МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматики и вычислительной техники

Кафедра АСУ

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Тема: Разработка фрагмента информационной системы автохозяйства, занимающегося междугородними грузовыми перевозками

Дисциплина «Базы данных», 5 семестр

Преподаватель: \_Эстрайх И.В.\_\_\_

Группа: \_\_\_\_АП-026\_\_\_\_

Студенты:

Почуев Н.А.

г. Новосибирск

2023 год

Оглавление

[ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ 1](#_Toc145458310)

[Реферат курсового проекта 3](#_Toc145458311)

[Введение 4](#_Toc145458312)

[Анализ предметной области 5](#_Toc145458313)

[Проектирование базы данных 7](#_Toc145458314)

[Конструктивный раздел 10](#_Toc145458315)

[Разработка эксплуатационной документации 12](#_Toc145458316)

[Заключение 13](#_Toc145458317)

[Приложение А ER-диаграмма 15](#_Toc145458318)

[Приложение Б Описание структуры таблиц 16](#_Toc145458319)

[Приложение В Примеры пользовательского интерфейса 17](#_Toc145458320)

[Приложение Г Созданные миграции 25](#_Toc145458321)

[Приложение Д Руководство пользователя 36](#_Toc145458322)

# Реферат курсового проекта

Курсовая работа по дисциплине «Базы данных» на тему «Разработка фрагмента информационной системы автохозяйства, занимающегося междугородними грузовыми перевозками» студента третьего курса Почуева Н.А.

Количество страниц – 40, рисунков – 20, таблиц – 2, количество используемых источников – 8, количество приложений – 5.

Ключевые слова, идентифицирующие предметную область – информационная система, базы данных, грузовые автоперевозки.

Объект исследования – информационная система грузовых автоперевозок.

Предмет исследования – программная реализация информационной системы грузовых автоперевозок.

Цель работы – разработать базу данных для поддержки деятельности автохозяйства, которое занимается междугородними грузовыми автоперевозками.

Задачи работы – проанализировать, как работают базы данных и сервисы грузовых перевозок, и на основе проделанного анализа спроектировать, разработать и протестировать программу-фрагмент информационной системы грузовых автоперевозок.

Методы исследования (используемые инструменты) – объектно-ориентированный язык программирования C#, технология пользовательского интерфейса Windows Presentation Foundation, среда разработки программного обеспечения Microsoft Visual Studio 2022, реляционная система управления базами данных – Microsoft SQL Server (MS SQL), PostgreSQL.

Краткая характеристика полученных результатов – работающая программа-фрагмент информационной системы грузовых автоперевозок, которая соответствует поставленным требованиями.

# Введение

Современное развитие технических и программных средств позволяет создавать базы данных высокого уровня с развитыми средствами анализа и высокой надежностью. Интуитивно понятный интерфейс приложения позволяет работать с базой данных даже сотрудникам, не обладающим глубокими знаниями компьютеров и программирования.

Современные СУБД позволяют создавать реляционные, распределенные базы данных, исключающие избыточность данных и обеспечивающие их целостность.

Информационная система (ИС) – это система, созданная для хранения, поиска и обработки информации. Информационные системы предназначены для обеспечения нужным людям необходимой информацией.

Актуальность выбранной темы обусловлена тем, что информационные системы для грузовых перевозок дают возможность рационализировать учет и реализацию заказов, анализировать потребности, оптимизировать перевозку грузов.

Объект исследования – информационная система грузовых автоперевозок.

Предмет исследования – программная реализация фрагмента информационной системы автохозяйства, занимающегося междугородними грузовыми перевозками.

Цель курсовой работы – разработать базу данных для поддержки деятельности автохозяйства, которое занимается междугородними грузовыми автоперевозками.

Задачи курсовой работы:

* проанализировать работу информационных систем и их связь с базами данных;
* проанализировать работу сервисов грузовых перевозок
* разработать информационно-логическую модель базы данных «Грузовые автоперевозки»
* разработать программу-фрагмент информационной системы грузовых автоперевозок;
* протестировать программу-фрагмент информационной системы грузовых автоперевозок.

Методы реализации (используемые инструменты) – объектно-ориентированный язык программирования C#, технология пользовательского интерфейса Windows Presentation Foundation, среда разработки программного обеспечения Microsoft Visual Studio 2022, реляционная система управления базами данных – Microsoft SQL Server (локальная база данных), PostgreSQL (удалённый сервер).

# Анализ предметной области

Необходимо разработать базу данных для поддержки деятельности автохозяйства, которое занимается междугородними грузовыми перевозками.

В БД должна присутствовать следующая информация: парк автомобилей, который представляет собой доступные машины и информацию о них, штат водителей, данные о клиенте, сами клиенты делятся на юридических и физических, данные о грузе, данные самого заказа, а также данные о рейсе заказа. Имея нужные права, пользователь должен иметь возможность читать, писать (добавлять новые записи, создавать новые документы и т.п.), редактировать и удалять нужные и дозволенные ему элементы таблиц базы данных.

Интерфейс должен иметь интуитивно понятное и простое в использовании меню, которое будет содержать все доступные операции с базой данных.

Входные данные программы-фрагмента сервиса грузовых автоперевозок представляют собой различные операции, которые пользователь может выполнять с базой данных. Среди них: получение списка всех заказов, поиск грузов по названию, заказу или единицам измерения, удаление записей из таблиц, добавление новых элементов в таблицу, редактирование элементов таблицы и т.д.

Выходные данные представляют собой таблицы после выполнения запросов пользователя. Требования к выходным данным: все ячейки таблиц должны быть заполнены.

Защита данных от несанкционированного доступа обеспечивается парольной защитой и контролем прав доступа. пароль хранится в зашифрованном виде. Используется алгоритм для шифрования пароля – MD5.

Пароль и логин задаются при регистрации пользователя, но пользователь может (а иногда и обязан) сменить пароль.

Пользователи, добавленные инициализатором для начала работы представлены в таблице 1.

Таблица 1 - Список пользователей

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя пользователя: | Пароль: | Right Name: | R | W | E | D |
| Admin | 1234 | Полный | True | True | True | True |
| User | 1234 | Просмотр | True | False | False | False |
| Nich | 12345 | Просмотр | True | False | False | False |

R, W, E, D – права пользователя на доступ и обработку данных: читать, писать (добавлять

новые записи, создавать новые документы и т.п.), редактировать записи и документы, удалять записи и документы.

Зарегистрированные новые пользователя изначально будут иметь Right.Name = Базовый, в котором по умолчанию доступ ко всему будет закрыт.

# Проектирование базы данных

Концептуальное представление данных в стандарте IDEF1X (или ER-диаграмма) представлена в приложении А. Описание структуры таблиц (реляционная модель) представлены приложении Б. Структуры таблиц в базе данных представлены в таблице 2.

Таблица 2 - Структуры таблиц в базе данных

|  |  |  |  |
| --- | --- | --- | --- |
| Название таблицы | Имя поля | Тип данных | Описание |
| Structure | Id | Счетчик | Данные, описывающие структуру меню. |
| ParentStructure | Числовой |
| DLLName | Текст |
| Function | Текст |
| Number | Числовой |
| Right | Id | Счетчик | Права пользователя, содержит название и булевые значения, обозначающие действия чтения, добавления и т.д. |
| Name | Текст |
| R | Логический |
| W | Логический |
| E | Логический |
| D | Логический |
| User | Id | Счетчик | Содержит данные пользователя – имя пользователя, фамилию, пароль, права. |
| Login | Текст |
| Surname | Текст |
| RightId | Числовой |
| Bank | Id | Счетчик | Содержит сведения о банках. |
| Name | Текст |
| Checking\_account | Id | Счетчик | Содержит сведения о расчетном счёте – счет и принадлежность к банку |
| Check | Текст |
| BankId | Числовой |
| Client | Id | Счетчик | Содержит информацию о клиенте - имя, фамилию, номер телефона. Если это физическое лицо, то также хранит серию и номер паспорта, дату получения и кем выдан. Если это юридическое лицо, то хранит руководителя, юридический адрес, расчетный счет и ИНН. |
| Name | Текст |
| PhoneNumber | Текст |
| IsPhysical | Логический |
| LegalPersonName | Текст |
| LegalAdress | Текст |
| Checking\_accountId | Числовой |
| Inn | Текст |
| SeriesAndNumberPass | Текст |
| DataOfIssue | Дата/Время |
| IssuedBy | Текст |
| Order | Id | Счетчик | Содержит сведения о заказе – дату и время заказа, клиента, адрес погрузки и разгрузки, длину маршрута, стоимость и рейс. |
| OrderData | Дата/Время |
| ClientId | Числовой |
| LoadingAddress | Текст |
| UnloadingAddress | Текст |
| RouteLength | Числовой |
| OrderCost | Числовой |
| FlightId | Числовой |
| Cargo | Id | Счетчик | Содержит информацию о грузе – название, количество, масса, единицы измерения, страховая стоимость, номер заказа. |
| Name | Текст |
| UnitId | Числовой |
| Amount | Числовой |
| Weight | Числовой |
| InsuranceValue | Числовой |
| OrderId | Числовой |
| Unit | Id | Счетчик | Содержит сведения о единицах измерений |
| Name | Текст |
| Flight | ArrivalDate | Дата/Время | Содержит сведения о рейсах – дата прибытия, экипаж, автомобиль. |
| CrewId | Числовой |
| AutomobileId | Числовой |
| Crew | Id | Счетчик | Содержит сведения о экипажах |
| Name | Текст |
| Driver | Id | Счетчик | Содержит сведения о водителях – имя, фамилия, экипаж, дата рождения, стаж, категория, классность. |
| Name | Текст |
| Surname | Текст |
| CrewId | Числовой |
| YearOfBirth | Дата/Время |
| WorkExperience | Текст |
| CategoryId | Числовой |
| ClassId | Числовой |
| DriverCategory | Id | Счетчик | Содержит информацию о категориях водителей |
| Name | Текст |
| DriverClass | Id | Счетчик | Содержит информацию о классах водителей |
| Name | Текст |
| Automobile | Id | Счетчик | Содержит сведения об автомобилях – название, государственный номер, бренд, грузоподъемность, предназначение, год выпуска, год ремонта, пробег. |
| Name | Текст |
| GosNumber | Текст |
| BrandId | Числовой |
| LoadCapacity | Числовой |
| Purpose | Текст |
| YearOfIssue | Дата/Время |
| YearOfRepair | Дата/Время |
| Millage | Числовой |
| Brand | Id | Счетчик | Содержит сведения о брендах |
| Name | Текст |

# Конструктивный раздел

**Архитектура приложения**

Проект практически полностью выполнен с использованием паттерна mvvm, поэтому его архитектуру довольно просто описать. У нас есть модель, в которой описаны используемые в приложении данные, а также логика, непосредственно связанная с этими данными. В нашем случае, для локального подключения и тестирования использовалась Microsoft SQL Server для хранения и управления данными. Впоследствии для подключения к удалённом серверу, база данных - PostgreSQL. Для создания и обновления схемы базы данных были созданы миграции, которые позволяют изменять структуру базы данных в соответствии с изменениями в модели. Миграции позволяют нам поддерживать актуальность базы данных без необходимости вручную вносить изменения в бд.

View определяет визуальный интерфейс, через который пользователь и взаимодействует с приложением, окно авторизации и отображение всех таблиц.

А также viewmodel, которая связывает модель и представление через привязку данных, для автоматического обновления отображения данных используется INotifyPropertyChanged и подобный метод (его можно заменить на INotifyPropertyChanged и ничего не изменится, логика и причины использования у них одинаковые), также был использован шаблон ICommand для реализации команд, которые связаны с действиями пользователя, такими как вход в систему, выполнение запросов к базе данных и другие действия.

Проверка ввода и изменения данных происходит с помощью Fluent Validation.

**Приложение**

Пример концептуальной схемы базы данных представлен на рисунке 1.

Пример логической схемы базы данных представлен на рисунке 2.

Пример окна авторизации представлен на рисунке 6.

Интерфейс приложения позволяет пользователям войти в систему, просматривать, добавлять, редактировать и удалять данные из базы данных. Вход в систему происходит с использованием логина и пароля, которые хранятся в отдельной таблице в базе данных. Пароль шифруется и сравнивается с зашифрованным паролем в базе данных перед предоставлением доступа.

Пример окна регистрации представлен на рисунке 3.

Пример регистрации пользователя представлен на рисунке 4.

Пример неудачной регистрации пользователя на рисунке 5.

Пользователи могут регистрироваться, если это нужно, через программу, вводя номер телефона, логин и пароль, все поля данных необходимо заполнить, иначе произойдёт ошибка.

Пример проверки регистра авторизации на рисунке 6.

Пример авторизации нового пользователя представлен на рисунке 7.

Зарегистрированные пользователи получают входа, а также базовые права доступа. Права содержат информацию о доступе пользователя к различным таблицам базы данных, включая права на чтение, добавление, редактирование и удаление записей.

Пример авторизации пользователя с правами представлен на рисунке 8.

Права определяют доступные возможности приложения.

Структура меню представлена на рисунке 18.

Из базы данных подгружается структура меню, с помощью которой можно обращаться к разным вкладкам и выполнять команды.

В главном меню программы отображаются названия таблиц базы данных. Пользователи могут выбрать таблицу, чтобы просмотреть ее содержимое.

Пример открытия окна “Добавить” представлен на рисунке 11.

Пример ошибки добавления представлен на рисунке 12.

Пример демонстрации удачного добавления представлен на рисунке 13.

Пример редактирования записи в таблице представлен на рисунке 14.

Они также могут удалить, добавить или редактировать записи в таблице. При неправильном вводе данных запись не будет добавлена или отредактирована и выведется ошибка.

Пример представления окна “Справка” представлен на рисунке 9.

Небольшое руководство по использованию находится во вкладке “Справка”

Пример смены пароля представлен на рисунке 10.

У каждого пользователя есть возможность смены пароля.

Пример изначальной таблицы представлен на рисунке 16.

Пример редактирования и изменения другой таблицы представлен на рисунке 17.

Изменение, добавление, удаление элементов таблицы автоматически изменяют остальные связанные таблицы.

Пример использования поиска представлен на рисунке 15.

Пользователи могут выполнять поиск данных в таблице по различным критериям. После ввода запроса и нажатия кнопки "Поиск", таблица обновляется, отображая только записи, удовлетворяющие поисковому запросу.

Пример использования экспорта в Excel представлен на рисунке 19.

Пример использования экспорта в Word представлен на рисунке 20.

В целом, пользовательский интерфейс программы грузовых автоперевозок предоставляет удобный способ управления данными в базе данных, позволяя пользователям взаимодействовать с информацией и выполнять различные операции, такие как просмотр, добавление, редактирование, удаление, поиск записей и создание нужной документации (экспорт).

# Разработка эксплуатационной документации

Руководство пользователя находится в соответствующем окне программы «Справка»

Справочная система включает в себя следующие разделы:

* О программе
* Возможности приложения

Раздел «О программе» включает в себя краткое описание приложения.

Раздел «Возможности приложения» в полном объёме описывает все возможности приложения с инструкциями по использованию.

# Заключение

В заключение, разработка базы данных для автохозяйства, занимающегося грузовыми перевозками, является необходимой и важной задачей. Такая база данных позволит эффективно управлять парком автомобилей, штатом водителей, клиентами и заказами. Интуитивно понятный интерфейс и разнообразные операции с базой данных обеспечат удобство использования сервиса грузовых автоперевозок. Корректное заполнение всех ячеек таблицы и правильное использование типов данных будет гарантировать точность и надежность информации. Разработка такой базы данных поможет оптимизировать процессы управления и повысить эффективность работы автохозяйства. Возможность создания документации с помощью экспорта позволит пользователю использовать данные вне приложения для любых целей.

**Список использованной литературы:**

[1] **ГОСТ Р 7.0.99—2018** Система стандартов по информации, библиотечному и издательскому делу. Реферат и аннотация. Общие требования [Текст]. – Введ. 01.01.2019. - М.: Стандартинформ, 2018. – 19 с.

[2] **ГОСТ Р ИСО 9127-94**. Системы обработки информации. Документация пользователя и информация на упаковке для потребительских программных пакетов [Текст] - Введён 01.07.1995 Переизд. 09.2003. - М.: ИПК Издательство стандартов, 2003. - 8 с.

[3] **ГОСТ 19.701-90** Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения [Текст]. – Взамен ГОСТ 19.002-80, ГОСТ 19.003-80 - Введ. 01.01.1992. – Переизд. 01 2010 г. - М.: Стандартинформ, 2010. – 24 с.

[4] **Дэвид Сеппа.** Программирование на Microsoft ADO.NET 2.0. Мастер-класс [Текст]/ Д. Сеппа – СПб.: Питер 2007 – 784 c. - ISBN: 978-5-91180-686-6

[5] **Проектирование баз данных. СУБД Microsoft Access**; Учебное пособие для вузов [Текст]/ Гринченко Н.Н, Гусев Е.Н, Макаров Н.П [и др.]. – М.: Горячая линия-Телеков, 2004.-240 с. – ISBN 5-93517-193-7

[6] **Марк Дж. Прайс.** C# 8 и .NET Core. Разработка и оптимизация [Текст] [пер. с анлг] /Павлов А. – СПб.: Питер, 2021 – 816 c. - ISBN: 978-5-4461-1700-0

[7] **Мартин Р.** Принципы, паттерны и методики гибкой разработки на языке C# [Текст]/ Р. Мартин, М. Мартин - СПб.: Символ-Плюс, 2011. – 768 c. – ISBN 978-5-8459-1684-6.

[8] **Фленов М. E.** Библия C# 2-е издание, перераб. и доп. / [Текст] - СПб.: БХВ-Петербург, 2013. – 560 c. – ISBN 978-5-9775-0655-7.

# Приложение А ER-диаграмма

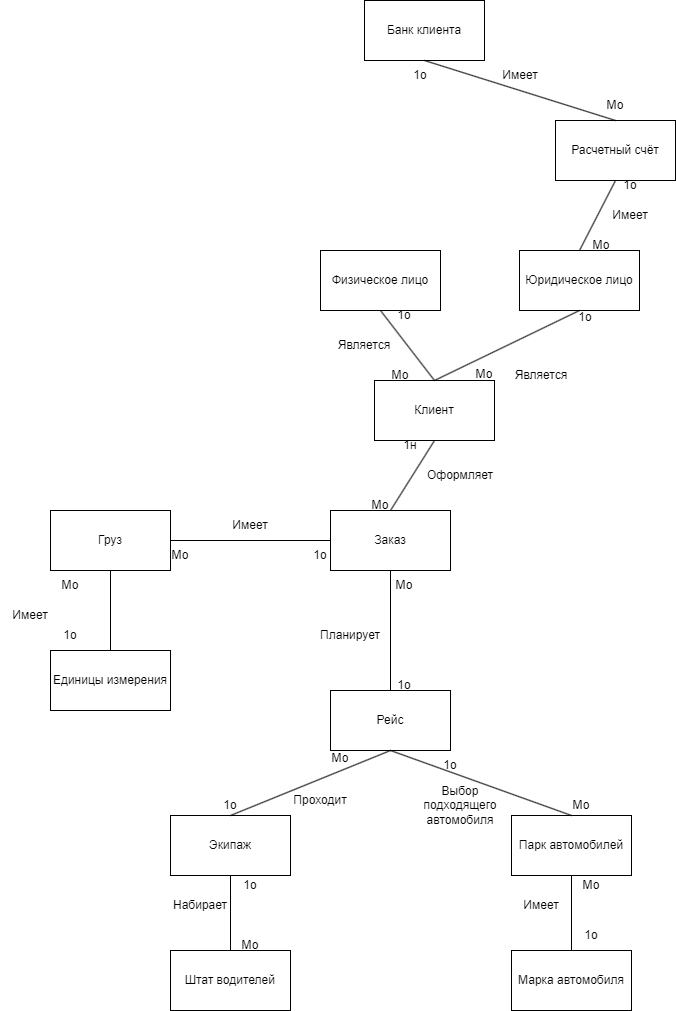


Рисунок 1 - Концептуальная схема базы данных

# Приложение Б Описание структуры таблиц на предварительном уровне

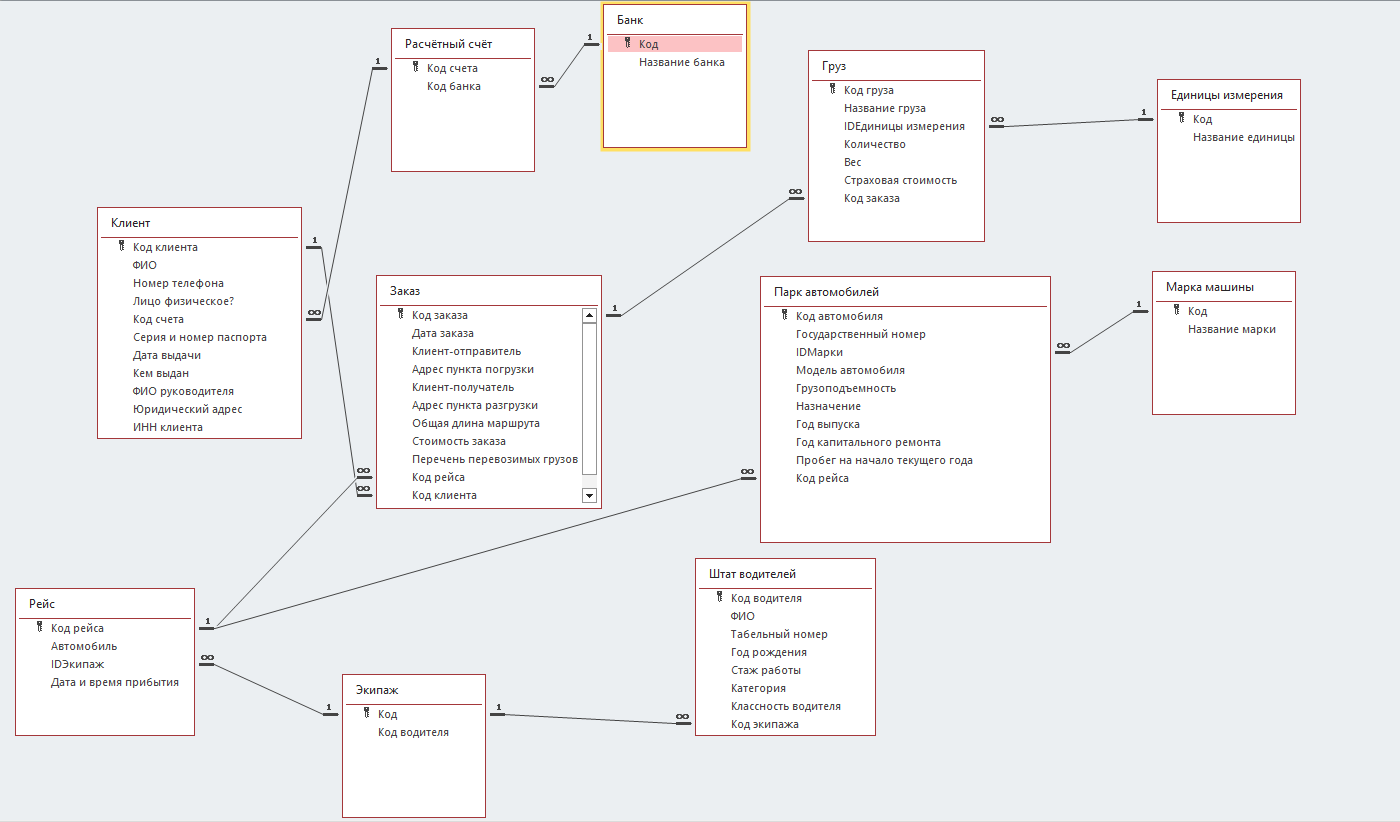


Рисунок 2 - Логическая схема базы данных

# Приложение В Примеры пользовательского интерфейса

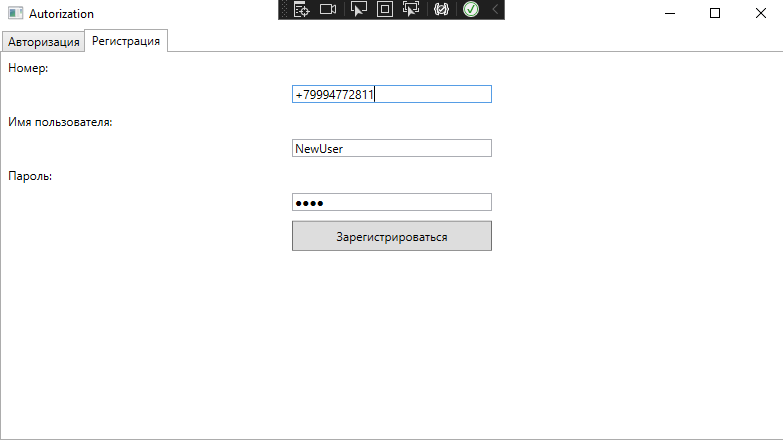


Рисунок 3 - Окно регистрации приложения

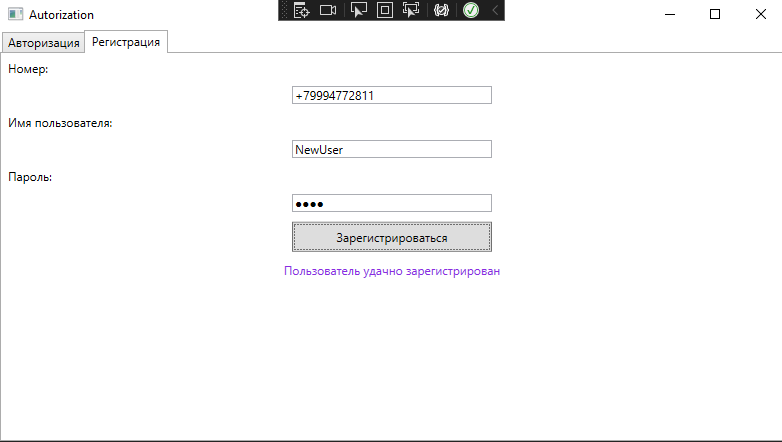


Рисунок 4 - Регистрация нового пользователя

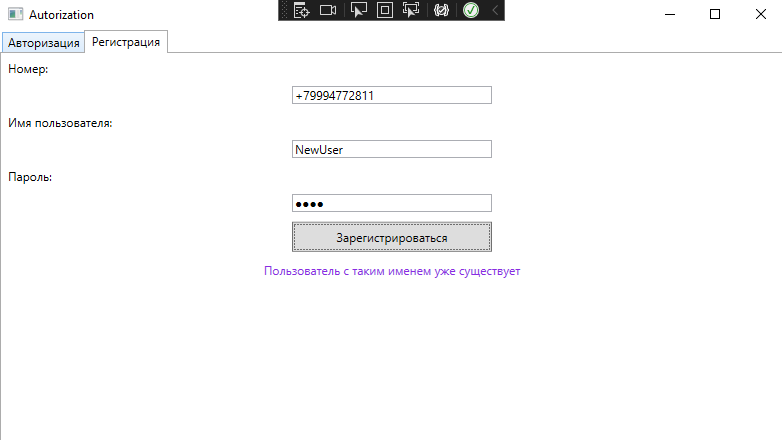


Рисунок 5 - Пример неудачной регистрации пользователя

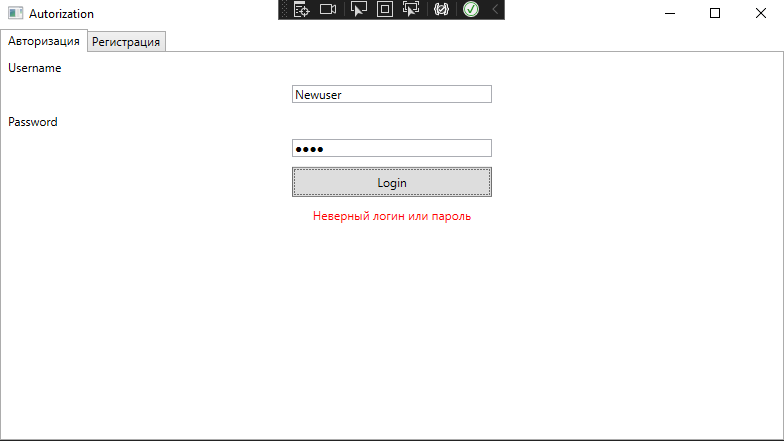


Рисунок 6 - Окно авторизации с проверкой регистра (зарегистрирован NewUser)

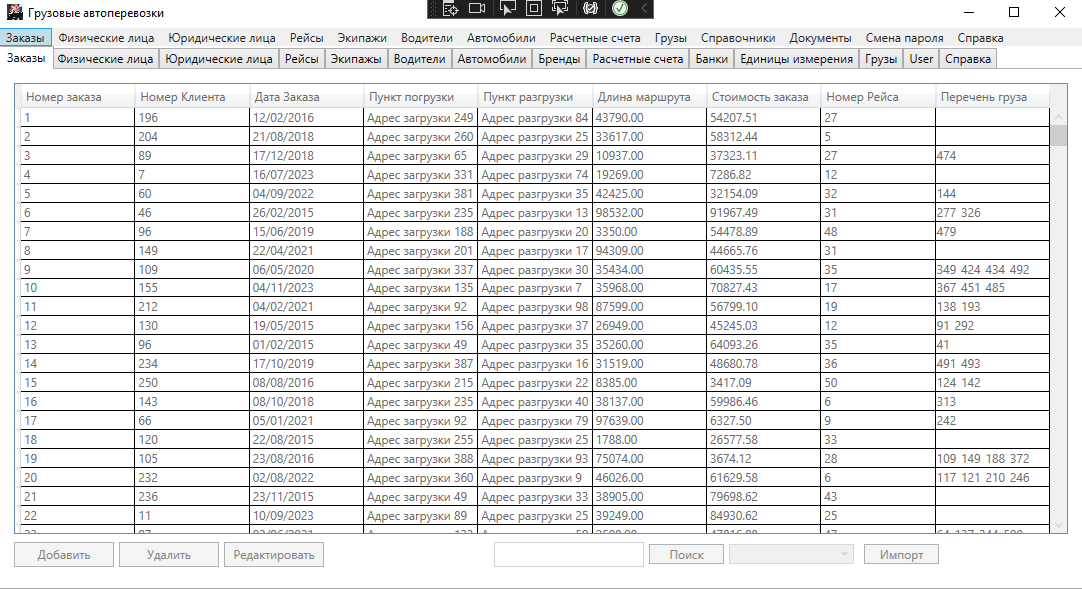


Рисунок 7 - Успешный вход на пользователе без прав

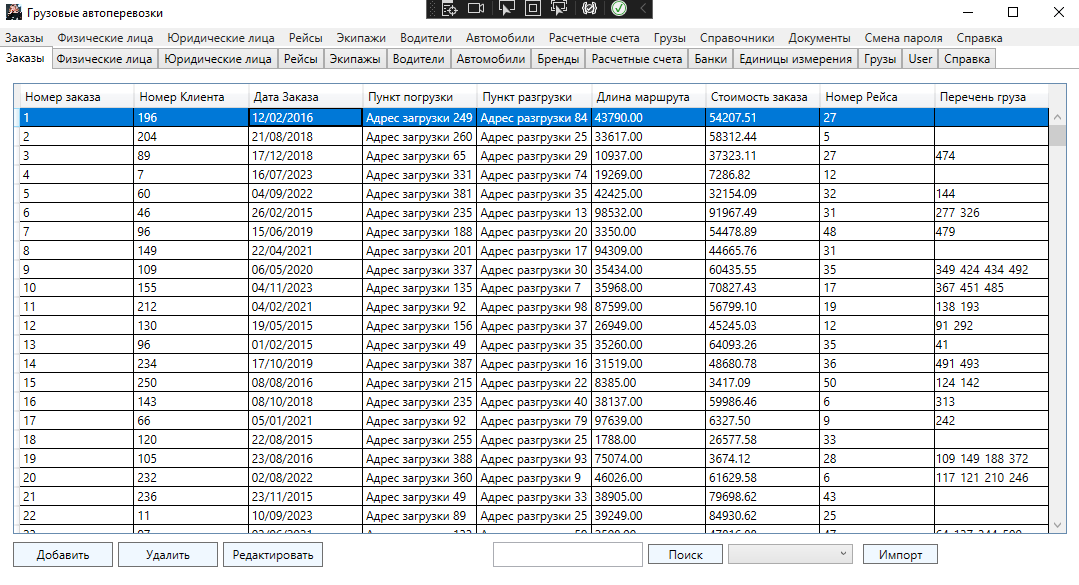


Рисунок 8 - Приложение у пользователя с правами

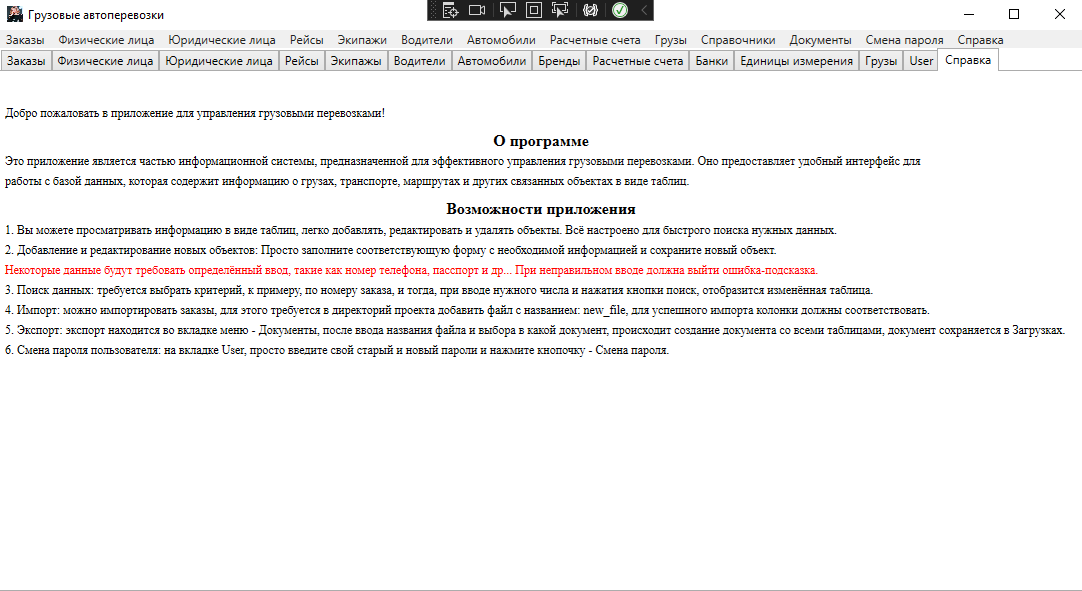


Рисунок 9 - Справка использования приложения

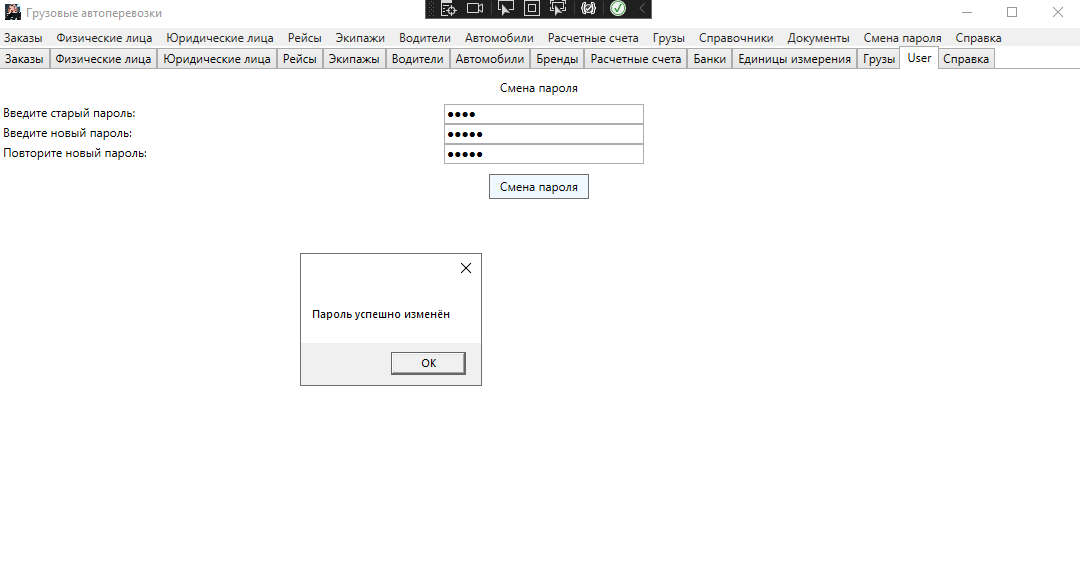


Рисунок 10 - Смена пароля пользователя



Рисунок 11 - Окно добавления нового заказа в таблицу

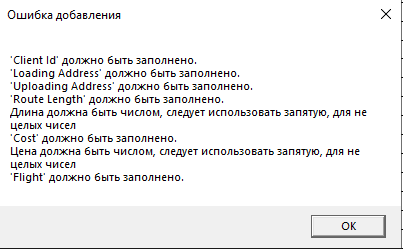


Рисунок 12 - Попытка добавления пустого заказа (результаты ошибок)

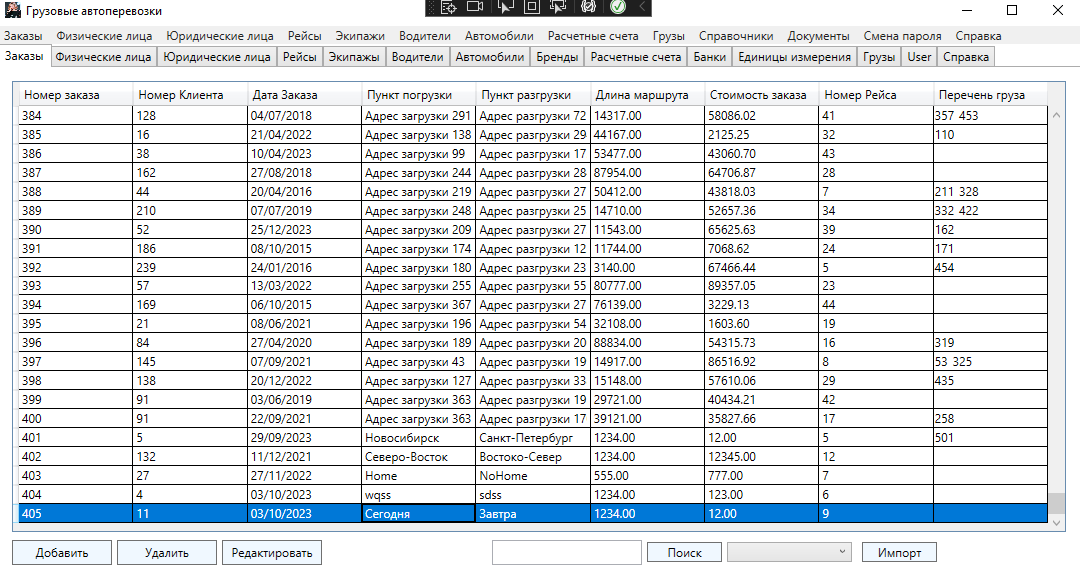


Рисунок 13 - Добавление нового заказа с учётом всех правил

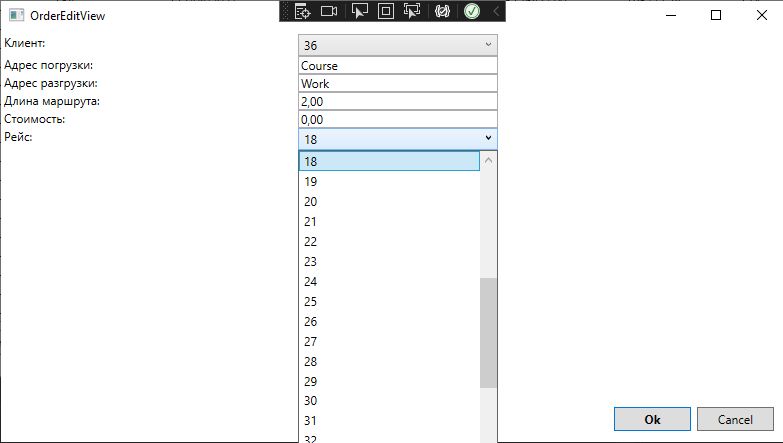


Рисунок 14 - Окно редактирования

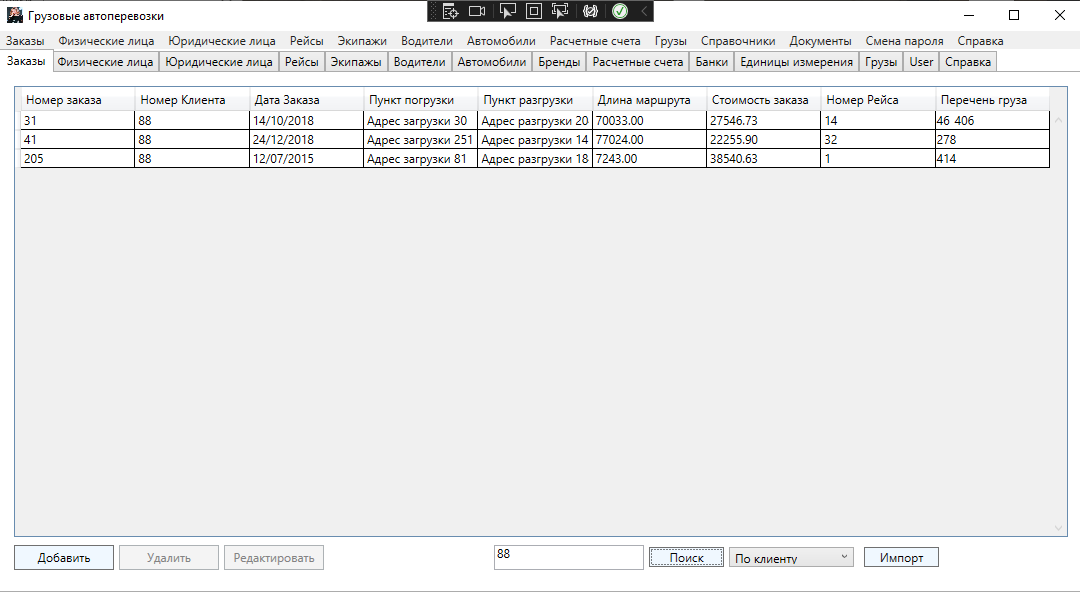


Рисунок 15 - Пример работы поиска по клиенту

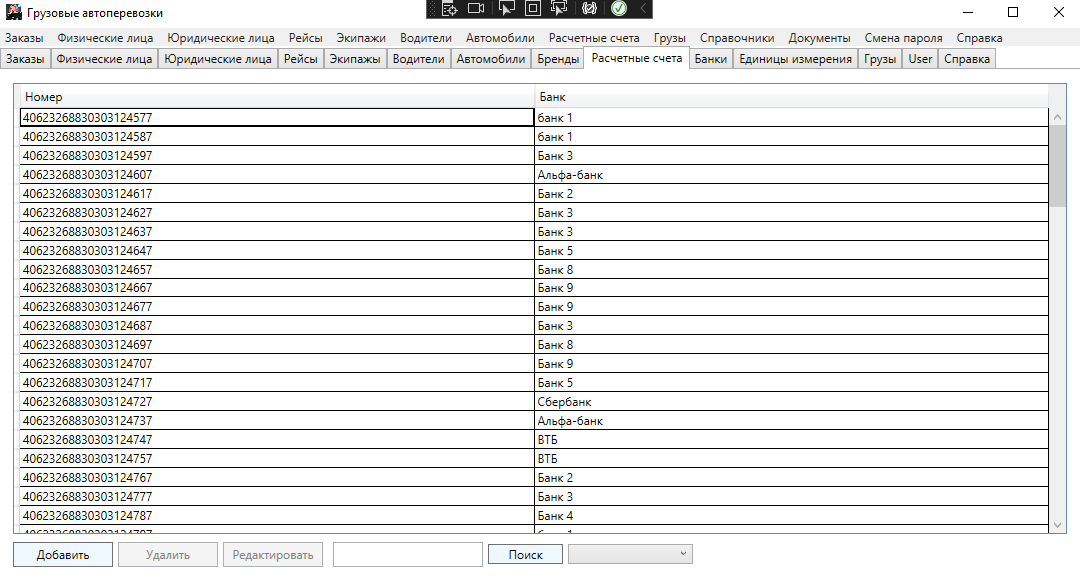


Рисунок 16 - Изначальная таблица расчетных счетов

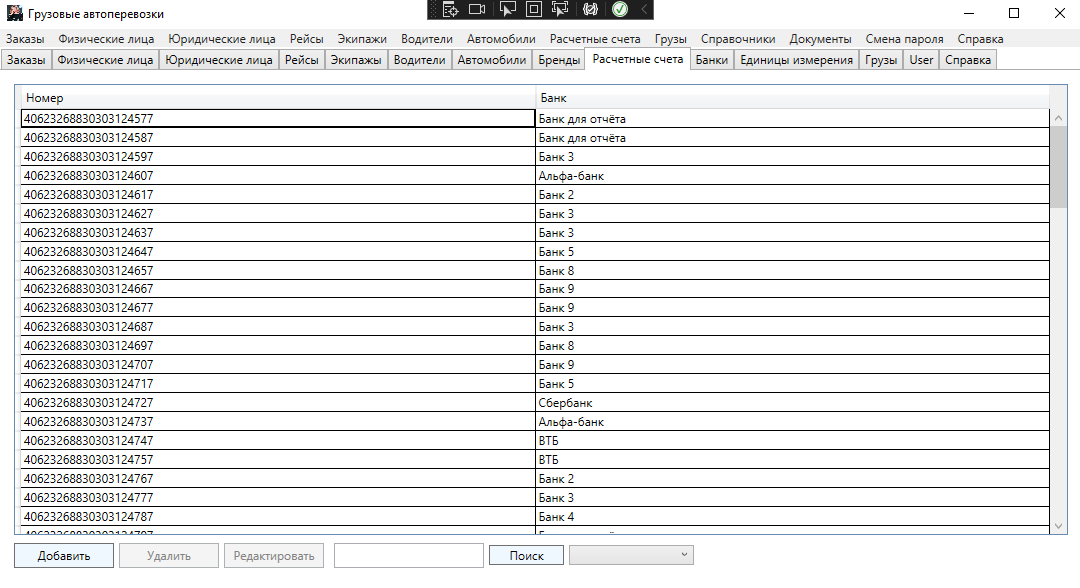


Рисунок 17 - Пример редактирования названия банка (изменения таблицы расчетных счетов)

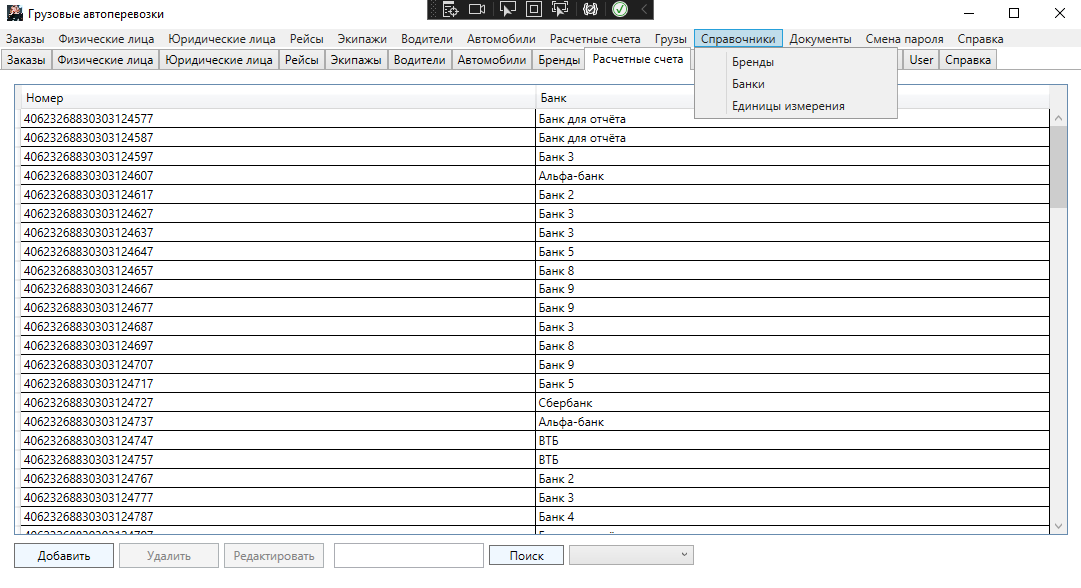


Рисунок 18 - Структура меню

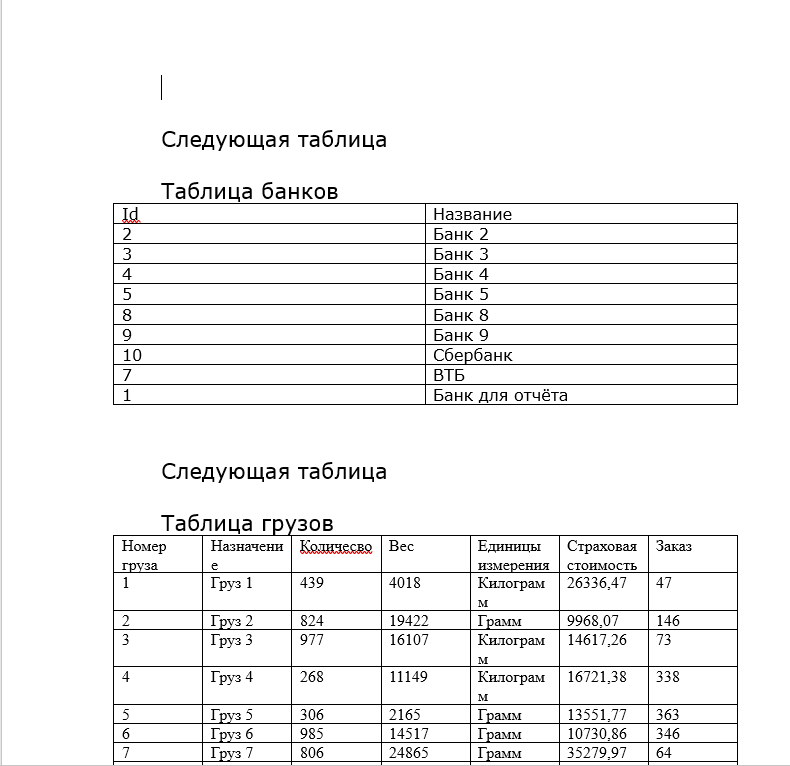


Рисунок 19 - Экспорт, создание документа Word

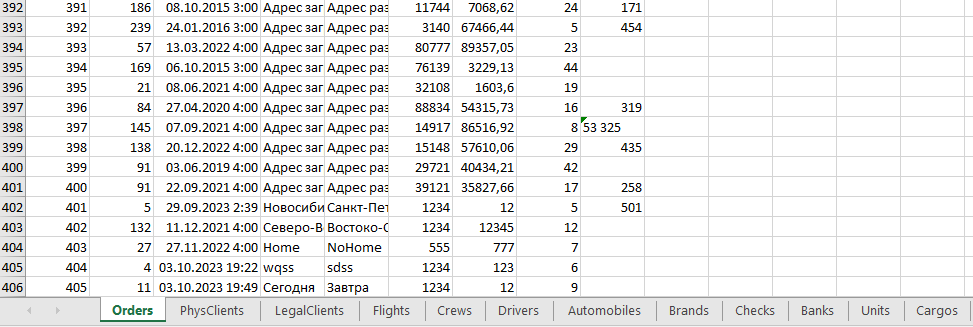


Рисунок 20 - Экспорт, создание документа Excel

# Приложение Г Созданные миграции

|  |
| --- |
| protected override void Up(MigrationBuilder migrationBuilder)  {  migrationBuilder.CreateTable(  name: "Banks",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Banks", x => x.Id);  });  migrationBuilder.CreateTable(  name: "Brands",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Brands", x => x.Id);  });  migrationBuilder.CreateTable(  name: "Crews",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Crews", x => x.Id);  });  migrationBuilder.CreateTable(  name: "DriverCategories",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_DriverCategories", x => x.Id);  });  migrationBuilder.CreateTable(  name: "DriverClasses",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_DriverClasses", x => x.Id);  });  migrationBuilder.CreateTable(  name: "Right",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  R = table.Column<bool>(type: "boolean", nullable: false),  W = table.Column<bool>(type: "boolean", nullable: false),  E = table.Column<bool>(type: "boolean", nullable: false),  D = table.Column<bool>(type: "boolean", nullable: false),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Right", x => x.Id);  });  migrationBuilder.CreateTable(  name: "Structures",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  ParentStructureId = table.Column<int>(type: "integer", nullable: true),  DLLName = table.Column<string>(type: "text", nullable: true),  function = table.Column<string>(type: "text", nullable: true),  number = table.Column<int>(type: "integer", nullable: false),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Structures", x => x.Id);  table.ForeignKey(  name: "FK\_Structures\_Structures\_ParentStructureId",  column: x => x.ParentStructureId,  principalTable: "Structures",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Units",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Units", x => x.Id);  });  migrationBuilder.CreateTable(  name: "Checking\_Accounts",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  BankId = table.Column<int>(type: "integer", nullable: true),  Check = table.Column<string>(type: "text", nullable: true)  },  constraints: table =>  {  table.PrimaryKey("PK\_Checking\_Accounts", x => x.Id);  table.ForeignKey(  name: "FK\_Checking\_Accounts\_Banks\_BankId",  column: x => x.BankId,  principalTable: "Banks",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Automobiles",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  GosNumber = table.Column<string>(type: "text", nullable: true),  BrandId = table.Column<int>(type: "integer", nullable: true),  LoadCapacity = table.Column<int>(type: "integer", nullable: false),  Purpose = table.Column<string>(type: "text", nullable: true),  YearOfIssue = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),  YearOfRepair = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),  Mileage = table.Column<int>(type: "integer", nullable: false),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Automobiles", x => x.Id);  table.ForeignKey(  name: "FK\_Automobiles\_Brands\_BrandId",  column: x => x.BrandId,  principalTable: "Brands",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Drivers",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  CrewId = table.Column<int>(type: "integer", nullable: true),  YearOfBirth = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),  WorkExperience = table.Column<string>(type: "text", nullable: true),  CategoryId = table.Column<int>(type: "integer", nullable: true),  ClassId = table.Column<int>(type: "integer", nullable: true),  Name = table.Column<string>(type: "text", nullable: false),  Surname = table.Column<string>(type: "text", nullable: true)  },  constraints: table =>  {  table.PrimaryKey("PK\_Drivers", x => x.Id);  table.ForeignKey(  name: "FK\_Drivers\_Crews\_CrewId",  column: x => x.CrewId,  principalTable: "Crews",  principalColumn: "Id");  table.ForeignKey(  name: "FK\_Drivers\_DriverCategories\_CategoryId",  column: x => x.CategoryId,  principalTable: "DriverCategories",  principalColumn: "Id");  table.ForeignKey(  name: "FK\_Drivers\_DriverClasses\_ClassId",  column: x => x.ClassId,  principalTable: "DriverClasses",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Users",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Password = table.Column<string>(type: "text", nullable: true),  RightId = table.Column<int>(type: "integer", nullable: true),  Name = table.Column<string>(type: "text", nullable: false),  Surname = table.Column<string>(type: "text", nullable: true)  },  constraints: table =>  {  table.PrimaryKey("PK\_Users", x => x.Id);  table.ForeignKey(  name: "FK\_Users\_Right\_RightId",  column: x => x.RightId,  principalTable: "Right",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Clients",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  Name = table.Column<string>(type: "text", nullable: false),  PhoneNumber = table.Column<string>(type: "text", nullable: true),  IsPhysical = table.Column<bool>(type: "boolean", nullable: false),  LegalPersonName = table.Column<string>(type: "text", nullable: true),  LegalAdress = table.Column<string>(type: "text", nullable: true),  Checking\_AccountId = table.Column<int>(type: "integer", nullable: true),  Inn = table.Column<string>(type: "text", nullable: true),  SeriesAndNumberPass = table.Column<string>(type: "text", nullable: true),  DataOfIssue = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),  IssuedBy = table.Column<string>(type: "text", nullable: true),  Surname = table.Column<string>(type: "text", nullable: true)  },  constraints: table =>  {  table.PrimaryKey("PK\_Clients", x => x.Id);  table.ForeignKey(  name: "FK\_Clients\_Checking\_Accounts\_Checking\_AccountId",  column: x => x.Checking\_AccountId,  principalTable: "Checking\_Accounts",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Flights",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  ArrivalDate = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),  CrewId = table.Column<int>(type: "integer", nullable: true),  AutomobileId = table.Column<int>(type: "integer", nullable: true)  },  constraints: table =>  {  table.PrimaryKey("PK\_Flights", x => x.Id);  table.ForeignKey(  name: "FK\_Flights\_Automobiles\_AutomobileId",  column: x => x.AutomobileId,  principalTable: "Automobiles",  principalColumn: "Id");  table.ForeignKey(  name: "FK\_Flights\_Crews\_CrewId",  column: x => x.CrewId,  principalTable: "Crews",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Orders",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  OrderData = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),  ClientId = table.Column<int>(type: "integer", nullable: true),  LoadingAddress = table.Column<string>(type: "text", nullable: true),  UnloadingAddress = table.Column<string>(type: "text", nullable: true),  RouteLength = table.Column<decimal>(type: "numeric(18,2)", nullable: false),  OrderCost = table.Column<decimal>(type: "numeric(18,2)", nullable: false),  FlightId = table.Column<int>(type: "integer", nullable: true)  },  constraints: table =>  {  table.PrimaryKey("PK\_Orders", x => x.Id);  table.ForeignKey(  name: "FK\_Orders\_Clients\_ClientId",  column: x => x.ClientId,  principalTable: "Clients",  principalColumn: "Id");  table.ForeignKey(  name: "FK\_Orders\_Flights\_FlightId",  column: x => x.FlightId,  principalTable: "Flights",  principalColumn: "Id");  });  migrationBuilder.CreateTable(  name: "Cargos",  columns: table => new  {  Id = table.Column<int>(type: "integer", nullable: false)  .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),  UnitId = table.Column<int>(type: "integer", nullable: true),  Amount = table.Column<int>(type: "integer", nullable: false),  Weight = table.Column<int>(type: "integer", nullable: false),  InsuranceValue = table.Column<decimal>(type: "numeric(18,2)", nullable: false),  OrderId = table.Column<int>(type: "integer", nullable: true),  Name = table.Column<string>(type: "text", nullable: false)  },  constraints: table =>  {  table.PrimaryKey("PK\_Cargos", x => x.Id);  table.ForeignKey(  name: "FK\_Cargos\_Orders\_OrderId",  column: x => x.OrderId,  principalTable: "Orders",  principalColumn: "Id");  table.ForeignKey(  name: "FK\_Cargos\_Units\_UnitId",  column: x => x.UnitId,  principalTable: "Units",  principalColumn: "Id");  });  migrationBuilder.CreateIndex(  name: "IX\_Automobiles\_BrandId",  table: "Automobiles",  column: "BrandId");  migrationBuilder.CreateIndex(  name: "IX\_Cargos\_OrderId",  table: "Cargos",  column: "OrderId");  migrationBuilder.CreateIndex(  name: "IX\_Cargos\_UnitId",  table: "Cargos",  column: "UnitId");  migrationBuilder.CreateIndex(  name: "IX\_Checking\_Accounts\_BankId",  table: "Checking\_Accounts",  column: "BankId");  migrationBuilder.CreateIndex(  name: "IX\_Clients\_Checking\_AccountId",  table: "Clients",  column: "Checking\_AccountId");  migrationBuilder.CreateIndex(  name: "IX\_Drivers\_CategoryId",  table: "Drivers",  column: "CategoryId");  migrationBuilder.CreateIndex(  name: "IX\_Drivers\_ClassId",  table: "Drivers",  column: "ClassId");  migrationBuilder.CreateIndex(  name: "IX\_Drivers\_CrewId",  table: "Drivers",  column: "CrewId");  migrationBuilder.CreateIndex(  name: "IX\_Flights\_AutomobileId",  table: "Flights",  column: "AutomobileId");  migrationBuilder.CreateIndex(  name: "IX\_Flights\_CrewId",  table: "Flights",  column: "CrewId");  migrationBuilder.CreateIndex(  name: "IX\_Orders\_ClientId",  table: "Orders",  column: "ClientId");  migrationBuilder.CreateIndex(  name: "IX\_Orders\_FlightId",  table: "Orders",  column: "FlightId");  migrationBuilder.CreateIndex(  name: "IX\_Structures\_ParentStructureId",  table: "Structures",  column: "ParentStructureId");  migrationBuilder.CreateIndex(  name: "IX\_Users\_RightId",  table: "Users",  column: "RightId");  }  /// <inheritdoc />  protected override void Down(MigrationBuilder migrationBuilder)  {  migrationBuilder.DropTable(  name: "Cargos");  migrationBuilder.DropTable(  name: "Drivers");  migrationBuilder.DropTable(  name: "Structures");  migrationBuilder.DropTable(  name: "Users");  migrationBuilder.DropTable(  name: "Orders");  migrationBuilder.DropTable(  name: "Units");  migrationBuilder.DropTable(  name: "DriverCategories");  migrationBuilder.DropTable(  name: "DriverClasses");  migrationBuilder.DropTable(  name: "Right");  migrationBuilder.DropTable(  name: "Clients");  migrationBuilder.DropTable(  name: "Flights");  migrationBuilder.DropTable(  name: "Checking\_Accounts");  migrationBuilder.DropTable(  name: "Automobiles");  migrationBuilder.DropTable(  name: "Crews");  migrationBuilder.DropTable(  name: "Banks");  migrationBuilder.DropTable(  name: "Brands");  }  } |

# Приложение Д Руководство пользователя

**Руководство пользователя**

Для начала работы с приложением, пользователю необходимо ввести свои учетные данные – логин и пароль. Если пользователь еще не зарегистрирован, он может нажать на кнопку "Регистрация" и ввести свой номер телефона, логин и пароль. После успешной регистрации, пользователь получит базовые права доступа.

После входа в систему, пользователь попадает на главное меню программы. Здесь отображаются названия таблиц базы данных. Пользователь может выбрать нужную таблицу, чтобы просмотреть ее содержимое.

Просмотр таблицы осуществляется путем прокручивания списка записей. Пользователь может также использовать поиск, чтобы найти определенную запись. Для этого необходимо ввести критерии поиска в соответствующие поля и нажать кнопку "Поиск". Таблица обновится, отображая только записи, удовлетворяющие поисковому запросу.

Если пользователь хочет добавить новую запись в таблицу, он должен нажать на кнопку "Добавить". Откроется форма, где пользователь может ввести информацию о новой записи. После заполнения полей, пользователь должен нажать кнопку "Сохранить", чтобы добавить запись в таблицу.

Если пользователь хочет изменить или удалить существующую запись, он должен выбрать нужную запись из таблицы и нажать на кнопку "Редактировать" или "Удалить". После этого откроется форма, где пользователь может внести изменения или подтвердить удаление.

Если пользователь хочет экспортировать базу данных в отдельный файл Excel или Word, нужно нажать на кнопку “Экспорт”, а также подтвердить это действие. После этого в загрузках создастся новый файл со всеми данными базы данных.

Также у пользователя есть возможность смены пароля.

Пользовательский интерфейс приложения для управления информационными ресурсами был специально адаптирован для удобства, скорости и безопасного способа взаимодействия с данными и информацией в организации.

Примеры использования приложения представлены в приложение B.

**О программе**

Данная программа является фрагментом информационной системы грузовых автоперевозок.

Лицензия: бесплатная программа для некоммерческого использования

Авторы:

Почуев Никита Александрович, 2023 г.