МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

Гомельский государственный технический университет имени П.О.Сухого

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальность 1-40 05 01-01 Информационные системы и технологии

(в проектировании и производстве)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

на тему «WPF приложение для учёта использования автотранспорта

предприятием»

Исполнитель: студентка гр. ИТП-21

Рожина А.К.

Руководитель: доцент

Курочка К.С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2023

СОДЕРЖАНИЕ

[Введение 3](#_Toc135797889)

[1 Описание средств разработки информационных систем 4](#_Toc135797890)

[1.1 Средства реализации приложения 4](#_Toc135797891)

[1.2 Основы объектно-ориентированного программирования 7](#_Toc135797892)

[1.3 Платформа *WPF* 9](#_Toc135797893)

[1.4 Слой для доступа к данным *ADO* 10](#_Toc135797894)

[1.5 Реализация приложения 12](#_Toc135797895)

[2 Архитектура разработанного приложения 13](#_Toc135797896)

[2.1 Проектирование архитектуры информационной системы 13](#_Toc135797897)

[2.2 Описание сущностей информационной системы 14](#_Toc135797898)

[2.3 Иерархическая схема интерфейса 15](#_Toc135797899)

[2.4 Общая структура классов 16](#_Toc135797900)

[2.5 Структура *XML* отчётов 18](#_Toc135797901)

[3 Этапы проведения верификации и тестирования информационной системы19](#_Toc135797902)

[3.1 Результаты проведения модульного тестирования 19](#_Toc135797903)

[3.2 Результаты проведения функционального тестирования 20](#_Toc135797904)

[Заключение 29](#_Toc135797905)

[Список использованных источников 30](#_Toc135797906)

[Приложение А Функциональная схема приложения 31](#_Toc135797907)

[Приложение Б Листинг программы 32](#_Toc135797908)

[Приложение В Листинг модульных тестов 77](#_Toc135797909)

[Приложение Г Руководство пользователя 86](#_Toc135797910)

[Приложение Д Руководство системного программиста 95](#_Toc135797911)

[Приложение Е Руководство программиста 96](#_Toc135797912)

# ВВЕДЕНИЕ

В современном мире в условиях развития информационных технологий практически во всех сферах деятельности человека присутствуют программные продукты, позволяющие хранить и обрабатывать огромный объем информации.

На транспортных предприятиях и организациях, а также в автотранспортных подразделениях торговых, производственных и прочих предприятиях для осуществления перевозок используется большое количество автомобилей. Перевозки осуществляются как грузовым, так и пассажирским транспортом разной вместительности.

При этом образуется большое количество путевых листов, требующих обработки. Чем больше транспорта, тем больший объём информации приходится обрабатывать с целью реализации функции контроля эксплуатации транспорта, эффективного управления рейсами, получения достоверной информации о состоянии используемого транспорта, его готовности к выполнению поставленных задач, об использовании автотехники.

Для оказания помощи в решении задач, выполняемых различными сотрудниками транспортного предприятия – диспетчерами, операторами, логистами, начальниками транспортных цехов, экономистами, необходима разработка программного продукта, позволяющего управлять автопарком, грузовыми и пассажирскими автоперевозками.

Причём при разработке программы необходимо разграничить информацию во избежание потери и преднамеренного искажения информации, кполучения данных «не в те руки». Для этого реализуется многопользовательский доступ к системе: администратор, оператор, водитель.

Таким образом исходя из всего вышесказанного подтверждается актуальность работы по созданию приложения для учёта использования автотранспорта предприятием.

# 1 ОПИСАНИЕ СРЕДСТВ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ

# Средства реализации приложения

# 

Любая информационная система предназначена для хранения, поиска и обработки информации, а также для своевременного обеспечения ее пользователей надлежащей информацией.

Существуют различные методы реализации информационных систем, классифицируемые по степени автоматизации:

– ручные («без компьютера»);

– автоматические;

– автоматизированные.

Автоматические и автоматизированные системы предусматривают обязательное наличие в их составе аппаратно-программных средств, которым отводится главная роль по обработке информации. Основным отличием автоматизированных информационных систем является необходимость постоянного участия в процессе обработки информации человека. Именно этот класс систем соответствует современному представлению об информационных системах.

В данном курсовом проекте используется автоматизированный метод реализации информационной системы, а именно создается автоматизированный программный комплекс на платформе *.NET Framework*, написанный на языке программирования *C#* при помощи объектно-ориентированного подхода в среде разработки *Visual Studio 2022* с использованием шаблонов проектирования и слоя доступа к данным, который взаимодействует с СУБД *MS SQL Server.* Для реализации пользовательского интерфейса использована технология *WPF* (*Windows Presentation Foundation*).

*.NET Framework* – программная платформа, выпущенная компанией *Microsoft* в 2002 году. Основой платформы является общеязыковая среда исполнения *CLR (Common Language Runtime)*, которая подходит для разных языков программирования. Функциональные возможности *CLR* доступны в любых языках программирования, использующих эту среду. Платформа *.NET* *Framework* поддерживает множество языков программирования, таких как *C#*, *Visual Basic*, *Iron Python*, *C*, *C++* и прочие благодаря тому, что программа сначала переводится компилятором в единый для *.NET* байт-код *CIL* (*Common Intermediate Language*), затем код либо исполняется виртуальной машиной *CLR*, либо транслируется утилитой *NGen.exe* в исполняемый код для конкретного целевого процессора. Благодаря *CLR* встроенный в неё *JIT* (*Just–in–time*)компилятор преобразует промежуточный байт-код в машинные коды нужного процессора [1].

Язык программирования *С#* – это один из языков, который предназначен для написания приложений для программной платформы *.NET Framework* (одной из основных платформ разработки, развертывания и исполнения распределенных приложений, которая разработана компанией *Microsoft*). Язык *C#* разработан в 1998-2001 годах группой инженеров компании *Microsoft*.

Язык *С#* обладает рядом достоинств: статическая типизация, перегрузка операторов, свойства, делегаты, *LINQ*, обобщённые типы. Также является профессиональным языком программирования, предназначенным для решения широкого спектра задач. Задачами могут выступать разработка приложений без или с графическим интерфейсом на *Windows*, на *Linux*, создание игр на базе *Unity*, создание веб-сервера на *ASP.NET*.

*WPF* (*Windows* *Presentation* *Foundation*) – это часть экосистемы .*NET*, которая представляет собой подсистему построения графических пользовательских интерфейсов. Преимуществом *WPF* является использование *DirectX* для отрисовки интерфейса, благодаря чему можно создавать более сложные и красивые представления, так как используется аппаратное ускорение графического процессора [2].

*SQL* – один из формальных языков, то есть средство, с помощью которого компьютеру передаются инструкции, называемые программой. Программное обеспечение базы данных выполняет эту программу, написанную на языке *SQL*. Это значит, что СУБД выполняет те запросы, которые ей переданы, и отображает результаты их работы, в том числе какое-нибудь сообщение об ошибке. Надо сказать, что языки программирования, называемые также формальными языками, отличаются от языков общения, называемых неформальными или естественными языками, главным образом тем, что создаются под конкретную цель, полностью лишены двусмысленности, имеют весьма ограниченные словарный запас и гибкость. Таким образом, если результат от работы программы, на который рассчитывали при ее написании не получен, это произошло потому, что программа содержит какую-либо ошибку (логическую или синтаксическую – в последнем случае, скорее всего, будет выведено соответствующее сообщение, описывающее ошибку), а не потому, что компьютер неправильно понял инструкции, формализованные в виде программы (эта информация проясняет, почему отладка программ считается одной из основных задач программирования).

Изначально *SQL* был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

Со временем *SQL* усложнился, обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях *SQL* остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов [3].

*LINQ to SQL* – наименование, присвоенное *API*–интерфейсу *IQueryable<T>*, который позволяет запросам *LINQ* (*Language Integrated Query –* язык интегрированных запросов) работать с базой данных *Microsoft SQL Server.*

Аббревиатура *LINQ* обозначает целый набор технологий, создающих и использующих возможности интеграции запросов непосредственно в язык *C#*. Традиционно запросы к данным выражаются в виде простых строк без проверки типов при компиляции или поддержки *IntelliSense*. Кроме того, разработчику приходится изучать различные языки запросов для каждого типа источников данных: баз данных *SQL*, *XML*-документов, различных веб-служб и т.д. Технологии *LINQ* превращают запросы в удобную языковую конструкцию, которая применяется аналогично классам, методам и событиям.

Наиболее очевидной частью *LINQ* является интегрированное выражение запроса. Выражения запроса используют декларативный синтаксис запроса. С помощью синтаксиса запроса можно выполнять фильтрацию, упорядочение и группирование данных из источника данных, обходясь минимальным объемом программного кода. Одни и те же базовые выражения запроса позволяют одинаково легко получать и преобразовывать данные из баз данных *SQL*, наборов данных *ADO* .*NET*, *XML*-документов, *XML*-потоков и коллекций .*NET*.

*LINQ to SQL* представляет технологию доступа и управления реляционными данными. Данная технология позволяет составлять запросы к базам данных в удобной форме с помощью операторов *LINQ*, которые затем трансформируются в *sql*-выражения. Ключевыми объектами здесь являются сущности, которые хранятся в базе данных, контекст данных и запрос *LINQ*.

Проблемы, связанные с базами данных, многочисленны. Первая сложность в том, что нельзя программно взаимодействовать с базой данных на уровне естественного языка. Это приводит к синтаксическим ошибкам, которые не проявляются вплоть до момента запуска. Неправильные ссылки на поля базы данных тоже не обнаруживаются. Это может пагубно отразиться на программе, особенно если произойдет во время выполнения кода обработки ошибок. Нет ничего хуже, чем сбой механизма обработки ошибок из-за синтаксически неверного кода, который никогда не тестировался. Иногда это неизбежно из-за непредсказуемого поведения ошибки. Наличие кода базы данных, который не проверяется во время компиляции, определенно может привести к этой проблеме.

Вторая проблема связана с неудобством, которое вызвано различными типами данных, используемыми определенным доменом данных, например, разница между типами базы данных или типами *XML* и типами данных в языке, на котором написана программа. В частности, серьезные сложности могут вызывать типы времени и даты.

Вместо того, чтобы просто добавить больше классов и методов для постепенного восполнения этих недостатков, в *Microsoft* решили пойти на один шаг дальше в абстрагировании основ запросов данных из этих конкретных доменов данных. В результате появился *LINQ* – технология *Microsoft*, предназначенная для поддержки запросов к данным всех типов на уровне языка. Эти типы включают массивы и коллекции в памяти, базы данных, документы *XML* и многое другое.

## 1.2 Основы объектно-ориентированного программирования

Объектно-ориентированное программирование (*object-oriented programming*) (далее – ООП) основано на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного типа, использующая механизм пересылки сообщений и классы, организованные в иерархию наследования. ООП представляет собой эффективный подход к программированию.

Методика ООП неотделима от *С#*, и поэтому все программы на *C#* являются объектно-ориентированными хотя бы в самой малой степени. В связи с этим очень важно и полезно усвоить основополагающие принципы ООП, прежде чем приступать к написанию самой простой программы на *С#*. Методики программирования претерпели существенные изменения с момента изобретения компьютера, постепенно приспосабливаясь, главным образом, к повышению сложности программ. Когда, например, появились первые ЭВМ, программирование заключалось в ручном переключении на разные двоичные машинные команды с переднего пульта управления ЭВМ. Такой подход был вполне оправданным, поскольку программы состояли всего из нескольких сотен команд. Дальнейшее усложнение программ привело к разработке языка ассемблера, который давал программистам возможность работать с более сложными программами, используя символическое представление отдельных машинных команд. Постоянное усложнение программ вызвало потребность в разработке и внедрении в практику программирования таких языков высокого уровня, как, например, *FORTRAN* и *COBOL*, которые предоставляли программистам больше средств для того, чтобы как-то справиться с постоянно растущей сложностью программ. Но как только возможности этих первых языков программирования были полностью исчерпаны, появились разработки языков структурного программирования, в том числе и *С.*

Программа, реализованная с помощью объектно-ориентированного программирования, состоит из взаимодействующих объектов.

Объект ­– это модель или абстракция реальной сущности в программной системе.

При написании программы необходимо соблюдать основные принципы объектно-ориентированного программирования:

* абстракция;
* инкапсуляция;
* наследование;
* полиморфизм.

Абстракция – в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций. Таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

Инкапсуляция – свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Наследование – позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию, например, реализация класса может быть изменена в процессе наследования.

На этапе разработки информационных систем выполняются следующие основные действия:

– кодирование;

– первоначальное тестирование;

– разработка справочной системы.

Кодирование представляет собой процесс преобразования результатов проектирования в готовый программный продукт. При кодировании происходит описание составленной модели информационной системы средствами языка программирования и СУБД [3].

# 1.3 Платформа *WPF*

Платформа *Windows Presentation Foundation* (*WPF*) позволяет создавать клиентские приложения для настольных систем *Windows* с привлекательным пользовательским интерфейсом.

В основе *WPF* лежит независимый от разрешения векторный модуль визуализации, использующий возможности современного графического оборудования. Возможности этого модуля расширяются с помощью 11 комплексного набора функций разработки приложений, которые включают в себя язык *XAML*, элементы управления, привязку к данным, макет, двумерную и трехмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции. *WPF* является частью .*NET*.

Преимущества *WPF*:

* использование традиционных языков .*NET*-платформы - *C#* и *VB.NET* для создания логики приложения;

– возможность декларативного определения графического интерфейса с помощью специального языка разметки *XAML*, основанном на *xml* и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать *XAML* и *C#/VB.NET*;

– независимость от разрешения экрана: поскольку в *WPF* все элементы измеряются в независимых от устройства единицах, приложения на *WPF* легко масштабируются под разные экраны с разным разрешением;

– новые возможности, которых сложно было достичь в *WinForms*, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др;

– хорошее взаимодействие с *WinForms*, благодаря чему, например, в приложениях *WPF* можно использовать традиционные элементы управления из *WinForms*;

– богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое;

– аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2*D* или 3*D*, графикой или текстом, все компоненты приложения транслируются в объекты, понятные *Direct*3*D*, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной;

– создание приложений под множество *ОС* семейства *Windows* - от *Windows XP* до *Windows* 10.

Однако стоит учитывать, что по сравнению с приложениями на *WindowsForms* объем программ на *WPF* и потребление ими памяти в процессе работы в среднем несколько выше. Но это с лихвой компенсируется более широкими графическими возможностями и повышенной производительностью при отрисовке графики.

## 1.4 Слой для доступа к данным *ADO*

*ADO.NET (ActiveX Data Object* для *.NET)* – технология, предоставляющая доступ и управление данными, хранящимися в базе данных или других источниках (*Microsoft SQL Server, Microsoft Access, Microsoft Excel, Microsoft Outlook, Microsoft Exchange, Oracle, OLE DB, ODBC, XML*, текстовые файлы), основанных на платформе *.NET Framework*. В отличие от технологии *ADO*, которая была в основном предназначена для тесно связанных клиент-серверных систем, *ADO.NET* больше нацелена на автономную работу с помощью объектов *DataSet.* Объекты *DataSet* представляют локальные копии взаимосвязанных таблиц данных, каждая из которых содержит набор строк и столбцов. Объекты *DataSet* позволяют вызывающей сборке (наподобие веб-страницы или программы, выполняющейся на настольном компьютере) работать с содержимым *DataSet*, изменять его, не требуя подключения к источнику данных, и отправлять обратно блоки измененных данных для обработки с помощью соответствующего адаптера данных. Но, пожалуй, самое фундаментальное различие между классической *ADO* и *ADO.NET* состоит в том, что *ADO.NET* является управляемой кодовой библиотекой, и, значит, подчиняется тем же правилам, что и любая управляемая библиотека. Типы, составляющие *ADO.NET*, используют протокол управления памятью *CLR*, принадлежат к той же системе типов (классы, интерфейсы, перечисления, структуры и делегаты), и доступ к ним возможен с помощью любого языка *.NET.* Классы *ADO.NET* находятся в сборке *System.Data.dll.* По умолчанию в *ADO.NET* имеются следующие встроенные провайдеры:

* провайдер для *MS SQL* *Server;*
* провайдер для *OLE DB* (предоставляет доступ к некоторым старым версиям *MS SQL Server*, а также к БД *Access*, *DB2*, *MySQL* и *Oracle*);
* провайдер для *ODBC* (провайдер для тех источников данных, для которых нет своих провайдеров);
* провайдер для *Oracle;*
* провайдер *EntityClient* (провайдер данных для технологии *ORM Entity Framework*);
* провайдер для сервера *SQL Server Compact* 4.0.

Кроме этих провайдеров, которые являются встроенными, существует также множество других, предназначенных для различных баз данных.

Основные пространства имен, которые используются в *ADO.NET*:

* *System.Data*: определяет классы, интерфейсы, делегаты, которые реализуют архитектуру *ADO.NET*;
* *System.Data.Common*: содержит классы, общие для всех провайдеров *ADO.NET*;
* *System.Data.Design*: определяет классы, которые используются для создания своих собственных наборов данных;
* *System.Data.Odbc*: определяет функциональность провайдера данных для *ODBC*;
* *System.Data.OleDb*: определяет функциональность провайдера данных для *OLE DB*;
* *System.Data.Sql*: хранит классы, которые поддерживают специфичную для *SQL Server* функциональность;
* *System.Data.OracleClient*: определяет функциональность провайдера для баз данных *Oracle*;
* *System.Data.SqlClient*: определяет функциональность провайдера для баз данных *MS SQL Server*;
* *System.Data.SqlServerCe*: определяет функциональность провайдера для *SQL Server Compact* 4.0;
* *System.Data.SqlTypes*: содержит классы для типов данных *MS SQL Servera*;
* *Microsoft.SqlServer.Server*: хранит компоненты для взаимодействия *SQL Server* и среды *CLR.*

# 1.5 Реализация приложения

Для реализации информационной системы будет создан автоматизированный программный комплекс на платформе *.NET Framework* написанный на языке программирования *C#* при помощи объектно-ориентированного подхода в среде разработки *Visual Studio* 2022, с использованием шаблонов проектирования.

Язык *С#* обладает рядом достоинств: статическая типизация, перегрузка операторов, свойства, делегаты, *LINQ* (*Language Integrated Query* – язык интегрированных запросов), обобщённые типы. Также является профессиональным языком программирования, предназначенным для решения широкого спектра задач. Задачами могут выступать разработка приложений без или с графическим интерфейсом на *Windows*, на *Linux*, создание двумерных и трёхмерных игр на движке *Unity*, простая разработка веб-приложений на *ASP.NET*.

Для реализации пользовательского интерфейса будет использована технология *WPF Application* – часть экосистемы .*NET*, которая представляет собой подсистему построения графических пользовательских интерфейсов.

.*Net Framework* – технологии, поддерживающая создание и выполнение нового поколения приложений и веб-служб, которая разработана корпорацией *Microsoft* в 2002 году.

**2 АРХИТЕКТУРА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

# 2.1 Проектирование архитектуры информационной системы

Анализ характеристик программного обеспечения помогает понять требования и ожидания, связанные с его функциональными и техническими аспектами. Если приложение сталкивается с конкуренцией на быстро меняющемся рынке, необходимо быстро адаптировать бизнес-модель. Чтобы обеспечить легкое расширение функциональности, программное обеспечение должно быть организовано в виде блоков и быть легким в обслуживании. Ключевыми характеристиками являются качество работы, высокая отказоустойчивость, масштабируемость и надежность.

Приложение должно выполнять следующие функции:

– вывод данных в корректной форме;

– обработка данных, полученных от пользователя;

– получение/обновление данных из источника.

Исходя из вышеописанных функций, можно выделить три основные части приложения:

– пользовательский интерфейс. Это та часть приложения, с которой взаимодействует пользователь, реализуется в виде графического пользовательского интерфейса, отображающего разрешенные в зависимости от роли пользователя элементы интерфейса;

– бизнес-логика. Содержит набор компонентов, с помощью которых обрабатывает полученные от уровня представлений данные, а также реализует всю необходимую логику приложения;

– доступ к данным. содержит компоненты для доступа к данным, в качестве источника данных.

Схема архитектура разрабатываемого приложения представлена на рисунке 2.1



Рисунок 2.1 – Схема архитектуры разрабатываемого приложения

Каждый уровень приложения должен взаимодействовать со своим «соседом». Это позволить обеспечить масштабируемость приложения.

# 2.2 Описание сущностей информационной системы

Исходя из требований задания можно выделить следующие сущности:

– пользователь;

– путевой лист;

– транспорт;

– марка;

– тип транспорта;

– роли пользователя.

Как в системе необходимо реализовать многопользовательский доступ, необходимо разработать класс *UserRole*, который будет содержать следующие поля:

– уникальный идентификатор *Id*;

– наименование роли *Role*.

Для моделирования сущности «пользователь» выступает класс *User*, которые имеет следующие поля:

– уникальный идентификатор *Id*;

– имя пользователя *Username*;

– пароль пользователя *Password*;

– фамилия, имя и отчество *FIO*;

– роль пользователя в системе *UserRole*.

Так как по заданию необходимо реализовать три вида транспорта, то был разработан класс *TransportType* со следующими полями:

– уникальный идентификатор *Id*;

– тип транспорта *Type*.

Класс Mark отражает сущность «марка» и имеет следующие поля:

– уникальный идентификатор *Id*;

– наименование марки *MarkName*.

Отражением сущности «транспорт» является класс *Transport*, который имеет следующие поля:

– уникальный идентификатор *Id*;

– уникальный номер транспорта *TransportNumber*;

– максимально возможное количество пассажиров *MaxPassenger*;

– максимально возможное количество груза *MaxCargo*;

– тип транспорта *TransportType*;

– марка транспорта *Mark*.

Для описания сущности путевой лист был разработан класс *Route* со следующими полями:

– уникальный идентификатор *Id*;

– расстояние маршрута *Distance*;

– дата маршрута *Date*;

– количество перевозимого груза *Cargo*;

– количество перевозимых пассажиров *Passenger*;

– водитель *Driver*;

– транспорт *Transport*.

# 2.3 Иерархическая схема интерфейса

Основа пользовательского интерфейса базируется на том, что приложение должно иметь систему пользователей. Поэтому при проектировании интерфейса были разработаны окна с разным функционалом для администраторов, операторов и водителей. Все эти окна находятся в проекте *UI*. Иерархия окон интерфейса изображена на рисунке 2.2.

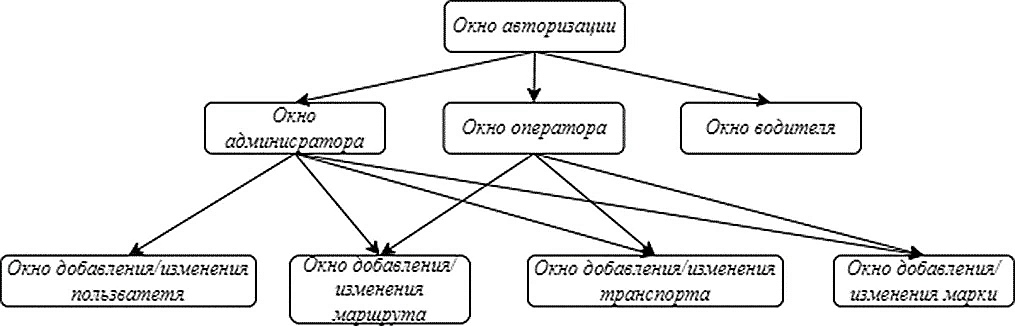


Рисунок 2.2 – Иерархия окон интерфейса

Окно авторизации отвечает за авторизацию пользователей. В нем находятся поля для ввода логина и пароля. Из этого окна, в зависимости от роли авторизированного пользователя, необходимо реализовать перенаправления на разные окна.

Водитель после авторизации должен быть перенаправлен в окно водителя*.* В нем водитель может просматривать свои путевые листы.

Оператор после авторизации будет перенаправлен в окно оператора. В этом окне должен быть добавлен функционал для просмотра путевых листов, а также информации о транспорте. Из этого окна необходимо реализовать переход в следующие окна:

– Окно добавления/изменения маршрута;

– Окно добавления/изменения транспорта;

– Окно добавления/изменения марки.

Эти окна должны позволять добавлять/изменять данные.

Администратора после авторизации необходимо перенаправить в окно администратора, в котором должны находится кнопки для перехода в следующие окна:

– Окно добавления/изменения пользователя;

– Окно добавления/изменения маршрута;

– Окно добавления/изменения транспорта;

– Окно добавления/изменения марки.

Программа заканчивает свою работу, когда все окна пользовательского интерфейса будут закрыты.

# 2.4 Общая структура классов

Приложение разделено на три проекта:

– библиотека классов, которая позволяет осуществлять доступ к базе данных. Данный проект также содержит классы бизнес-логики;

– проект, содержащий графические окна и элементы управления *WPF*.

Первый проект описывает основные сущности приложения. Библиотека также позволяет взаимодействовать с базой данных используя паттерн *Repository*. Для реализации паттерна был создан интерфейс *IRepository*, который содержит такие *CRUD* методы как: *Get, GetAll, Create, Update, Delete*. Бизнес-логика реализована при помощи классов-сервисов, которые позволяют определить правила взаимодействия пользователя с данными [6].

Структура классов первого проекта приведена на рисунке 2.3.

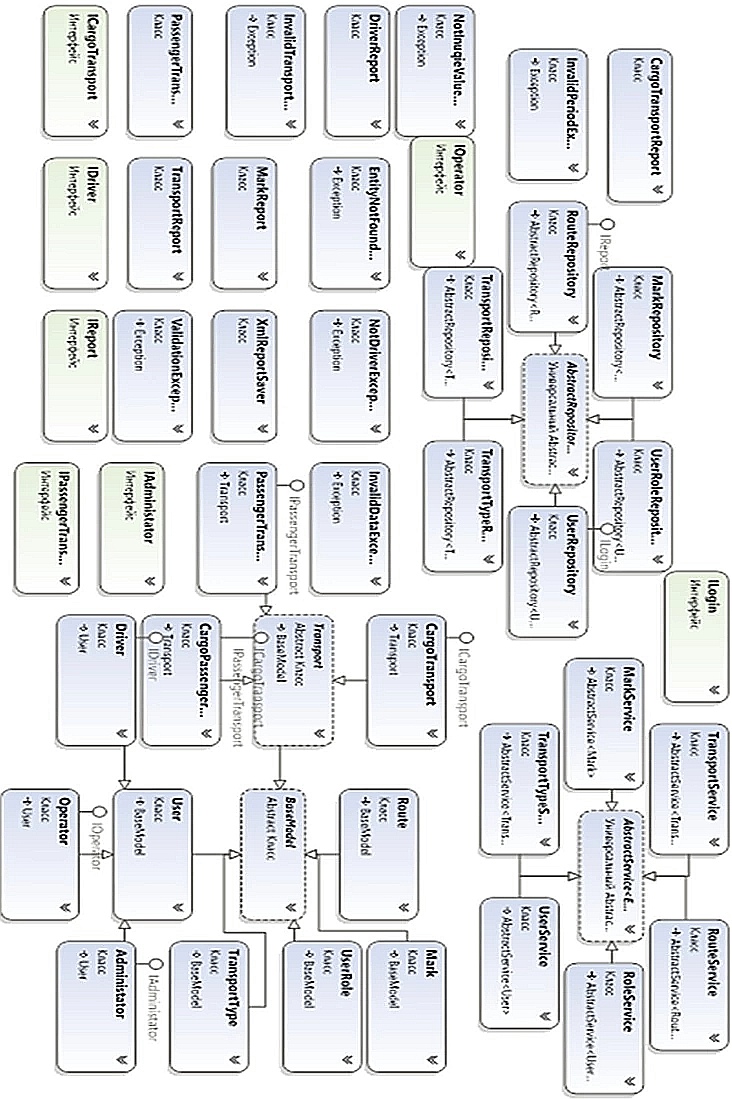


Рисунок 2.3 – Структура классов первого проекта

В папке *exceptions* хранятся классы-исключения. Папка *models* содержит классы-модели для работы с базой данных, а также для сохранения отчётов. Папка *repositories* хранит классы-репозитории для работы с базой данных. В папке *services* лежат классы-сервисы, которые реализуют бизнес-логику. Папка *xml* хранит класс, который реализует сохранения отчётов в *xml* файл.

Второй проект содержит в себе пользовательские элементы управления и все графические окна для всех видов пользователей. Структура классов второго проекта приведена на рисунке 2.4.



Рисунок 2.4 – Структура классов второго проекта

В папках *Admin, Driver, Operator* содержатся окна, которые доступы администратору, водителю и оператору соответственно. Класс *ConnectionStringUtils* содержит метод для извлечения строки подключения, которая хранится в файле *config.json*. Элементы схемы интерфейса и их реализация:

– окно авторизации – *LoginWindow.xaml*;

– окно администратора – *AdminWindow.xaml*;

– окно оператора – *OperatorWindow.xaml*;

– окно водителя – *DriverWindow.xaml*;

– окно добавления/изменения маршрута – *AddEditRouteWindow.xaml*;

– окно добавления/изменения пользователя – *AddEditUserWindow.xaml*;

– окно добавления/изменения транспорта – *AddEditTransportWindow.xaml*;

– окно добавления изменения марки – *AddEditMarkWindow*.

Листинг программы приведен в приложении Б.

Третий проект содержит классы модульных тестов.

# 2.5 Структура *XML* отчётов

Для сохранения отчётов в *XML* файл внутри проекта создана папка *reports*, которая имеет следующую структуру папок:

– *cargoTransport;*

*– drivers;*

*– marks;*

*– passengerTranport;*

*– transport.*

Внутри каждой папки создана подпапка, именем которой будет служить интервал, который был выбран при создании отчёта.

Пример *XML* отчётa приведён на рисункe 2.5.

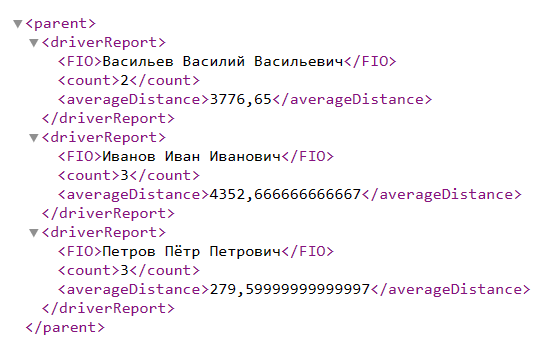


Рисунок 2.5 – Отчёт по водителям

1. **ЭТАПЫ ПРОВЕДЕНИЯ ВЕРИФИКАЦИИ И ТЕСТИРОВАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

# 3.1 Результаты проведения модульного тестирования

Одной из главных составляющих программы является ее тестирование на работоспособность.

Модульное тестирование, или юнит-тестирование (англ. *unit testing*) – это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования.

Тестирование проводится на уровне отдельных модулей программы, которые могут быть протестированы независимо друг от друга. Такой подход позволяет выявлять ошибки в работе программы на ранних этапах разработки и упрощает процесс отладки.

Концепция заключается в том, чтобы писать тесты для каждой нетривиальной функции или метода, что позволяет довольно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в ранее оттестированных местах программы, кроме того, облегчает обнаружение и устранение подобных ошибок.

Модульные тесты занимают миллисекунды, выполняются простым нажатием кнопки и не обязательно требуют знаний о всей системе в целом. Успешность прохождения теста зависит от средства выполнения теста, а не от пользователя [8].

Таким образом характеристиками хорошего модульного теста являются:

– быстрота;

– изолированность от внешних факторов, таких как файловая система или база данных;

– повторяемость;

– самопроверка без участия пользователя.

После реализации основного функционала приложения, проведено модульное тестирование. На рисунке 3.1 приведены результаты проведения модульного тестирования. Листинг модульных тестов приведен в приложении В.

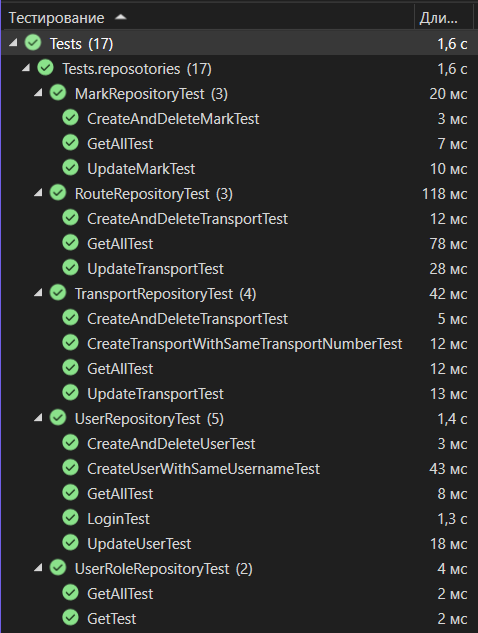


Рисунок 3.1 – Результат выполнения модульного тестирования

* 1. **Результаты проведения функционального тестирования**

Функциональное тестирование позволяет убедиться, что приложение правильно отображает данные, верно их изменяет и сохраняет. Такая проверка помогает найти слабые места и возможные ошибки, которые не выявило модульное тестирование.

Взаимодействие пользователя с приложением не должно вызывать ошибок или неполадок. Все непоследовательные действия пользователя должны быть предусмотрены и пользователю должно выводиться соответствующее сообщение о неверных действиях. Особое внимание уделено текстовым полям для ввода информации. Ввод некорректных данных проверяется и выводится соответствующее сообщение.

Для работы с программой необходимо открыть файл «*TransportUsage.sln*»*.* После появления оконного приложения пользователю предлагается ввести логин и пароль и авторизоваться в системе. Логин и пароль должны соответствовать параметрам заполнения, после этого проверяется их наличие в базе данных. Переход на следующую страницу происходит по нажатии на кнопку «*Enter*». Окно авторизации представлено на рисунке 3.2.

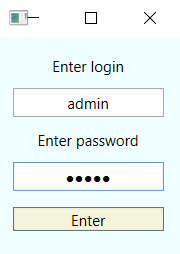


Рисунок 3.2 – Cостояние окна авторизации после ввода данных

Система предупреждает пользователя, если тот ввёл некорректные данные при входе. Пример результата некорректного ввода представлен на рисунке 3.3.

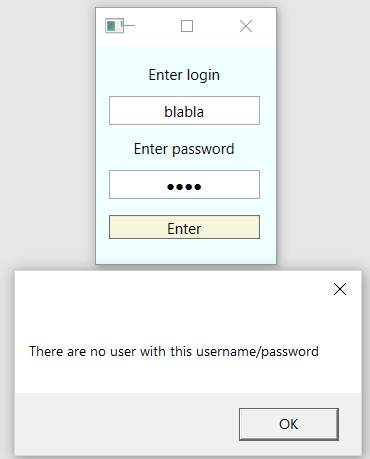


Рисунок 3.3 – Пример результата некорректного ввода данных

Если данные введены корректно, пользователь увидит сообщение, представленное на рисунке 3.4

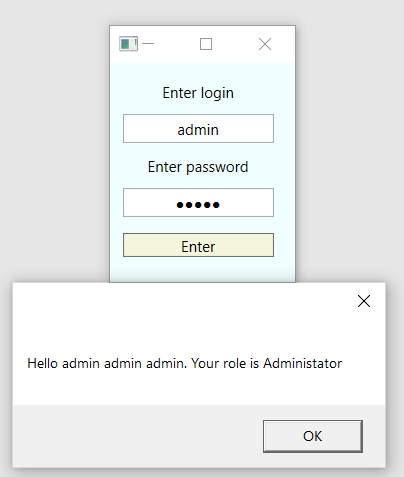


Рисунок 3.4 – Пример результата корректного ввода данных

После успешного входа в аккаунт пользователя в роли администратора, система перенаправляет его на вкладку главного окна. Администратор имеет полные права доступа: функции просмотра, добавления, редактирования и удаления всех пользователей системы, информации о маршрутах, транспорте и марках автомобилей, создание отчётов за произвольный период. Главное окно с функцией просмотра, добавления, редактирования и удаления пользователей бражено на рисунке 3.5.

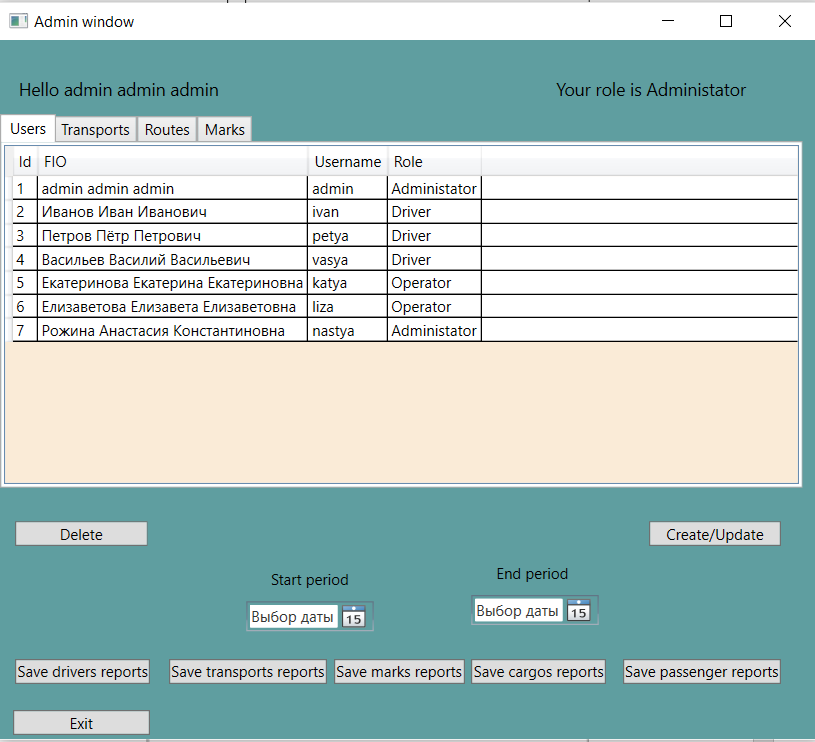


Рисунок 3.5 – Окно администратора для просмотра, добавления,

редактирования и удаления пользователей

Главное окно администратора с функцией просмотра, добавления, редактирования и удаления транспорта изображено на рисунке 3.6. Для перехода на вкладку транспорта необходимо нажать на кнопку *«Transport».*

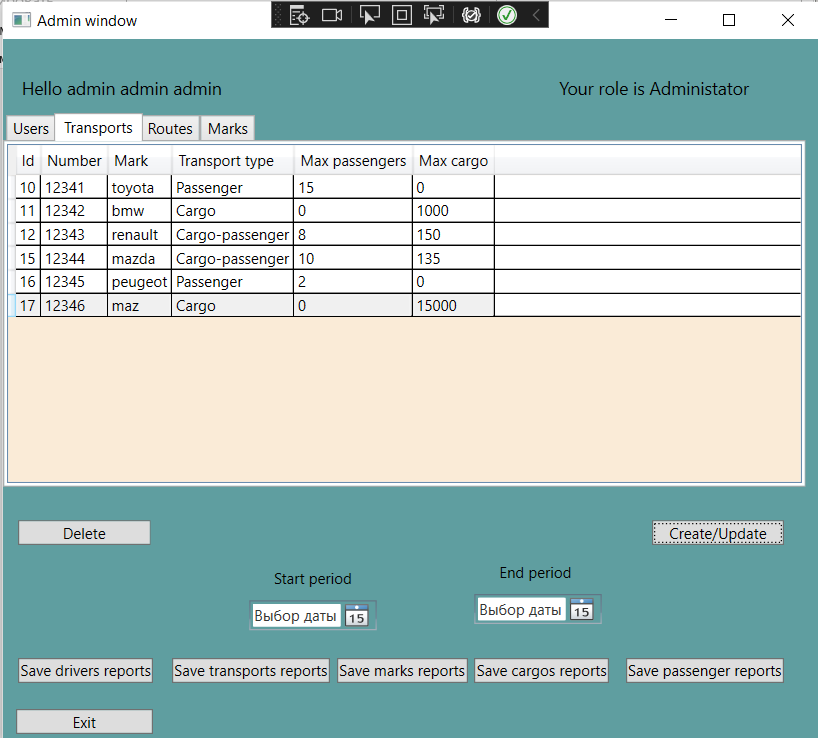


Рисунок 3.6 – Окно администратора для просмотра, добавления,

редактирования и удаления транспорта

Главное окно администратора с функцией просмотра, добавления, редактирования и удаления маршрутов изображено на рисунке 3.7. Для перехода на вкладку маршрутов необходимо нажать на кнопку *«Routes».*

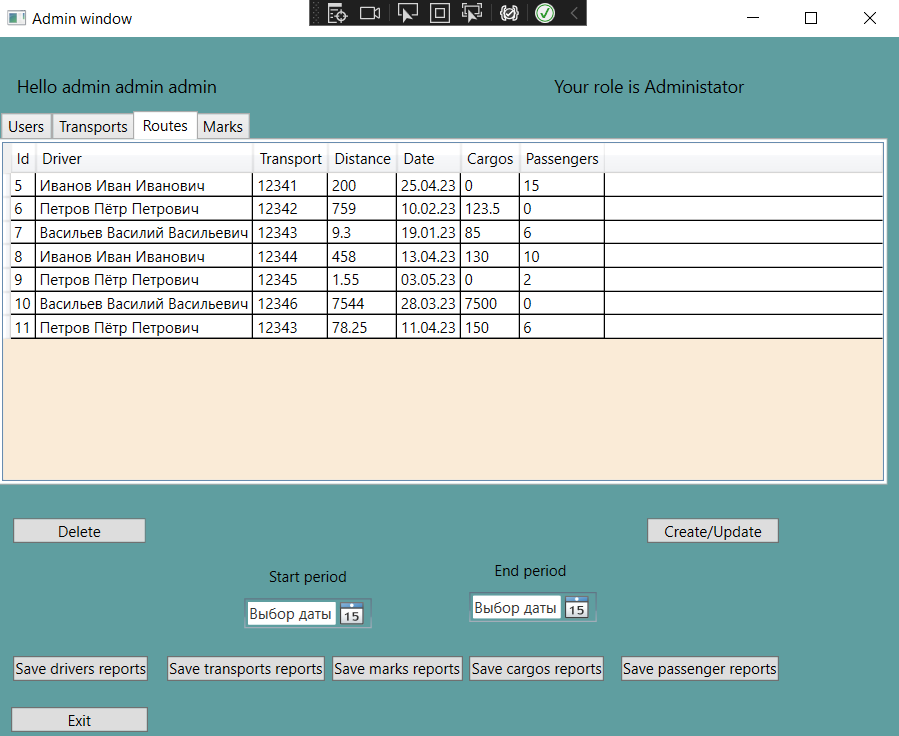


Рисунок 3.7 – Окно администратора для просмотра, добавления,

редактирования и удаления маршрутов

Главное окно администратора с функцией просмотра, добавления, редактирования и удаления марок автомобилей изображено на рисунке 3.8. Для перехода на вкладку маршрутов необходимо нажать на кнопку *«Marks».*

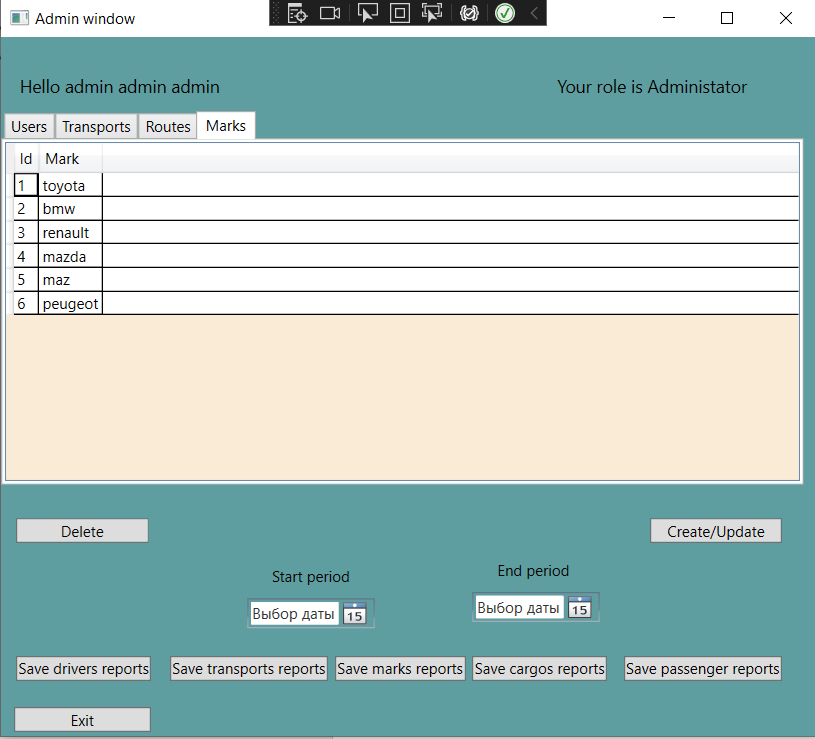


Рисунок 3.8 – Окно администратора для просмотра, добавления,

редактирования и удаления марок автомобилей

Пример корректного добвления информации на вкладку, где располагаются маршруты представлен на рисунке 3.9.

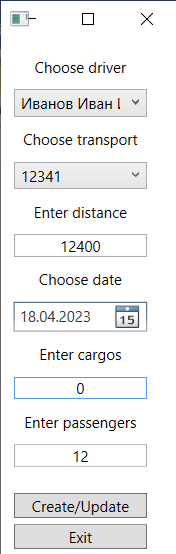


Рисунок 3.9 – Окно администратора для добавления маршрута

После добавления данных на вкладке маршрутов появится новая строка (рисунок 3.10).

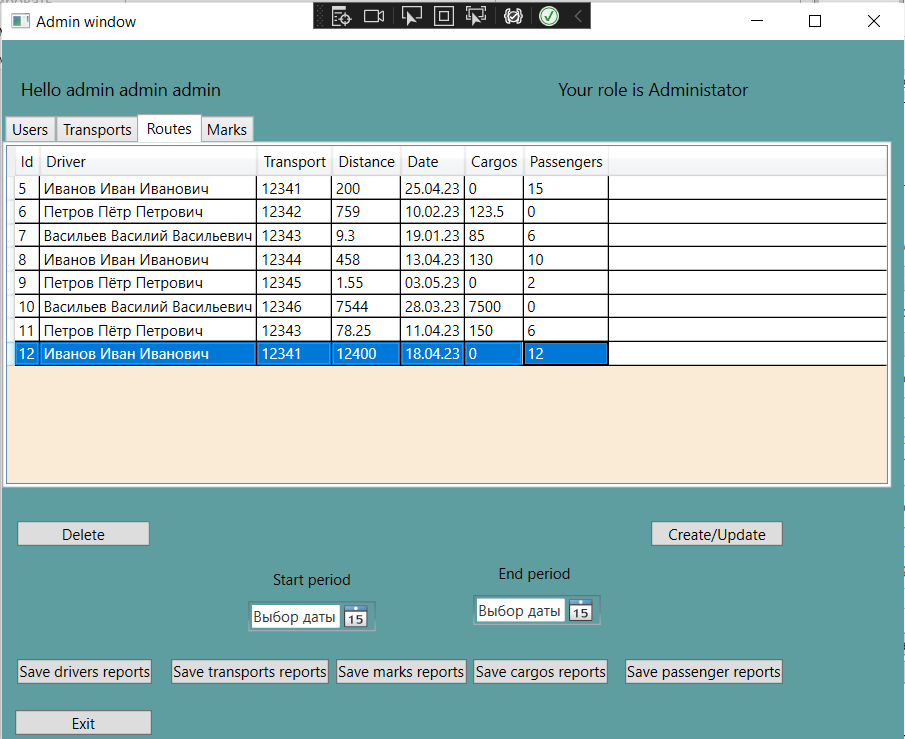


Рисунок 3.10 – Окно администратора с новым созданным маршрутом

Если транспорт уже используется в этот день, количество груза и пассажиров превышает заданное, либо пассажирскому транспорту будет присвоено значение грузов, а грузовому – количество пассажиров, то пользователь увидит сообщение об ошибке (рисунки 3.11 – 3.13).

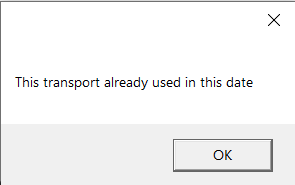


Рисунок 3.11 – Пример результата некорректного ввода даты

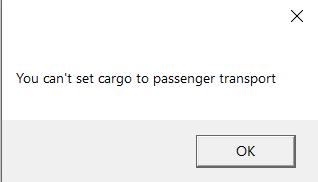


Рисунок 3.12 – Пример результата некорректного ввода количества груза

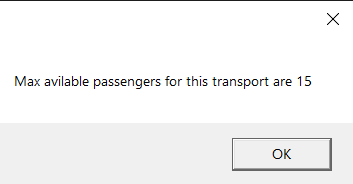


Рисунок 3.13 – Пример результата превышения количества пассажиров

В случае входа в аккаунт пользователя в роли оператора система также перенаправляет его на вкладку главного окна с доступными функциями (рисунок 3.13). Для оператора реализованы права доступа: функции просмотра, добавления, редактирования и удаления информации о маршрутах, транспорте и марках автомобилей.

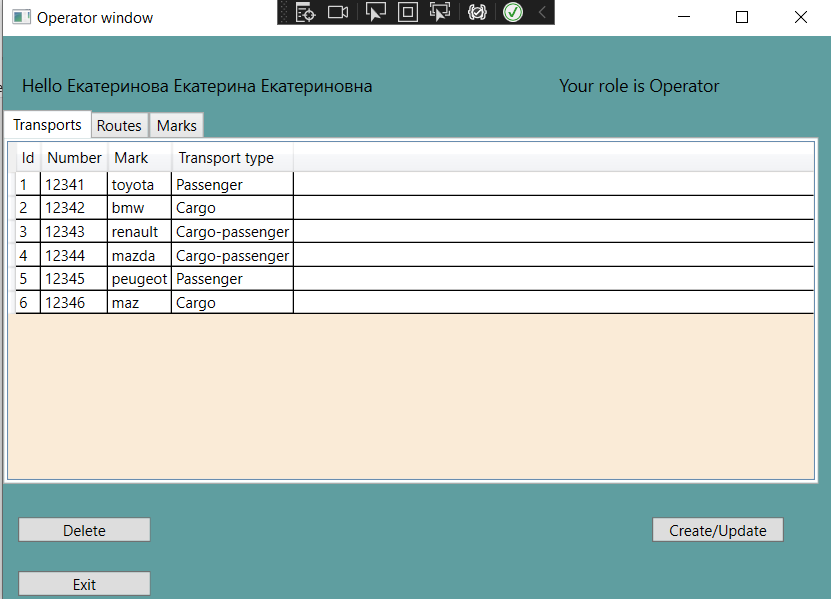


Рисунок 3.13 – Окно оператора с предоставленными функциями

Для пользователя в роли водителя доступна только функция просмотра путевых листов. Вид окна водителя приведен на рисунке 3.14.

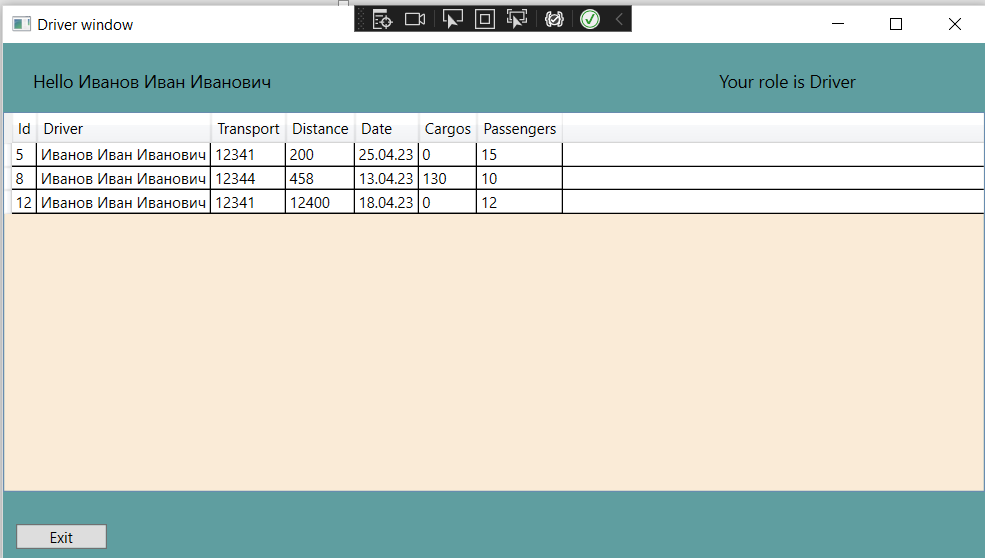


Рисунок 3.14 – Окно водителя с предоставленными функциями

После нажатия на кнопку «*Exit»* будет осуществлён выход из системы и переход на вкладку окна авторизации.

Пользователь в роли администратора, кроме вышеперечисленных функций, имеет возможность создания отчётов за произвольный период.

Для создания отчета в формате *XML* документа необходимо выбрать временной диапозон для выборки заказов. Шаблон интерфейса окна для создания отчета о заказах в заданный период представлен на рисунке 3.15.

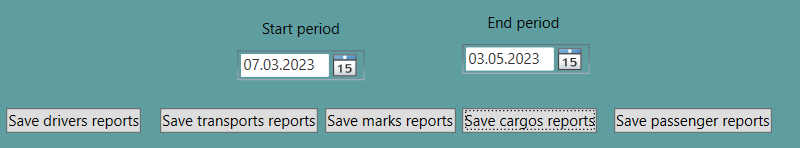


Рисунок 3.15 – Шаблон интерфейса окна для создания отчета о заказах в

указанный период

По нажатии на кнопки «*Save drivers reports*», «*Save transports reports*», «*Save marks reports*», «*Save cargos reports*», «*Save passenger reports*» создаются отчеты в формате *XML:*

* по каждому водителю (сколько сделано рейсов и какой у него средний проезд в километрах);
* по каждому автомобилю и марке автомобиля о суммарном пробеге за период;
* по грузовым автомобилям­ (сколько было перевезено грузов);
* по пассажирским автомобилям (сколько было перевезено пассажиров).

При успешном создании отчёта на экране будет выведено окно, в котором указан путь к отчёту (рисунок 3.16).

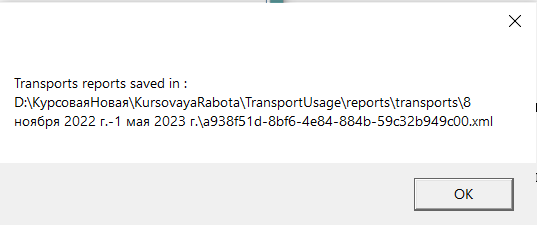


Рисунок 3.16 – Шаблон окна пути к отчёту

Пример одного из отчётов в виде электронной таблицы (на базе *Excel*) указан на рисунке 3.17.

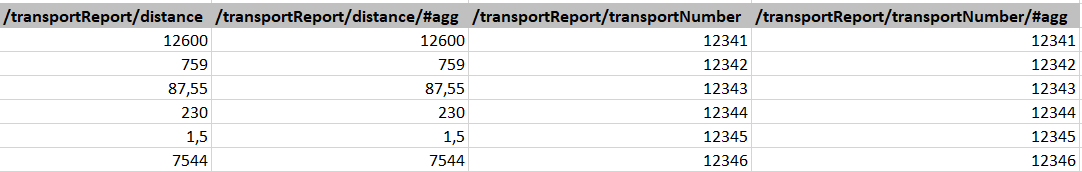


Рисунок 3.17 – Отчёт по транспорту в виде электронной таблицы

Весь функционал программного комплекса протестирован с помощью интегрированного тестирования и доказывает работоспособность программы.

В приложениях Г, Д, Ж приведены подробные руководства пользователей, системных программистов и программистов.

# ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является созданное *WPF* приложение для учёта использования автотранспорта предприятием. Целью разработки приложения было упрощение работы персонала с документацией касательно путевых листов и транспорта, а также управление процессами перевозок. Каждый зарегистрированный пользователь может заходить под одной из трёх ролей, взаимодействовать с функциями программы, основная задача которых заключается в предоставлении данных о рейсах согласно путевым листам.

Проведен аналитический обзор средств разработки, достаточных для освоения заданной темы. Затем были выделены несколько основных задач и составлен алгоритм разработки программного обеспечения. Основным этапом являлась разработка основных классов и логики программы. Для последующей расширяемости использованы шаблоны проектирования.

В разработке активно применялись современные технологии и подходы к решениям задач: для хранения исходных данных использованы *XML* документы, приложение написано с использованием объектно-ориентированного языка программирования *C#*,для графического интерфейса пользователя было выбрано приложение *WPF Application*.

Разработанное приложение позволяет ускорить процесс работы с путевыми листами, выводить структурированные отчёты и получать необходимую информацию о транспорте предприятия.

Проведённые модульные и функциональные тесты доказывают работоспособность данной программы и функционирование всех требуемых ролей и методов. Созданная информационная система полностью удовлетворяет представленным требованиям и выполняет все необходимые задачи. Полученное приложение готово для работы с пользователем.

Также приложение является актуальным и востребованным из-за его функционала и простоты использования.

Курсовой проект проверен в системе «Антиплагиат». Процент оригинальности составляет 90,29% процентов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вайсфельд, М. Объектно-ориентированное мышление в программировании / М.Вайсфельд – СПб.: Питер, 2014. – 304 с.
2. Объектно-ориентированный подход к программированию: Файловый архив для студентов. *Studfiles.* – Электрон. данные. – Режим доступа: https://studfile.net/preview/2059940/page:16/. – Дата доступа: 12.03.2023.
3. Введение в базы данных. Основные понятия и определения. – Электрон. данные. – Режим доступа: <https://siblec.ru/informatika-i-vychislitelnaya-tekhnika/bazy-dannykh>. – Дата доступа: 15.03.2023.
4. Обзор языка *C#* – Руководство по *C#*: *Microsoft Docs*. – Электрон. данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/>. – Дата доступа: 16.03.2023.
5. Принципы *SOLID* в *C#*. – Электрон. данные. – Режим доступа: <https://professorweb.ru/my/it/blog/net/solid.php>. – Дата доступа: 17.03.2023.
6. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# – ООО Издательство «Питер», 2013. – 896 с.
7. Шаблоны проектирования: Библиотека программиста. – Электрон. данные. – Режим доступа: https://proglib.io/p/creational-patterns/. – Дата доступа: 12.14.2023.
8. Основы тестирования программного обеспечения: Файловый архив для студентов. *Studfiles*. – Электрон. данные. – Режим доступа: https://studfile.net/preview/4494386/. – Дата доступа: 18.03.2023.
9. Дипломное проектирование: учеб.-метод. пособие для студентов специальностей 1-40 04 01 «Информатика и технологии программирования» и 1-40 05 01 «Информационные системы и технологии (по направлениям)» днев. формы обучения / сост.: Т.А. Трохова, И.А. Мурашко, К.С. Курочка. – Гомель: ГГТУ им. П.О. Сухого, 2019. – 55 с. – Режим доступа: <https://elib.gstu.by>. – Дата доступа: 20.04.2023.

# ПРИЛОЖЕНИЕ А

(обязательное)

**ФУНКЦИОНАЛЬНАЯ СХЕМА ПРИЛОЖЕНИЯ**

# ПРИЛОЖЕНИЕ Б

(обязательное)

**ЛИСТИНГ ПРОГРАММЫ**

**Код класса *LoginWindow.xaml.cs***

using ClassLibrary.models;

using ClassLibrary.models.transports;

using ClassLibrary.models.users;

using ClassLibrary.models.users.interfaces;

using ClassLibrary.services;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using UI.Admin;

using UI.Driver;

using UI.Operator;

namespace UI

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private readonly AbstractService<Mark> markService;

private readonly AbstractService<Route> routeService;

private readonly AbstractService<Transport> transportService;

private readonly AbstractService<User> userService;

private readonly AbstractService<UserRole> roleService;

private readonly AbstractService<TransportType> transportTypeService;

private readonly IDbConnection connection;

public MainWindow()

{

InitializeComponent();

var connectionString = ConnectionStringUtils.GetConnectionString();

connection = new SqlConnection(connectionString);

connection.Open();

markService = new MarkService(connection);

routeService = new RouteService(connection);

transportService = new TransportService(connection);

userService = new UserService(connection);

roleService = new RoleService(connection);

transportTypeService = new TransportTypeService(connection);

}

private void EnterButton\_Click(object sender, RoutedEventArgs e)

{

if(!userService.GetAll().Any())

{

userService.Create(new User

{

FIO = "admin admin admin",

Username = "admin",

Password = "admin",

UserRole = new UserRole

{

Id = 1

}

});

}

var username = LoginTextBox.Text;

var password = PasswordTextBox.Password;

var user = ((UserService)userService).Login(username, password);

if (user != null)

{

MessageBox.Show($"Hello {user.FIO}. Your role is {user.UserRole.Role}");

PasswordTextBox.Password = null;

Window window = null;

if(user is IAdministator)

{

window = new AdminWindow(markService, routeService, transportService, userService, roleService, transportTypeService, user);

}

else if(user is IOperator)

{

window = new OperatorWindow(markService, routeService, transportService, userService, transportTypeService, user);

}

else

{

window = new DriverWindow(routeService.GetAll().Where(route => route.Driver.Equals(user)), (ClassLibrary.models.users.Driver)user);

}

SetParentToWindowAndCloseParent(window);

}

else

{

MessageBox.Show("There are no user with this username/password");

}

}

private void SetParentToWindowAndCloseParent(Window window)

{

window.Owner = this;

Visibility = Visibility.Collapsed;

window.Show();

}

private void Window\_Closed(object sender, EventArgs e)

{

connection.Close();

}

}

}

**Код класса *AdminWindow.xaml.cs***

using ClassLibrary.exceptions;

using ClassLibrary.models;

using ClassLibrary.models.transports;

using ClassLibrary.models.users;

using ClassLibrary.services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace UI.Admin

{

/// <summary>

/// Логика взаимодействия для AdminWindow.xaml

/// </summary>

public partial class AdminWindow : Window

{

private readonly AbstractService<Mark> \_markService;

private readonly AbstractService<Route> \_routeService;

private readonly AbstractService<Transport> \_transportService;

private readonly AbstractService<User> \_userService;

private readonly AbstractService<UserRole> \_roleService;

private readonly AbstractService<TransportType> \_transportTypeService;

private readonly Administator \_user;

private IEnumerable<User> \_users;

public AdminWindow(AbstractService<Mark> markService, AbstractService<Route> routeService, AbstractService<Transport> transportService, AbstractService<User> userService, AbstractService<UserRole> roleService, AbstractService<TransportType> transportTypeService, User user)

{

InitializeComponent();

\_markService = markService;

\_routeService = routeService;

\_transportService = transportService;

\_userService = userService;

\_roleService = roleService;

\_transportTypeService = transportTypeService;

\_user = (Administator?)user;

FIOLabel.Content = $"Hello {\_user.FIO}";

RoleLabel.Content = $"Your role is {\_user.UserRole.Role}";

Init();

}

private void Init()

{

InitUsersDataGrid();

InitTransportDataGrid();

InitRoutesDataGrid();

InitMarksDataGrid();

}

private void Window\_Closed(object sender, EventArgs e)

{

Owner.Visibility = Visibility.Visible;

}

private void InitUsersDataGrid()

{

UsersDataGrid.ItemsSource = \_userService.GetAll();

UsersDataGrid.AutoGenerateColumns = false;

}

private void InitTransportDataGrid()

{

TransportsDataGrid.ItemsSource = \_transportService.GetAll();

TransportsDataGrid.AutoGenerateColumns = false;

}

private void InitRoutesDataGrid()

{

try

{

RoutesDataGrid.ItemsSource = \_routeService.GetAll();

}

catch(InvalidTransportTypeException)

{

}

RoutesDataGrid.AutoGenerateColumns = false;

}

private void InitMarksDataGrid()

{

MarksDataGrid.ItemsSource = \_markService.GetAll();

MarksDataGrid.AutoGenerateColumns = false;

}

private void CreateUpdateButton\_Click(object sender, RoutedEventArgs e)

{

Window window = null;

switch (Tabs.SelectedIndex)

{

case 0:

var selectedUser = UsersDataGrid.SelectedItem;

window = new AddEditUserWindow(\_userService, \_roleService, (User)selectedUser, () => InitUsersDataGrid());

break;

case 1:

var selectedTransport = TransportsDataGrid.SelectedItem;

window = new AddEditTransportWindow(\_transportService, \_markService, \_transportTypeService, (Transport)selectedTransport, () => InitTransportDataGrid());

break;

case 2:

var selectedRoute = RoutesDataGrid.SelectedItem;

window = new AddEditRouteWindow(\_routeService, \_transportService, \_userService, (Route)selectedRoute, () => InitRoutesDataGrid());

break;

case 3:

var selectedMark = MarksDataGrid.SelectedItem;

window = new AddEditMarkWindow(\_markService, (Mark)selectedMark, () => InitMarksDataGrid());

break;

}

window.Owner = this;

window.Show();

Visibility = Visibility.Collapsed;

}

private void DeleteButton\_Click(object sender, RoutedEventArgs e)

{

switch (Tabs.SelectedIndex)

{

case 0:

var selectedUser = UsersDataGrid.SelectedItem;

if (selectedUser != null)

{

if(((User)selectedUser).Id != \_user.Id)

{

\_userService.Delete(((User)selectedUser).Id);

InitUsersDataGrid();

}

else

{

MessageBox.Show("You can't delete yourself");

}

}

else

{

MessageBox.Show("Choose user to delete");

}

break;

case 1:

var selectedTransport = TransportsDataGrid.SelectedItem;

if (selectedTransport != null)

{

\_transportService.Delete(((Transport)selectedTransport).Id);

InitTransportDataGrid();

}

else

{

MessageBox.Show("Choose transport to delete");

}

break;

case 2:

var selectedRoute = RoutesDataGrid.SelectedItem;

if (selectedRoute != null)

{

\_routeService.Delete(((Route)selectedRoute).Id);

InitRoutesDataGrid();

}

else

{

MessageBox.Show("Choose route to delete");

}

break;

case 3:

var selectedMark = MarksDataGrid.SelectedItem;

if (selectedMark != null)

{

\_markService.Delete(((Mark)selectedMark).Id);

InitMarksDataGrid();

}

else

{

MessageBox.Show("Choose mark to delete");

}

break;

}

}

private void CloseButton\_Click(object sender, RoutedEventArgs e)

{

Owner.Visibility = Visibility.Visible;

Close();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

var beginDate = BeginDatePicker.SelectedDate;

var endDate = EndDatePicker.SelectedDate;

if(beginDate != default && endDate != default)

{

try

{

MessageBox.Show($"Drivers reports saved in : {((RouteService)\_routeService).SaveDriversReports(beginDate.Value.Date, endDate.Value.Date)}");

}

catch(Exception exc)

{

MessageBox.Show(exc.Message);

}

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

var beginDate = BeginDatePicker.SelectedDate;

var endDate = EndDatePicker.SelectedDate;

if (beginDate != default && endDate != default)

{

try

{

MessageBox.Show($"Transports reports saved in : {((RouteService)\_routeService).SaveTransportsReports(beginDate.Value.Date, endDate.Value.Date)}");

}

catch (Exception exc)

{

MessageBox.Show(exc.Message);

}

}

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

var beginDate = BeginDatePicker.SelectedDate;

var endDate = EndDatePicker.SelectedDate;

if (beginDate != default && endDate != default)

{

try

{

MessageBox.Show($"Marks reports saved in : {((RouteService)\_routeService).SaveMarksReports(beginDate.Value.Date, endDate.Value.Date)}");

}

catch (Exception exc)

{

MessageBox.Show(exc.Message);

}

}

}

private void Button\_Click\_3(object sender, RoutedEventArgs e)

{

var beginDate = BeginDatePicker.SelectedDate;

var endDate = EndDatePicker.SelectedDate;

if (beginDate != default && endDate != default)

{

try

{

MessageBox.Show($"Cargo transports reports saved in : {((RouteService)\_routeService).SaveCargoTransportReports(beginDate.Value.Date, endDate.Value.Date)}");

}

catch (Exception exc)

{

MessageBox.Show(exc.Message);

}

}

}

private void Button\_Click\_4(object sender, RoutedEventArgs e)

{

var beginDate = BeginDatePicker.SelectedDate;

var endDate = EndDatePicker.SelectedDate;

if (beginDate != default && endDate != default)

{

try

{

MessageBox.Show($"Passenger transports reports saved in : {((RouteService)\_routeService).SavePassengerTransportReports(beginDate.Value.Date, endDate.Value.Date)}");

}

catch (Exception exc)

{

MessageBox.Show(exc.Message);

}

}

}

}

}

**Код класса *AddEditUserWindow.cs***

using ClassLibrary.models;

using ClassLibrary.models.users;

using ClassLibrary.services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace UI.Admin

{

/// <summary>

/// Логика взаимодействия для AddEditUserWindow.xaml

/// </summary>

public partial class AddEditUserWindow : Window

{

private AbstractService<User> \_userService;

private AbstractService<UserRole> \_roleService;

private User \_user;

private Action \_action;

public AddEditUserWindow(AbstractService<User> userService, AbstractService<UserRole> roleService, User user, Action action)

{

InitializeComponent();

\_userService = userService;

\_roleService = roleService;

\_user = user;

\_action = action;

InitRoleComboBox();

InitFields();

}

private void InitRoleComboBox()

{

RoleCB.ItemsSource = \_roleService.GetAll();

RoleCB.DisplayMemberPath = "Role";

RoleCB.SelectedValuePath = "Id";

}

private void InitFields()

{

if(\_user != null)

{

FIOTB.Text = \_user.FIO;

UsernameTB.Text = \_user.Username;

PasswordTB.Text = \_user.Password;

RoleCB.SelectedValue = \_user.RoleId;

}

else

{

\_user = new User();

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

\_user.FIO = FIOTB.Text;

\_user.Username = UsernameTB.Text;

\_user.Password = PasswordTB.Text;

\_user.RoleId = (int)RoleCB.SelectedValue;

try

{

if (\_user.Id > 0)

{

\_userService.Update(\_user);

}

else

{

\_userService.Create(\_user);

}

Owner.Visibility = Visibility.Visible;

Close();

\_action();

}

catch(Exception exc)

{

MessageBox.Show(exc.Message);

}

}

private void Window\_Closed(object sender, EventArgs e)

{

Owner.Visibility = Visibility.Visible;

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

Owner.Visibility = Visibility.Visible;

Close();

}

}

}

**Код класса *OperatorWindow.xaml.cs***

using ClassLibrary.models;

using ClassLibrary.models.transports;

using ClassLibrary.models.users;

using ClassLibrary.services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using UI.Admin;

namespace UI.Operator

{

/// <summary>

/// Логика взаимодействия для OperatorWindow.xaml

/// </summary>

public partial class OperatorWindow : Window

{

private readonly AbstractService<Mark> \_markService;

private readonly AbstractService<Route> \_routeService;

private readonly AbstractService<Transport> \_transportService;

private readonly AbstractService<User> \_userService;

private readonly AbstractService<TransportType> \_transportTypeService;

private readonly ClassLibrary.models.users.Operator \_user;

public OperatorWindow(AbstractService<Mark> markService, AbstractService<Route> routeService, AbstractService<Transport> transportService, AbstractService<User> userService, AbstractService<TransportType> transportTypeService, User user)

{

InitializeComponent();

\_markService = markService;

\_routeService = routeService;

\_transportService = transportService;

\_userService = userService;

\_transportTypeService = transportTypeService;

\_user = (ClassLibrary.models.users.Operator?)user;

FIOLabel.Content = $"Hello {\_user.FIO}";

RoleLabel.Content = $"Your role is {\_user.UserRole.Role}";

Init();

}

private void Init()

{

InitTransportDataGrid();

InitRoutesDataGrid();

InitMarksDataGrid();

}

private void Window\_Closed(object sender, EventArgs e)

{

Owner.Visibility = Visibility.Visible;

}

private void InitTransportDataGrid()

{

TransportsDataGrid.ItemsSource = \_transportService.GetAll();

TransportsDataGrid.AutoGenerateColumns = false;

}

private void InitRoutesDataGrid()

{

RoutesDataGrid.ItemsSource = \_routeService.GetAll();

RoutesDataGrid.AutoGenerateColumns = false;

}

private void InitMarksDataGrid()

{

MarksDataGrid.ItemsSource = \_markService.GetAll();

MarksDataGrid.AutoGenerateColumns = false;

}

private void CreateUpdateButton\_Click(object sender, RoutedEventArgs e)

{

Window window = null;

switch (Tabs.SelectedIndex)

{

case 0:

var selectedTransport = TransportsDataGrid.SelectedItem;

window = new AddEditTransportWindow(\_transportService, \_markService, \_transportTypeService, (Transport)selectedTransport, () => InitTransportDataGrid());

break;

case 1:

var selectedRoute = RoutesDataGrid.SelectedItem;

window = new AddEditRouteWindow(\_routeService, \_transportService, \_userService, (Route)selectedRoute, () => InitRoutesDataGrid());

break;

case 2:

var selectedMark = MarksDataGrid.SelectedItem;

window = new AddEditMarkWindow(\_markService, (Mark)selectedMark, () => InitMarksDataGrid());

break;

}

window.Owner = this;

window.Show();

Visibility = Visibility.Collapsed;

}

private void DeleteButton\_Click(object sender, RoutedEventArgs e)

{

switch (Tabs.SelectedIndex)

{

case 0:

var selectedTransport = TransportsDataGrid.SelectedItem;

if (selectedTransport != null)

{

\_transportService.Delete(((Transport)selectedTransport).Id);

InitTransportDataGrid();

}

else

{

MessageBox.Show("Choose transport to delete");

}

break;

case 1:

var selectedRoute = RoutesDataGrid.SelectedItem;

if (selectedRoute != null)

{

\_routeService.Delete(((Route)selectedRoute).Id);

InitRoutesDataGrid();

}

else

{

MessageBox.Show("Choose route to delete");

}

break;

case 2:

var selectedMark = MarksDataGrid.SelectedItem;

if (selectedMark != null)

{

\_markService.Delete(((Mark)selectedMark).Id);

InitMarksDataGrid();

}

else

{

MessageBox.Show("Choose mark to delete");

}

break;

}

}

private void CloseButton\_Click(object sender, RoutedEventArgs e)

{

Owner.Visibility = Visibility.Visible;

Close();

}

}

}

**Код класса *DriverWindow.xaml.cs***

using ClassLibrary.models;

using ClassLibrary.models.users;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace UI.Driver

{

/// <summary>

/// Логика взаимодействия для DriverWindow.xaml

/// </summary>

public partial class DriverWindow : Window

{

public DriverWindow(IEnumerable<Route> routes, ClassLibrary.models.users.Driver user)

{

InitializeComponent();

InitRoutesDataGrid(routes);

FIOLabel.Content = $"Hello {user.FIO}";

RoleLabel.Content = $"Your role is {user.UserRole.Role}";

}

private void InitRoutesDataGrid(IEnumerable<Route> routes)

{

RoutesDataGrid.ItemsSource = routes;

RoutesDataGrid.AutoGenerateColumns = false;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

Owner.Visibility = Visibility.Visible;

Close();

}

}

}

**Код класса *BaseModel.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models

{

public abstract class BaseModel

{

public int Id { get; set; }

public override bool Equals(object? obj)

{

return obj is BaseModel model &&

Id == model.Id;

}

}

}

**Код класса *Mark.cs***

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models

{

public class Mark : BaseModel

{

public string MarkName { get; set; }

public override string ToString()

{

return MarkName;

}

}

}

**Код класса *Route.cs***

using ClassLibrary.exceptions;

using ClassLibrary.models.transports;

using ClassLibrary.models.transports.interfaces;

using ClassLibrary.models.users;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models

{

public class Route : BaseModel

{

public double Distance { get; set; }

public DateTime Date { get; set; }

public double Cargo { get; set; }

public int Passenger { get; set; }

public Driver Driver { get; set; }

public Transport Transport { get; set; }

}

}

**Код класса *TransportType.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models

{

public class TransportType : BaseModel

{

public string Type { get; set; }

public override string ToString()

{

return Type;

}

}

}

**Код класса *UserRole.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models

{

public class UserRole : BaseModel

{

public string Role { get; set; }

public override string ToString()

{

return Role;

}

}

}

**Код класса *IAdministrator.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users.interfaces

{

public interface IAdministator

{

}

}

**Код класса *IOperator.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users.interfaces

{

public interface IOperator

{

}

}

**Код класса *IDriver.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users.interfaces

{

public interface IDriver

{

}

}

**Код класса *Administraror.cs***

using ClassLibrary.models.users.interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users

{

public class Administator : User, IAdministator

{

}

}

**Код класса *Operator.cs***

using ClassLibrary.models.users.interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users

{

public class Operator : User, IOperator

{

}

}

**Код класса *Driver.cs***

using ClassLibrary.models.users.interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users

{

public class Driver : User, IDriver

{

}

}

**Код класса *User.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.users

{

public class User : BaseModel

{

public string Username { get; set; }

public string Password { get; set; }

public string FIO { get; set; }

public int RoleId { get; set; }

public UserRole UserRole { get; set; }

public override string ToString()

{

return FIO;

}

}

}

**Код класса *ICargoTransport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.transports.interfaces

{

public interface ICargoTransport

{

}

}

**Код класса *IPassengerTransport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.transports.interfaces

{

public interface IPassengerTransport

{

}

}

**Код класса *CargoPassengerTransport.cs***

using ClassLibrary.models.transports.interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.transports

{

public class CargoPassengerTransport : Transport, ICargoTransport, IPassengerTransport

{

}

}

**Код класса *CargoTransrort.cs***

using ClassLibrary.models.transports.interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.transports

{

public class CargoTransport : Transport, ICargoTransport

{

}

}

**Код класса *PassengerTransrort.cs***

using ClassLibrary.models.transports.interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.transports

{

public class PassengerTransport : Transport, IPassengerTransport

{

}

}

**Код класса *Transport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.transports

{

public abstract class Transport : BaseModel

{

public string TransportNumber { get; set; }

public int MaxPassenger { get; set; }

public double MaxCargo { get; set; }

public TransportType TransportType { get; set; }

public Mark Mark { get; set; }

public override string ToString()

{

return $"{TransportNumber}";

}

}

}

**Код класса *CargoTransportReport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.reports

{

public class CargoTransportReport

{

public string TransportNumber { get; set; }

public double Cargo { get; set; }

}

}

**Код класса *DriverReport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.reports

{

public class DriverReport

{

public string FIO { get; set; }

public int Count { get; set; }

public double Distance { get; set; }

}

}

**Код класса *MarkReport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.reports

{

public class MarkReport

{

public string MarkName { get; set; }

public double Distance { get; set; }

}

}

**Код класса *PassengerTransportReport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.reports

{

public class PassengerTransportReport

{

public string TransportNumber { get; set; }

public int Passengers { get; set; }

}

}

**Код класса *TransportReport.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.models.reports

{

public class TransportReport

{

public string TransportNumber { get; set; }

public double Distance { get; set; }

}

}

**Код класса *XmlReportSaver.cs***

using ClassLibrary.models.reports;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml;

namespace ClassLibrary.xml

{

public class XmlReportSaver

{

public static string SaveDriversReports(IEnumerable<DriverReport> reports, DateTime beginDate, DateTime endDate)

{

XmlDocument xmlReports = new XmlDocument();

XmlElement parent = xmlReports.CreateElement("parent");

xmlReports.AppendChild(parent);

foreach (var report in reports)

{

XmlElement driverReport = xmlReports.CreateElement("driverReport");

XmlElement driverFIO = xmlReports.CreateElement("FIO");

driverFIO.InnerText = report.FIO;

XmlElement count = xmlReports.CreateElement("count");

count.InnerText = report.Count.ToString();

XmlElement averageDistance = xmlReports.CreateElement("averageDistance");

averageDistance.InnerText = report.Distance.ToString();

driverReport.AppendChild(driverFIO);

driverReport.AppendChild(count);

driverReport.AppendChild(averageDistance);

parent.AppendChild(driverReport);

}

return SaveXmlAndReturnPath(xmlReports, "drivers", beginDate, endDate);

}

public static string SaveTransportsReports(IEnumerable<TransportReport> reports, DateTime beginDate, DateTime endDate)

{

XmlDocument xmlReports = new XmlDocument();

XmlElement parent = xmlReports.CreateElement("parent");

xmlReports.AppendChild(parent);

foreach (var report in reports)

{

XmlElement transportReport = xmlReports.CreateElement("transportReport");

XmlElement transportNumber = xmlReports.CreateElement("transportNumber");

transportNumber.InnerText = report.TransportNumber;

XmlElement distance = xmlReports.CreateElement("distance");

distance.InnerText = report.Distance.ToString();

transportReport.AppendChild(transportNumber);

transportReport.AppendChild(distance);

parent.AppendChild(transportReport);

}

return SaveXmlAndReturnPath(xmlReports, "transports", beginDate, endDate);

}

public static string SaveMarksReports(IEnumerable<MarkReport> reports, DateTime beginDate, DateTime endDate)

{

XmlDocument xmlReports = new XmlDocument();

XmlElement parent = xmlReports.CreateElement("parent");

xmlReports.AppendChild(parent);

foreach (var report in reports)

{

XmlElement markReport = xmlReports.CreateElement("markReport");

XmlElement markName = xmlReports.CreateElement("markName");

markName.InnerText = report.MarkName;

XmlElement distance = xmlReports.CreateElement("distance");

distance.InnerText = report.Distance.ToString();

markReport.AppendChild(markName);

markReport.AppendChild(distance);

parent.AppendChild(markReport);

}

return SaveXmlAndReturnPath(xmlReports, "marks", beginDate, endDate);

}

public static string SavePassengerTransportReports(IEnumerable<PassengerTransportReport> reports, DateTime beginDate, DateTime endDate)

{

XmlDocument xmlReports = new XmlDocument();

XmlElement parent = xmlReports.CreateElement("parent");

xmlReports.AppendChild(parent);

foreach (var report in reports)

{

XmlElement passengerTransportReport = xmlReports.CreateElement("passengerTransportReport");

XmlElement transportNumber = xmlReports.CreateElement("transportNumber");

transportNumber.InnerText = report.TransportNumber;

XmlElement passenger = xmlReports.CreateElement("passenger");

passenger.InnerText = report.Passengers.ToString();

passengerTransportReport.AppendChild(transportNumber);

passengerTransportReport.AppendChild(passenger);

parent.AppendChild(passengerTransportReport);

}

return SaveXmlAndReturnPath(xmlReports, "passengerTransport", beginDate, endDate);

}

public static string SaveCargoTransportReports(IEnumerable<CargoTransportReport> reports, DateTime beginDate, DateTime endDate)

{

XmlDocument xmlReports = new XmlDocument();

XmlElement parent = xmlReports.CreateElement("parent");

xmlReports.AppendChild(parent);

foreach (var report in reports)

{

XmlElement cargoTransportReport = xmlReports.CreateElement("cargoTransportReport");

XmlElement transportNumber = xmlReports.CreateElement("transportNumber");

transportNumber.InnerText = report.TransportNumber;

XmlElement cargo = xmlReports.CreateElement("cargo");

cargo.InnerText = report.Cargo.ToString();

cargoTransportReport.AppendChild(transportNumber);

cargoTransportReport.AppendChild(cargo);

parent.AppendChild(cargoTransportReport);

}

return SaveXmlAndReturnPath(xmlReports, "cargoTransport", beginDate, endDate);

}

private static string SaveXmlAndReturnPath(XmlDocument document, string reportType, DateTime beginDate, DateTime endDate)

{

var dir = Directory.CreateDirectory($"{GetDir()}\\{reportType}\\{GetDateFromDateTime(beginDate)}-{GetDateFromDateTime(endDate)}").ToString();

var path = $"{dir}\\{Guid.NewGuid()}.xml";

document.Save(path);

return path;

}

private static string GetDir()

{

return $"{Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.Parent}\\reports";

}

private static string GetDateFromDateTime(DateTime date)

{

return date.ToString("D");

}

}

}

**Код класса *AbsrtactRepository.cs***

using ClassLibrary.models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories

{

public abstract class AbstractRepository<T> where T : BaseModel

{

protected IDbConnection Connection { get; }

private readonly string createQuery;

private readonly string updateQuery;

private readonly string deleteQuery;

private readonly string getByIdQuery;

private readonly string getAllQuery;

public AbstractRepository(IDbConnection connection, string createQuery, string updateQuery, string deleteQuery, string getByIdQuery, string getAllQuery)

{

Connection = connection;

this.createQuery = createQuery;

this.updateQuery = updateQuery;

this.deleteQuery = deleteQuery;

this.getByIdQuery = getByIdQuery;

this.getAllQuery = getAllQuery;

}

public T Create(T entity)

{

var command = Connection.CreateCommand();

command.CommandText = createQuery;

SetParametersToCommand(command, entity);

command.ExecuteNonQuery();

return GetAll().Last();

}

public void Delete(int id)

{

var command = Connection.CreateCommand();

command.CommandText = deleteQuery;

SetIdToCommand(command, id);

command.ExecuteNonQuery();

}

public T GetById(int id)

{

T entity = null;

var command = Connection.CreateCommand();

command.CommandText = getByIdQuery;

SetIdToCommand(command, id);

var reader = command.ExecuteReader();

while (reader.Read())

{

entity = GetEntityFromReader(reader);

}

reader.Close();

return entity;

}

public IEnumerable<T> GetAll()

{

var entities = new List<T>();

var command = Connection.CreateCommand();

command.CommandText = getAllQuery;

var reader = command.ExecuteReader();

while (reader.Read())

{

entities.Add(GetEntityFromReader(reader));

}

reader.Close();

return entities;

}

public T Update(T entity)

{

var command = Connection.CreateCommand();

command.CommandText = updateQuery;

SetIdToCommand(command, entity.Id);

SetParametersToCommand(command, entity);

command.ExecuteNonQuery();

return GetById(entity.Id);

}

private static void SetIdToCommand(IDbCommand command, int id)

{

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

}

protected abstract T GetEntityFromReader(IDataReader reader);

protected abstract void SetParametersToCommand(IDbCommand command, T entity);

}

}

**Код класса *MarkRepository.cs***

using ClassLibrary.models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.impls

{

public class MarkRepository : AbstractRepository<Mark>

{

private static readonly string createQuery = "insert into marks values(@mark\_name)";

private static readonly string deleteQuery = "delete from marks where id = @id";

private static readonly string getByIdQuery = "select \* from marks where id = @id";

private static readonly string getAllQuery = "select \* from marks order by id";

private static readonly string updateQuery = "update marks set mark = @mark\_name where id = @id";

public MarkRepository(IDbConnection connection) : base(connection, createQuery, updateQuery, deleteQuery, getByIdQuery, getAllQuery)

{

}

protected override Mark GetEntityFromReader(IDataReader reader)

{

return new Mark

{

Id = reader.GetInt32(0),

MarkName = reader.GetString(1)

};

}

protected override void SetParametersToCommand(IDbCommand command, Mark entity)

{

var markName = command.CreateParameter();

markName.ParameterName = "@mark\_name";

markName.Value = entity.MarkName;

command.Parameters.Add(markName);

}

}

}

**Код класса *RouteRepository.cs***

using ClassLibrary.models;

using ClassLibrary.models.reports;

using ClassLibrary.models.transports;

using ClassLibrary.models.users;

using ClassLibrary.repositories.interfaces;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.impls

{

public class RouteRepository : AbstractRepository<Route>, IReport

{

private const string createQuery = "insert into routes values(@driver\_id, @transport\_id, @distance, @date, @cargo, @passenger)";

private const string deleteQuery = "delete from routes where id = @id";

private const string getByIdQuery = "select r.id, r.distance, date, cargo, passenger, users.id, FIO, username, password, transports.id, transports.transport\_number, transports.transport\_type\_id from routes as r inner join users on users.id = r.driver\_id inner join transports on transports.id = r.transport\_id where r.id = @id";

private const string getAllQuery = "select r.id, r.distance, date, cargo, passenger, users.id, FIO, username, password, transports.id, transports.transport\_number, transports.transport\_type\_id from routes as r inner join users on users.id = r.driver\_id inner join transports on transports.id = r.transport\_id";

private const string updateQuery = "update routes set driver\_id = @driver\_id, transport\_id=@transport\_id, distance=@distance, date=@date, cargo=@cargo, passenger=@passenger where id = @id;";

private const string cargoTransportReportQuery = "select transports.transport\_number, sum(routes.cargo) " +

"from routes " +

"inner join transports on transports.id=routes.transport\_id " +

"where routes.date between @startDate and @endDate " +

"and transports.transport\_type\_id=2 " +

"group by transports.transport\_number";

private const string passengerTransportReportQuery = "select transports.transport\_number, sum(routes.passenger) " +

"from routes " +

"inner join transports on transports.id=routes.transport\_id " +

"where routes.date between @startDate and @endDate " +

"and transports.transport\_type\_id=1 " +

"group by transports.transport\_number";

private const string marksReportQuery = "select marks.mark, sum(routes.distance) " +

"from routes " +

"inner join transports on transports.id=routes.transport\_id " +

"inner join marks on marks.id=transports.mark\_id " +

"where routes.date between @startDate and @endDate " +

"group by marks.mark";

private const string transportsReportQuery = "select transports.transport\_number, sum(routes.distance) " +

"from routes " +

"inner join transports on transports.id=routes.transport\_id " +

"where routes.date between @startDate and @endDate " +

"group by transports.transport\_number";

private const string driversReportQuery = "select users.fio, count(routes.id), sum(routes.distance) " +

"from routes " +

"inner join users on users.id=routes.driver\_id " +

"where routes.date between @startDate and @endDate " +

"group by users.FIO";

public RouteRepository(IDbConnection connection) : base(connection, createQuery, updateQuery, deleteQuery, getByIdQuery, getAllQuery)

{

}

public IEnumerable<DriverReport> GetDriversReports(DateTime startDate, DateTime endDate)

{

var result = new List<DriverReport>();

var command = Connection.CreateCommand();

command.CommandText = driversReportQuery;

SetStartAndEndDatesToCommand(command, startDate, endDate);

var reader = command.ExecuteReader();

while (reader.Read())

{

var count = reader.GetInt32(1);

result.Add(new DriverReport

{

FIO = reader.GetString(0),

Count = count,

Distance = reader.GetDouble(2) / count

});

}

reader.Close();

return result;

}

public IEnumerable<TransportReport> GetTransportReports(DateTime startDate, DateTime endDate)

{

var result = new List<TransportReport>();

var command = Connection.CreateCommand();

command.CommandText = transportsReportQuery;

SetStartAndEndDatesToCommand(command, startDate, endDate);

var reader = command.ExecuteReader();

while (reader.Read())

{

result.Add(new TransportReport

{

TransportNumber = reader.GetString(0),

Distance = reader.GetDouble(1)

});

}

reader.Close();

return result;

}

public IEnumerable<MarkReport> GetMarksReports(DateTime startDate, DateTime endDate)

{

var result = new List<MarkReport>();

var command = Connection.CreateCommand();

command.CommandText = marksReportQuery;

SetStartAndEndDatesToCommand(command, startDate, endDate);

var reader = command.ExecuteReader();

while (reader.Read())

{

result.Add(new MarkReport

{

MarkName = reader.GetString(0),

Distance = reader.GetDouble(1)

});

}

reader.Close();

return result;

}

public IEnumerable<PassengerTransportReport> GetPassengerTransportsReports(DateTime startDate, DateTime endDate)

{

var result = new List<PassengerTransportReport>();

var command = Connection.CreateCommand();

command.CommandText = passengerTransportReportQuery;

SetStartAndEndDatesToCommand(command, startDate, endDate);

var reader = command.ExecuteReader();

while (reader.Read())

{

result.Add(new PassengerTransportReport

{

TransportNumber = reader.GetString(0),

Passengers = reader.GetInt32(1)

});

}

reader.Close();

return result;

}

public IEnumerable<CargoTransportReport> GetCargoTransportsReports(DateTime startDate, DateTime endDate)

{

var result = new List<CargoTransportReport>();

var command = Connection.CreateCommand();

command.CommandText = cargoTransportReportQuery;

SetStartAndEndDatesToCommand(command, startDate, endDate);

var reader = command.ExecuteReader();

while (reader.Read())

{

result.Add(new CargoTransportReport

{

TransportNumber = reader.GetString(0),

Cargo = reader.GetDouble(1)

});

}

return result;

}

private static void SetStartAndEndDatesToCommand(IDbCommand command, DateTime startDate, DateTime endDate)

{

var startDateParameter = command.CreateParameter();

startDateParameter.ParameterName = "@startDate";

startDateParameter.Value = startDate.Date;

var endDateParameter = command.CreateParameter();

endDateParameter.ParameterName = "@endDate";

endDateParameter.Value = endDate.Date;

command.Parameters.Add(startDateParameter);

command.Parameters.Add(endDateParameter);

}

protected override Route GetEntityFromReader(IDataReader reader)

{

var driver = new Driver

{

Id = reader.GetInt32(5),

FIO = reader.GetString(6),

Username = reader.GetString(7),

Password = reader.GetString(8)

};

Transport transport;

var transportId = reader.GetInt32(9);

var transportNumber = reader.GetString(10);

var transportTypeId = reader.GetInt32(11);

if (transportTypeId == 1)

{

transport = new PassengerTransport

{

Id = transportId,

TransportNumber = transportNumber,

};

}

else if (transportTypeId == 2)

{

transport = new CargoTransport

{

Id = transportId,

TransportNumber = transportNumber,

};

}

else

{

transport = new CargoPassengerTransport

{

Id = transportId,

TransportNumber = transportNumber,

};

}

return new Route

{

Id = reader.GetInt32(0),

Driver = driver,

Transport = transport,

Distance = reader.GetDouble(1),

Date = reader.GetDateTime(2),

Cargo = reader.GetDouble(3),

Passenger = reader.GetInt32(4)

};

}

protected override void SetParametersToCommand(IDbCommand command, Route entity)

{

var driverIdParam = command.CreateParameter();

driverIdParam.ParameterName = "@driver\_id";

driverIdParam.Value = entity.Driver.Id;

var transportIdParam = command.CreateParameter();

transportIdParam.ParameterName = "@transport\_id";

transportIdParam.Value = entity.Transport.Id;

var distanceParam = command.CreateParameter();

distanceParam.ParameterName = "@distance";

distanceParam.Value = entity.Distance;

var dateParam = command.CreateParameter();

dateParam.ParameterName = "@date";

dateParam.Value = DateTime.Parse(entity.Date.ToString());

var cargoParam = command.CreateParameter();

cargoParam.ParameterName = "@cargo";

cargoParam.Value = entity.Cargo;

var passengerParam = command.CreateParameter();

passengerParam.ParameterName = "@passenger";

passengerParam.Value = entity.Passenger;

command.Parameters.Add(driverIdParam);

command.Parameters.Add(transportIdParam);

command.Parameters.Add(distanceParam);

command.Parameters.Add(dateParam);

command.Parameters.Add(cargoParam);

command.Parameters.Add(passengerParam);

}

}

}

**Код класса *TransportRepository.cs***

using ClassLibrary.models;

using ClassLibrary.models.transports;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.impls

{

public class TransportRepository : AbstractRepository<Transport>

{

private static readonly string createQuery = "insert into transports values(@transport\_number, @transport\_type\_id, @mark\_id)";

private static readonly string deleteQuery = "delete from transports where id = @id";

private static readonly string getByIdQuery = "select transports.id, transport\_number, marks.id, marks.mark, transport\_types.id, transport\_types.transport\_type " +

"from transports " +

"inner join transport\_types on transport\_type\_id = transport\_types.id " +

"inner join marks on marks.id = transports.mark\_id where transports.id = @id";

private static readonly string getAllQuery = "select transports.id, transport\_number, marks.id, marks.mark, transport\_types.id, transport\_types.transport\_type " +

" from transports " +

" inner join transport\_types on transport\_type\_id = transport\_types.id " +

" inner join marks on marks.id = transports.mark\_id";

private static readonly string updateQuery = "update transports set transport\_number = @transport\_number, transport\_type\_id=@transport\_type\_id, mark\_id=@mark\_id where id = @id;";

public TransportRepository(IDbConnection connection) : base(connection, createQuery, updateQuery, deleteQuery, getByIdQuery, getAllQuery)

{

}

protected override Transport GetEntityFromReader(IDataReader reader)

{

Transport transport;

var id = reader.GetInt32(0);

var transportNumber = reader.GetString(1);

var transportTypeId = reader.GetInt32(4);

var mark = new Mark

{

Id = reader.GetInt32(2),

MarkName = reader.GetString(3),

};

var transportType = new TransportType

{

Id = reader.GetInt32(4),

Type = reader.GetString(5),

};

if (transportTypeId == 1)

{

transport = new PassengerTransport

{

Id = id,

TransportNumber = transportNumber,

TransportType = transportType,

Mark = mark

};

}

else if (transportTypeId == 2)

{

transport = new CargoTransport

{

Id = id,

TransportNumber = transportNumber,

TransportType = transportType,

Mark = mark

};

}

else

{

transport = new CargoPassengerTransport

{

Id = id,

TransportNumber = transportNumber,

TransportType = transportType,

Mark = mark

};

}

return transport;

}

protected override void SetParametersToCommand(IDbCommand command, Transport entity)

{

var transportNumberParam = command.CreateParameter();

transportNumberParam.ParameterName = "@transport\_number";

transportNumberParam.Value = entity.TransportNumber;

var transportTypeIdParam = command.CreateParameter();

transportTypeIdParam.ParameterName = "@transport\_type\_id";

transportTypeIdParam.Value = entity.TransportType.Id;

var markIdParam = command.CreateParameter();

markIdParam.ParameterName = "@mark\_id";

markIdParam.Value = entity.Mark.Id;

command.Parameters.Add(transportNumberParam);

command.Parameters.Add(transportTypeIdParam);

command.Parameters.Add(markIdParam);

}

}

}

**Код класса *TransportTypeRepository.cs***

using ClassLibrary.models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.impls

{

public class TransportTypeRepository : AbstractRepository<TransportType>

{

private static readonly string getByIdQuery = "select \* from transport\_types where id = @id";

private static readonly string getAllQuery = "select \* from transport\_types order by id";

public TransportTypeRepository(IDbConnection connection) : base(connection, null, null, null, getByIdQuery, getAllQuery)

{

}

protected override TransportType GetEntityFromReader(IDataReader reader)

{

return new TransportType

{

Id = reader.GetInt32(0),

Type = reader.GetString(1)

};

}

protected override void SetParametersToCommand(IDbCommand command, TransportType entity)

{

throw new NotImplementedException();

}

}

}

**Код класса *UserRepository.cs***

using ClassLibrary.models.users;

using ClassLibrary.repositories.interfaces;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.impls

{

public class UserRepository : AbstractRepository<User>, ILogin

{

private static readonly string createQuery = "insert into users values(@username, @password, @FIO, @role\_id)";

private static readonly string deleteQuery = "delete from users where id = @id";

private static readonly string getByIdQuery = "select \* from users where id = @id";

private static readonly string getAllQuery = "select \* from users";

private static readonly string updateQuery = "update users set username = @username, password=@password, FIO=@FIO, role\_id=@role\_id where id = @id;";

private static readonly string loginQuery = "select \* from users where username=@username";

public UserRepository(IDbConnection connection) : base(connection, createQuery, updateQuery, deleteQuery, getByIdQuery, getAllQuery)

{

}

public User GetByUsername(string username)

{

User entity = null;

var command = Connection.CreateCommand();

command.CommandText = loginQuery;

SetUsernameToCommand(command, username);

var reader = command.ExecuteReader();

while (reader.Read())

{

entity = GetEntityFromReader(reader);

}

reader.Close();

return entity;

}

protected override User GetEntityFromReader(IDataReader reader)

{

int roleId = reader.GetInt32(4);

if(roleId == 1)

{

return new Administator

{

Id = reader.GetInt32(0),

Username = reader.GetString(1),

Password = reader.GetString(2),

FIO = reader.GetString(3),

RoleId = roleId

};

}

else if(roleId == 2)

{

return new Operator

{

Id = reader.GetInt32(0),

Username = reader.GetString(1),

Password = reader.GetString(2),

FIO = reader.GetString(3),

RoleId = roleId

};

}

else

{

return new Driver

{

Id = reader.GetInt32(0),

Username = reader.GetString(1),

Password = reader.GetString(2),

FIO = reader.GetString(3),

RoleId = roleId

};

}

}

private static void SetUsernameToCommand(IDbCommand command, string username)

{

var usernameParameter = command.CreateParameter();

usernameParameter.ParameterName = "@username";

usernameParameter.Value = username;

command.Parameters.Add(usernameParameter);

}

protected override void SetParametersToCommand(IDbCommand command, User entity)

{

SetUsernameToCommand(command, entity.Username);

var passwordParameter = command.CreateParameter();

passwordParameter.ParameterName = "@password";

passwordParameter.Value = entity.Password;

var fioParameter = command.CreateParameter();

fioParameter.ParameterName = "@FIO";

fioParameter.Value = entity.FIO;

var roleIdParameter = command.CreateParameter();

roleIdParameter.ParameterName = "@role\_id";

roleIdParameter.Value = entity.RoleId;

command.Parameters.Add(fioParameter);

command.Parameters.Add(roleIdParameter);

command.Parameters.Add(passwordParameter);

}

}

}

**Код класса *UserRoleRepository cs***

using ClassLibrary.models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.impls

{

public class UserRoleRepository : AbstractRepository<UserRole>

{

private const string getByIdQuery = "select \* from user\_roles where id = @id";

private const string getAllQuery = "select \* from user\_roles order by id";

public UserRoleRepository(IDbConnection connection) : base(connection, null, null, null, getByIdQuery, getAllQuery)

{

}

protected override UserRole GetEntityFromReader(IDataReader reader)

{

return new UserRole

{

Id = reader.GetInt32(0),

Role = reader.GetString(1)

};

}

protected override void SetParametersToCommand(IDbCommand command, UserRole entity)

{

throw new NotImplementedException();

}

}

}

**Код класса *Ilogin.cs***

using ClassLibrary.models.users;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.interfaces

{

public interface ILogin

{

User GetByUsername(string username);

}

}

**Код класса *IReport.cs***

using ClassLibrary.models.reports;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.repositories.interfaces

{

public interface IReport

{

IEnumerable<DriverReport> GetDriversReports(DateTime startDate, DateTime endDate);

IEnumerable<TransportReport> GetTransportReports(DateTime startDate, DateTime endDate);

IEnumerable<MarkReport> GetMarksReports(DateTime startDate, DateTime endDate);

IEnumerable<PassengerTransportReport> GetPassengerTransportsReports(DateTime startDate, DateTime endDate);

IEnumerable<CargoTransportReport> GetCargoTransportsReports(DateTime startDate, DateTime endDate);

}

}

**Код класса *AbstractService.cs***

using ClassLibrary.models;

using ClassLibrary.repositories;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.services

{

public abstract class AbstractService<Entity> where Entity : BaseModel

{

protected readonly AbstractRepository<Entity> \_repository;

public AbstractService(AbstractRepository<Entity> repository)

{

\_repository = repository;

}

public virtual void Create(Entity entity)

{

ValidateEntity(entity);

\_repository.Create(entity);

}

public virtual void Delete(int id)

{

\_repository.Delete(id);

}

public virtual IEnumerable<Entity> GetAll()

{

return \_repository.GetAll();

}

public virtual Entity Get(int id)

{

return \_repository.GetById(id);

}

public virtual void Update(Entity entity)

{

ValidateEntity(entity);

\_repository.Update(entity);

}

protected virtual void ValidateEntity(Entity entity)

{

if (entity == null)

{

throw new NullReferenceException("Empty entity were passed");

}

}

}

}

**Код класса *MarkService.cs***

using ClassLibrary.exceptions;

using ClassLibrary.models;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System.Data;

namespace ClassLibrary.services

{

public class MarkService : AbstractService<Mark>

{

public MarkService(AbstractRepository<Mark> repository) : base(repository)

{

}

public MarkService(IDbConnection connection) : base(new MarkRepository(connection))

{

}

protected override void ValidateEntity(Mark entity)

{

base.ValidateEntity(entity);

if(entity.MarkName == null || entity.MarkName.Length < 3)

{

throw new ValidationException($"Mark name too short: {entity.MarkName}. Must be greater that 2 symbols");

}

}

}

}

**Код класса *RoleService.cs***

using ClassLibrary.models;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.services

{

public class RoleService : AbstractService<UserRole>

{

public RoleService(AbstractRepository<UserRole> repository) : base(repository)

{

}

public RoleService(IDbConnection connection) : base(new UserRoleRepository(connection))

{

}

public override void Create(UserRole entityDTO)

{

throw new NotImplementedException();

}

public override void Delete(int id)

{

throw new NotImplementedException();

}

public override void Update(UserRole entityDTO)

{

throw new NotImplementedException();

}

}

}

**Код класса *RouteService.cs***

using ClassLibrary.exceptions;

using ClassLibrary.models;

using ClassLibrary.models.reports;

using ClassLibrary.models.transports;

using ClassLibrary.models.transports;

using ClassLibrary.models.transports.interfaces;

using ClassLibrary.models.users;

using ClassLibrary.models.users.interfaces;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using ClassLibrary.repositories.interfaces;

using ClassLibrary.xml;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using InvalidDataException = ClassLibrary.exceptions.InvalidDataException;

namespace ClassLibrary.services

{

public class RouteService : AbstractService<Route>

{

private readonly AbstractRepository<Transport> \_transportRepository;

private readonly AbstractRepository<User> \_userRepository;

public RouteService(AbstractRepository<Route> repository, AbstractRepository<Transport> transportRepository, AbstractRepository<User> userRepository) : base(repository)

{

\_transportRepository = transportRepository;

\_userRepository = userRepository;

}

public RouteService(IDbConnection connection) : base(new RouteRepository(connection))

{

\_transportRepository = new TransportRepository(connection);

\_userRepository = new UserRepository(connection);

}

public string SaveDriversReports(DateTime beginDate, DateTime endDate)

{

var reports = ((RouteRepository)\_repository).GetDriversReports(beginDate, endDate);

if (reports == null)

{

throw new InvalidPeriodException("No data during choosed period");

}

return XmlReportSaver.SaveDriversReports(reports, beginDate, endDate);

}

public string SaveTransportsReports(DateTime beginDate, DateTime endDate)

{

var reports = ((RouteRepository)\_repository).GetTransportReports(beginDate, endDate);

if (reports == null)

{

throw new InvalidPeriodException("No data during choosed period");

}

return XmlReportSaver.SaveTransportsReports(reports, beginDate, endDate);

}

public string SaveMarksReports(DateTime beginDate, DateTime endDate)

{

var reports = ((RouteRepository)\_repository).GetMarksReports(beginDate, endDate);

if (reports == null)

{

throw new InvalidPeriodException("No data during choosed period");

}

return XmlReportSaver.SaveMarksReports(reports, beginDate, endDate);

}

public string SavePassengerTransportReports(DateTime beginDate, DateTime endDate)

{

var reports = ((RouteRepository)\_repository).GetPassengerTransportsReports(beginDate, endDate);

if (reports == null)

{

throw new InvalidPeriodException("No data during choosed period");

}

return XmlReportSaver.SavePassengerTransportReports(reports, beginDate, endDate);

}

public string SaveCargoTransportReports(DateTime beginDate, DateTime endDate)

{

var reports = ((RouteRepository)\_repository).GetCargoTransportsReports(beginDate, endDate);

if (reports == null)

{

throw new InvalidPeriodException("No data during choosed period");

}

return XmlReportSaver.SaveCargoTransportReports(reports, beginDate, endDate);

}

protected override void ValidateEntity(Route entity)

{

base.ValidateEntity(entity);

if (entity.Distance <= 0)

{

throw new ValidationException("Distance must be greater than 0");

}

if (entity.Date == default)

{

throw new ValidationException("Invalide date");

}

try

{

var user = \_userRepository.GetById(entity.Driver.Id);

if (user is not IDriver)

{

throw new NotDriverException("Only driver can be set to the route");

}

}

catch

{

throw new EntityNotFoundException("There no driver");

}

try

{

\_transportRepository.GetById(entity.Transport.Id);

}

catch

{

throw new EntityNotFoundException("There no transport");

}

var all = \_repository.GetAll().ToList();

if (all.Any(route => route.Transport.Equals(entity.Transport) && route.Date.Date.Equals(entity.Date.Date) && entity.Id != route.Id))

{

throw new InvalidDataException("This transport already used in this date");

}

if (all.Any(route => route.Driver.Equals(entity.Driver) && route.Date.Date.Equals(entity.Date.Date) && entity.Id != route.Id))

{

throw new InvalidDataException("This driver already used in this date");

}

var transport = \_transportRepository.GetById(entity.Transport.Id);

if (transport is not ICargoTransport && entity.Cargo > 0)

{

throw new InvalidTransportTypeException("You can't set cargo to passenger transport");

}

else if (transport is not IPassengerTransport && entity.Passenger > 0)

{

throw new InvalidTransportTypeException("You can't set passengers to cargo transport");

}

if(entity.Cargo > transport.MaxCargo)

{

throw new ExceededException(@$"Max avilable cargo for this transport are {transport.MaxCargo}");

}

else if(entity.Passenger > transport.MaxPassenger)

{

throw new ExceededException(@$"Max avilable passengers for this transport are {transport.MaxPassenger}");

}

}

}

}

**Код класса *TransportService.cs***

using ClassLibrary.exceptions;

using ClassLibrary.models;

using ClassLibrary.models.transports;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.services

{

public class TransportService : AbstractService<Transport>

{

private readonly AbstractRepository<TransportType> \_typeRepository;

private readonly AbstractRepository<Mark> \_markRepository;

public TransportService(AbstractRepository<Transport> repository, AbstractRepository<TransportType> typeRepository, AbstractRepository<Mark> markRepository) : base(repository)

{

\_typeRepository = typeRepository;

\_markRepository = markRepository;

}

public TransportService(IDbConnection connection) : base(new TransportRepository(connection))

{

\_typeRepository = new TransportTypeRepository(connection);

\_markRepository = new MarkRepository(connection);

}

public override IEnumerable<Transport> GetAll()

{

return base.GetAll().Select(transport =>

{

SetChildToTransport(transport);

return transport;

});

}

public override Transport Get(int id)

{

var transport = base.Get(id);

SetChildToTransport(transport);

return transport;

}

public override void Create(Transport entity)

{

try

{

base.Create(entity);

}

catch(SqlException e)

{

if (e.Message.Contains("UNIQUE KEY"))

{

throw new NotInuqieValueException("Transport number must be unique");

}

throw e;

}

}

private void SetChildToTransport(Transport transport)

{

transport.TransportType = \_typeRepository.GetById(transport.TypeId);

transport.Mark = \_markRepository.GetById(transport.MarkId);

}

protected override void ValidateEntity(Transport entity)

{

base.ValidateEntity(entity);

if(entity.TransportNumber == null || entity.TransportNumber.Length < 5)

{

throw new ValidationException($"Transport number too short: {entity.TransportNumber}. Must be greater that 4 symbols");

}

try

{

\_typeRepository.GetById(entity.TypeId);

}

catch

{

throw new EntityNotFoundException("There no transport type");

}

try

{

\_markRepository.GetById(entity.MarkId);

}

catch

{

throw new EntityNotFoundException("There no mark");

}

}

}

}

**Код класса *TransportTypeService.cs***

using ClassLibrary.models;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.services

{

public class TransportTypeService : AbstractService<TransportType>

{

public TransportTypeService(AbstractRepository<TransportType> repository) : base(repository)

{

}

public TransportTypeService(IDbConnection connection) : base(new TransportTypeRepository(connection))

{

}

public override void Create(TransportType entity)

{

throw new NotImplementedException();

}

public override void Delete(int id)

{

throw new NotImplementedException();

}

public override void Update(TransportType entity)

{

throw new NotImplementedException();

}

}

}

**Код класса *UserService.cs***

using ClassLibrary.exceptions;

using ClassLibrary.models;

using ClassLibrary.models.users;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using ClassLibrary.repositories.interfaces;

using Isopoh.Cryptography.Argon2;

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.services

{

public class UserService : AbstractService<User>

{

private readonly AbstractRepository<UserRole> \_roleRepository;

public UserService(AbstractRepository<User> repository, AbstractRepository<UserRole> roleRepository) : base(repository)

{

\_roleRepository = roleRepository;

}

public UserService(IDbConnection connection) : base(new UserRepository(connection))

{

\_roleRepository = new UserRoleRepository(connection);

}

public override IEnumerable<User> GetAll()

{

return base.GetAll().Select(user =>

{

SetChildToUser(user);

return user;

});

}

public override User Get(int id)

{

var user = base.Get(id);

SetChildToUser(user);

return user;

}

public override void Create(User entity)

{

try

{

entity.Password = Argon2.Hash(entity.Password);

base.Create(entity);

}

catch (SqlException e)

{

if (e.Message.Contains("UNIQUE KEY"))

{

throw new NotInuqieValueException("Username must be unique");

}

throw e;

}

}

public User Login(string username, string password)

{

var user = ((ILogin)\_repository).GetByUsername(username);

if(user != null && Argon2.Verify(user.Password, password))

{

SetChildToUser(user);

return user;

}

return null;

}

private void SetChildToUser(User user)

{

user.UserRole = \_roleRepository.GetById(user.RoleId);

}

protected override void ValidateEntity(User entity)

{

base.ValidateEntity(entity);

if (entity.Username == null || entity.Username.Length < 4)

{

throw new ValidationException($"Username too short: {entity.Username}. Must be greater that 3 symbols");

}

else if (entity.Password == null || entity.Password.Length < 5)

{

throw new ValidationException($"Password too short: {entity.Password}. Must be greater that 4 symbols");

}

else if (entity.FIO == null || entity.FIO.Length < 15)

{

throw new ValidationException($"FIO too short: {entity.FIO}. Must be greater that 14 symbols");

}

try

{

\_roleRepository.GetById(entity.RoleId);

}

catch

{

throw new EntityNotFoundException("There no user role");

}

}

}

}

***Код класса EntityNotFoundException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class EntityNotFoundException : Exception

{

public EntityNotFoundException(string message) : base(message)

{

}

}

}

***Код класса InvalidDataException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class InvalidDataException : Exception

{

public InvalidDataException(string message) : base(message)

{

}

}

}

***Код класса InvalidPeriodException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class InvalidPeriodException : Exception

{

public InvalidPeriodException(string message) : base(message) { }

}

}

***Код класса InvalidTransportTypeException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class InvalidTransportTypeException : Exception

{

public InvalidTransportTypeException(string message) : base(message)

{

}

}

}

***Код класса NotDriverException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class NotDriverException : Exception

{

public NotDriverException(string message) : base(message)

{

}

}

}

***Код класса*** ***NotInuqieValueException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class NotInuqieValueException : Exception

{

public NotInuqieValueException(string message) : base(message)

{

}

}

}

**Код класса *ValidationException.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary.exceptions

{

public class ValidationException : Exception

{

public ValidationException(string message) : base(message)

{

}

}

}

# ПРИЛОЖЕНИЕ В

(обязательное)

**ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ**

**Код класса *MarkRepositoryTest.cs***

using ClassLibrary.models;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tests.reposotories

{

[TestClass]

public class MarkRepositoryTest

{

private AbstractRepository<Mark> \_repository;

SqlConnection connection;

[TestInitialize]

public void SetUp()

{

connection = new SqlConnection(ConnectionStringUtils.GetConnectionString());

\_repository = new MarkRepository(connection);

connection.Open();

}

[TestCleanup]

public void CleanUp()

{

connection.Close();

}

[TestMethod]

public void CreateAndDeleteMarkTest()

{

var expectedCount = \_repository.GetAll().Count();

var mark = new Mark

{

MarkName = "test"

};

var createdMark = \_repository.Create(mark);

\_repository.Delete(createdMark.Id);

Assert.AreEqual(mark.MarkName, createdMark.MarkName);

Assert.AreEqual(expectedCount, \_repository.GetAll().Count());

}

[TestMethod]

public void UpdateMarkTest()

{

var createdMark = \_repository.Create(new Mark

{

MarkName = "test"

});

createdMark.MarkName = "updated";

var updatedMark = \_repository.Update(createdMark);

Assert.AreEqual("updated", updatedMark.MarkName);

\_repository.Delete(updatedMark.Id);

}

[TestMethod]

public void GetAllTest()

{

var expectedCount = \_repository.GetAll().Count() + 2;

var mark1 = \_repository.Create(new Mark

{

MarkName = "test"

});

var mark2 = \_repository.Create(new Mark

{

MarkName = "test"

});

var actualCount = \_repository.GetAll().Count();

\_repository.Delete(mark1.Id);

\_repository.Delete(mark2.Id);

Assert.AreEqual(expectedCount, actualCount);

}

}

}

**Код класса *RouteRepositoryTest.cs***

using ClassLibrary.models;

using ClassLibrary.models.transports;

using ClassLibrary.models.transports;

using ClassLibrary.models.users;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tests.reposotories

{

[TestClass]

public class RouteRepositoryTest

{

AbstractRepository<Route> \_repository;

AbstractRepository<User> \_userRepository;

AbstractRepository<Transport> \_transportRepository;

AbstractRepository<Mark> \_markRepository;

private Route \_route;

private User \_driver;

private Transport \_transport;

private Mark \_mark;

SqlConnection connection;

[TestInitialize]

public void SetUp()

{

connection = new SqlConnection(ConnectionStringUtils.GetConnectionString());

\_repository = new RouteRepository(connection);

\_userRepository = new UserRepository(connection);

\_transportRepository = new TransportRepository(connection);

\_markRepository = new MarkRepository(connection);

connection.Open();

\_mark = \_markRepository.Create(new Mark

{

MarkName = "test"

});

\_transport = \_transportRepository.Create(new PassengerTransport

{

TransportNumber = "passenger",

TypeId = 1,

MarkId = \_mark.Id

});

\_driver = \_userRepository.Create(new User

{

Username = "test",

Password = "test",

FIO = "test",

RoleId = 3

});

\_route = new Route

{

DriverId = \_driver.Id,

TransportId = \_transport.Id,

Distance = 100,

Date = DateTime.Now,

Passenger = 100

};

}

[TestCleanup]

public void CleanUp()

{

\_markRepository.Delete(\_mark.Id);

\_userRepository.Delete(\_driver.Id);

connection.Close();

}

[TestMethod]

public void CreateAndDeleteTransportTest()

{

var expectedCount = \_repository.GetAll().Count();

var createdRoute = \_repository.Create(\_route);

\_repository.Delete(createdRoute.Id);

Assert.AreEqual(createdRoute.DriverId, createdRoute.DriverId);

Assert.AreEqual(createdRoute.TransportId, createdRoute.TransportId);

Assert.AreEqual(createdRoute.Distance, createdRoute.Distance);

Assert.AreEqual(createdRoute.Date, createdRoute.Date);

Assert.AreEqual(createdRoute.Cargo, createdRoute.Cargo);

Assert.AreEqual(createdRoute.Passenger, createdRoute.Passenger);

Assert.AreEqual(expectedCount, \_repository.GetAll().Count());

}

[TestMethod]

public void UpdateTransportTest()

{

var createdRoute = \_repository.Create(\_route);

createdRoute.Distance = 120;

createdRoute.Date = createdRoute.Date.AddDays(1);

createdRoute.Passenger = 120;

var updatedRoute = \_repository.Update(createdRoute);

Assert.AreEqual(120, updatedRoute.Distance);

Assert.AreEqual(createdRoute.Date, updatedRoute.Date);

Assert.AreEqual(120, updatedRoute.Passenger);

}

[TestMethod]

public void GetAllTest()

{

var expectedCount = \_repository.GetAll().Count() + 2;

var route1 = \_repository.Create(\_route);

var route2 = \_repository.Create(\_route);

var actualCount = \_repository.GetAll().Count();

\_repository.Delete(route1.Id);

\_repository.Delete(route2.Id);

Assert.AreEqual(expectedCount, actualCount);

}

}

}

**Код класса *TransportRepositoryTest.cs***

using ClassLibrary.models;

using ClassLibrary.models.transports;

using ClassLibrary.models.transports;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tests.reposotories

{

[TestClass]

public class TransportRepositoryTest

{

private AbstractRepository<Transport> \_repository;

private AbstractRepository<Mark> \_markRepository;

private Mark \_mark;

private Transport \_passengerTransport;

SqlConnection connection;

[TestInitialize]

public void SetUp()

{

connection = new SqlConnection(ConnectionStringUtils.GetConnectionString());

\_repository = new TransportRepository(connection);

\_markRepository = new MarkRepository(connection);

connection.Open();

\_mark = \_markRepository.Create(new Mark

{

MarkName = "test"

});

\_passengerTransport = new PassengerTransport

{

TransportNumber = "passenger",

TypeId = 1,

MarkId = \_mark.Id

};

}

[TestCleanup]

public void CleanUp()

{

\_markRepository.Delete(\_mark.Id);

connection.Close();

}

[TestMethod]

public void CreateAndDeleteTransportTest()

{

var expectedCount = \_repository.GetAll().Count();

var createdTransport = \_repository.Create(\_passengerTransport);

\_repository.Delete(createdTransport.Id);

Assert.AreEqual(\_passengerTransport.TransportNumber, createdTransport.TransportNumber);

Assert.AreEqual(\_passengerTransport.TypeId, createdTransport.TypeId);

Assert.AreEqual(\_passengerTransport.MarkId, createdTransport.MarkId);

Assert.AreEqual(expectedCount, \_repository.GetAll().Count());

}

[TestMethod]

public void UpdateTransportTest()

{

var createdTransport = \_repository.Create(\_passengerTransport);

createdTransport.TransportNumber = "updated";

createdTransport.TypeId = 2;

var updatedTransport = \_repository.Update(createdTransport);

Assert.AreEqual("updated", updatedTransport.TransportNumber);

Assert.AreEqual(2, updatedTransport.TypeId);

Assert.IsTrue(updatedTransport is CargoTransport);

\_repository.Delete(updatedTransport.Id);

}

[TestMethod]

public void GetAllTest()

{

var expectedCount = \_repository.GetAll().Count() + 2;

var passengerTransport = \_repository.Create(\_passengerTransport);

var cargoTransport = \_repository.Create(new CargoTransport

{

TransportNumber = "cargo",

TypeId = 2,

MarkId = \_mark.Id

});

var actualCount = \_repository.GetAll().Count();

\_repository.Delete(passengerTransport.Id);

\_repository.Delete(cargoTransport.Id);

Assert.AreEqual(expectedCount, actualCount);

}

[TestMethod]

[ExpectedException(typeof(SqlException))]

public void CreateTransportWithSameTransportNumberTest()

{

int id = 0;

try

{

id = \_repository.Create(\_passengerTransport).Id;

\_repository.Create(\_passengerTransport);

}

catch (Exception ex)

{

\_repository.Delete(id);

throw ex;

}

}

}

}

**Код класса *UserRepositoryTest.cs***

using ClassLibrary.models.users;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using ClassLibrary.repositories.interfaces;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tests.reposotories

{

[TestClass]

public class UserRepositoryTest

{

private AbstractRepository<User> \_repository;

private readonly User \_user = new()

{

Username = "user",

Password = "password",

FIO = "fio",

RoleId = 2

};

SqlConnection connection;

[TestInitialize]

public void SetUp()

{

connection = new SqlConnection(ConnectionStringUtils.GetConnectionString());

\_repository = new UserRepository(connection);

connection.Open();

}

[TestCleanup]

public void Cleanup()

{

connection.Close();

}

[TestMethod]

public void CreateAndDeleteUserTest()

{

var expectedCount = \_repository.GetAll().Count();

var createdUser = \_repository.Create(\_user);

\_repository.Delete(createdUser.Id);

Assert.AreEqual(\_user.Username, createdUser.Username);

Assert.AreEqual(\_user.Password, createdUser.Password);

Assert.AreEqual(\_user.FIO, createdUser.FIO);

Assert.AreEqual(\_user.RoleId, createdUser.RoleId);

Assert.AreEqual(expectedCount, \_repository.GetAll().Count());

}

[TestMethod]

public void UpdateUserTest()

{

var updated = "updated";

var createdUser = \_repository.Create(\_user);

createdUser.Username = updated;

createdUser.Password = updated;

createdUser.FIO = updated;

createdUser.RoleId = 3;

var updatedUser = \_repository.Update(createdUser);

Assert.AreEqual(updated, updatedUser.Username);

Assert.AreEqual(updated, updatedUser.Password);

Assert.AreEqual(updated, updatedUser.FIO);

Assert.AreEqual(3, updatedUser.RoleId);

\_repository.Delete(updatedUser.Id);

}

[TestMethod]

public void GetAllTest()

{

var expectedCount = \_repository.GetAll().Count() + 2;

var user1 = \_repository.Create(\_user);

var user2 = \_repository.Create(new User

{

Username = "user1",

Password = "password",

FIO = "fio",

RoleId = 2

});

var actualCount = \_repository.GetAll().Count();

\_repository.Delete(user1.Id);

\_repository.Delete(user2.Id);

Assert.AreEqual(expectedCount, actualCount);

}

[TestMethod]

public void LoginTest()

{

var createdUser = \_repository.Create(\_user);

Assert.IsNotNull(((ILogin)\_repository).GetByUsername(\_user.Username));

\_repository.Delete(createdUser.Id);

}

[TestMethod]

[ExpectedException(typeof(SqlException))]

public void CreateUserWithSameUsernameTest()

{

int id = 0;

try

{

id = \_repository.Create(\_user).Id;

\_repository.Create(\_user);

}

catch (Exception ex)

{

\_repository.Delete(id);

throw ex;

}

}

}

}

**Код класса *UserRoleRepositoryTest.cs***

using ClassLibrary.models;

using ClassLibrary.repositories;

using ClassLibrary.repositories.impls;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tests.reposotories

{

[TestClass]

public class UserRoleRepositoryTest

{

private AbstractRepository<UserRole> \_repository;

SqlConnection connection;

[TestInitialize]

public void SetUp()

{

connection = new SqlConnection(ConnectionStringUtils.GetConnectionString());

\_repository = new UserRoleRepository(connection);

connection.Open();

}

[TestCleanup]

public void CleanUp()

{

connection.Close();

}

[TestMethod]

public void GetTest()

{

Assert.AreEqual("Operator", \_repository.GetById(2).Role);

}

[TestMethod]

public void GetAllTest()

{

Assert.AreEqual(3, \_repository.GetAll().Count());

}

}

}

**Код класса *Usings.cs***

global using Microsoft.VisualStudio.TestTools.UnitTesting;

**Код класса *ConnectionStringUtils.cs***

using Microsoft.Extensions.Configuration;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UI

{

public class ConnectionStringUtils

{

public static string GetConnectionString()

{

var curDir = Directory.GetCurrentDirectory();

var baseDir = Directory.GetParent(curDir).Parent.Parent;

var config = new ConfigurationBuilder().AddJsonFile(@$"{baseDir}\config.json").Build();

return config["ConnectionStrings:defaultConnection"];

}

}

}

# ПРИЛОЖЕНИЕ Г

(обязательное)

**РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Руководство пользователя содержит материал, необходимый для работы с разработанной программой. В руководстве описывается уровень допустимых знаний пользователя, особенности работы и функционирования системы, порядок действий по внесению и изменению данных, формированию и просмотру отчетов.

Пользователь системы должен обладать следующей квалификацией:

– пользовательские навыки работы с ЭВМ (электронная вычислительная машина);

– понимание предметной области.

Список документов, предоставляемых к обязательному ознакомлению перед началом работы:

– настоящее руководство пользователя;

– указания по технике безопасности при работе с ЭВМ.

Для работы пользователю необходимо запустить исполняемый файл разработанного приложения.

При запуске программы осуществляется вход в систему и открывается окно авторизации, в котором пользователю необходимо ввести логин и пароль, а затем нажать кнопку *«Enter»*. После этого пользователь увидит окно приветсвтия, и для дальнейшей работы необходимо нажать клавишу «*OK*». Внешний вид начального окна программы показан на рисунке Г.1.

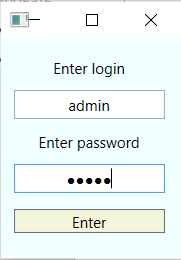


Рисунок Г.1 – Начальное окно программы (окно авторизации)

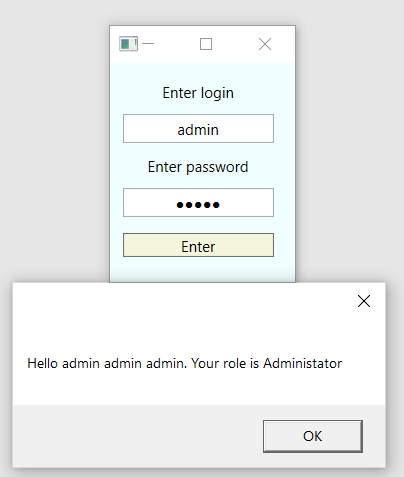


Рисунок Г.2 – Окно приветствия

Вид главного окна зависит от роли пользователя.

В программе первостепенным является пользователь с ролью администратора. При входе администратора в систему открывается окно с вкладками пользователей, маршрутов, видами транспорта и марками автомобилей, в которых администратор имеет возможность просмотра, создания, редактирования и удаления данных, а также формирования отчетов за выбранный период. Внешний вид окна администратора приведен на рисунке Г.3.

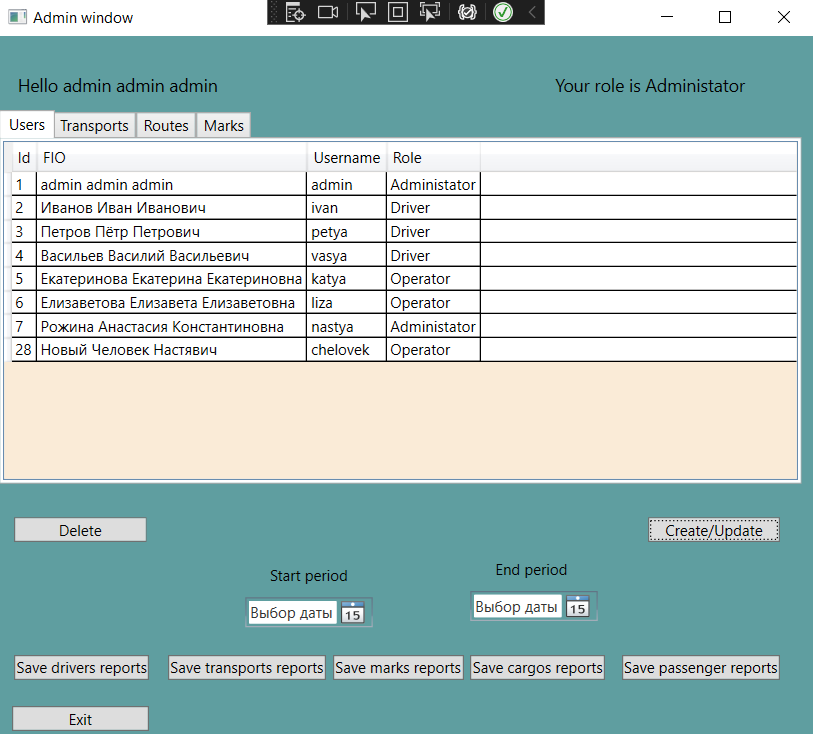


Рисунок Г.3 – Окно администратора

Создавать аккаунты для всех пользователей имеет право только администратор. Для этого необходимо нажать на кнопку *«Create/Update»*. Далее пользователь увидит окно регистрации, где должен заполнить каждое поле и выбрать роль из выпадающего списка. Регистрация будет окончена при нажатии кнопки *«Add/Update»*. Если пользователь захочет изменить ранее добавленные данные, нужно нажать на кнопку *«Create/Update»*, изменить необходимые поля и затем сохранить изменения путём нажатия кнопки *«Add/Update»*. Если пользователь в процессе передумал добавлять данные о пользователях и в последующем о марках автомобилей, путевых листах или транспорте, есть возможность вернуться на вкладку главного окна путём нажатия кнопки *«Exit»*. Если нажать клавишу *«Exit»* на вкладке главного окна, произойдёт выход из системы и перенаправление на окно авторизации. Окно регистрации приведено на рисунке Г.4.

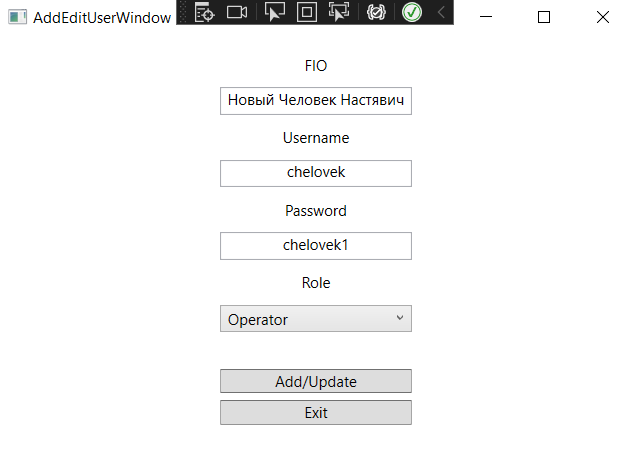


Рисунок Г.4 – Окно регистрации

Администратор может вносить новые марки автомобилей. Для этого нужно перейти на вкладку *«Marks»* и нажать кнопку *«Create/Update»*. На экране появится окно для заполнения марки автомобиля, поле которого нужно заполнить, а затем нажать кнопку *«Create/Update»*. Окно администратора, в котором указаны марки автомобилей и окно для добавления марки автомобиля указаны на рисунках Г.5 и Г.6 соответственно.

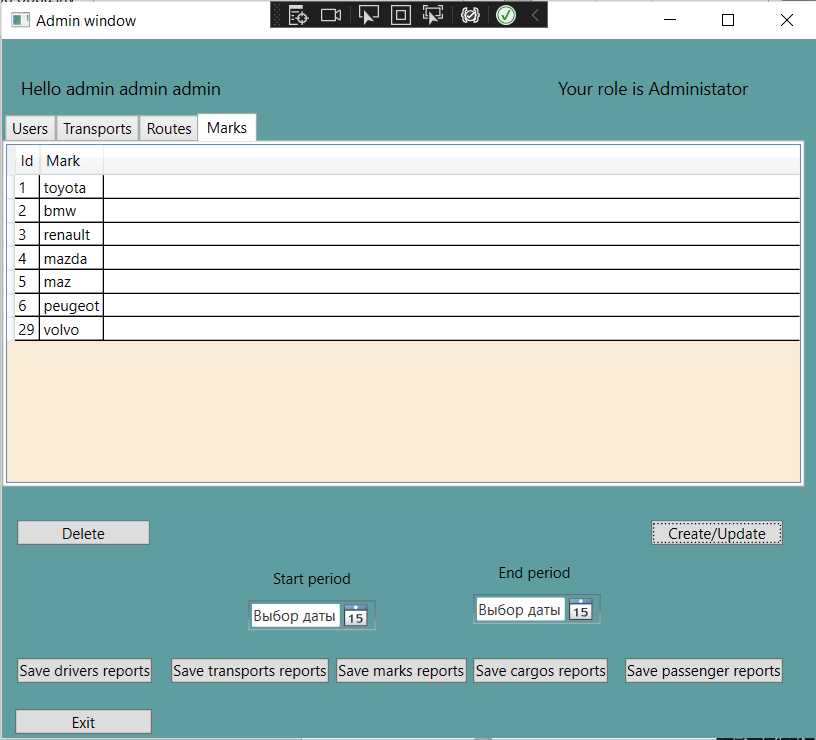


Рисунок Г.5 – Окно администратора, в котором указаны марки автомобилей

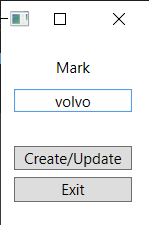


Рисунок Г.6 – Окно добавления марки автомобиля

Помимо марок автомобилей, администратор имеет возможность добавлять данные о видах транспорта. Для этого нужно перейти на вкладку *«Transports»* и нажать кнопку *«Create/Update»*. На экране появится окно для заполнения номера автомобиля, а также два выпадающих списка с типами транспорта и заранее созданными марками автомобилей. После прохождения этапа ввода данных, необходимо нажать кнопку *«Create/Update»*. Окно администратора, в котором указан транспорт и окно для добавления транспорта указаны на рисунках Г.7 и Г.8 соответственно.

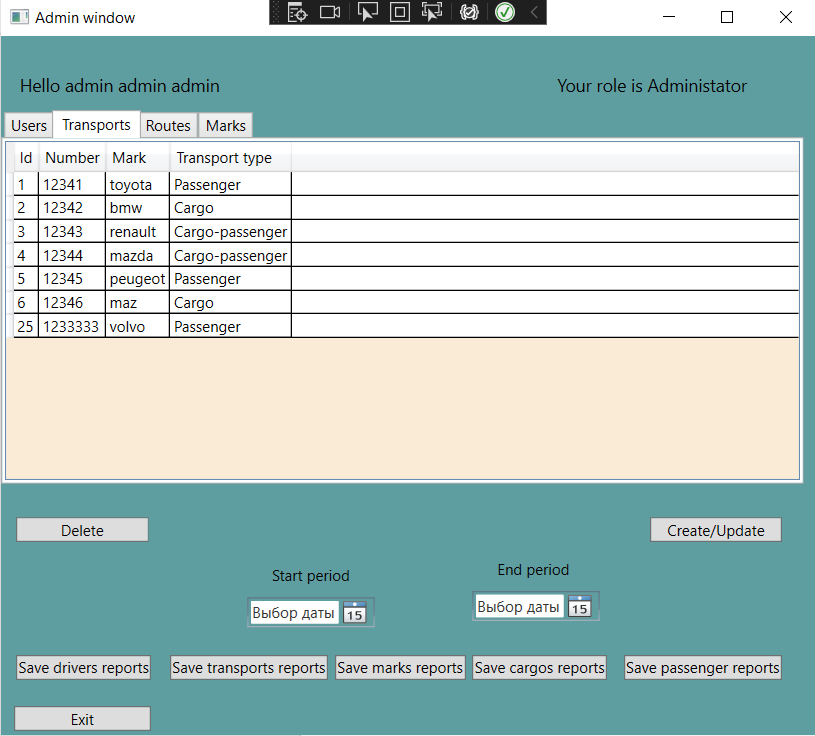


Рисунок Г.7 – Окно администратора, в котором указан транспорт

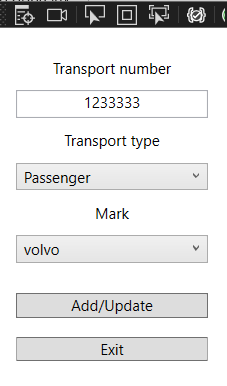


Рисунок Г.8 – Окно добавления транспорта

Администратору доступна возможность взаимодействия с путевыми листами. Для этого нужно перейти на вкладку *«Routes»* и нажать кнопку *«Create/Update»*. На экране появится окно, в котором будут доступны списки с водителями, транспортом, выбором даты маршрута, а также полями для заполнения дистанции, количества груза и пассажиров. Если выбран грузовой транспорт, количество пассажиров должно быть равно нулю, а если пассажирский, то, наоборот, количество груза будет нуль. В противном случае система выдаст сообщение об ошибке. Также нельзя использовать одного водителя на несколько маршрутов в день, иначе будет вывдено окно об ошибке. После корректного ввода данных необходимо нажать кнопку *«Create/Update»*. Окно администратора, в котором указаны маршруты, окно для добавления маршрута, сообщение об ошибке из-за неправильного количества груза, окно об ошибке использования водителя на маршруте указаны на рисунках Г.9, Г.10, Г.11 и Г.12 соответственно.

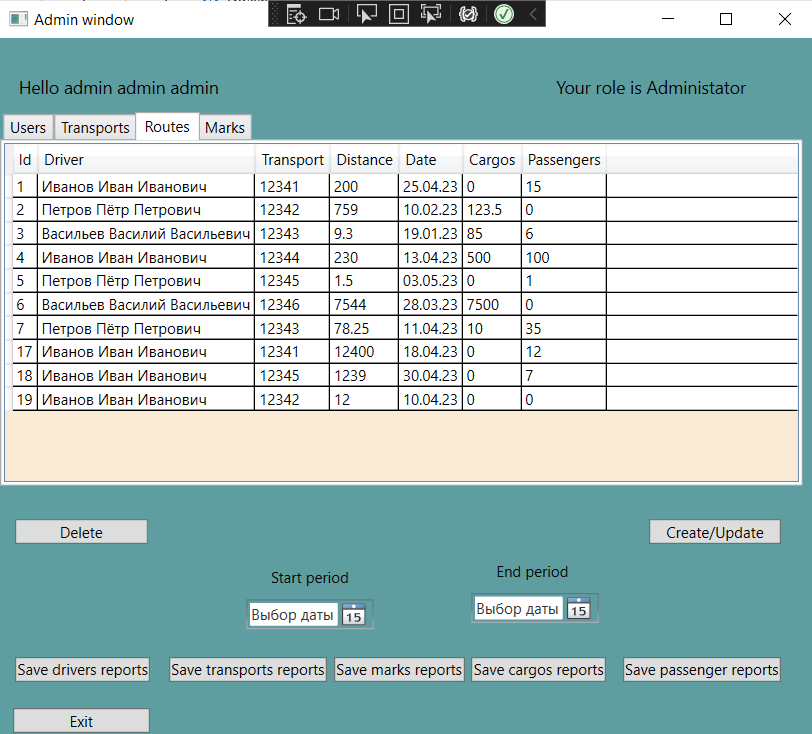


Рисунок Г.9 – Окно администратора, в котором указаны маршруты

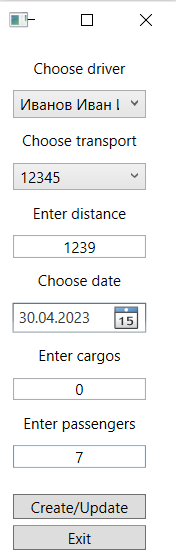


Рисунок Г.10 – Окно добавления маршрутов

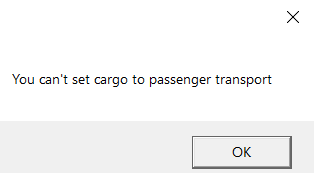


Рисунок Г.11 – Окно об ошибке из-за количества груза

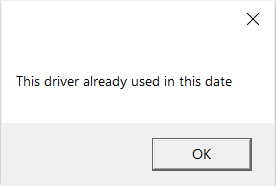


Рисунок Г.12 – Окно об ошибке из-за использования водителя более 1 раза в день на маршруте

Заключительной функцией администратора является сохранение отчётов по любой из категорий. По нажатии на кнопки «*Save drivers reports*», «*Save transports reports*», «*Save marks reports*», «*Save cargos reports*», «*Save passenger reports*» (рисунок Г.3) создаются отчеты в формате *XML*, путь к которым указан на рисунке Г.13.

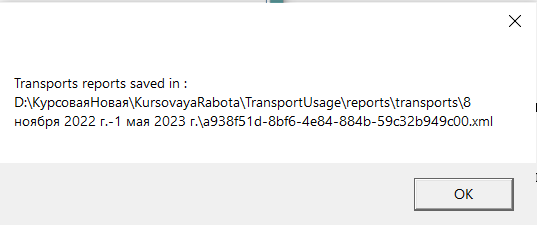


Рисунок Г.13 – Окно, в котором указан путь к сохранённому отчёту

Когда пользователь при авторизации входит под ролью оператора, доступны функции просмотра, создания, редактирования и удаления данных во вкладках маршрутов, марок автомобилей и видов транспорта. Взаимодействовать с ними можно точно так же, как и администратор. Окно оператора представлено на рисунке Г.14.

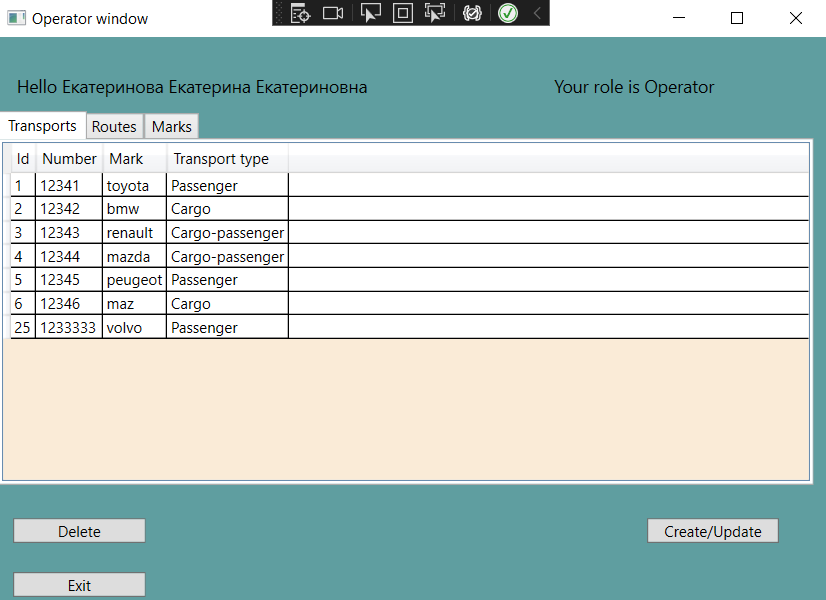


Рисунок Г.14 – Окно оператора

Пользователь с ролью водитель не имеет прав доступа на внесение каких-либо данных. В его возможности входит только просмотр своих путевых листов. Внешний вид окна водителя представлен на рисунке Г.15.

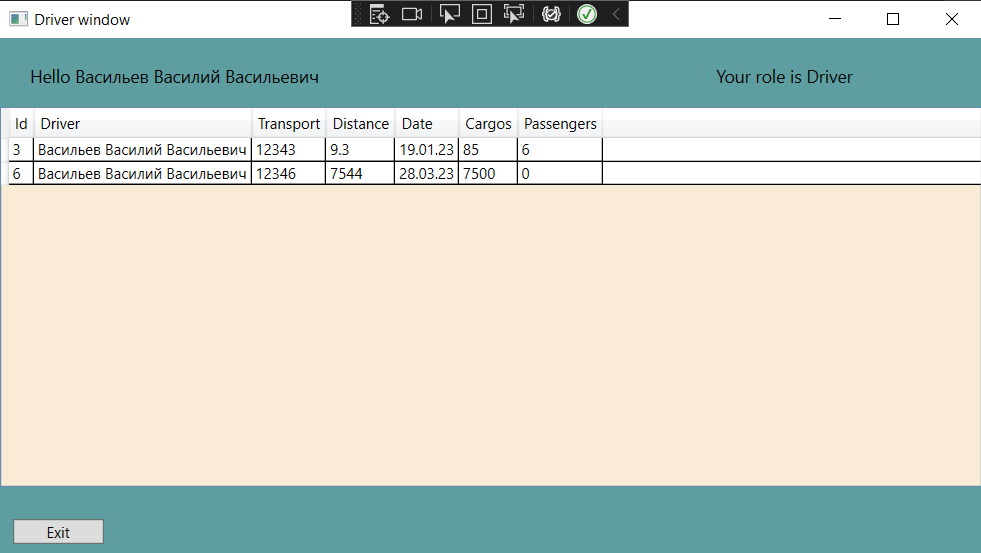


Рисунок Г.15 – Окно водителя

Полностью закрыть систему можно нажав кнопку с символом креста в верхнем правом углу окна.

# ПРИЛОЖЕНИЕ Д

(обязательное)

**РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА**

Программа предназначена для создания информационной системы учета автотранспорта на предприятии. Приложение предоставляет возможность авторизации в системе, взаимодействия с пользователями, маршрутами, видами транспорта и марками автомобиля и составления отчетов за произвольный промежуток времени.

Данное приложение выполняет следующие функции:

− управление данными: добавление, удаление, просмотр, редактирование данных;

− выборка данных для предоставления отчётов;

− получение данных из базы данных.

Для функционирования программно-аппаратного комплекса на стороне клиента необходимо наличие *.exe* файла с приложением.

Для работы программы требуется ЭВМ с установленной операционной системой *Windows*, которая содержитпрограммную платформу .*NET Framework*. Характеристики ЭВМ должны быть достаточными для работы с операционными системами семейства *Windows*: *Windows* 7 / 8 / 8.1 / 10/ 11.

Для функционирования программы не требуется другое специализированное программное обеспечение. Программе не нужна установка, так как она является переносимой и работает в любом каталоге, из которого была запущена.

Для запуска программы нужно запустить файл T*ransportUsage.exe*. После запуска исполняемого файла и ожидания в течение нескольких секунд появляется графическое окно программы.

Проверить работоспособность по авторизации пользователя можно с помощью входа от имени пользователя с правами администратора, который изначально по умолчанию имеет логин «*admin*» и пароль «*admin*».

В случае непредвиденных ошибок требуется перезапустить программу, если ошибки не исчезли рекомендуется проверить корректность введённых данных. Если ошибки так и не удалось устранить, требуется обратится к разработчику программы либо поддерживающему на данный момент её программисту.

# ПРИЛОЖЕНИЕ Е

(обязательное)

**РУКОВОДСТВО ПРОГРАММИСТА**

Разработанное приложение позволяет осуществлять доступ к данным, содержащимся в *LINQ*. При работе с программой пользователю предоставляется доступ к функциям просмотра, добавления, изменения, фильтрации данных. Программа разработана для системы учета использования автотранспорта предприятием.

Для функционирования программно-аппаратного комплекса на стороне клиента необходимо наличие *.exe* файла с приложением.

Никаких ограничений и требований для работы и запуска программы не налагается. Для контроля правильности выполнения программы предусмотрены сообщения пользователю о некорректно введённых данных.

Входными данными для приложения являются данные из базы данных, а также данные, вводимые пользователем в программе, где это необходимо.

К выходным данным можно отнести результаты выполнения различных действий в программе, а также различные сообщения о некорректном вводе.

При регистрации пользователю необходимо ввести имя, логин и пароль. В программе используются механизмы валидации вводимых пользователем данных. При неправильном или некорректном вводе появляются соответствующие сообщения об ошибках.

Если ввести логин несуществующего пользователя или неверный пароль для существующего пользователя, то появится сообщение об ошибке неправильного логина или пароля. Для устранения ошибки нужно ввести логин пользователя, который сохранён в системе.