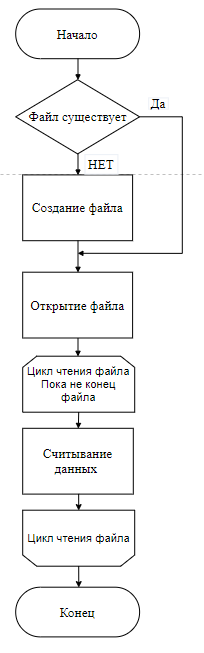


Рисунок Г4 – Блок-схема считывания данных из файла