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 проектирование и программирование»

на тему «*WPF* приложение для учёта использования автотранспорта

предприятием»

Исполнитель: студент гр. ИТП-21

Шинкарук В.Г.

Руководитель: доцент

Курочка К.С.

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

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

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

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

Подписи членов комиссии

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

Гомель 2024

**СОДЕРЖАНИЕ**

Введение

1 Технические подходы к реализации приложения для учёта использования автотранспорта 4

1.1 Теоретические сведение о среде разработки 4

1.2 Средства для создания приложения 6

1.3 Сравнение выбранной базы данных и языка с аналогами 9

1.4 Методы разработки приложения 11

2 Архитектура разработанного приложения 12

2.1 Общая архитектура приложения 12

2.2 Структура базы данных приложения 13

2.3 Структура классов приложения 15

2.4 Общая структура окон приложения 18

2.5 Итоговые *XML*-отчёты 19

3 Этапы проведения верификации и тестирования информационной системы 21

3.1 Результаты модульного тестирования 21

3.2 Основные окна программы 22

3.3 Результаты ручного тестирования 24

Заключение 30

Список использованных источников 31

Приложение А Функциональная схема приложения 32

Приложение Б Графический интерфейс пользователя 33

Приложение В Листинг программы 36

Приложение Г Листинг модульных тестов 83

Приложение Д Руководство пользователя 92

Приложение Ж Руководство программиста 100

Приложение И Руководство системного программиста 101

**ВВЕДЕНИЕ**

В эру информационных технологий, когда цифровизация проникает во все аспекты нашей жизни, программные решения становятся неотъемлемой частью эффективного функционирования любой сферы деятельности. Это касается и транспортной отрасли, где учет и контроль за автопарком, организация перевозок и обработка сопутствующей документации требуют современного программного обеспечения для обеспечения прозрачности, точности и оперативности управления.

Транспортные предприятия сталкиваются с необходимостью автоматизации управления значительным числом автомобилей, организацией грузовых и пассажирских перевозок. Для удобства управления данным и надёжного хранения их в базе данных необходимо применять все существующие средства управления и автоматизации.

Помимо простого управления данными, система должна быть безопасной и надежной, чтобы не вызывать никаких искривлений в данных. Современные технологии учитывают и этот момент, поэтому правильная разработка даёт положительный результат для всего предприятия.

Учитывая необходимость обеспечения многопользовательского доступа с разграничением прав доступа для разных категорий сотрудников (администраторы, операторы, водители), становится очевидной актуальность разработки такого программного обеспечения, которое сможет предоставлять только нужные данные и только тому, кому они доступны.

Таким образом, разработка специализированного программного продукта для учёта использования автотранспорта предприятия является не только актуальной, но и необходимой задачей, решение которой позволит повысить качество управления информации и обеспечить безопасное использование её без потери данных.

**1 ТЕХНИЧЕСКИЕ ПОДХОДЫ К РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ ДЛЯ УЧЁТА ИСПОЛЬЗОВАНИЯ АВТОТРАНСПОРТА**

* 1. **Теоретические сведения о среде разработки**

В современном мире существует множество различных языков программирования, имеющих свои преимущества и недостатки. Данный курсовой проект написан на языке *C#* – объектно-ориентированном языке программирования, работающего на платформе *.NET Framework*. На *C#* будет создано приложение с интерфейсом *Windows Presentation Foundation (WPF)* для учета автотранспорта предприятия. Данные для приложения будут храниться в базе данных *Microsoft SQL Server*, а взаимодействие с базой осуществляться через *Language Integrated Query (LINQ)*.

*.NET Framework*, разработанный *Microsoft* в 2002 году, изначально был ответом на популярность *Java* от *Sun Microsystems* (теперь часть *Oracle*). Основу *.NET* составляет *Common Language Runtime (CLR)* – среда выполнения, поддерживающая множество языков программирования и предоставляющая обширные возможности для разработки. Платформа запатентована *Microsoft* и оптимизирована для *Windows*, хотя проекты вроде *Mono* и *Portable.NET* позволяют запускать .*NET*-приложения и на других операционных системах, что может сопровождаться специфическими ошибками. Современное развитие платформы представлено *.NET Core*, фокусирующимся на кроссплатформенности и выборочном использовании компонентов для улучшения производительности в разработке.

*C#* является типобезопасным, объектно-ориентированным языком, спроектированным для создания различных приложений на платформе *.NET Framework*. Язык отличается чистотой, строгостью и простотой, что облегчает процесс разработки. *C#* сочетает в себе новаторство, позволяя быстро разрабатывать приложения, при этом сохраняя элегантность и выразительность, характерные для современных языков программирования.

*Visual C#* от *Microsoft* представляет собой развитие языка *C#* с поддержкой в среде разработки *Visual Studio*. В *Visual Studio* разработчики имеют доступ к обширным средствам, включая редактор кода, инструменты для проектирования, шаблоны проектов и мощный отладчик. *.NET Framework* предлагает богатую библиотеку классов, упрощающих доступ к системным службам и ускоряющих процесс разработки.

Основными преимуществами языка *C#* являются:

– объектно-ориентированный язык;

– управляемый код и автоматическое управление памятью;

– богатая стандартная библиотека;

– интеграция с *.NET Framework* и *.NET Core*;

– поддержка *LINQ* для работы с данными;

– интеграция с *Visual Studio* для удобной разработки.

*SQL* (*Structured Query Language*, или язык структурированных запросов) – это декларативный язык программирования (язык запросов), использующийся для создания, обработки и хранения данных в реляционных базах данных. Чистый *SQL* не предоставляет возможности написания – он предназначен только для взаимодействия с базами данных: получения, добавления, изменения и удаления информации в них, управления доступом и другими возможностями, реализованными в базах данных. Данные в реляционной базе данных хранятся в виде таблиц, они структурированы и разложены по строкам и столбцам для облегчения процесса оперирования таблицами. В целом, способ хранения информации, описанный выше, и есть реляционная база данных. Такое название легко объяснить переводом (от англ. *relation* – «отношения»). Название указывает на то, что объекты в такой базе связаны определенными отношениями. *SQL* настолько тесно связан с реляционными базами, что все нереляционные базы данных в противовес со временем стали называть *NoSQL*. Тем самым получилось, что *SQL* – это язык программирования, а *NoSQL* – тип баз данных.

Основными преимуществами *MS* *SQL* над другими системами управления базами данных являются:

– высокая оптимизация для больших объемов данных;

– глубокая интеграция с другими продуктами *Microsoft*;

– масштабируемость и поддержка высокой доступности;

– гибкость в развертывании.

*ADO.NET* является компонентом платформы *.NET*, предназначенным для связи приложений с базами данных, позволяя управлять данными по мере необходимости. Это инструмент, который облегчает доступ приложений к данным из реляционных баз данных и других источников, и позволяет изменять и обрабатывать эти данные. Важно понимать, что *ADO.NET* само по себе не является системой управления базами данных, а предоставляет разработчикам интерфейс для взаимодействия с разнообразными базами данных.

Архитектура *ADO.NET* строится вокруг двух основных элементов: поставщиков данных и *disconnected model*. Ключевую роль также играет компонент *DataAdapter*. Поставщик данных, представляющий собой набор библиотек, позволяет приложениям подключаться к источникам данных и извлекать из них информацию, используя объекты, такие как *Connection* и *DataReader*. В зависимости от используемой базы данных выбирается соответствующий поставщик, например, для работы с *SQL Server* используется специфический поставщик данных.

*Disconnected model*, или автономная модель, предназначена для хранения данных на стороне клиента. Основой этой модели является *DataSet*, который можно представить как виртуальную базу данных, хранящую данные, извлеченные из реальной базы. *DataSet* может включать данные из разных источников, таким образом обеспечивая работу с множественными базами данных в одной модели. Для синхронизации изменений данных между *DataSet* и базой данных применяется объект *DataAdapter*, который передает изменения из автономной модели обратно в базу данных.

В отличие от предыдущей версии *ADO*, которая была ориентирована на тесное взаимодействие в клиент-серверных приложениях, *ADO.NET* акцентирует внимание на работе с данными в автономном режиме через объекты *DataSet*. Эти объекты представляют собой копии данных, которые могут быть легко манипулированы и обновлены без необходимости постоянного подключения к источнику данных. *DataSet* поддерживает работу с данными в приложениях, таких как веб-страницы или настольные программы, позволяя обрабатывать и отправлять изменения данных обратно в базу данных с помощью *DataAdapter*.

Особенностью *ADO.NET* является то, что оно представляет собой управляемую коллекцию классов, следующую правилам платформы *.NET*. Это означает, что все компоненты *ADO.NET*, включая классы, интерфейсы и другие типы, подчиняются правилам управления памятью *CLR* и могут быть использованы в любом языке программирования *.NET*. Классы *ADO.NET* находятся в сборке *System.Data.dll*, обеспечивая тесную интеграцию с остальной частью платформы *.NET*.

**1.2 Средства для создания приложения**

Для разработки приложения по учёту использования и обслуживания автотранспорта предприятия, состоящего из пассажирского, грузового и грузопассажирского транспорта, важно подобрать правильно сочетание программ. Эти программы должны обеспечить надежность, производительность, масштабируемость и удобство разработки и поддержки. Использование СУБД *MS SQL Server* в качестве источника данных, языка программирования *C#* с *WPF*-приложением и *LINQ* для доступа к базе данных является крайне правильным подходом для реализации приложения для учёта автотранспорта предприятием по нескольким причинам.

Главная из них – надежность и безопасность с *MS SQL Server*.

При разработке корпоративных приложений важно выбирать проверенные решения, которые гарантируют безопасность и надежность хранения данных. *MS SQL Server* – это не просто база данных. Это целая экосистема, предлагающая мощные инструменты для администрирования, анализа и безопасного хранения информации. Благодаря ей, не возникнет проблем с разделением данных. Данная среда позволяет создать отдельные таблицы под водителей, транспортные средства, рейсы и путевые листы. В сравнении с другими системами управления базами данных, *MS SQL Server* выделяется своей способностью обрабатывать огромные объемы данных с высоким уровнем производительности и надежности, что критически важно для корпоративных приложений. *MS SQL Server* предлагает глубокую интеграцию с другими продуктами и сервисами *Microsoft*, такими как *Visual Studio* и *.NET Framework*. Это позволяет работать в уже привычной среде и иметь поддержку от множества существующих ресурсов. Для приложения можно эффективно использовать *ADO.NET* для доступа к данным, *LINQ* для их обработки и *WPF* для создания пользовательского интерфейса, опираясь на сильные стороны каждой из этих технологий. Все они отлично работают вместе и дадут в результате мощное приложение с большим функционалом.

*MS SQL Server* имеет встроенные функции обеспечения безопасности, такие как шифрование данных, аутентификация на уровне сервера и базы данных, а также поддержка политик паролей и управление доступом на основе ролей. Это особенно важно для корпоративных приложений, где требуется защита конфиденциальных данных о транспортных средствах, водителях и маршрутах. Как раз данное приложение для учёта автотранспорта будет гарантированно давать доступ к данным только тем, кто эту информацию может получать в соответствии с ролями пользователей (администратор, оператор, водитель).

*MS SQL Server* поддерживает работу с огромной базой данных. Даже если автомобилей или водителей будет десятки тысяч, приложение будет эффективно обрабатывать данные без потери скорости работы и актуальности информации.

Следующая причина – производительность и удобство разработки с *C#*.

*C#* является мощной средой разработки. Этот язык программирования обладает огромной популярностью, делая написание приложения именно на нём важным в современном мире, а также полезным с точки зрения имеющихся возможностей в языке. В нём разработчики обеспечены всем необходимым для создания мощных и эффективных приложений.

*C#* сочетает в себе производительность компилируемых языков и удобство интерпретируемых. Благодаря оптимизациям *.NET CLR* (*Common Language Runtime*), приложения на *C#* могут предложить высокую скорость выполнения и эффективное управление ресурсами. Это критически важно для приложения, где обработка большого количества данных о рейсах, автомобилях и водителях должна происходить быстро и без задержек.

*C#* тесно связан с *.NET Framework*, что даёт доступ к широкому спектру фреймворков и библиотек, ускоряющих процесс разработки. Также большое количество возможностей языка положительно влияет на эффективность реализации из-за возможности сосредоточится на разработке самой логики приложения.

Более того, *C#* является объектно-ориентированным языком программирования, что облегчает создание модульных и легко поддерживаемых приложений. ООП подход облегчает моделирование реальных сущностей системы, таких как автомобили, водители и рейсы, в виде объектов с соответствующими свойствами и методами. Объединение всех принципов объектно-ориентированного языка в один механизм делает его удобным, быстрым, надёжным и качественным, а следовательно, и саму программу, обладающую теми же свойствами.

Поддержка *C#* разработки модульных тестов способствует созданию надежного кода и обеспечению высокого качества приложения. Использование фреймворков для модульного тестирования, таких как *NUnit* или *MSTest*, позволяет автоматизировать тестирование компонентов системы, убедиться в их корректной работе и предотвратить регрессию функциональности при внесении изменений.

Также важным является практичность *WPF*.

*WPF* (*Windows Presentation Foundation*) представляет собой ключевую составляющую при выборе технологии для разработки настольных приложений, особенно когда речь идет о создании сложных и интерактивных пользовательских интерфейсов. Его роль в разработке приложений выходит за рамки простого создания визуальных элементов, предлагая глубокую интеграцию с *.NET Framework* и широкий спектр возможностей для дизайнеров и разработчиков.

*WPF* использует векторную графику. Это обеспечивает чёткое отображение элементов интерфейса при использовании любого разрешения экрана, а также при смене масштаба. Тем самым, интерфейсы выглядят одинаково хорошо на различных устройствах, от экранов телефонов до стандартных мониторов и даже проекторов. Продвинутые возможности *WPF* по работе со стилями позволяют создавать не только красивые, но и функционально разнообразные пользовательские интерфейсы без необходимости изменения логики приложения. В нём можно реализовать различные динамические анимации и применять различные стили, что может сильно повысить вовлеченность пользователя, однако для приложения для учёта автотранспорта предприятием анимации трудно назвать крайне полезной возможностью, но темы могут быть крайне полезны. Стили и темы могут быть использованы для выделения важной информации, улучшения навигации, да и в целом помочь сделать приложение красивее. *WPF* позволяет быстро адаптировать интерфейс под цвета и логотипы предприятия, что делает приложение сразу особенным и привлекательным для пользователей.

*Windows Presentation Foundation* поддерживает модель разработки *MVVM* (*Model-View-ViewModel*) – шаблон проектирования архитектуры приложения. В технологии *WPF* есть концепция «связывания данных», позволяющая связывать данные с визуальными элементами в обе стороны. *MVVM* способствует разделению пользовательского интерфейса от бизнес-логики приложения. Поскольку логика интерфейса отделена от бизнес-логики, вносить изменения в пользовательский интерфейс становится проще, так как не обязательно затрагивать основную логику приложения. Это упрощает тестирование, поддержку и разработку приложений. В контексте приложения для учёта использования автотранспорта предприятием эта модель позволит легко реализовать разные представления и функциональности для разных типов пользователей (администратор, оператор, водитель), так как каждый из них должен иметь собственные права доступа, и тот же водитель должен видеть только часть имеющейся базы.

Удобство *LINQ* также помогает сделать приложение простым в разработке и понятным для других пользователей.

*LINQ* (*Language Integrated Query*) представляет собой мощный компонент языка *C#*, обеспечивающий единообразный подход к запросам данных из различных источников, будь то коллекции объектов в памяти, базы данных или *XML*-файлы. *LINQ* играет ключевую роль в облегчении и ускорении разработки благодаря своей гибкости и мощности.

Вместо написания сложных запросов *SQL* или использования традиционных методов доступа к данным, можно использовать выразительный и типобезопасный синтаксис *C#* для формирования запросов непосредственно в коде. Это значительно упрощает работу с данными, делая код более читаемым и уменьшая вероятность ошибок.

С течением времени *LINQ* развивался, предлагая новые оптимизации и улучшения производительности. Это означает, что запросы, написанные с использованием *LINQ*, могут быть не только более выразительными, но и более эффективными с точки зрения выполнения. Оптимизация запросов *LINQ*, особенно в *LINQ to SQL*, помогает минимизировать нагрузку на базу данных и ускорить обработку данных.

Также *LINQ* предоставляет широкие возможности для выполнения сложных операций с данными, таких как фильтрация, сортировка, группировка и агрегация, используя лаконичный и понятный синтаксис. Учитывая необходимость расчёта среднего пробега по марке автомобиля или количеству рейсов у водителя, сложные запросы будут создаваться упрощенно.

**1.3 Сравнение выбранной базы данных и языка с аналогами**

Для убеждения, что для приложения выбраны наилучшие средства разработки необходимо сравнить их с аналогичными и выделить преимущества.

Сравнив *MS SQL Server* с другими популярными СУБД, такими как *MySQL* и *PostgreSQL*, можно выделить несколько причин, по которым *MS SQL Server* представляется более предпочтительным выбором для данной задачи. В первую очередь он глубоко интегрирован с другими продуктами *Microsoft*, что обеспечивает беспрепятственную работу в среде, основанной на технологиях *Microsoft*. Это включает в себя тесную интеграцию с *.NET Framework, Visual Studio и Azure*, предоставляя разработчикам удобные инструменты для создания, тестирования и развертывания приложений. Он известен своей высокой производительностью и масштабируемостью, что позволяет обрабатывать большие объемы данных. В отличие от *MySQL*, который традиционно оптимизирован для веб-приложений и легких операций, *MS SQL Server* предлагает более продвинутые возможности для обработки сложных транзакций. Также он поддерживается обширной документацией, сообществом разработчиков и профессиональной поддержкой от *Microsoft*, что обеспечивает доступ к ресурсам для решения возникающих вопросов и проблем. Да, *MySQL* и *PostgreSQL* также имеют активные сообщества, но широта и глубина поддержки, предоставляемой *Microsoft*, гораздо больше.

Выбор *C#* в качестве основного языка программирования для разработки информационной системы учета использования и обслуживания автотранспорта предприятия обусловлен рядом весомых причин. Первая из них – это снова специфика разработки. Он разработан специально для работы в экосистеме *.NET*, что обеспечивает беспрепятственную интеграцию с множеством полезных библиотек и инструментов. *C#* полностью поддерживает объектно-ориентированное программирование, предоставляя возможности для создания четко структурированного и модульного кода. ООП упрощает проектирование сложных систем, позволяя разбивать приложение на объекты и классы, что облегчает разработку, тестирование и поддержку программы. Если сравнить принципы работы с базами данных в других языках программирования, то можно прийти к выводам, что *Java* предоставляет широкие возможности для работы с различными СУБД через *JDBC* и *JPA*, однако настройка и оптимизация подключения к базам данных могут потребовать больше усилий по сравнению с интегрированными решениями в *C#*. А *Python* обладает гибкостью в выборе СУБД и простотой кода при работе с базами данных через *ORM*, но может уступать по производительности и возможностям оптимизации запросов, доступным в *C#* и .*NET*.

*WPF* (*Windows Presentation Foundation*) и *WinForms* (*Windows Forms*) – две популярные технологии от *Microsoft* для создания настольных приложений на платформе *.NET*. В то время как обе технологии позволяют разрабатывать интерфейс пользователя, между ними есть существенные различия, делающие *WPF* более предпочтительным выбором для многих сценариев разработки, особенно в контексте современных приложений. *WinForms* опирается на растровую графику, что может приводить к потере качества при масштабировании или на дисплеях высокой четкости, в отличие от векторной, которая позволяет изменять масштаб без потери качества. Также *WinForms* традиционно следует подходу с сильной связью между пользовательским интерфейсом и бизнес-логикой, что может негативно сказаться на соблюдении всех правил объектно-ориентированного программирования, а также *WinForms* не имеет модель *MVVM*, которая обеспечивает четкое разделение логики приложения и пользовательского интерфейса.

**1.4 Методы разработки приложения**

В ходе анализа было выявлено, что использование C# в сочетании с .*NET Framework*, *WPF* для разработки интерфейса и *MS SQL Server* для управления базой данных является наиболее подходящим решением для создания информационной системы по учету использования и обслуживания автотранспорта предприятия. Этот выбор обусловлен несколькими ключевыми факторами, включая тесную интеграцию технологий, высокую производительность, гибкость в разработке и обширную поддержку со стороны *Microsoft*.

Преимущества использования *C#* заключаются в его мощности и гибкости, что делает язык идеальным для создания надежных и масштабируемых приложений. Объектно-ориентированный подход *C#*, интеграция с *.NET Framework* обеспечивает удобную разработку и эффективное управление памятью.

*WPF* выделяется векторной графикой, поддержкой разделения логики и интерфейса через модель *MVV.*

*MS SQL Server* был выбран как система управления базой данных из-за его высокой производительности, надежности, обширных функций безопасности и масштабируемости. Глубокая интеграция с продуктами *Microsoft*, включая *Visual Studio* и .*NET Framework*, упрощает разработку и обеспечивает эффективное взаимодействие приложения с базой данных.

*LINQ* значительно упрощает работу с данными, предоставляя единообразный и выразительный способ запросов к коллекциям объектов, базам данных и *XML*.

В целом, выбор данных технологий для разработки информационной системы учета автотранспорта предприятия обусловлен их взаимной совместимостью, обширной функциональностью и поддержкой со стороны *Microsoft*, что обеспечивает высокую производительность приложения, его надежность и легкость в поддержке и развитии.

**2 АРХИТЕКТУРА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

**2.1 Общая архитектура приложения**

Правильный анализ до проектирования имеет огромную роль для обеспечения стабильной работы и правильного функционирования всего приложения. Изучение особенностей программного продукта обеспечивает понимания необходимых требований и ожиданий в функциональных и технических задачах приложения. Для упрощения добавления новых функций, различных возможностей, архитектура приложения должна быть модульной и обеспечивать простое и понятное обслуживания. Основными параметрами, которым должно соответствовать приложение, являются:

– корректное и чёткое отображения информации;

– безопасность;

– высокая производительность;

– простота обновления.

В соответствие с правилами объектно-ориентированного программирования, приложение должно быть разделено на несколько основных частей.

– слой взаимодействия с базой данных;

– интерфейсы, определяющие взаимодействие с другими слоями;

– слой моделей, содержащий основную структуру данных приложения. Он используется для представления и передачи данных между различными уровнями приложения;

– классы сервисов, предназначенные для обработки данных, получаемых из репозиториев и их последующую передачу в пользовательский интерфейс. Данный слой содержит бизнес-логику приложения;

– пользовательский интерфейс, реализованный с использованием технологии *WPF*. Предназначен для определения того, как данные будут отображаться и как пользователь сможет взаимодействовать с приложением;

Для разработки приложения был выбран подход, основанный на удовлетворении всех заданных требований объектно-ориентированного языка и принципов *SOLID* с возможностью быстрой и понятной для пользователя работой. Основной подход имеет вид:

– создание базы данных;

– разработка слоя доступа к данным и всех необходимым методов для последующего изменения;

– создание бизнес-классов для обработки данных со слоя доступа;

– обеспечение вывода информации, а также выполнения различных операций путём создания слоя *WPF* интерфейса.

– тестирование созданных методов путём написания модульных тестов.

На рисунке 2.1 представлена общая схема архитектуры приложения.



Рисунок 2.1 – Общая схема архитектуры приложения

**2.2 Структура базы данных приложения**

Основными сущностями, исходя из имеющегося задания, можно выделить:  
 – путевые листы;

– пользователи;

– транспорт;

– марки транспорта.

Все сущности содержат дополнительные поля, позволяющие сделать логику приложения более широкой и многогранной.

Для обеспечения эффективной работы приложения и корректного анализа данных необходим процесс нормализации данных. Нормализация данных – это процесс организации информации в базе данных таким образом, чтобы избежать избыточности и обеспечить целостность данных. Для этого были выбраны таблицы таким образом, чтобы обеспечить отсутствие повторяющихся групп данных. Также были добавлены связи с помощью ключевых полей для правильно взаимодействия таблиц базы между собой. После процесса нормализации данные становятся более структурированными и легко управляемыми, что улучшает производительность приложения и уменьшает риск ошибок при работе с данными.

Таблица *TripSheets*, представляющая собой таблицу путевых листов, содержит поля, позволяющие не только обеспечивать взаимодействие с другими таблицами базы данных, но и выделять различные типы транспорта, в соответствии с заданием:  
 – идентификатор *TripSheetID*, содержащий уникальное значение для каждого путевого листа;

– идентификатор *VehicleID*, предназначенный для связи с таблицей *Vehicles*;

– *Date*, представляющий собой дату выполненной поездки;

– Расстояние, которое проехал автомобиль, – *Distance*;

– *CargoAmount*, предназначенная для хранения размера груза, перевезенного грузовым и грузопассажирским транспортом;

– *PassengerAmount*, предназначенная для хранения количества пассажиров, перевезенного пассажирским и грузопассажирским транспортом.

Для полноценного управления системой, обеспечения полной безопасности в доступном функционале и защиты от возможного изменения данных от не имеющих должных прав пользователей, в приложении реализован многопользовательский уровень доступа. Для этого выделено специальное поле в таблице пользователей. Сама таблица имеет вид:

– идентификатор *DriverID*, предназначенный для хранения уникального значения пользователя. Он имеет такое название из-за превосходящей роли водителя в управлении данными;

– *Username* для хранения логина пользователя;

– *FIO*, содержащий фамилию, имя и отчество пользователя системы;

– *PasswordHash*, представляющий собой пароль. В процессе работы программы он хешируется и хранится в базе зашифрованным.

Таблица марок транспорта предназначена для хранения названия каждой марки и последующего использования в *Vehicles*. Содержание таблицы *Marks*:  
 – идентификатор *MarkID*, содержащий уникальное значение для каждой марки автомобиля;

– *Name*, представляющий собой название марки автомобиля.

Для описания сущности транспорта был разработана таблица в базе данных *Vehicles*. Она содержит такие поля:

– идентификатор *VehicleID* для хранения уникального значения транспорта;

– идентификатор *MarkID* для хранения марки автомобиля и связи с таблицей Marks;

– *TypeVehicle* представляет собой тип транспорта;

– *PassengerCapacity*, содержащий вместимость пассажиров для пассажирского и грузопассажирского транспорта;

– *CargoCapacity*, представляющий собой грузоподъёмность грузового и грузопассажирского транспорта;

– *RegistrationNumber*, являющийся регистрационным номером транспорта. В базе данных он заполнен разным видом значений для отличия разных типов транспорта между собой.

На рисунке представлена схема базы данных на рисунке 2.2.

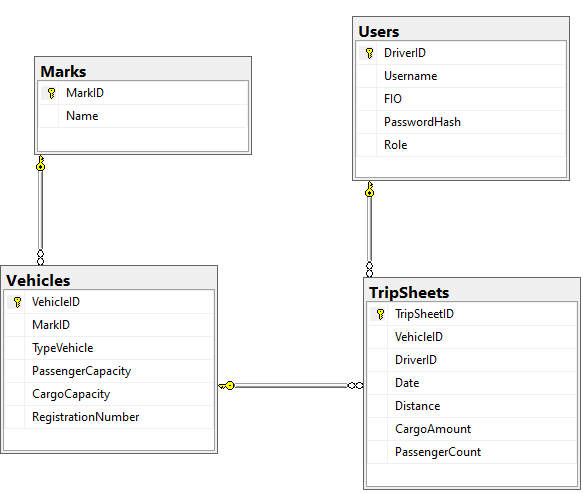


Рисунок 2.2 – Схема базы данных

**2.3 Структура классов приложения**

Классы приложения можно разделить на две разные группы. В программе они выделяются как *DataLayer* для управления данными и *ViewLayer* для представления этих данных.

*DataLayer* предназначен для обработки операций с базой данных и включает классы бизнес-логики. В данном слое выделяется 3 основных группы, предназначенных для работы с данными: *Models*, *Repositories*, *Services*.

M*odels* содержит классы-модели для работы с базой данных. Они отражают основные сущности базы данных. Благодаря им, облегчается взаимодействие с базой данных, поскольку они представляют столбцы таблиц в виде объектов. В приложении Models разделен ещё на 3 группы: *Transport*, *User*, *Reports*. Первые две предназначены для разделения транспорта и пользователей на разные типы, а *Reports* для генерации отчётов. Структура основных классов *DataLayer* приведена на рисунке 2.3.

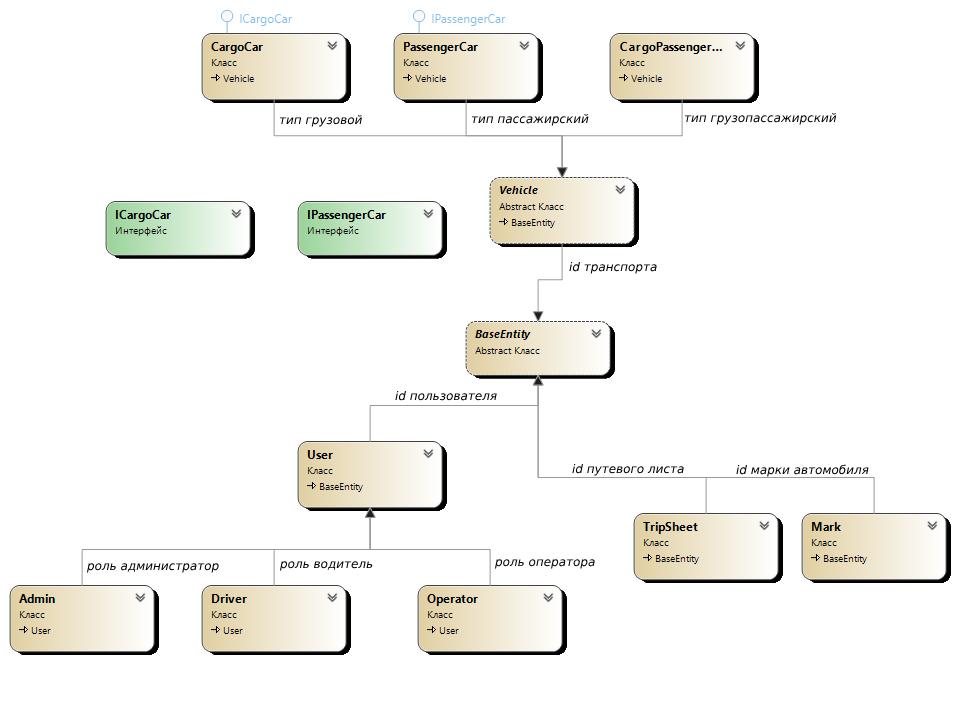


Рисунок 2.3 – Структура основных классов *DataLayer*

Библиотека обеспечивает интерактивность с базой данных через шаблон «*Repository*». В рамках этого подхода был разработан интерфейс «*IRepository*», который поддерживает базовые *CRUD* операции, такие как получение (*Get*), получение всех данных (*GetAll*), создание (*Create*), обновление (*Update*) и удаление (*Delete*). Классы-сервисы в этой библиотеке задают правила, согласно которым пользователи будут взаимодействовать с данными.

Слой *Services* обеспечивает реализацию бизнес-логики и координацию различных операций за счёт изоляции слоя доступа к данным (*Repository*) от остальных частей приложения. На рисунке 2.4 представлена структура классов *Repository* и *Service* в *DataLayer*.

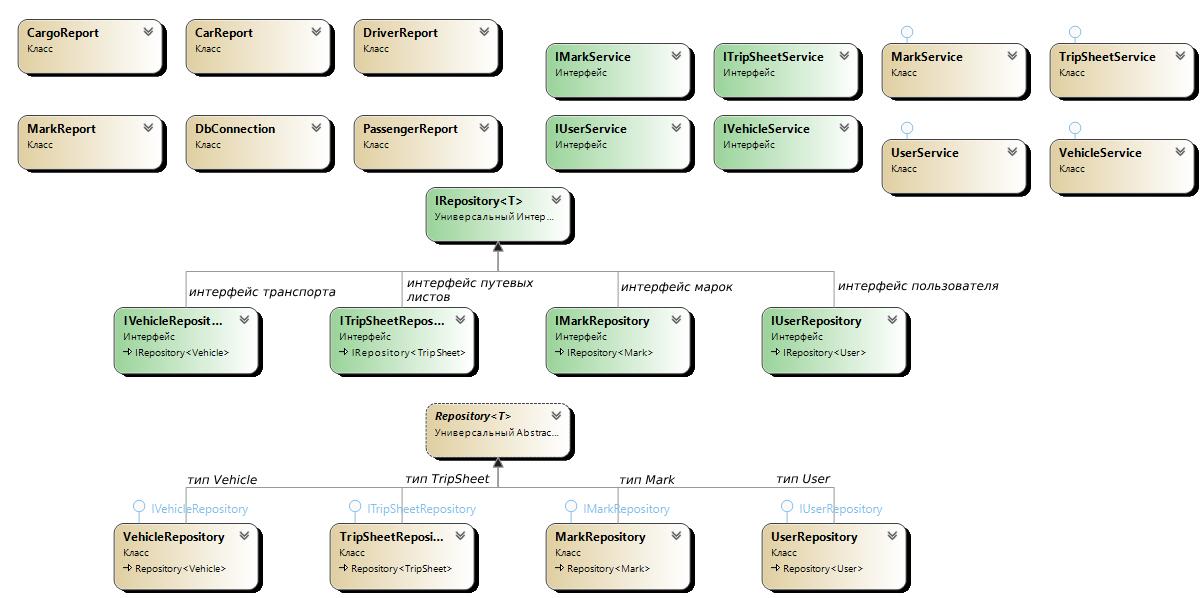


Рисунок 2.4 – Структура классов *Repository* и *Service* в *DataLayer*

Слой *ViewLayer* отвечает за предоставление графического интерфейса, который пользователи могут использовать для взаимодействия с функционалом приложения.

Графический интерфейс приложения, реализованный в *ViewLayer*, включает в себя пользовательские элементы управления и все необходимые окна для всех категорий пользователей.

Для управления подключением к базе данных в приложении используется класс *DbConnection*. Он использует файл конфигурации *app.config*. Чтобы избежать создания множества соединений и эффективно использовать ресурсы, используется паттерн «*Singleton*». Статическое свойство «*Instance*» возвращает только один экземпляр подключения. Интерфейса содержит следующие окна:

– окно авторизации – *LoginWindow.xaml*;

– окно регистрации – *RegistrationWindow.xaml*;

– окно администратора – *AdminWindow.xaml*;

– окно оператора – *OperatorWindow.xaml*;

– окно водителя – *DriverWindow.xaml*.

Листинг программы приведен в приложении Б.

Структура классов *ViewLayer* приведена на рисунке 2.5.

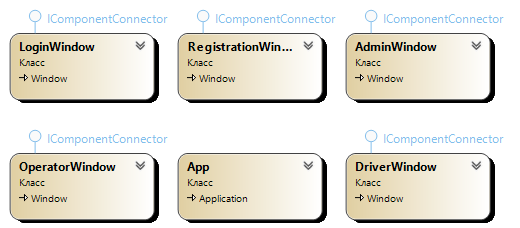


Рисунок 2.5 – Структура классов *ViewLayer*

**2.4 Общая структура окон приложения**

Пользовательский интерфейс (*UI*) разработан с учетом необходимости наличия различных типов пользователей в приложении. Для этого были разработаны различные окна с уникальной функциональностью для различных категорий пользователей: администраторов, операторов и водителей. Все эти окна находятся в рамках проекта *UI*.

Окно авторизации отвечает за проверку подлинности пользователей. В случае правильного ввода данных в полях логина и пароля открывается окно, специфичное для соответствующей роли пользователя. Роль задана в базе данных.  
 Окно регистрации предназначено для добавления нового пользователя в систему, если таков ещё не зарегистрирован. Оно содержит поля для ввода логина, ФИО и пароля. Также реализован форма, обязующая подтвердить согласие с правилами пользования.

Когда оператор проходит авторизацию, он попадает в окно оператора. Здесь он может просматривать путевые листы и информацию о транспорте. Также в соответствии с его полномочиями, реализованы следующие возможности:  
 – редактировать данные о путевых листах, добавляя новые и редактируя или удаляя существующие;

– редактировать данные о транспортных средствах, используя все вытекающие из *CRUD* операции.

Если администратор попадает в своё окно, то он видит все таблицы базы данных. Он может читать всю информацию, кроме паролей пользователей. Помимо этого, на форме администратора реализованы следующие возможности:  
 – полное редактирование таблицы марок машин;

– все возможности для изменения в таблице транспорта;

– обновление, добавление, удаление путевых листов;

– полное изменение всех данных, за исключением пароля, возможность добавить нового пользователя без использования формы регистрации.

Для завершения работы необходимо закрыть форму. Вместе с ней закрывается и соединение.

На рисунке 2.6 представлена диаграмма вариантов использования.

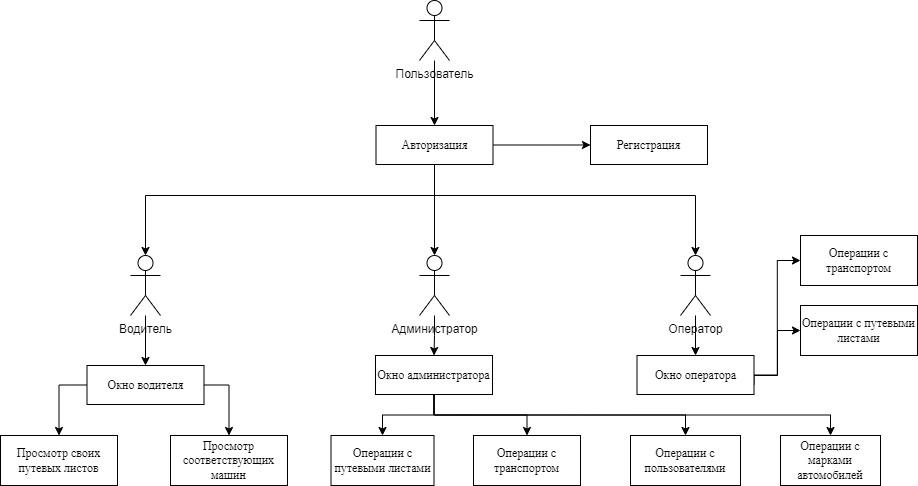


Рисунок 2.6 – Диаграмма вариантов использования

**2.5 Итоговые *XML*-отчёты**

Папка *Models*, содержащая модели данных, имеет также подпапку *Reports*. В ней содержатся классы, предназначенные для генерации и вывода отчётов на форме и с возможностью сохранения в формате *XML*. *Reports* имеет такие классы:

– *DriverReport* для отчёта по количеству рейсов и среднему проезду километров для каждого водителя;  
 – *CarReport* для отчёта о суммарном пробеге за период по автомобилю;

– *MarkReport* для отчёта о суммарном пробеге за период по марке автомобиля;

– *CargoReport* для формирования отчёта по перевезенному товару грузовым транспортом;

– *PassengerReport* для отчёта по количеству перевезенных людей пассажирским транспортом.

С помощью специальных кнопок на форме окна администратора, выбрав период времени, за который нужно сформировать отчёты, можно сгенерировать и вывести отчёты на форму.

Сами отчёты реализованы в классе *TripSheetRepository*. Посредством выполнения *SQL* запросов, а также использования вышеописанных классов-отчётов создаются специальные классы. Эти классы вызываются в зависимости от выбранной пользователем кнопки для генерации отчёта. Для удовлетворения требованию, что отчёт должен быть за определённый период, реализован метод считывания выбранного пользователем промежутка времени. Дополнительно, реализована возможность сохранения отчётов в формате *XML*. Для этого использован *XmlSerializer*. Они сохраняются по стандартному пути с понятным для пользователя названием.

На рисунке 2.7 представлен пример *XML* отчёта по маркам транспорта за период.

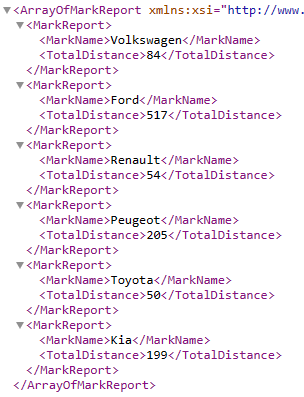


Рисунок 2.7 – Пример отчёта по маркам

На рисунке 2.8 представлен пример *XML* отчёта по пассажирскому транспорту за период.



Рисунок 2.8 – Пример отчёта по пассажирскому транспорту

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

# **3.1 Результаты модульного тестирования**

Ключевым аспектом разработки программного обеспечения является его тестирование для проверки работоспособности и надежности.

Лучшим вариантом для тестирования, помимо ручной проверки всех функций системы, является модульное тестирование. Оно направлено на проверку отдельных модулей, функций или классов программы с целью выявления локализованных ошибок в их реализации. Главная задача модульного тестирования заключается в обнаружении проблем на ранних этапах разработки и предотвращении их перехода на более продвинутые уровни тестирования.

Тестирование проводится на уровне отдельных компонентов программы. Для этого пишутся отдельные функции теста, которые помимо вызова функции, которую необходимо проверить, содержат и результат, которому должен соответствовать тест. Это позволяет проверить их работу независимо друг от друга. Такой подход упрощает процесс отладки и обеспечивает быструю реакцию на изменения в коде. Написание тестов для каждой значимой функции или метода позволяет оперативно выявлять ошибки и как следствие, быстро устранять их, не доводя до серьезных проблем во всем приложении.

Помимо этого, модульные тесты удобно отображают что ожидал пользователь, и что получил тест. Это значительно ускоряет процесс обнаружения ошибок, если изначально тест завершился неудачно.

Как и все основные составляющие программирования, модульные тесты должны обладать характеристиками, которые будут оправдывать их использование в приложении и доказывать эффективность применения в конкретной среде разработки. Эффективные и качественные модульные тесты должны:

– выполняться быстро, за миллисекунды, что позволяет проводить их часто и не замедляет процесс разработки;

– охватывать как можно больше кода целевого компонента или функции;

– быть легко изменяемыми и обновляться в случае изменения требований кода приложения;

– быть независимыми от внешних условий и окружения, таких как операционная система, наличие сети или состояние базы данных;

– быть легко расширяемыми для добавления новых сценариев тестирования или проверки новых функциональных возможностей;

– давать одинаковый результат каждый раз при выполнении в одинаковых условиях;

На рисунке 3.1 представлен результат выполнения модульных тестов. Листинг модульных тестов приведён в приложении В.

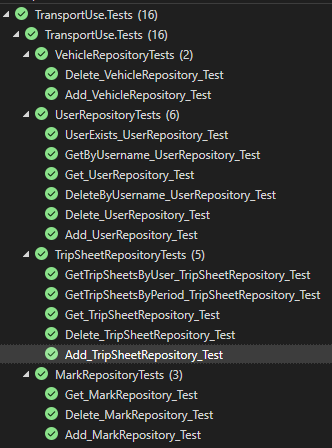


Рисунок 3.1 – Результат выполнения модульных тестов

# **3.2 Основные окна программы**

При запуске приложения открывается окно входа в систему. Для перехода на последующие окна пользователь должен либо ввести логин и пароль, либо, в случае отсутствия аккаунта, нажать кнопку «Зарегистрироваться». Логин и пароль должны соответствовать уже имеющимся в базе данных. Если логин и пароль верны, то при нажатии на кнопку «Войти» пользователь получит доступ к окну, соответствующему его роли, хранящейся в базе данных. Само окно входа в систему представлено на рисунке 3.2.

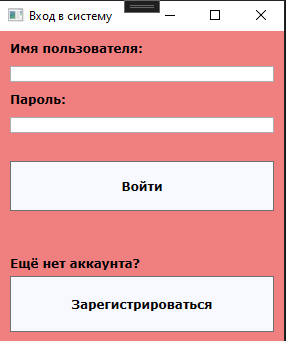


Рисунок 3.2 – Окно входа в систему

Если же пользователь ещё не имеет свой аккаунт, то он может зарегистрироваться. Для этого он должен перейти в окно регистрации и ввести логин, ФИО и пароль, а также поставить галочку «Я согласен с правилами». Окно регистрации представлено на рисунке 3.3.

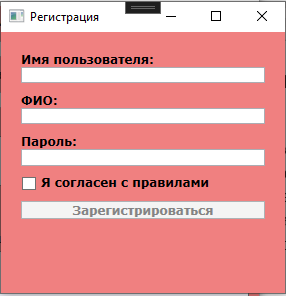


Рисунок 3.3 – Окно регистрации пользователя

Если роль пользователя администратор и он ввёл верные данные при входе, то откроется окно *AdminWindow* на вкладке «Транспорт». Окно администратора представлено на рисунке 3.4.

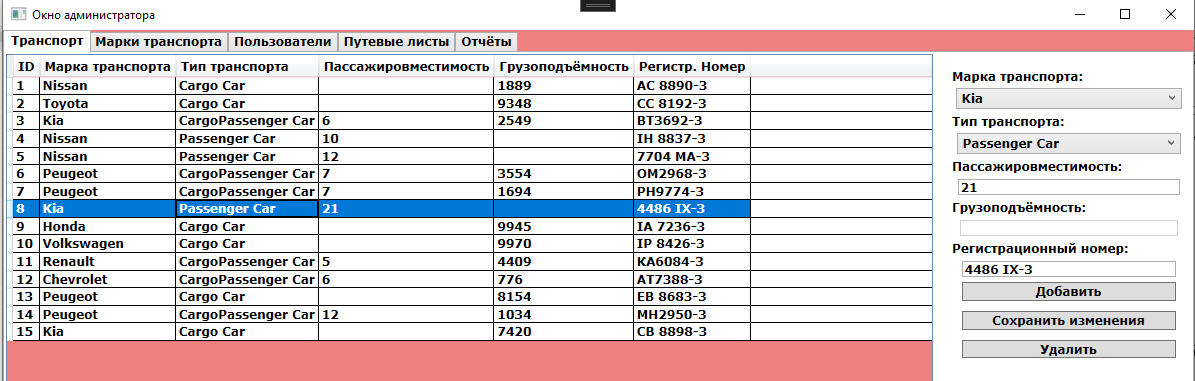


Рисунок 3.4 – Окно администратора

Если пользователь является оператором, и он ввёл верные данные при входе, то откроется окно *OperatorWindow*. Оператор может вносить данные о путевых листах и автомобилях. Поэтому ему доступны только 2 таблицы: транспорт и путевые листы. Окно оператора представлено на рисунке 3.5.

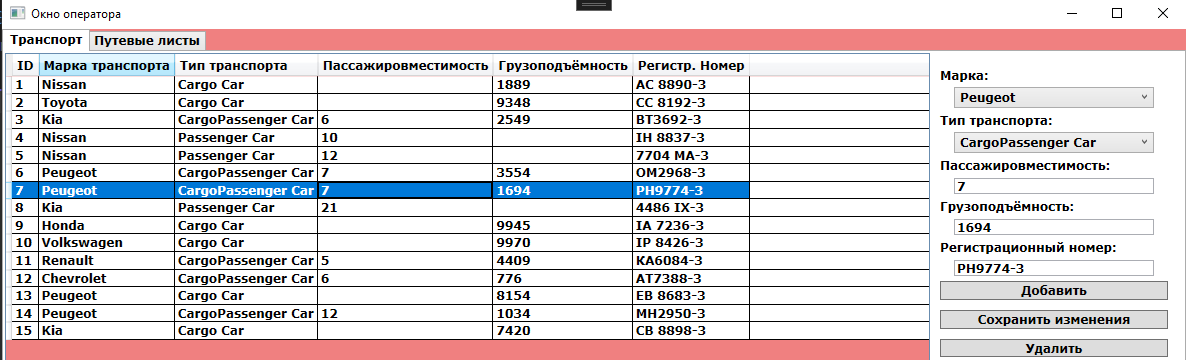


Рисунок 3.5 – Окно оператора

Если уровень доступа у пользователя «Водитель», и он ввёл верные данные при входе, то откроется окно *DriverWindow*. Водитель может просматривать информацию только о себе. Редактировать и удалять, а также добавлять своё он не может. На рисунках 3.6 – 3.7 представлены вкладка транспорта и путевых листов окна водителя с существующими записями.

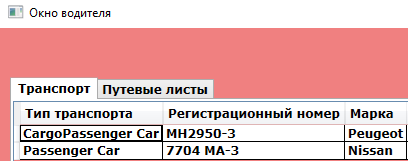


Рисунок 3.6 – Вкладка транспорта окна водителя



Рисунок 3.7 – Вкладка путевых листов окна водителя

# **3.2 Результаты ручного тестирования**

Валидация, или ручное тестирование, является неотъемлемой частью процесса проверки программного продукта перед его выпуском. Этот метод тестирования включает в себя активное взаимодействие пользователя с приложением с целью выявления потенциальных проблем и несоответствий между ожидаемым и фактическим поведением программы.

В процессе валидации важно выполнить ряд различных действий, охватывающих все функции приложения, а также возможные ошибки с целью выявления ещё необработанных проблем или обнаружение обработчика неверного действия. Процесс валидации может включать в себя ввод данных, навигацию по интерфейсу, выполнение определенных действий и анализ реакции приложения на эти действия. Задача состоит в том, чтобы убедиться, что приложение работает корректно, соответствует заявленным требованиям и ожиданиям пользователей.

Хотя ручное тестирование требует больше времени и усилий по сравнению с автоматизированными методами тестирования, оно остается необходимым инструментом для обеспечения высокого качества программного продукта и удовлетворения всех потребностей.

Если пользователь ввёл неверные данные при входе в систему, то появится соответствующее сообщение. Следовательно, пользователь не получит доступ ни к одному из окон приложения. На рисунке 3.8 представлена данная ошибка.

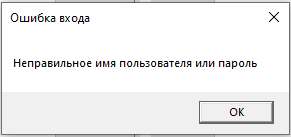


Рисунок 3.8 – Ошибка при вводе неправильных данных

Для регистрации пользователя в системе используется окно регистрации. Когда пользователь нажмет кнопку «Зарегистрироваться», то он вернется в окно входа в систему. В системе только зарегистрированный пользователь будет иметь роль водителя.

Если при регистрации пользователь ввёл уже существующий логин, то появится ошибка. На рисунке 3.9 представлена ошибка при попытке регистрации с уже существующим именем пользователя.

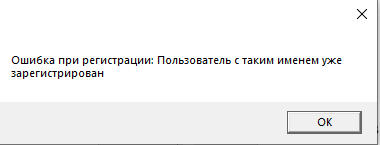


Рисунок 3.9 – Ошибка повторного логина при регистрации

Но заполнить в регистрации можно не все поля. Тогда тоже будет выведена ошибка. Ошибка, если не все поля заполнены, представлена на рисунке 3.10.

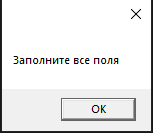


Рисунок 3.10 – Ошибка при наполненности одного из полей

Если пользователь является администратором и получил доступ к соответствующему окну, то он будет иметь полные права. Они позволяют ему добавлять новый транспорт, редактировать или удалить существующий.

Для этих целей выделено отдельное меню справа от данных с таблицы. Выбранная пользователем строка отслеживается и автоматически заполняет все необходимые поля.

Так как тип транспорта напрямую влияет на возможность перевозить пассажиров или груз, добавлена проверка типа транспорта. Если тип транспорта «*Cargo Car*», то поле «Пассажировместимость» будет недоступно для изменения или заполнения. Если тип транспорта «*Passenger Car*», то поле «Грузоподъемность» будет недоступно. При типе «*CargoPassenger Car*» все поля доступны. На рисунке 3.11 представлена блокировка поля груза в случае пассажирского транспорта.

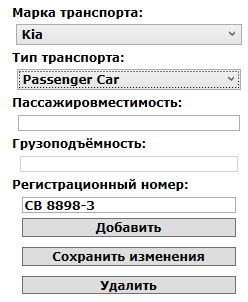


Рисунок 3.11 – Блокировка поля в зависимости от типа транспорта

Второй вкладкой окна администратора является «Марки транспорта». Она не содержит много информации, но также позволяет выполнять все *CRUD* операции, такие как добавление, изменения или удаления.

Во вкладке «Пользователи» окна администратора реализована возможность добавления, удаления и редактирования пользователей. Главной функцией здесь является возможность смены роли пользователя. Это особенно важно, учитывая факт того, что новый пользователь получает уровень доступа «Driver» и только администратор после проверки может изменить это поле.

Из побуждений сохранения безопасности системы, пароль на форму не выводится. В меню редактирования поле пароля присутствует, но тут оно имеется для добавления пользователя.

Если администратор попытается сохранить изменения, но при этом не введёт логин, то появляется ошибка. Ошибка изменения данных пользователя представлена на рисунке 3.12.

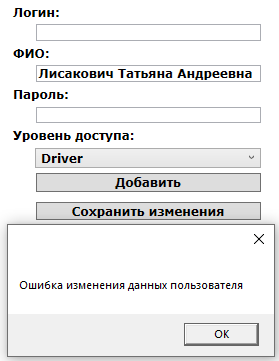


Рисунок 3.12 – Ошибка изменения при пустом логине

Вкладка путевых листов окна администратора представляет собой полноценное окно взаимодействия с базой за счёт связи этой таблицы с другими. На форме выводится регистрационный номер, а также пользователь в виде *ComboBox* только для возможности просматривать другие регистрационные номера транспортных средств и ФИО пользователей. Справа, как и на других вкладках, расположены поля для добавления, редактирования и удаления данных в путевых листах.

Для генерации отчётов выделена вкладка «Отчёты». На ней расположены специальные элементы для выбора промежутка времени, за который будут выводится отчёты.

При нажатии на кнопку «Сгенерировать отчёты» генерируется сразу все 5 отчётов.

Также отчёты должны выводится не только на форму, но и иметь возможность сохранятся в формате *XML*. Для этого предназначена кнопка «Сохранить в *XML*». На рисунке 3.13 представлен результат нажатия на кнопку сохранения в *XML*.

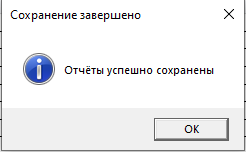


Рисунок 3.13 – Удачное сохранение отчётов в формате *XML*

Если же администратор не выберет период, то получит сообщение о необходимости выбора дат начала и окончания периода. На рисунке 3.14 представлено соответствующее сообщение.

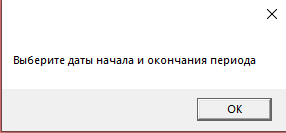


Рисунок 3.14 – Ошибка генерации отчётов из-за невыбранного периода

В случае неправильного выбора дат администратор получит сообщение о ошибке. На рисунке 3.15 представлена ошибка неправильного выбора дат.

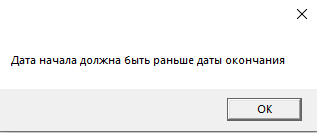


Рисунок 3.15 – Ошибка генерации отчётов из-за невыбранного периода

Для реализации действий оператора выделено отдельное меню справа от данных с таблицы. Выбранная пользователем строка отслеживается и автоматически заполняет все необходимые поля. На рисунке 3.16 представлена вкладка путевых листов окна оператора с полным редактированием и возможностью просматривать других машин и пользователей напрямую с элемента *Grid*.

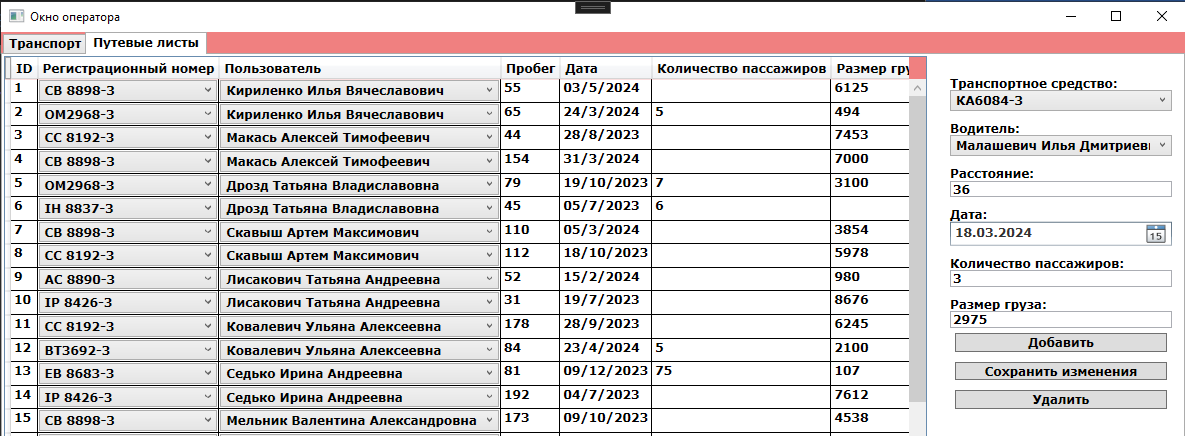


Рисунок 3.16 – Вкладка путевых листов окна оператора с полными правами редактирования

В окне водителя доступно всего две вкладки. При входе в систему открывается вкладка транспорта и выполняется автоматическая проверка на существование записей для водителя. Если же соответствующих записей для него нет, то выводится сообщение. Сообщения об отсутствии записей под водителя представлены на рисунках 3.17 – 3.18.

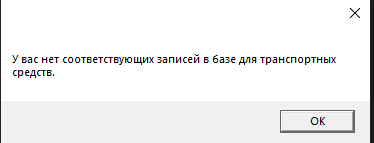


Рисунок 3.17 – Сообщение об отсутствии записей для водителя в таблице транспорта

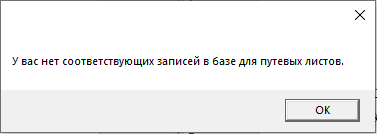


Рисунок 3.18 – Сообщение об отсутствии записей для водителя в таблице путевых листов

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

# **ЗАКЛЮЧЕНИЕ**

Результатов курсовой работы является приложение для учёта использования автотранспорта предприятием, позволяющее вести полноценный учёт пользователей системы и имеющегося транспорта. Важнейшим аспектом является возможность учёта перевозок различными типами транспорта с вытекающими характеристиками, актуальными именно для этого транспорта.

Данное приложение является отличным решением для упрощения учёта использования автотранспорта. Первоначально, оно решает проблему сложности работы с документацией по путевым листам. Благодаря его функционалу, все данные о рейсах и транспортных средствах структурированы и легко доступны, что позволяет сократить время, затрачиваемое на поиск нужной информации. Важным аспектом является также его способность упростить взаимодействие между сотрудниками предприятия. Выделение разных ролей у пользователей позволяет делать приложение безопасным от изменения данных неопытными пользователями.

Для создания приложения учёта использования автотранспорта предприятием использовались современные технологии и подходы, включая крайне популярный язык программирования *C#*, *WPF* для создания графического интерфейса. Удобство СУБД *MS SQL Server* помогло сделать работу с базой данных легкой и понятной для разных разработчиков.

Также, немаловажным является наличие модульного тестирования, показывающее правильность и адекватность работы созданных методов. Разделение данных на слои делает приложение чётко организованным и удовлетворяющим правилам объектно-ориентированного программирования, а также *SOLID* и *Code Convention*.

В результате разработки, получилось простое, но в то же время качественное и лёгкое приложения для управления всем необходимым для предприятия с различными типами транспорта. Графический интерфейс обеспечивает интуитивное взаимодействие между пользователем и базой, не вызывая необходимости долгого освоения принципов работы базы данных. Генерация отчётов и вовсе позволяет вывести нужные данные в отдельные файлы для дальнейшего анализа.

Данное приложение отличается от себе подобных простым интерфейсом, полным и структурированным подходом к учёту автотранспорта и перевозок, возможностью генерации отчётов напрямую из формы. Шаблона проектирования, используемые в приложении, делают его гибким и расширяемым, в отличие от многих современным аналогов.

В целом, функционал и подход к работе с данными делает приложение актуальным и востребованным для самых различных предприятий.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Прайс, М. *С#* 10 и .*NET* 6. Современная кроссплатформенная разработка / М.Прайс – Издательство «Питер», 2023. – 1846 с.
2. 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: 28.03.2024.
3. *Ivanov*, *N*. Портфолиоразработчика / *N.Ivanov* – Россия, 2023. – 246 с.
4. Принципы *SOLID* в *C#*. – Электрон. данные. – Режим доступа: <https://professorweb.ru/my/it/blog/net/solid.php>. – Дата доступа: 04.04.2024.
5. Мюллер, Д. *С#* для чайников / Джон Мюллер, Билл Семпф, Чак Сфер. – Издательство «Диалектика-Вильямс», 2019. – 608 с.
6. Гриффитс, И. Программируем на C# 8.0 / И. Гриффитс – Издательство «Питер», 2021. – 944 с.
7. Моргунов, E. *PostgreSQL.* Основы языка SQL / Е.Моргунов – Россия, 2024. – 336 с.
8. 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: 27.04.2024.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**ФУНКЦИОНАЛЬНАЯ СХЕМА ПРИЛОЖЕНИЯ**

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ**

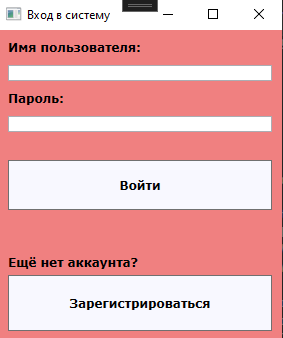
****

Рисунок Б.1 – Окно входа в систему

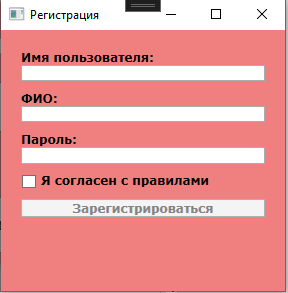
****

Рисунок Б.2 – Окно регистрации

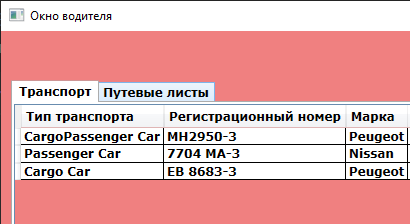
****

Рисунок Б.3 – Окно водителя

****

Рисунок Б.4 – Окно оператора

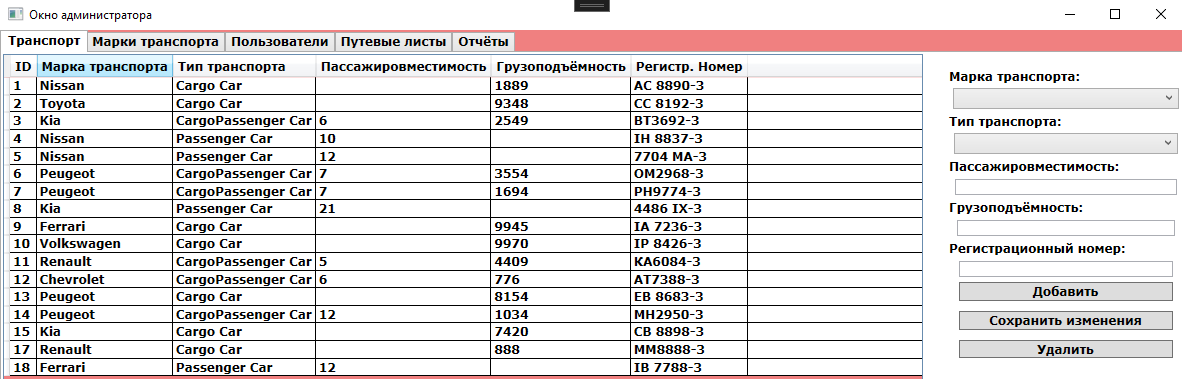
****

Рисунок Б.5 – Вкладка транспорта окна администратора

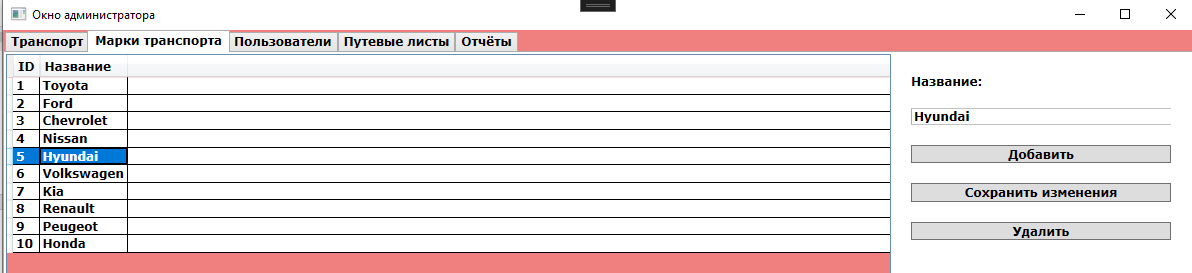


Рисунок Б.6 – Вкладка марок транспорта окна администратора

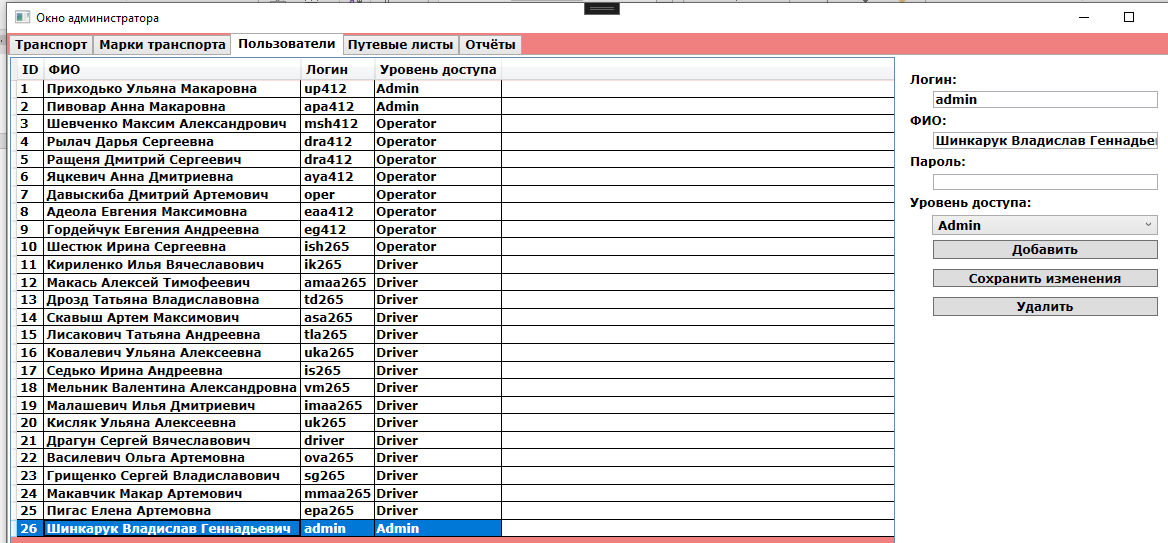


Рисунок Б.7 – Вкладка пользователей окна администратора

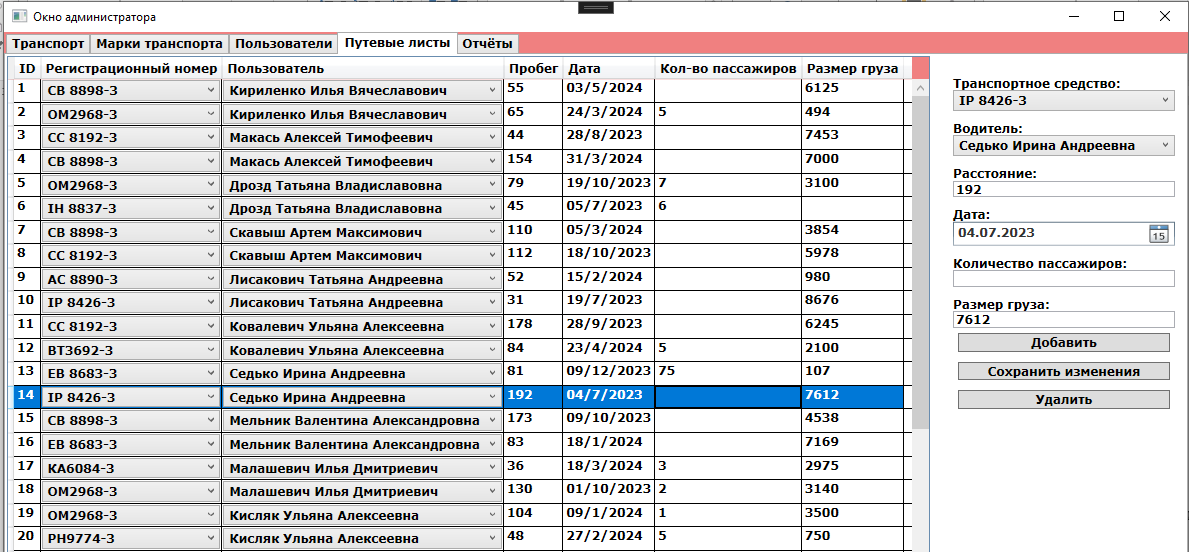


Рисунок Б.8 – Вкладка путевых листов окна администратора

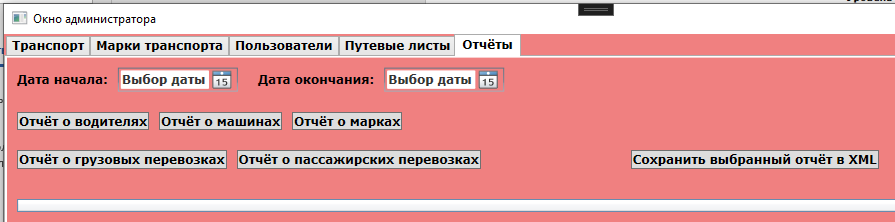


Рисунок Б.9 – Вкладка отчётов в окне администратора

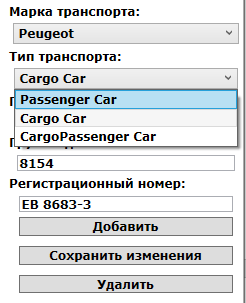
****

Рисунок Б.10 – Выпадающие списки в окнах

**ПРИЛОЖЕНИЕ В**

**ЛИСТИНГ ПРОГРАММЫ**

**Код класса *LoginWindow.xaml.cs***

using System;

using System.Windows;

using System.Data.SqlClient;

using TransportUse.DataLayer.Models.User;

using TransportUse.DataLayer.Repositories;

using TransportUse.DataLayer.Interfaces;

using TransportUse.DataLayer.Services.Interfaces;

using TransportUse.DataLayer.Services;

namespace TransportUseWPF.ViewLayer

{

/// <summary>

/// Окно для входа в систему.

/// </summary>

public partial class LoginWindow : Window

{

private readonly UserRepository \_userRepository;

/// <summary>

/// Конструктор класса LoginWindow.

/// </summary>

public LoginWindow()

{

InitializeComponent();

\_userRepository = new UserRepository();

}

/// <summary>

/// Обработчик события нажатия кнопки входа.

/// </summary>

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

{

// Получение имя пользователя и пароля из полей ввода.

string username = UsernameTextBox.Text;

string password = PasswordBox.Password;

try

{

// Инициализация сервиса пользователей и попытка входа в систему.

var userService = new UserService(\_userRepository);

User user = userService.Login(username, password);

// Проверка успешности входа и открытие соответствующего окна.

if (user != null)

{

// Открытие окна в зависимости от роли пользователя.

switch (user.Role)

{

case "Admin":

OpenAdminWindow();

break;

case "Operator":

OpenOperatorWindow();

break;

case "Driver":

OpenDriverWindow(username);

break;

default:

MessageBox.Show("Неизвестная роль пользователя");

break;

}

Close(); // Закрытие окна входа.

}

else

{

MessageBox.Show("Неправильное имя пользователя или пароль", "Ошибка входа");

}

}

catch (SqlException ex)

{

MessageBox.Show($"Ошибка базы данных: {ex.Message}");

}

catch (Exception ex)

{

MessageBox.Show($"Необработанное исключение: {ex.Message}");

}

}

/// <summary>

/// Открытие окна регистрации нового пользователя.

/// </summary>

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

{

// Инициализация сервиса пользователей и открытие окна регистрации.

IUserRepository userRepository = new UserRepository();

IUserService userService = new UserService(userRepository);

RegistrationWindow registrationWindow = new RegistrationWindow(userService);

registrationWindow.Show();

}

// Методы для открытия соответствующих окон в зависимости от роли пользователя.

private void OpenAdminWindow()

{

var markRepository = new MarkRepository();

var vehicleRepository = new VehicleRepository();

var userRepositoryAdmin = new UserRepository();

var tripSheetRepository = new TripSheetRepository();

var markService = new MarkService(markRepository);

var vehicleService = new VehicleService(vehicleRepository);

var userServiceAdmin = new UserService(userRepositoryAdmin);

var tripSheetService = new TripSheetService(tripSheetRepository);

AdminWindow adminWindow = new AdminWindow(markService, vehicleService, userServiceAdmin, tripSheetService);

adminWindow.Show();

}

private void OpenOperatorWindow()

{

var vehicleRepositoryOperator = new VehicleRepository();

var tripSheetRepositoryOperator = new TripSheetRepository();

var userRepositoryOperator = new UserRepository();

var markRepositoryOperator = new MarkRepository();

var vehicleServiceOperator = new VehicleService(vehicleRepositoryOperator);

var tripSheetServiceOperator = new TripSheetService(tripSheetRepositoryOperator);

var userServiceOperator = new UserService(userRepositoryOperator);

var markServiceOperator = new MarkService(markRepositoryOperator);

OperatorWindow operatorWindow = new OperatorWindow(

vehicleServiceOperator,

tripSheetServiceOperator,

userServiceOperator,

markServiceOperator

);

operatorWindow.Show();

}

private void OpenDriverWindow(string username)

{

var vehicleRepositoryDriver = new VehicleRepository();

var tripSheetRepositoryDriver = new TripSheetRepository();

var vehicleServiceDriver = new VehicleService(vehicleRepositoryDriver);

var tripSheetServiceDriver = new TripSheetService(tripSheetRepositoryDriver);

DriverWindow driverWindow = new DriverWindow(

username,

vehicleServiceDriver,

tripSheetServiceDriver

);

driverWindow.Show();

}

}

}

**Код шаблона *RegisterWindow.cshtml***

using System;

using System.Windows;

using TransportUse.DataLayer.Services.Interfaces;

namespace TransportUseWPF.ViewLayer

{

/// <summary>

/// Окно регистрации нового пользователя.

/// </summary>

public partial class RegistrationWindow : Window

{

private readonly IUserService \_userService;

/// <summary>

/// Конструктор класса RegistrationWindow.

/// </summary>

/// <param name="userService">Сервис пользователей для регистрации.</param>

public RegistrationWindow(IUserService userService)

{

InitializeComponent();

\_userService = userService;

RegisterButton.IsEnabled = false;

}

/// <summary>

/// Обработчик события выбора пользователем согласия с условиями использования.

/// </summary>

private void AgreementCheckBox\_Checked(object sender, RoutedEventArgs e)

{

RegisterButton.IsEnabled = AgreementCheckBox.IsChecked == true;

}

/// <summary>

/// Обработчик события нажатия кнопки регистрации.

/// </summary>

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

{

// Проверка согласия с условиями использования.

if (AgreementCheckBox.IsChecked != true)

{

MessageBox.Show("Согласитесь с условиями использования");

return;

}

// Получение данных пользователя из полей ввода.

string username = UsernameTextBox.Text;

string fio = FIOTextBox.Text;

string password = PasswordBox.Password;

// Проверка заполненности всех полей.

if (string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(fio) || string.IsNullOrWhiteSpace(password))

{

MessageBox.Show("Заполните все поля");

return;

}

try

{

// Регистрация нового пользователя.

var newUser = \_userService.Register(username, fio, password);

MessageBox.Show("Регистрация прошла успешно");

Close();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при регистрации: {ex.Message}");

}

}

}

}

**Код класса *AdminWindow.xaml.cs***

using System;

using System.Windows;

using TransportUse.DataLayer.Models;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using TransportUse.DataLayer.Models.Transport;

using TransportUse.DataLayer.Models.User;

using System.Windows.Controls;

using TransportUse.DataLayer.Services.Interfaces;

using System.IO;

using System.Xml.Serialization;

using TransportUse.DataLayer.Models.Reports;

namespace TransportUseWPF.ViewLayer

{

/// <summary>

/// Окно администратора системы.

/// </summary>

public partial class AdminWindow : Window

{

public List<Mark> Marks { get; set; }

public Mark SelectedMark { get; set; }

private IVehicleService \_vehicleService;

private IMarkService \_markService;

private IUserService \_userService;

private ITripSheetService \_tripSheetService;

public ObservableCollection<string> VehicleTypes { get; } = new ObservableCollection<string>

{

"Passenger Car", "Cargo Car", "CargoPassenger Car"

};

public ObservableCollection<Vehicle> Vehicles { get; set; }

public ObservableCollection<User> Users { get; set; }

public ObservableCollection<User> Drivers { get; set; }

/// <summary>

/// Конструктор класса AdminWindow.

/// </summary>

/// <param name="markService">Сервис марок транспорта.</param>

/// <param name="vehicleService">Сервис транспортных средств.</param>

/// <param name="userService">Сервис пользователей.</param>

/// <param name="tripSheetService">Сервис путевых листов.</param>

public AdminWindow(IMarkService markService, IVehicleService vehicleService, IUserService userService, ITripSheetService tripSheetService)

{

InitializeComponent();

\_markService = markService;

\_vehicleService = vehicleService;

\_userService = userService;

\_tripSheetService = tripSheetService;

Marks = \_markService.GetAll().ToList();

Vehicles = new ObservableCollection<Vehicle>(\_vehicleService.GetAll());

Users = new ObservableCollection<User>(\_userService.GetAll());

Drivers = new ObservableCollection<User>(\_userService.GetAll().Where(u => u.Role == "Driver"));

VehiclesDataGrid.ItemsSource = Vehicles;

UsersDataGrid.ItemsSource = Users;

DataContext = this;

LoadData();

}

/// <summary>

/// Загрузка данных в окно.

/// </summary>

private void LoadData()

{

var marks = \_markService.GetAll();

MarksDataGrid.ItemsSource = marks;

var vehicles = \_vehicleService.GetAll();

VehiclesDataGrid.ItemsSource = vehicles;

var users = \_userService.GetAll();

UsersDataGrid.ItemsSource = users;

var tripSheets = \_tripSheetService.GetAll();

TripSheetsDataGrid.ItemsSource = tripSheets;

}

/// <summary>

/// Добавление транспорта.

/// </summary>

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

{

string typeVehicle = VehicleTypeComboBox.SelectedItem as string;

string registrationNumber = RegistrationNumberTextBox.Text;

Mark selectedMark = VehicleMarkComboBox.SelectedItem as Mark;

string passengerCapacity = PassengerCapacityTextBox.Text;

string cargoCapacity = CargoCapacityTextBox.Text;

if (!\_vehicleService.AddVehicle(typeVehicle, registrationNumber, selectedMark, passengerCapacity, cargoCapacity))

{

MessageBox.Show("Неверный формат данных");

return;

}

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

}

/// <summary>

/// Редактирование транспорта.

/// </summary>

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

{

if (VehiclesDataGrid.SelectedItem is Vehicle selectedVehicle)

{

string typeVehicle = VehicleTypeComboBox.SelectedItem as string;

string registrationNumber = RegistrationNumberTextBox.Text;

Mark selectedMark = VehicleMarkComboBox.SelectedItem as Mark;

string passengerCapacity = PassengerCapacityTextBox.Text;

string cargoCapacity = CargoCapacityTextBox.Text;

if (!\_vehicleService.UpdateVehicle(selectedVehicle, typeVehicle, registrationNumber, selectedMark, passengerCapacity, cargoCapacity))

{

MessageBox.Show("Заполните все поля");

return;

}

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

}

}

/// <summary>

/// Удаление транспорта.

/// </summary>

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

{

if (VehiclesDataGrid.SelectedItem is Vehicle selectedVehicle)

{

\_vehicleService.DeleteVehicle(selectedVehicle);

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

}

}

/// <summary>

/// Заполнение выпадающего списка транспорта.

/// </summary>

private void VehicleTypeComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

var selectedType = VehicleTypeComboBox.SelectedItem as string;

switch (selectedType)

{

case "Passenger Car":

PassengerCapacityTextBox.IsEnabled = true;

CargoCapacityTextBox.IsEnabled = false;

CargoCapacityTextBox.Clear();

break;

case "Cargo Car":

PassengerCapacityTextBox.IsEnabled = false;

PassengerCapacityTextBox.Clear();

CargoCapacityTextBox.IsEnabled = true;

break;

case "CargoPassenger Car":

PassengerCapacityTextBox.IsEnabled = true;

CargoCapacityTextBox.IsEnabled = true;

break;

default:

PassengerCapacityTextBox.IsEnabled = false;

CargoCapacityTextBox.IsEnabled = false;

break;

}

}

/// <summary>

/// Проверка выбранной строки в транспорте.

/// </summary>

private void VehiclesDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (VehiclesDataGrid.SelectedItem is Vehicle selectedVehicle)

{

VehicleTypeComboBox.SelectedItem = selectedVehicle.TypeVehicle;

RegistrationNumberTextBox.Text = selectedVehicle.RegistrationNumber;

VehicleMarkComboBox.SelectedItem = Marks.FirstOrDefault(m => m.Id == selectedVehicle.Mark.Id);

switch (selectedVehicle)

{

case PassengerCar p:

PassengerCapacityTextBox.Text = p.PassengerCapacity.ToString();

CargoCapacityTextBox.Clear();

break;

case CargoCar c:

CargoCapacityTextBox.Text = c.CargoCapacity.ToString();

PassengerCapacityTextBox.Clear();

break;

case CargoPassengerCar cp:

PassengerCapacityTextBox.Text = cp.PassengerCapacity.ToString();

CargoCapacityTextBox.Text = cp.CargoCapacity.ToString();

break;

default:

PassengerCapacityTextBox.Clear();

CargoCapacityTextBox.Clear();

break;

}

}

}

/// <summary>

/// Добавление марки.

/// </summary>

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

{

if (\_markService.AddMark(MarkNameTextBox.Text))

{

MarksDataGrid.ItemsSource = \_markService.GetAll();

MarkNameTextBox.Clear();

}

}

/// <summary>

/// Редактирование марки.

/// </summary>

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

{

if (MarksDataGrid.SelectedItem is Mark selectedMark && \_markService.UpdateMark(selectedMark, MarkNameTextBox.Text))

{

MarksDataGrid.ItemsSource = \_markService.GetAll();

}

}

/// <summary>

/// Удаление марки.

/// </summary>

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

{

if (MarksDataGrid.SelectedItem is Mark selectedMark && \_markService.DeleteMark(selectedMark.Id))

{

MarksDataGrid.ItemsSource = \_markService.GetAll();

}

}

/// <summary>

/// Проверка выбранной строки в марках.

/// </summary>

private void MarksDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (e.AddedItems.Count > 0)

{

var selectedMark = e.AddedItems[0] as Mark;

MarkNameTextBox.Text = selectedMark?.Name;

}

}

/// <summary>

/// Добавление пользователя.

/// </summary>

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

{

var username = UsernameTextBox.Text;

var fio = FIOTextBox.Text;

var password = PasswordBox.Password;

var role = RoleComboBox.SelectedItem as string;

if (string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(fio) || string.IsNullOrWhiteSpace(password) || role == null)

{

MessageBox.Show("Все поля должны быть заполнены");

return;

}

if (\_userService.AddUser(username, fio, password, role))

{

UsersDataGrid.ItemsSource = \_userService.GetAll();

}

else

{

MessageBox.Show("Пользователь с таким именем уже зарегистрирован");

}

}

/// <summary>

/// Редактирование пользователя.

/// </summary>

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

{

if (UsersDataGrid.SelectedItem is User selectedUser)

{

var username = UsernameTextBox.Text;

var fio = FIOTextBox.Text;

var password = PasswordBox.Password;

var role = RoleComboBox.SelectedItem as string;

if (\_userService.UpdateUser(selectedUser, username, fio, password, role))

{

UsersDataGrid.ItemsSource = \_userService.GetAll();

}

else

{

MessageBox.Show("Ошибка изменения данных пользователя");

}

}

}

/// <summary>

/// Удаление пользователя.

/// </summary>

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

{

if (UsersDataGrid.SelectedItem is User selectedUser)

{

if (\_userService.DeleteUser(selectedUser))

{

UsersDataGrid.ItemsSource = \_userService.GetAll();

}

else

{

MessageBox.Show("Ошибка удаления пользователя");

}

}

}

/// <summary>

/// Проверка выбранной строки в пользователях.

/// </summary>

private void UsersDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (UsersDataGrid.SelectedItem is User selectedUser)

{

UsernameTextBox.Text = selectedUser.Username;

FIOTextBox.Text = selectedUser.FIO;

PasswordBox.Password = "";

RoleComboBox.SelectedItem = selectedUser.Role;

}

}

/// <summary>

/// Добавление путевого листа.

/// </summary>

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

{

int vehicleId = (int)VehicleComboBox.SelectedValue;

int driverId = (int)UserComboBox.SelectedValue;

DateTime date = (DateDatePicker.SelectedDate ?? DateTime.Now).Date;

float distance = (float)Convert.ToDouble(DistanceTextBox.Text);

int? cargoAmount = string.IsNullOrEmpty(CargoAmountTextBox.Text) ? (int?)null : int.Parse(CargoAmountTextBox.Text);

int? passengerCount = string.IsNullOrEmpty(PassengerCountTextBox.Text) ? (int?)null : int.Parse(PassengerCountTextBox.Text);

var tripSheet = new TripSheet(vehicleId, driverId, date, distance, cargoAmount, passengerCount);

if (!\_tripSheetService.AddTripSheet(tripSheet))

{

MessageBox.Show("Ошибка добавления данных о путевых листах");

}

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

/// <summary>

/// Редактирование путевого листа.

/// </summary>

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

{

if (TripSheetsDataGrid.SelectedItem is TripSheet selectedTripSheet)

{

selectedTripSheet.VehicleId = (int)VehicleComboBox.SelectedValue;

selectedTripSheet.DriverId = (int)UserComboBox.SelectedValue;

selectedTripSheet.Date = (DateDatePicker.SelectedDate ?? DateTime.Now).Date;

selectedTripSheet.Distance = (float)Convert.ToDouble(DistanceTextBox.Text);

selectedTripSheet.CargoAmount = string.IsNullOrEmpty(CargoAmountTextBox.Text) ? (int?)null : int.Parse(CargoAmountTextBox.Text);

selectedTripSheet.PassengerCount = string.IsNullOrEmpty(PassengerCountTextBox.Text) ? (int?)null : int.Parse(PassengerCountTextBox.Text);

if (!\_tripSheetService.UpdateTripSheet(selectedTripSheet))

{

MessageBox.Show("Ошибка изменения данных пользователя");

}

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

}

/// <summary>

/// Удаление путевого листа.

/// </summary>

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

{

if (TripSheetsDataGrid.SelectedItem is TripSheet selectedTripSheet)

{

if (!\_tripSheetService.DeleteTripSheet(selectedTripSheet.Id))

{

MessageBox.Show("Ошибка удаления данных о путевых листах");

}

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

}

/// <summary>

/// Проверка выбранной строки в путевых листах.

/// </summary>

private void TripSheetsDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (TripSheetsDataGrid.SelectedItem is TripSheet selectedTripSheet)

{

VehicleComboBox.SelectedValue = selectedTripSheet.VehicleId;

UserComboBox.SelectedValue = selectedTripSheet.DriverId;

DateDatePicker.SelectedDate = selectedTripSheet.Date;

DistanceTextBox.Text = selectedTripSheet.Distance.ToString();

CargoAmountTextBox.Text = selectedTripSheet.CargoAmount?.ToString() ?? "";

PassengerCountTextBox.Text = selectedTripSheet.PassengerCount?.ToString() ?? "";

}

}

/// <summary>

/// Метод для генерации отчётов.

/// </summary>

private void GenerateReport<T>(Func<DateTime, DateTime, IEnumerable<T>> reportGenerator, DataGrid dataGrid)

{

if (StartDatePicker.SelectedDate == null || EndDatePicker.SelectedDate == null)

{

MessageBox.Show("Выберите даты начала и окончания периода");

return;

}

DateTime startDate = StartDatePicker.SelectedDate.Value;

DateTime endDate = EndDatePicker.SelectedDate.Value;

if (startDate > endDate)

{

MessageBox.Show("Дата начала должна быть раньше даты окончания");

return;

}

string errorMessage = "Ошибка при генерации отчёта: ";

try

{

var report = reportGenerator(startDate.Date, endDate.Date);

dataGrid.ItemsSource = report;

}

catch (Exception ex)

{

MessageBox.Show(errorMessage + ex.Message);

}

}

// Методы для генерации различных отчётов.

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

{

GenerateReport(\_tripSheetService.GetDriversReports, ReportsDataGrid);

}

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

{

GenerateReport(\_tripSheetService.GetCarsReports, ReportsDataGrid);

}

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

{

GenerateReport(\_tripSheetService.GetMarksReports, ReportsDataGrid);

}

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

{

GenerateReport(\_tripSheetService.GetCargoReports, ReportsDataGrid);

}

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

{

GenerateReport(\_tripSheetService.GetPassengerReports, ReportsDataGrid);

}

/// <summary>

/// Метод для сохранения в XML.

/// </summary>

private void SaveReportsToXml<T>(IEnumerable<T> reports, string filePath)

{

XmlSerializer serializer = new XmlSerializer(typeof(List<T>));

using (StreamWriter writer = new StreamWriter(filePath))

{

serializer.Serialize(writer, reports.ToList());

}

MessageBox.Show("Отчет успешно сохранен в файл " + filePath, "Успешное сохранение", MessageBoxButton.OK, MessageBoxImage.Information);

}

/// <summary>

/// Обработка нажатия на сохранения отчётов в XML.

/// </summary>

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

{

if (ReportsDataGrid.ItemsSource == null)

{

MessageBox.Show("Нет сгенерированных отчетов для сохранения", "Предупреждение", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (ReportsDataGrid.ItemsSource is IEnumerable<DriverReport>)

{

SaveReportsToXml((IEnumerable<DriverReport>)ReportsDataGrid.ItemsSource, "driverReports.xml");

}

else if (ReportsDataGrid.ItemsSource is IEnumerable<CarReport>)

{

SaveReportsToXml((IEnumerable<CarReport>)ReportsDataGrid.ItemsSource, "carReports.xml");

}

else if (ReportsDataGrid.ItemsSource is IEnumerable<MarkReport>)

{

SaveReportsToXml((IEnumerable<MarkReport>)ReportsDataGrid.ItemsSource, "markReports.xml");

}

else if (ReportsDataGrid.ItemsSource is IEnumerable<CargoReport>)

{

SaveReportsToXml((IEnumerable<CargoReport>)ReportsDataGrid.ItemsSource, "cargoReports.xml");

}

else if (ReportsDataGrid.ItemsSource is IEnumerable<PassengerReport>)

{

SaveReportsToXml((IEnumerable<PassengerReport>)ReportsDataGrid.ItemsSource, "passengerReports.xml");

}

}

}

}

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

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Models.Transport;

using TransportUse.DataLayer.Models.User;

using TransportUse.DataLayer.Services.Interfaces;

namespace TransportUseWPF.ViewLayer

{

/// <summary>

/// Окно оператора системы.

/// </summary>

public partial class OperatorWindow : Window

{

public List<Mark> Marks { get; set; }

private IVehicleService \_vehicleService;

private ITripSheetService \_tripSheetService;

private IUserService \_userService;

private IMarkService \_markService;

public ObservableCollection<string> VehicleTypes { get; } = new ObservableCollection<string>

{

"Passenger Car", "Cargo Car", "CargoPassenger Car"

};

public ObservableCollection<Vehicle> Vehicles { get; set; }

public ObservableCollection<User> Drivers { get; set; }

/// <summary>

/// Конструктор класса OperatorWindow.

/// </summary>

/// <param name="vehicleService">Сервис транспортных средств.</param>

/// <param name="tripSheetService">Сервис путевых листов.</param>

/// <param name="userService">Сервис пользователей.</param>

/// <param name="markService">Сервис марок транспорта.</param>

public OperatorWindow(IVehicleService vehicleService, ITripSheetService tripSheetService, IUserService userService, IMarkService markService)

{

InitializeComponent();

\_vehicleService = vehicleService;

\_tripSheetService = tripSheetService;

\_userService = userService;

\_markService = markService;

Marks = \_markService.GetAll().ToList();

Vehicles = new ObservableCollection<Vehicle>(\_vehicleService.GetAll());

Drivers = new ObservableCollection<User>(\_userService.GetAll().Where(u => u.Role == "Driver"));

VehiclesDataGrid.ItemsSource = Vehicles;

DataContext = this;

LoadData();

}

/// <summary>

/// Загрузка данных в окно.

/// </summary>

private void LoadData()

{

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

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

{

string typeVehicle = VehicleTypeComboBox.SelectedItem as string;

string registrationNumber = RegistrationNumberTextBox.Text;

Mark selectedMark = VehicleMarkComboBox.SelectedItem as Mark;

string passengerCapacity = PassengerCapacityTextBox.Text;

string cargoCapacity = CargoCapacityTextBox.Text;

if (!\_vehicleService.AddVehicle(typeVehicle, registrationNumber, selectedMark, passengerCapacity, cargoCapacity))

{

MessageBox.Show("Неверный формат данных");

return;

}

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

}

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

{

if (VehiclesDataGrid.SelectedItem is Vehicle selectedVehicle)

{

string typeVehicle = VehicleTypeComboBox.SelectedItem as string;

string registrationNumber = RegistrationNumberTextBox.Text;

Mark selectedMark = VehicleMarkComboBox.SelectedItem as Mark;

string passengerCapacity = PassengerCapacityTextBox.Text;

string cargoCapacity = CargoCapacityTextBox.Text;

if (!\_vehicleService.UpdateVehicle(selectedVehicle, typeVehicle, registrationNumber, selectedMark, passengerCapacity, cargoCapacity))

{

MessageBox.Show("Заполните все поля");

return;

}

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

}

}

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

{

if (VehiclesDataGrid.SelectedItem is Vehicle selectedVehicle)

{

\_vehicleService.DeleteVehicle(selectedVehicle);

VehiclesDataGrid.ItemsSource = \_vehicleService.GetAll();

}

}

private void VehicleTypeComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

var selectedType = VehicleTypeComboBox.SelectedItem as string;

switch (selectedType)

{

case "Passenger Car":

PassengerCapacityTextBox.IsEnabled = true;

CargoCapacityTextBox.IsEnabled = false;

CargoCapacityTextBox.Clear();

break;

case "Cargo Car":

PassengerCapacityTextBox.IsEnabled = false;

PassengerCapacityTextBox.Clear();

CargoCapacityTextBox.IsEnabled = true;

break;

case "CargoPassenger Car":

PassengerCapacityTextBox.IsEnabled = true;

CargoCapacityTextBox.IsEnabled = true;

break;

default:

PassengerCapacityTextBox.IsEnabled = false;

CargoCapacityTextBox.IsEnabled = false;

break;

}

}

private void VehiclesDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (VehiclesDataGrid.SelectedItem is Vehicle selectedVehicle)

{

VehicleTypeComboBox.SelectedItem = selectedVehicle.TypeVehicle;

RegistrationNumberTextBox.Text = selectedVehicle.RegistrationNumber;

VehicleMarkComboBox.SelectedItem = Marks.FirstOrDefault(m => m.Id == selectedVehicle.Mark.Id);

switch (selectedVehicle)

{

case PassengerCar p:

PassengerCapacityTextBox.Text = p.PassengerCapacity.ToString();

CargoCapacityTextBox.Clear();

break;

case CargoCar c:

CargoCapacityTextBox.Text = c.CargoCapacity.ToString();

PassengerCapacityTextBox.Clear();

break;

case CargoPassengerCar cp:

PassengerCapacityTextBox.Text = cp.PassengerCapacity.ToString();

CargoCapacityTextBox.Text = cp.CargoCapacity.ToString();

break;

default:

PassengerCapacityTextBox.Clear();

CargoCapacityTextBox.Clear();

break;

}

}

}

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

{

int vehicleId = (int)VehicleComboBox.SelectedValue;

int driverId = (int)UserComboBox.SelectedValue;

DateTime date = DateDatePicker.SelectedDate ?? DateTime.Now;

float distance = (float)Convert.ToDouble(DistanceTextBox.Text);

int? cargoAmount = string.IsNullOrEmpty(CargoAmountTextBox.Text) ? (int?)null : int.Parse(CargoAmountTextBox.Text);

int? passengerCount = string.IsNullOrEmpty(PassengerCountTextBox.Text) ? (int?)null : int.Parse(PassengerCountTextBox.Text);

var tripSheet = new TripSheet(vehicleId, driverId, date, distance, cargoAmount, passengerCount);

if (!\_tripSheetService.AddTripSheet(tripSheet))

{

MessageBox.Show("Ошибка добавления данных о путевых листах");

}

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

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

{

if (TripSheetsDataGrid.SelectedItem is TripSheet selectedTripSheet)

{

selectedTripSheet.VehicleId = (int)VehicleComboBox.SelectedValue;

selectedTripSheet.DriverId = (int)UserComboBox.SelectedValue;

selectedTripSheet.Date = DateDatePicker.SelectedDate ?? DateTime.Now;

selectedTripSheet.Distance = (float)Convert.ToDouble(DistanceTextBox.Text);

selectedTripSheet.CargoAmount = string.IsNullOrEmpty(CargoAmountTextBox.Text) ? (int?)null : int.Parse(CargoAmountTextBox.Text);

selectedTripSheet.PassengerCount = string.IsNullOrEmpty(PassengerCountTextBox.Text) ? (int?)null : int.Parse(PassengerCountTextBox.Text);

if (!\_tripSheetService.UpdateTripSheet(selectedTripSheet))

{

MessageBox.Show("Ошибка обновления данных о путевых листах");

}

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

}

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

{

if (TripSheetsDataGrid.SelectedItem is TripSheet selectedTripSheet)

{

if (!\_tripSheetService.DeleteTripSheet(selectedTripSheet.Id))

{

MessageBox.Show("Ошибка удаления данных о путевых листах");

}

TripSheetsDataGrid.ItemsSource = \_tripSheetService.GetAll();

}

}

private void TripSheetsDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (TripSheetsDataGrid.SelectedItem is TripSheet selectedTripSheet)

{

VehicleComboBox.SelectedValue = selectedTripSheet.VehicleId;

UserComboBox.SelectedValue = selectedTripSheet.DriverId;

DateDatePicker.SelectedDate = selectedTripSheet.Date;

DistanceTextBox.Text = selectedTripSheet.Distance.ToString();

CargoAmountTextBox.Text = selectedTripSheet.CargoAmount?.ToString() ?? "";

PassengerCountTextBox.Text = selectedTripSheet.PassengerCount?.ToString() ?? "";

}

}

}

}

**Код класса *DriverWindow.xaml.cs***

using System;

using System.Linq;

using System.Windows;

using TransportUse.DataLayer.Services.Interfaces;

namespace TransportUseWPF.ViewLayer

{

/// <summary>

/// Окно водителя системы.

/// </summary>

public partial class DriverWindow : Window

{

private readonly IVehicleService \_vehicleService;

private readonly ITripSheetService \_tripSheetService;

private readonly string \_username;

/// <summary>

/// Конструктор класса DriverWindow.

/// </summary>

/// <param name="username">Имя пользователя водителя.</param>

/// <param name="vehicleService">Сервис транспортных средств.</param>

/// <param name="tripSheetService">Сервис путевых листов.</param>

public DriverWindow(string username, IVehicleService vehicleService, ITripSheetService tripSheetService)

{

InitializeComponent();

\_username = username;

\_vehicleService = vehicleService;

\_tripSheetService = tripSheetService;

LoadVehicles();

LoadTripSheets();

}

/// <summary>

/// Загрузка транспортных средств, связанных с текущим водителем.

/// </summary>

private void LoadVehicles()

{

try

{

var vehicles = \_vehicleService.GetVehiclesByUser(\_username);

if (vehicles.Any())

{

VehiclesDataGrid.ItemsSource = vehicles;

}

else

{

MessageBox.Show("У вас нет соответствующих записей в базе для транспортных средств.");

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при загрузке данных о транспорте: " + ex.Message);

}

}

/// <summary>

/// Загрузка путевых листов, связанных с текущим водителем.

/// </summary>

private void LoadTripSheets()

{

try

{

var tripSheets = \_tripSheetService.GetTripSheetsByUser(\_username);

if (tripSheets.Any())

{

TripsDataGrid.ItemsSource = tripSheets;

}

else

{

MessageBox.Show("У вас нет соответствующих записей в базе для путевых листов.");

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при загрузке данных о путевых листах: " + ex.Message);

}

}

}

}

**Код класса *BaseEntity.cs***

using System;

namespace TransportUse.DataLayer.Models

{

/// <summary>

/// Абстрактный класс, представляющий базовую сущность с идентификатором.

/// </summary>

public abstract class BaseEntity

{

/// <summary>

/// Уникальный идентификатор сущности.

/// </summary>

public int Id { get; set; }

/// <summary>

/// Переопределение метода для сравнения объектов на равенство по их идентификаторам.

/// </summary>

public override bool Equals(object obj)

{

if (obj == null || obj.GetType() != GetType())

return false;

BaseEntity other = (BaseEntity)obj;

return Id == other.Id;

}

/// <summary>

/// Получение хэш-кода объекта на основе его идентификатора.

/// </summary>

/// <returns>Хэш-код объекта.</returns>

public override int GetHashCode()

{

return Id.GetHashCode();

}

}

}

**Код класса *Mark.cs***

using System;

namespace TransportUse.DataLayer.Models

{

/// <summary>

/// Марка транспортного средства.

/// </summary>

public class Mark : BaseEntity

{

private string \_name;

/// <summary>

/// Создание нового экземпляра класса Mark с указанным именем.

/// </summary>

public Mark(string name)

{

\_name = name;

}

/// <summary>

/// Получение имя марки транспортного средства.

/// </summary>

public string Name { get { return \_name; } set { \_name = value; } }

}

}

**Код класса *TripSheet.cs***

using System;

namespace TransportUse.DataLayer.Models

{

/// <summary>

/// Представляет путевой лист, содержащий информацию о поездке на транспортном средстве.

/// </summary>

public class TripSheet : BaseEntity

{

private int \_vehicle;

private int \_driver;

private DateTime \_date;

private float \_distance;

private int? \_cargoAmount;

private int? \_passengerCount;

/// <summary>

/// Создание нового экземпляра класса TripSheet с указанными данными.

/// </summary>

public TripSheet(int vehicle, int driver, DateTime date, float distance, int? cargoAmount, int? passengerCount)

{

\_vehicle = vehicle;

\_driver = driver;

\_date = date;

\_distance = distance;

\_cargoAmount = cargoAmount;

\_passengerCount = passengerCount;

}

/// <summary>

/// Получение идентификатора транспортного средства.

/// </summary>

public int VehicleId { get { return \_vehicle; } set { \_vehicle = value; } }

/// <summary>

/// Получение идентификатора водителя.

/// </summary>

public int DriverId { get { return \_driver; } set { \_driver = value; } }

/// <summary>

/// Получение даты поездки.

/// </summary>

public DateTime Date { get { return \_date; } set { \_date = value; } }

/// <summary>

/// Получение пройденного расстояния.

/// </summary>

public float Distance { get { return \_distance; } set { \_distance = value; } }

/// <summary>

/// Получение количества перевезенного груза.

/// </summary>

public int? CargoAmount { get { return \_cargoAmount; } set { \_cargoAmount = value; } }

/// <summary>

/// Получение количество пассажиров.

/// </summary>

public int? PassengerCount { get { return \_passengerCount; } set { \_passengerCount = value; } }

}

}

**Код класса *Vehicle.cs***

using System;

namespace TransportUse.DataLayer.Models.Transport

{

/// <summary>

/// Представляет абстрактное транспортное средство.

/// </summary>

public abstract class Vehicle : BaseEntity

{

private string \_typeVehicle;

private string \_registrationNumber;

private Mark \_mark;

/// <summary>

/// Создание нового экземпляра класса Vehicle с указанными данными.

/// </summary>

public Vehicle(string typeVehicle, string registrationNumber, Mark mark)

{

\_typeVehicle = typeVehicle;

\_registrationNumber = registrationNumber;

\_mark = mark;

}

/// <summary>

/// Получение типа транспортного средства.

/// </summary>

public string TypeVehicle { get { return \_typeVehicle; } set { \_typeVehicle = value; } }

/// <summary>

/// Получение регистрационного номера.

/// </summary>

public string RegistrationNumber { get { return \_registrationNumber; } set { \_registrationNumber = value; } }

/// <summary>

/// Получение марки транспортного средства.

/// </summary>

public Mark Mark { get { return \_mark; } set { \_mark = value; } }

/// <summary>

/// Получение типа транспортного средства.

/// </summary>

/// <returns>Тип транспортного средства.</returns>

public abstract string GetTransportType();

}

}

**Код класса *IPassengerCar.cs***

using System;

namespace TransportUse.DataLayer.Models.Transport.Interfaces

{

public interface IPassengerCar

{

int GetPassengerCapacity();

}

}

**Код класса *ICargoCar.cs***

using System;

namespace TransportUse.DataLayer.Models.Transport.Interfaces

{

public interface ICargoCar

{

float GetCargoCapacity();

}

}

**Код класса *PassengerCar.cs***

using TransportUse.DataLayer.Models.Transport.Interfaces;

namespace TransportUse.DataLayer.Models.Transport

{

/// <summary>

/// Представляет пассажирский автомобиль.

/// </summary>

public class PassengerCar : Vehicle, IPassengerCar

{

private int \_passengerCapacity;

public int PassengerCapacity { get { return \_passengerCapacity; } set { \_passengerCapacity = value; } }

public PassengerCar(string typeVehicle, string registrationNumber, int passengerCapacity, Mark mark)

: base(typeVehicle, registrationNumber, mark)

{

\_passengerCapacity = passengerCapacity;

}

public int GetPassengerCapacity()

{

return \_passengerCapacity;

}

public override string GetTransportType()

{

return "Passenger";

}

}

}

**Код класса *CargoCar.cs***

using TransportUse.DataLayer.Models.Transport.Interfaces;

namespace TransportUse.DataLayer.Models.Transport

{

/// <summary>

/// Представляет грузовой автомобиль.

/// </summary>

public class CargoCar : Vehicle, ICargoCar

{

private float \_cargoCapacity;

public float CargoCapacity { get { return \_cargoCapacity; } set { \_cargoCapacity = value; } }

public CargoCar(string typeVehicle, string registrationNumber, float cargoCapacity, Mark mark)

: base(typeVehicle, registrationNumber, mark)

{

\_cargoCapacity = cargoCapacity;

}

public float GetCargoCapacity()

{

return \_cargoCapacity;

}

public override string GetTransportType()

{

return "Cargo";

}

}

}

**Код класса *CargoPassengerCar.cs***

using System;

namespace TransportUse.DataLayer.Models.Transport

{

/// <summary>

/// Представляет грузопассажирский автомобиль.

/// </summary>

public class CargoPassengerCar : Vehicle

{

private int \_passengerCapacity;

private float \_cargoCapacity;

public int PassengerCapacity { get { return \_passengerCapacity; } set { \_passengerCapacity = value; } }

public float CargoCapacity { get { return \_cargoCapacity; } set { \_cargoCapacity = value; } }

public CargoPassengerCar(string typeVehicle, string registrationNumber, int passengerCapacity, float cargoCapacity, Mark mark)

: base(typeVehicle, registrationNumber, mark)

{

\_passengerCapacity = passengerCapacity;

\_cargoCapacity = cargoCapacity;

}

public float GetCargoCapacity()

{

return \_cargoCapacity;

}

public int GetPassengerCapacity()

{

return \_passengerCapacity;

}

public override string GetTransportType()

{

return "CargoPassenger";

}

}

}

**Код класса *User.cs***

using System;

namespace TransportUse.DataLayer.Models.User

{

/// <summary>

/// Представляет пользователя системы.

/// </summary>

public class User : BaseEntity

{

private string \_username;

private string \_fio;

private string \_passwordHash;

private string \_role;

public User(string username, string fio, string passwordHash, string role)

{

\_username = username;

\_fio = fio;

\_passwordHash = passwordHash;

\_role = role;

}

public string Username { get { return \_username; } set { \_username = value; } }

public string FIO { get { return \_fio; } set { \_fio = value; } }

public string PasswordHash { get { return \_passwordHash; } set { \_passwordHash = value; } }

public string Role { get { return \_role; } set { \_role = value; } }

}

}

**Код класса *Driver.cs***

using System;

namespace TransportUse.DataLayer.Models.User

{

/// <summary>

/// Представляет водителя.

/// </summary>

public class Driver : User

{

public Driver(string username, string fio, string passwordHash) : base(username, fio, "Driver", passwordHash)

{

}

}

}

**Код класса *Operator.cs***

using System;

namespace TransportUse.DataLayer.Models.User

{

/// <summary>

/// Представляет опертора.

/// </summary>

public class Operator : User

{

public Operator(string username, string fio, string passwordHash) : base(username, fio, "Operator", passwordHash)

{

}

}

}

**Код класса *Admin.cs***

using System;

namespace TransportUse.DataLayer.Models.User

{

/// <summary>

/// Представляет администратора.

/// </summary>

public class Admin : User

{

public Admin(string username, string fio, string passwordHash) : base(username, fio, "Admin", passwordHash)

{

}

}

}

**Код класса *DbConnection.cs***

using System;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

namespace TransportUse.DataLayer

{

/// <summary>

/// Класс для установления подключения к базе данных.

/// </summary>

public class DbConnection

{

private static IDbConnection instance;

private static readonly object lockObject = new object();

/// <summary>

/// Делегат для фабрики создания подключения к базе данных.

/// </summary>

public static Func<IDbConnection> ConnectionFactory = CreateSqlConnection;

private DbConnection() { }

/// <summary>

/// Создание нового подключения к базе данных.

/// </summary>

/// <returns>Новое подключение к базе данных.</returns>

private static IDbConnection CreateSqlConnection()

{

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

}

/// <summary>

/// Получение экземпляра подключения к базе данных.

/// </summary>

public static IDbConnection Instance

{

get

{

lock (lockObject)

{

if (instance == null || instance.State == ConnectionState.Closed)

{

instance = ConnectionFactory();

}

return instance;

}

}

}

}

}

**Код класса *IRepository.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Models;

namespace TransportUse.DataLayer.Interfaces

{

public interface IRepository<T> where T : BaseEntity

{

T Add(T item);

T Get(int id);

IEnumerable<T> GetAll();

T Update(T item);

bool Delete(int id);

}

}

**Код класса *IMarkRepository.cs***

using TransportUse.DataLayer.Models;

namespace TransportUse.DataLayer.Interfaces

{

public interface IMarkRepository : IRepository<Mark>

{

}

}

**Код класса *ITripSheetRepository.cs***

using System;

using System.Collections.Generic;

using TransportUse.DataLayer.Models;

namespace TransportUse.DataLayer.Interfaces

{

public interface ITripSheetRepository : IRepository<TripSheet>

{

IEnumerable<TripSheet> GetTripSheetsByUser(string username);

IEnumerable<TripSheet> GetTripSheetsByPeriod(DateTime startDate, DateTime endDate);

}

}

**Код класса *IVehicleRepository.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Models.Transport;

namespace TransportUse.DataLayer.Interfaces

{

public interface IVehicleRepository : IRepository<Vehicle>

{

IEnumerable<Vehicle> GetVehiclesByUser(string username);

IEnumerable<Vehicle> GetAllVehiclesWithMarks();

}

}

**Код класса *IUserRepository.cs***

using TransportUse.DataLayer.Models.User;

namespace TransportUse.DataLayer.Interfaces

{

public interface IUserRepository : IRepository<User>

{

User GetByUsername(string username);

void DeleteByUsername(string username);

bool UserExists(string username);

}

}

**Код класса *Repository.cs***

using System.Collections.Generic;

using System.Data;

using TransportUse.DataLayer.Models;

namespace TransportUse.DataLayer.Repositories

{

public abstract class Repository<T> where T : BaseEntity

{

protected IDbConnection Connection => DbConnection.Instance;

private readonly string createQuery;

private readonly string updateQuery;

private readonly string deleteQuery;

private readonly string getByIdQuery;

private readonly string getAllQuery;

public Repository(string createQuery, string updateQuery, string deleteQuery, string getByIdQuery, string getAllQuery)

{

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}; SELECT CAST(SCOPE\_IDENTITY() as int)";

SetParametersToCommand(command, entity);

var id = (int)command.ExecuteScalar();

return GetById(id);

}

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;

SetParametersToCommand(command, entity);

command.ExecuteNonQuery();

return GetById(entity.Id);

}

public 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 System;

using System.Data;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Interfaces;

namespace TransportUse.DataLayer.Repositories

{

public class MarkRepository : Repository<Mark>, IMarkRepository

{

public MarkRepository()

: base(

"INSERT INTO Marks (Name) VALUES (@Name); SELECT CAST(SCOPE\_IDENTITY() as int)",

"UPDATE Marks SET Name = @Name WHERE MarkID = @id",

"DELETE FROM Marks WHERE MarkID = @id",

"SELECT \* FROM Marks WHERE MarkID = @id",

"SELECT \* FROM Marks")

{

}

protected override Mark GetEntityFromReader(IDataReader reader)

{

return new Mark(reader["Name"].ToString())

{

Id = int.Parse(reader["MarkID"].ToString())

};

}

protected override void SetParametersToCommand(IDbCommand command, Mark entity)

{

var nameParameter = command.CreateParameter();

nameParameter.ParameterName = "@Name";

nameParameter.Value = entity.Name;

command.Parameters.Add(nameParameter);

if (command.CommandText.Contains("@id"))

{

SetIdToCommand(command, entity.Id);

}

}

public Mark Add(Mark mark)

{

var command = Connection.CreateCommand();

command.CommandText = "INSERT INTO Marks (Name) VALUES (@Name); SELECT CAST(SCOPE\_IDENTITY() as int)";

var nameParameter = command.CreateParameter();

nameParameter.ParameterName = "@Name";

nameParameter.Value = mark.Name;

command.Parameters.Add(nameParameter);

mark.Id = Convert.ToInt32(command.ExecuteScalar());

return mark;

}

public Mark Get(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "SELECT \* FROM Marks WHERE MarkID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

IDataReader reader = null;

try

{

reader = command.ExecuteReader();

if (reader.Read())

{

return GetEntityFromReader(reader);

}

}

finally

{

reader.Close();

}

return null;

}

public new bool Delete(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "DELETE FROM Marks WHERE MarkID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

return command.ExecuteNonQuery() > 0;

}

}

}

**Код класса *TripSheetRepository.cs***

using System;

using System.Data;

using System.Collections.Generic;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Interfaces;

namespace TransportUse.DataLayer.Repositories

{

public class TripSheetRepository : Repository<TripSheet>, ITripSheetRepository

{

public TripSheetRepository()

: base(

"INSERT INTO TripSheets (VehicleID, DriverID, Date, Distance, CargoAmount, PassengerCount) VALUES (@VehicleID, @DriverID, @Date, @Distance, @CargoAmount, @PassengerCount); SELECT CAST(SCOPE\_IDENTITY() as int)",

"UPDATE TripSheets SET VehicleID = @VehicleID, DriverID = @DriverID, Date = @Date, Distance = @Distance, CargoAmount = @CargoAmount, PassengerCount = @PassengerCount WHERE TripSheetID = @id",

"DELETE FROM TripSheets WHERE TripSheetID = @id",

"SELECT \* FROM TripSheets WHERE TripSheetID = @id",

"SELECT \* FROM TripSheets")

{

}

protected override TripSheet GetEntityFromReader(IDataReader reader)

{

return new TripSheet(

int.Parse(reader["VehicleID"].ToString()),

int.Parse(reader["DriverID"].ToString()),

DateTime.Parse(reader["Date"].ToString()),

float.Parse(reader["Distance"].ToString()),

reader["CargoAmount"] == DBNull.Value ? (int?)null : int.Parse(reader["CargoAmount"].ToString()),

reader["PassengerCount"] == DBNull.Value ? (int?)null : int.Parse(reader["PassengerCount"].ToString())

)

{

Id = int.Parse(reader["TripSheetID"].ToString())

};

}

protected override void SetParametersToCommand(IDbCommand command, TripSheet entity)

{

var vehicleIdParameter = command.CreateParameter();

vehicleIdParameter.ParameterName = "@VehicleID";

vehicleIdParameter.Value = entity.VehicleId;

command.Parameters.Add(vehicleIdParameter);

var driverIdParameter = command.CreateParameter();

driverIdParameter.ParameterName = "@DriverID";

driverIdParameter.Value = entity.DriverId;

command.Parameters.Add(driverIdParameter);

var dateParameter = command.CreateParameter();

dateParameter.ParameterName = "@Date";

dateParameter.Value = entity.Date;

command.Parameters.Add(dateParameter);

var distanceParameter = command.CreateParameter();

distanceParameter.ParameterName = "@Distance";

distanceParameter.Value = entity.Distance;

command.Parameters.Add(distanceParameter);

var cargoAmountParameter = command.CreateParameter();

cargoAmountParameter.ParameterName = "@CargoAmount";

cargoAmountParameter.Value = entity.CargoAmount == null ? (object)DBNull.Value : entity.CargoAmount;

command.Parameters.Add(cargoAmountParameter);

var passengerCountParameter = command.CreateParameter();

passengerCountParameter.ParameterName = "@PassengerCount";

passengerCountParameter.Value = entity.PassengerCount == null ? (object)DBNull.Value : entity.PassengerCount;

command.Parameters.Add(passengerCountParameter);

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = entity.Id;

command.Parameters.Add(idParameter);

}

public TripSheet Add(TripSheet tripSheet)

{

var command = Connection.CreateCommand();

command.CommandText = "INSERT INTO TripSheets (VehicleID, DriverID, Date, Distance, CargoAmount, PassengerCount) VALUES (@VehicleID, @DriverID, @Date, @Distance, @CargoAmount, @PassengerCount); SELECT SCOPE\_IDENTITY();";

SetParametersToCommand(command, tripSheet);

tripSheet.Id = Convert.ToInt32(command.ExecuteScalar());

return tripSheet;

}

public TripSheet Get(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "SELECT \* FROM TripSheets WHERE TripSheetID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

return GetEntityFromReader(reader);

}

}

return null;

}

public new bool Delete(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "DELETE FROM TripSheets WHERE TripSheetID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

return command.ExecuteNonQuery() > 0;

}

public IEnumerable<TripSheet> GetTripSheetsByUser(string username)

{

var command = Connection.CreateCommand();

command.CommandText =

"SELECT ts.\* FROM TripSheets ts " +

"JOIN Users u ON ts.DriverId = u.DriverId " +

"WHERE u.Username = @username";

var usernameParameter = command.CreateParameter();

usernameParameter.ParameterName = "@username";

usernameParameter.Value = username;

command.Parameters.Add(usernameParameter);

var tripSheets = new List<TripSheet>();

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

tripSheets.Add(GetEntityFromReader(reader));

}

}

return tripSheets;

}

public IEnumerable<TripSheet> GetTripSheetsByPeriod(DateTime startDate, DateTime endDate)

{

var tripSheets = new List<TripSheet>();

var command = Connection.CreateCommand();

command.CommandText = @"

SELECT ts.VehicleID, ts.DriverID, ts.Date, ts.Distance, ts.CargoAmount, ts.PassengerCount

FROM TripSheets ts

WHERE ts.Date >= @StartDate AND ts.Date <= @EndDate";

var startDateParameter = command.CreateParameter();

startDateParameter.ParameterName = "@StartDate";

startDateParameter.Value = startDate;

command.Parameters.Add(startDateParameter);

var endDateParameter = command.CreateParameter();

endDateParameter.ParameterName = "@EndDate";

endDateParameter.Value = endDate;

command.Parameters.Add(endDateParameter);

var reader = command.ExecuteReader();

try

{

while (reader.Read())

{

int vehicleId = reader.GetInt32(reader.GetOrdinal("VehicleID"));

int driverId = reader.GetInt32(reader.GetOrdinal("DriverID"));

DateTime date = reader.GetDateTime(reader.GetOrdinal("Date"));

float distance = (float)reader.GetDouble(reader.GetOrdinal("Distance"));

int? cargoAmount = reader.IsDBNull(reader.GetOrdinal("CargoAmount")) ? null : (int?)reader.GetInt32(reader.GetOrdinal("CargoAmount"));

int? passengerCount = reader.IsDBNull(reader.GetOrdinal("PassengerCount")) ? null : (int?)reader.GetInt32(reader.GetOrdinal("PassengerCount"));

var tripSheet = new TripSheet(vehicleId, driverId, date, distance, cargoAmount, passengerCount);

tripSheets.Add(tripSheet);

}

}

finally

{

reader.Close();

}

return tripSheets;

}

}

**Код класса *VehicleRepository.cs***

using System;

using System.Data;

using System.Collections.Generic;

using TransportUse.DataLayer.Models.Transport;

using TransportUse.DataLayer.Models;

using System.Data.SqlClient;

using TransportUse.DataLayer.Interfaces;

namespace TransportUse.DataLayer.Repositories

{

public class VehicleRepository : Repository<Vehicle>, IVehicleRepository

{

public VehicleRepository()

: base(

"INSERT INTO Vehicles (MarkID, TypeVehicle, PassengerCapacity, CargoCapacity, RegistrationNumber) VALUES (@MarkID, @TypeVehicle, @PassengerCapacity, @CargoCapacity, @RegistrationNumber); SELECT CAST(SCOPE\_IDENTITY() as int)",

"UPDATE Vehicles SET MarkID = @MarkID, TypeVehicle = @TypeVehicle, PassengerCapacity = @PassengerCapacity, CargoCapacity = @CargoCapacity, RegistrationNumber = @RegistrationNumber WHERE VehicleID = @id",

"DELETE FROM Vehicles WHERE VehicleID = @id",

"SELECT v.\*, m.Name FROM Vehicles v JOIN Marks m ON v.MarkID = m.MarkID WHERE VehicleID = @id",

"SELECT v.\*, m.Name FROM Vehicles v JOIN Marks m ON v.MarkID = m.MarkID")

{

}

protected override Vehicle GetEntityFromReader(IDataReader reader)

{

var id = reader.GetInt32(reader.GetOrdinal("VehicleID"));

var markId = reader.GetInt32(reader.GetOrdinal("MarkID"));

var typeVehicle = reader.GetString(reader.GetOrdinal("TypeVehicle"));

var registrationNumber = reader.GetString(reader.GetOrdinal("RegistrationNumber"));

var markName = reader.GetString(reader.GetOrdinal("Name"));

var mark = new Mark(markName) { Id = markId };

Vehicle vehicle = null;

int? passengerCapacity = reader.IsDBNull(reader.GetOrdinal("PassengerCapacity"))

? (int?)null

: reader.GetInt32(reader.GetOrdinal("PassengerCapacity"));

double? cargoCapacity = reader.IsDBNull(reader.GetOrdinal("CargoCapacity"))

? (double?)null

: reader.GetDouble(reader.GetOrdinal("CargoCapacity"));

switch (typeVehicle)

{

case "Passenger Car":

vehicle = new PassengerCar(typeVehicle, registrationNumber, passengerCapacity ?? 0, mark) { Id = id };

break;

case "Cargo Car":

vehicle = new CargoCar(typeVehicle, registrationNumber, (float)(cargoCapacity ?? 0.0), mark) { Id = id };

break;

case "CargoPassenger Car":

vehicle = new CargoPassengerCar(typeVehicle, registrationNumber, passengerCapacity ?? 0, (float)(cargoCapacity ?? 0.0), mark) { Id = id };

break;

default:

throw new InvalidOperationException("Неизвестный тип транспортного средства: " + typeVehicle);

}

return vehicle;

}

protected override void SetParametersToCommand(IDbCommand command, Vehicle entity)

{

var markIdParameter = command.CreateParameter();

markIdParameter.ParameterName = "@MarkID";

markIdParameter.Value = entity.Mark.Id;

command.Parameters.Add(markIdParameter);

var typeVehicleParameter = command.CreateParameter();

typeVehicleParameter.ParameterName = "@TypeVehicle";

typeVehicleParameter.Value = entity.TypeVehicle;

command.Parameters.Add(typeVehicleParameter);

var registrationNumberParameter = command.CreateParameter();

registrationNumberParameter.ParameterName = "@RegistrationNumber";

registrationNumberParameter.Value = entity.RegistrationNumber;

command.Parameters.Add(registrationNumberParameter);

var passengerCapacityParameter = command.CreateParameter();

passengerCapacityParameter.ParameterName = "@PassengerCapacity";

passengerCapacityParameter.Value = entity is PassengerCar pc ? (object)pc.PassengerCapacity : entity is CargoPassengerCar cpc ? (object)cpc.PassengerCapacity : DBNull.Value;

command.Parameters.Add(passengerCapacityParameter);

var cargoCapacityParameter = command.CreateParameter();

cargoCapacityParameter.ParameterName = "@CargoCapacity";

cargoCapacityParameter.Value = entity is CargoCar cc ? (object)cc.CargoCapacity : entity is CargoPassengerCar capc ? (object)capc.CargoCapacity : DBNull.Value;

command.Parameters.Add(cargoCapacityParameter);

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = entity.Id;

command.Parameters.Add(idParameter);

}

public Vehicle Add(Vehicle vehicle)

{

var command = Connection.CreateCommand();

command.CommandText = "INSERT INTO Vehicles (MarkID, TypeVehicle, PassengerCapacity, CargoCapacity, RegistrationNumber) VALUES (@MarkID, @TypeVehicle, @PassengerCapacity, @CargoCapacity, @RegistrationNumber); SELECT SCOPE\_IDENTITY();";

SetParametersToCommand(command, vehicle);

vehicle.Id = Convert.ToInt32(command.ExecuteScalar());

return vehicle;

}

public Vehicle Get(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "SELECT \* FROM Vehicles WHERE VehicleID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

IDataReader reader = null;

try

{

reader = command.ExecuteReader();

if (reader.Read())

{

return GetEntityFromReader(reader);

}

}

finally

{

reader.Close();

}

return null;

}

public new bool Delete(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "DELETE FROM Vehicles WHERE VehicleID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

return command.ExecuteNonQuery() > 0;

}

public IEnumerable<Vehicle> GetVehiclesByUser(string username)

{

var vehicles = new List<Vehicle>();

var command = Connection.CreateCommand();

command.CommandText = @"

SELECT v.VehicleID, v.MarkID, v.TypeVehicle, v.RegistrationNumber,

v.PassengerCapacity, v.CargoCapacity, m.Name

FROM Vehicles v

JOIN TripSheets ts ON v.VehicleID = ts.VehicleID

JOIN Users u ON ts.DriverID = u.DriverID

JOIN Marks m ON v.MarkID = m.MarkID

WHERE u.Username = @Username";

command.Parameters.Add(new SqlParameter("@Username", username));

IDataReader reader = null;

try

{

reader = command.ExecuteReader();

while (reader.Read())

{

var vehicle = GetEntityFromReader(reader);

vehicles.Add(vehicle);

}

}

finally

{

reader.Close();

}

return vehicles;

}

public IEnumerable<Vehicle> GetAllVehiclesWithMarks()

{

var vehicles = new List<Vehicle>();

var command = Connection.CreateCommand();

command.CommandText = @"

SELECT v.VehicleID, v.MarkID, v.TypeVehicle, v.RegistrationNumber,

v.PassengerCapacity, v.CargoCapacity, m.Name

FROM Vehicles v

JOIN Marks m ON v.MarkID = m.MarkID";

IDataReader reader = null;

try

{

reader = command.ExecuteReader();

while (reader.Read())

{

var vehicle = GetEntityFromReader(reader);

vehicles.Add(vehicle);

}

}

finally

{

if (reader != null)

{

reader.Close();

}

}

return vehicles;

}

}

}

**Код класса *UserRepository.cs***

using System.Data;

using TransportUse.DataLayer.Models.User;

using TransportUse.DataLayer.Interfaces;

using System;

namespace TransportUse.DataLayer.Repositories

{

public class UserRepository : Repository<User>, IUserRepository

{

public UserRepository()

: base(

"INSERT INTO Users (Username, FIO, PasswordHash, Role) VALUES (@Username, @FIO, @PasswordHash, @Role); SELECT CAST(SCOPE\_IDENTITY() as int)",

"UPDATE Users SET Username = @Username, FIO = @FIO, PasswordHash = @PasswordHash, Role = @Role WHERE DriverID = @id",

"DELETE FROM Users WHERE DriverID = @id",

"SELECT \* FROM Users WHERE DriverID = @id",

"SELECT \* FROM Users"

)

{

}

protected override User GetEntityFromReader(IDataReader reader)

{

return new User(

reader["Username"].ToString(),

reader["FIO"].ToString(),

reader["PasswordHash"].ToString(),

reader["Role"].ToString())

{

Id = int.Parse(reader["DriverID"].ToString())

};

}

protected override void SetParametersToCommand(IDbCommand command, User entity)

{

var usernameParameter = command.CreateParameter();

usernameParameter.ParameterName = "@Username";

usernameParameter.Value = entity.Username;

command.Parameters.Add(usernameParameter);

var fioParameter = command.CreateParameter();

fioParameter.ParameterName = "@FIO";

fioParameter.Value = entity.FIO;

command.Parameters.Add(fioParameter);

var passwordParameter = command.CreateParameter();

passwordParameter.ParameterName = "@PasswordHash";

passwordParameter.Value = entity.PasswordHash;

command.Parameters.Add(passwordParameter);

var roleParameter = command.CreateParameter();

roleParameter.ParameterName = "@Role";

roleParameter.Value = entity.Role;

command.Parameters.Add(roleParameter);

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@Id";

idParameter.Value = entity.Id;

command.Parameters.Add(idParameter);

}

public User Add(User user)

{

var command = Connection.CreateCommand();

command.CommandText = "INSERT INTO Users (Username, FIO, PasswordHash, Role) VALUES (@Username, @FIO, @PasswordHash, @Role); SELECT CAST(SCOPE\_IDENTITY() as int)";

SetParametersToCommand(command, user);

user.Id = Convert.ToInt32(command.ExecuteScalar());

return user;

}

public User Get(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "SELECT \* FROM Users WHERE DriverID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

var reader = command.ExecuteReader();

try

{

if (reader.Read())

{

return GetEntityFromReader(reader);

}

}

finally

{

reader.Close();

}

return null;

}

public new bool Delete(int id)

{

var command = Connection.CreateCommand();

command.CommandText = "DELETE FROM Users WHERE DriverID = @id";

var idParameter = command.CreateParameter();

idParameter.ParameterName = "@id";

idParameter.Value = id;

command.Parameters.Add(idParameter);

return command.ExecuteNonQuery() > 0;

}

public User GetByUsername(string username)

{

var command = Connection.CreateCommand();

command.CommandText = "SELECT \* FROM Users WHERE Username = @Username";

var usernameParameter = command.CreateParameter();

usernameParameter.ParameterName = "@Username";

usernameParameter.Value = username;

command.Parameters.Add(usernameParameter);

var reader = command.ExecuteReader();

try

{

if (reader.Read())

{

return GetEntityFromReader(reader);

}

}

finally

{

reader.Close();

}

return null;

}

public bool UserExists(string username)

{

var command = Connection.CreateCommand();

command.CommandText = "SELECT COUNT(1) FROM Users WHERE Username = @Username";

var usernameParameter = command.CreateParameter();

usernameParameter.ParameterName = "@Username";

usernameParameter.Value = username;

command.Parameters.Add(usernameParameter);

int result = (int)command.ExecuteScalar();

return result > 0;

}

public void DeleteByUsername(string username)

{

var command = Connection.CreateCommand();

command.CommandText = "DELETE FROM Users WHERE Username = @Username";

var usernameParameter = command.CreateParameter();

usernameParameter.ParameterName = "@Username";

usernameParameter.Value = username;

command.Parameters.Add(usernameParameter);

command.ExecuteNonQuery();

}

}

}

**Код класса *IMarkService.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Models;

namespace TransportUse.DataLayer.Services.Interfaces

{

public interface IMarkService

{

bool AddMark(string name);

bool UpdateMark(Mark mark, string newName);

bool DeleteMark(int markId);

IEnumerable<Mark> GetAll();

}

}

**Код класса *ITripSheetService.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Models;

namespace TransportUse.DataLayer.Services.Interfaces

{

public interface ITripSheetService

{

bool AddTripSheet(TripSheet tripSheet);

bool UpdateTripSheet(TripSheet tripSheet);

bool DeleteTripSheet(int id);

IEnumerable<TripSheet> GetAll();

IEnumerable<TripSheet> GetTripSheetsByUser(string username);

}

}

**Код класса *IVehicleService.cs***

using TransportUse.DataLayer.Models;

using System.Collections.Generic;

using TransportUse.DataLayer.Models.Transport;

namespace TransportUse.DataLayer.Services.Interfaces

{

public interface IVehicleService

{

bool AddVehicle(string typeVehicle, string registrationNumber, Mark selectedMark, string passengerCapacity, string cargoCapacity);

bool UpdateVehicle(Vehicle vehicle, string typeVehicle, string registrationNumber, Mark selectedMark, string passengerCapacity, string cargoCapacity);

void DeleteVehicle(Vehicle vehicle);

Vehicle Get(int id);

IEnumerable<Vehicle> GetAll();

IEnumerable<Vehicle> GetVehiclesByUser(string username);

}

}

**Код класса *IUserService.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Models.User;

namespace TransportUse.DataLayer.Services.Interfaces

{

public interface IUserService

{

User Register(string username,string fio, string password);

bool AddUser(string username, string fio, string password, string role);

bool UpdateUser(User user, string username, string fio, string password, string role);

bool DeleteUser(User user);

IEnumerable<User> GetAll();

}

}

**Код класса *IReportService.cs***

using System;

using System.Collections.Generic;

using TransportUse.DataLayer.Models.Reports;

namespace TransportUse.DataLayer.Services.Interfaces

{

public interface IReportService

{

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

IEnumerable<CarReport> GenerateCarReport(DateTime startDate, DateTime endDate);

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

IEnumerable<CargoReport> GenerateCargoReport(DateTime startDate, DateTime endDate);

IEnumerable<PassengerReport> GeneratePassengerReport(DateTime startDate, DateTime endDate);

}

}

**Код класса *DriverReport.cs***

using System;

namespace TransportUse.DataLayer.Models.Reports

{

public class DriverReport

{

public int DriverId { get; set; }

public int TotalTrips { get; set; }

public double AverageDistance { get; set; }

}

}

**Код класса *MarkReport.cs***

using System;

namespace TransportUse.DataLayer.Models.Reports

{

public class MarkReport

{

public string MarkName { get; set; }

public double TotalDistance { get; set; }

}

}

**Код класса *CarReport.cs***

using System;

namespace TransportUse.DataLayer.Models.Reports

{

public class CarReport

{

public int VehicleId { get; set; }

public int TotalDistance { get; set; }

}

}

**Код класса *CargoReport.cs***

using System;

namespace TransportUse.DataLayer.Models.Reports

{

public class CargoReport

{

public string MarkName { get; set; }

public int TotalCargo { get; set; }

}

}

**Код класса *PassengerReport.cs***

using System;

namespace TransportUse.DataLayer.Models.Reports

{

public class PassengerReport

{

public string MarkName { get; set; }

public int TotalPassengers { get; set; }

}

}

**Код класса *MarkService.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Interfaces;

using TransportUse.DataLayer.Services.Interfaces;

namespace TransportUse.DataLayer.Services

{

public class MarkService : IMarkService

{

private readonly IMarkRepository \_markRepository;

public MarkService(IMarkRepository markRepository)

{

\_markRepository = markRepository;

}

public bool AddMark(string name)

{

if (string.IsNullOrWhiteSpace(name))

return false;

var newMark = new Mark(name);

\_markRepository.Add(newMark);

return true;

}

public bool UpdateMark(Mark mark, string newName)

{

if (mark == null || string.IsNullOrWhiteSpace(newName))

return false;

mark.Name = newName;

\_markRepository.Update(mark);

return true;

}

public bool DeleteMark(int markId)

{

if (markId <= 0)

return false;

\_markRepository.Delete(markId);

return true;

}

public IEnumerable<Mark> GetAll()

{

return \_markRepository.GetAll();

}

}

}

**Код класса *TripSheetService.cs***

using System.Collections.Generic;

using TransportUse.DataLayer.Interfaces;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Services.Interfaces;

namespace TransportUse.DataLayer.Services

{

public class TripSheetService : ITripSheetService

{

private readonly ITripSheetRepository \_tripSheetRepository;

public TripSheetService(ITripSheetRepository tripSheetRepository)

{

\_tripSheetRepository = tripSheetRepository;

}

public bool AddTripSheet(TripSheet tripSheet)

{

if (tripSheet == null)

return false;

\_tripSheetRepository.Add(tripSheet);

return true;

}

public bool UpdateTripSheet(TripSheet tripSheet)

{

if (tripSheet == null)

return false;

\_tripSheetRepository.Update(tripSheet);

return true;

}

public bool DeleteTripSheet(int id)

{

\_tripSheetRepository.Delete(id);

return true;

}

public IEnumerable<TripSheet> GetAll()

{

return \_tripSheetRepository.GetAll();

}

public IEnumerable<TripSheet> GetTripSheetsByUser(string username)

{

return \_tripSheetRepository.GetTripSheetsByUser(username);

}

}

}

**Код класса *VehicleService.cs***

using TransportUse.DataLayer.Models.Transport;

using TransportUse.DataLayer.Interfaces;

using TransportUse.DataLayer.Services.Interfaces;

using TransportUse.DataLayer.Models;

using System.Collections.Generic;

namespace TransportUse.DataLayer.Services

{

public class VehicleService : IVehicleService

{

private readonly IVehicleRepository \_vehicleRepository;

public VehicleService(IVehicleRepository vehicleRepository)

{

\_vehicleRepository = vehicleRepository;

}

public bool AddVehicle(string typeVehicle, string registrationNumber, Mark selectedMark, string passengerCapacity, string cargoCapacity)

{

int? parsedPassengerCapacity = null;

float? parsedCargoCapacity = null;

if (typeVehicle == "Passenger Car" || typeVehicle == "CargoPassenger Car")

{

if (int.TryParse(passengerCapacity, out int capacity))

{

parsedPassengerCapacity = capacity;

}

else

{

return false;

}

}

if (typeVehicle == "Cargo Car" || typeVehicle == "CargoPassenger Car")

{

if (float.TryParse(cargoCapacity, out float capacity))

{

parsedCargoCapacity = capacity;

}

else

{

return false;

}

}

Vehicle newVehicle;

switch (typeVehicle)

{

case "Passenger Car":

newVehicle = new PassengerCar(typeVehicle, registrationNumber, parsedPassengerCapacity.Value, selectedMark);

break;

case "Cargo Car":

newVehicle = new CargoCar(typeVehicle, registrationNumber, parsedCargoCapacity.Value, selectedMark);

break;

case "CargoPassenger Car":

newVehicle = new CargoPassengerCar(typeVehicle, registrationNumber, parsedPassengerCapacity.Value, parsedCargoCapacity.Value, selectedMark);

break;

default:

return false;

}

\_vehicleRepository.Add(newVehicle);

return true;

}

public bool UpdateVehicle(Vehicle vehicle, string typeVehicle, string registrationNumber, Mark selectedMark, string passengerCapacity, string cargoCapacity)

{

if (vehicle == null || selectedMark == null)

return false;

int? parsedPassengerCapacity = null;

float? parsedCargoCapacity = null;

if (typeVehicle == "Passenger Car" || typeVehicle == "CargoPassenger Car")

{

if (int.TryParse(passengerCapacity, out int pCapacity))

{

parsedPassengerCapacity = pCapacity;

}

else

{

return false;

}

}

if (typeVehicle == "Cargo Car" || typeVehicle == "CargoPassenger Car")

{

if (float.TryParse(cargoCapacity, out float cCapacity))

{

parsedCargoCapacity = cCapacity;

}

else

{

return false;

}

}

vehicle.TypeVehicle = typeVehicle;

vehicle.RegistrationNumber = registrationNumber;

vehicle.Mark = selectedMark;

if (vehicle is PassengerCar passengerCar)

{

passengerCar.PassengerCapacity = parsedPassengerCapacity ?? passengerCar.PassengerCapacity;

}

if (vehicle is CargoCar cargoCar)

{

cargoCar.CargoCapacity = parsedCargoCapacity ?? cargoCar.CargoCapacity;

}

if (vehicle is CargoPassengerCar cargoPassengerCar)

{

cargoPassengerCar.PassengerCapacity = parsedPassengerCapacity ?? cargoPassengerCar.PassengerCapacity;

cargoPassengerCar.CargoCapacity = parsedCargoCapacity ?? cargoPassengerCar.CargoCapacity;

}

\_vehicleRepository.Update(vehicle);

return true;

}

public void DeleteVehicle(Vehicle vehicle)

{

\_vehicleRepository.Delete(vehicle.Id);

}

public IEnumerable<Vehicle> GetAll()

{

return \_vehicleRepository.GetAll();

}

public IEnumerable<Vehicle> GetVehiclesByUser(string username)

{

return \_vehicleRepository.GetVehiclesByUser(username);

}

public Vehicle Get(int id)

{

return \_vehicleRepository.Get(id);

}

}

}

**Код класса *UserService.cs***

using TransportUse.DataLayer.Models.User;

using TransportUse.DataLayer.Interfaces;

using TransportUse.DataLayer.Services.Interfaces;

using System.Collections.Generic;

using System;

using System.Security.Cryptography;

using System.Text;

namespace TransportUse.DataLayer.Services

{

public class UserService : IUserService

{

private readonly IUserRepository \_userRepository;

public UserService(IUserRepository userRepository)

{

\_userRepository = userRepository;

}

private string HashPassword(string password)

{

using (var sha256 = SHA256.Create())

{

var hashedBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));

return BitConverter.ToString(hashedBytes).Replace("-", "").ToLowerInvariant();

}

}

public User Register(string username, string fio, string password)

{

if (\_userRepository.UserExists(username))

{

throw new InvalidOperationException("Пользователь с таким именем уже зарегистрирован");

}

var passwordHash = HashPassword(password);

var user = new User(username, fio, passwordHash, "Driver");

return \_userRepository.Add(user);

}

public User Login(string username, string password)

{

var user = \_userRepository.GetByUsername(username);

var hashedPassword = HashPassword(password);

if (user != null && user.PasswordHash == hashedPassword)

{

return user;

}

return null;

}

public bool AddUser(string username, string fio, string password, string role)

{

if (string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(fio) || string.IsNullOrWhiteSpace(password) || string.IsNullOrWhiteSpace(role))

return false;

if (UserExists(username))

{

return false;

}

var passwordHash = HashPassword(password);

var user = new User(username, fio, passwordHash, role);

\_userRepository.Add(user);

return true;

}

private bool UserExists(string username)

{

return \_userRepository.GetByUsername(username) != null;

}

public bool UpdateUser(User user, string username, string fio, string newPassword, string role)

{

if (user == null || string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(role))

return false;

user.Username = username;

user.FIO = fio;

user.Role = role;

if (!string.IsNullOrWhiteSpace(newPassword))

{

if (user.PasswordHash != HashPassword(newPassword))

{

user.PasswordHash = HashPassword(newPassword);

}

}

\_userRepository.Update(user);

return true;

}

public bool DeleteUser(User user)

{

if (user == null || string.IsNullOrEmpty(user.Username))

return false;

\_userRepository.DeleteByUsername(user.Username);

return true;

}

public IEnumerable<User> GetAll()

{

return \_userRepository.GetAll();

}

}

}

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ**

**Код класса *MarkRepositoryTests.cs***

using Moq;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

using TransportUse.DataLayer;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Repositories;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TransportUse.Tests

{

[TestClass]

public class MarkRepositoryTests

{

private MarkRepository? \_repository;

[TestMethod]

public void Add\_MarkRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteScalar()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new MarkRepository();

var mark = new Mark("Test Name");

var result = \_repository.Add(mark);

Assert.AreEqual(1, result.Id);

mockCommand.Verify(cmd => cmd.ExecuteScalar(), Times.Once);

DbConnection.ConnectionFactory = () => {

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

};

}

[TestMethod]

public void Get\_MarkRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockDataReader = new Mock<IDataReader>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockDataReader.SetupSequence(r => r.Read()).Returns(true).Returns(false);

mockDataReader.Setup(r => r["Name"]).Returns("Test Name");

mockDataReader.Setup(r => r["MarkID"]).Returns(1);

mockCommand.Setup(cmd => cmd.ExecuteReader()).Returns(mockDataReader.Object);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new MarkRepository();

var result = \_repository.Get(1);

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Id);

Assert.AreEqual("Test Name", result.Name);

mockCommand.Verify(cmd => cmd.ExecuteReader(), Times.Once);

DbConnection.ConnectionFactory = () => {

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

};

}

[TestMethod]

public void Delete\_MarkRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteNonQuery()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new MarkRepository();

var result = \_repository.Delete(1);

Assert.IsTrue(result);

mockCommand.Verify(cmd => cmd.ExecuteNonQuery(), Times.Once);

DbConnection.ConnectionFactory = () => {

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

};

}

}

}

**Код класса *TripSheetRepositoryTests.cs***

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Moq;

using System;

using System.Data;

using TransportUse.DataLayer.Models;

using TransportUse.DataLayer.Repositories;

using TransportUse.DataLayer;

namespace TransportUse.Tests

{

[TestClass]

public class TripSheetRepositoryTests

{

private TripSheetRepository? \_repository;

[TestMethod]

public void Add\_TripSheetRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteScalar()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new TripSheetRepository();

var tripSheet = new TripSheet(1, 1, DateTime.Now, 100, 50, 4);

var result = \_repository.Add(tripSheet);

Assert.AreEqual(1, result.Id);

mockCommand.Verify(cmd => cmd.ExecuteScalar(), Times.Once);

}

[TestMethod]

public void Get\_TripSheetRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockDataReader = new Mock<IDataReader>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockDataReader.SetupSequence(r => r.Read()).Returns(true).Returns(false);

mockDataReader.Setup(r => r["VehicleID"]).Returns(1);

mockDataReader.Setup(r => r["DriverID"]).Returns(1);

mockDataReader.Setup(r => r["Date"]).Returns(DateTime.Now);

mockDataReader.Setup(r => r["Distance"]).Returns(100);

mockDataReader.Setup(r => r["CargoAmount"]).Returns(50);

mockDataReader.Setup(r => r["PassengerCount"]).Returns(4);

mockDataReader.Setup(r => r["TripSheetID"]).Returns(1);

mockCommand.Setup(cmd => cmd.ExecuteReader()).Returns(mockDataReader.Object);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new TripSheetRepository();

var result = \_repository.Get(1);

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Id);

Assert.AreEqual(1, result.VehicleId);

Assert.AreEqual(1, result.DriverId);

mockCommand.Verify(cmd => cmd.ExecuteReader(), Times.Once);

}

[TestMethod]

public void Delete\_TripSheetRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteNonQuery()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new TripSheetRepository();

var result = \_repository.Delete(1);

Assert.IsTrue(result);

mockCommand.Verify(cmd => cmd.ExecuteNonQuery(), Times.Once);

}

[TestMethod]

public void GetTripSheetsByUser\_TripSheetRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockDataReader = new Mock<IDataReader>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockDataReader.SetupSequence(r => r.Read()).Returns(true).Returns(false);

mockDataReader.Setup(r => r["VehicleID"]).Returns(1);

mockDataReader.Setup(r => r["DriverID"]).Returns(1);

mockDataReader.Setup(r => r["Date"]).Returns(DateTime.Now);

mockDataReader.Setup(r => r["Distance"]).Returns(100);

mockDataReader.Setup(r => r["CargoAmount"]).Returns(50);

mockDataReader.Setup(r => r["PassengerCount"]).Returns(4);

mockDataReader.Setup(r => r["TripSheetID"]).Returns(1);

mockCommand.Setup(cmd => cmd.ExecuteReader()).Returns(mockDataReader.Object);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new TripSheetRepository();

var result = \_repository.GetTripSheetsByUser("testuser");

Assert.IsNotNull(result);

Assert.AreEqual(1, ((List<TripSheet>)result).Count);

mockCommand.Verify(cmd => cmd.ExecuteReader(), Times.Once);

}

[TestMethod]

public void GetTripSheetsByPeriod\_TripSheetRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockDataReader = new Mock<IDataReader>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockDataReader.SetupSequence(r => r.Read()).Returns(true).Returns(false);

mockDataReader.Setup(r => r["VehicleID"]).Returns(1);

mockDataReader.Setup(r => r["DriverID"]).Returns(1);

mockDataReader.Setup(r => r["Date"]).Returns(DateTime.Now);

mockDataReader.Setup(r => r["Distance"]).Returns(100);

mockDataReader.Setup(r => r["CargoAmount"]).Returns(50);

mockDataReader.Setup(r => r["PassengerCount"]).Returns(4);

mockDataReader.Setup(r => r["TripSheetID"]).Returns(1);

mockCommand.Setup(cmd => cmd.ExecuteReader()).Returns(mockDataReader.Object);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new TripSheetRepository();

var result = \_repository.GetTripSheetsByPeriod(DateTime.Now.AddDays(-7), DateTime.Now);

Assert.IsNotNull(result);

Assert.AreEqual(1, ((List<TripSheet>)result).Count);

mockCommand.Verify(cmd => cmd.ExecuteReader(), Times.Once);

}

}

}

**Код класса *VehicleRepositoryTests.cs***

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Moq;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

using TransportUse.DataLayer;

using TransportUse.DataLayer.Models.Transport;

using TransportUse.DataLayer.Repositories;

using TransportUse.DataLayer.Models;

namespace TransportUse.Tests

{

[TestClass]

public class VehicleRepositoryTests

{

private VehicleRepository? \_repository;

[TestMethod]

public void Add\_VehicleRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteScalar()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new VehicleRepository();

var vehicle = new PassengerCar("Passenger Car", "ABC123", 4, new Mark("Toyota"));

var result = \_repository.Add(vehicle);

Assert.AreEqual(1, result.Id);

mockCommand.Verify(cmd => cmd.ExecuteScalar(), Times.Once);

DbConnection.ConnectionFactory = () =>

{

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

};

}

[TestMethod]

public void Delete\_VehicleRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteNonQuery()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new VehicleRepository();

var result = \_repository.Delete(1);

Assert.IsTrue(result);

mockCommand.Verify(cmd => cmd.ExecuteNonQuery(), Times.Once);

DbConnection.ConnectionFactory = () =>

{

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

};

}

}

}

**Код класса *UserRepositoryTests.cs***

using Moq;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

using TransportUse.DataLayer;

using TransportUse.DataLayer.Repositories;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using TransportUse.DataLayer.Models.User;

namespace TransportUse.Tests

{

[TestClass]

public class UserRepositoryTests

{

private UserRepository? \_repository;

[TestMethod]

public void Add\_UserRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteScalar()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new UserRepository();

var user = new User("testuser", "Test User", "password", "user");

var result = \_repository.Add(user);

Assert.AreEqual(1, result.Id);

mockCommand.Verify(cmd => cmd.ExecuteScalar(), Times.Once);

DbConnection.ConnectionFactory = CreateSqlConnection;

}

[TestMethod]

public void Get\_UserRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockDataReader = new Mock<IDataReader>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockDataReader.SetupSequence(r => r.Read()).Returns(true).Returns(false);

mockDataReader.Setup(r => r["Username"]).Returns("testuser");

mockDataReader.Setup(r => r["FIO"]).Returns("Test User");

mockDataReader.Setup(r => r["PasswordHash"]).Returns("password");

mockDataReader.Setup(r => r["Role"]).Returns("user");

mockDataReader.Setup(r => r["DriverID"]).Returns(1);

mockCommand.Setup(cmd => cmd.ExecuteReader()).Returns(mockDataReader.Object);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new UserRepository();

var result = \_repository.Get(1);

Assert.IsNotNull(result);

Assert.AreEqual("testuser", result.Username);

mockCommand.Verify(cmd => cmd.ExecuteReader(), Times.Once);

DbConnection.ConnectionFactory = CreateSqlConnection;

}

[TestMethod]

public void Delete\_UserRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteNonQuery()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new UserRepository();

var result = \_repository.Delete(1);

Assert.IsTrue(result);

mockCommand.Verify(cmd => cmd.ExecuteNonQuery(), Times.Once);

DbConnection.ConnectionFactory = CreateSqlConnection;

}

[TestMethod]

public void GetByUsername\_UserRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockDataReader = new Mock<IDataReader>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockDataReader.SetupSequence(r => r.Read()).Returns(true).Returns(false);

mockDataReader.Setup(r => r["Username"]).Returns("testuser");

mockDataReader.Setup(r => r["FIO"]).Returns("Test User");

mockDataReader.Setup(r => r["PasswordHash"]).Returns("password");

mockDataReader.Setup(r => r["Role"]).Returns("user");

mockDataReader.Setup(r => r["DriverID"]).Returns(1);

mockCommand.Setup(cmd => cmd.ExecuteReader()).Returns(mockDataReader.Object);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new UserRepository();

var result = \_repository.GetByUsername("testuser");

Assert.IsNotNull(result);

Assert.AreEqual("testuser", result.Username);

mockCommand.Verify(cmd => cmd.ExecuteReader(), Times.Once);

}

[TestMethod]

public void UserExists\_UserRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteScalar()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new UserRepository();

var result = \_repository.UserExists("testuser");

Assert.IsTrue(result);

mockCommand.Verify(cmd => cmd.ExecuteScalar(), Times.Once);

}

[TestMethod]

public void DeleteByUsername\_UserRepository\_Test()

{

var mockConnection = new Mock<IDbConnection>();

var mockCommand = new Mock<IDbCommand>();

var mockParameterCollection = new Mock<IDataParameterCollection>();

mockConnection.Setup(conn => conn.CreateCommand()).Returns(mockCommand.Object);

mockCommand.Setup(cmd => cmd.CreateParameter()).Returns(new Mock<IDbDataParameter>().Object);

mockCommand.Setup(cmd => cmd.Parameters).Returns(mockParameterCollection.Object);

mockCommand.Setup(cmd => cmd.ExecuteNonQuery()).Returns(1);

DbConnection.ConnectionFactory = () => mockConnection.Object;

\_repository = new UserRepository();

\_repository.DeleteByUsername("testuser");

mockCommand.Verify(cmd => cmd.ExecuteNonQuery(), Times.Once);

}

private IDbConnection CreateSqlConnection()

{

var connectionString = ConfigurationManager.ConnectionStrings["TransportConnection"].ConnectionString;

var connection = new SqlConnection(connectionString);

connection.Open();

return connection;

}

}

}

**Код файла конфигурации App.config**<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<connectionStrings>

<add name="TransportConnection" connectionString="Data Source=DESKTOP-FKCCTFK\SQLEXPRESS;Initial Catalog=TransportDB;Integrated Security=True" providerName="System.Data.SqlClient"/>

</connectionStrings>

</configuration>

**ПРИЛОЖЕНИЕ Д**

(обязательное)

**РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Руководство пользователя представляет собой незаменимый источник информации, который охватывает все аспекты работы с разработанным приложением. Оно описывает требования к уровню знаний пользователя, рассказывает о том, как использовать программу, объясняет особенности ее функционирования и подробно описывает процедуры внесения и изменения данных, а также создания и просмотра отчетов.

Пользователь системы должен обладать определенным уровнем квалификации, который включает в себя базовые навыки работы с компьютером и понимание сущности предметной области.

Перед началом работы пользователям предоставляется список обязательных для изучения документов, включая настоящее руководство пользователя и инструкции по обеспечению безопасности при работе с компьютером.

В целом, руководство пользователя является неотъемлемой частью разработанного приложения, которая помогает пользователям освоить его функциональность и достичь оптимальной производительности при работе с программой.

Чтобы начать работу с программой, пользователю необходимо запустить исполняемый файл. После запуска открывается окно авторизации, где пользователь должен ввести свой логин и пароль, а затем нажать кнопку "Войти". Если введённые данные верны, то пользователь увидит окно. Какое из окон откроется зависит от прав доступа пользователя. Общий вид начального окна программы показан на рисунке Г.1.

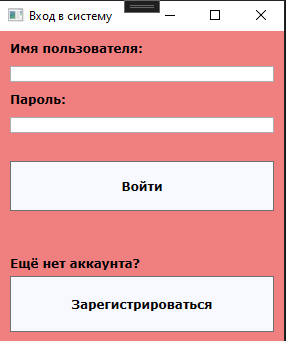


Рисунок Г.1 – Окно входа в систему

Если роль пользователя администратор и он ввёл верные данные при входе, то откроется окно *AdminWindow* на вкладке «Транспорт». Полные права администратора позволяют ему добавлять новый транспорт, редактировать или удалить существующий.

При открытии, администратор увидит сразу не только содержимое каждой таблицы, но и специальную форму, содержащую поля, которые предназначены для редактирования. Они автоматически заполняются в зависимости от выбранной строки. Внешний вид вкладки транспорта окна администратора представлен на рисунке Г.2.

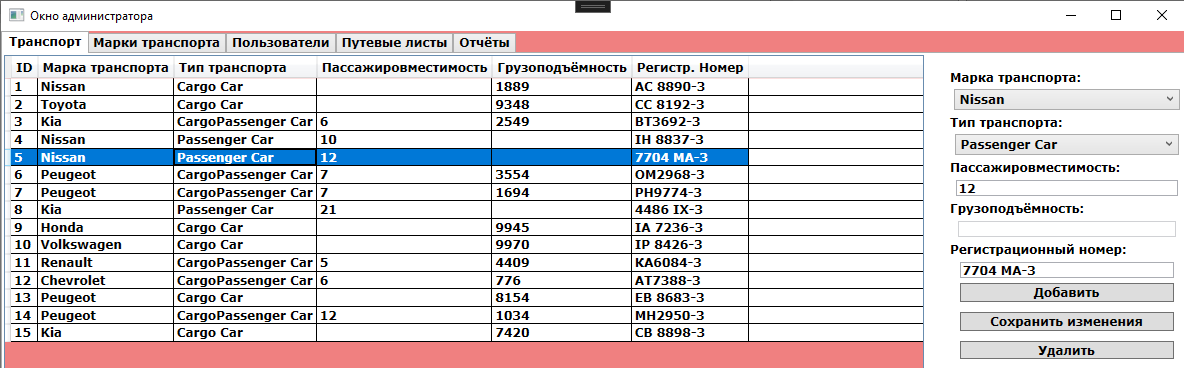


Рисунок Г.2 – Вкладка транспорта окна администратора

Если администратор хочет что-то изменить в данных вкладки транспорта, то ему нужно обратить внимание на:

– интерфейс автоматически отслеживает выбранную строку и заполняет поля данными;

– марка транспорта является *ComboBox*, а значит содержит в себе все типы марки транспорта из таблицы *Marks*;

– тип транспорта также является *ComboBox*. Он содержит 3 вида автомобилей: *Cargo Car* (грузовой), *Passenger Car* (пассажирский), *CargoPassenger Car* (грузопассажирский);

– Поля пассажировместимость и грузоподъёмность автоматически блокируются в зависимости от типа транспорта. Например, при типе *Passenger Car*, ввести значение грузоподъемности не получится;

Если пользователь захочет добавить новую запись или отредактировать имеющуюся, то он должен выбрать строку. Она будет служить образцом для добавления, либо сразу будет изменяемой строкой. Затем пользователь должен внести желаемые изменения. Например, сменить марку транспорта в *ComboBox* и затем вывести нажать кнопку «Добавить», либо кнопку «Сохранить изменения». Пример правильного заполнения формы, а затем добавления новой строки представлен на рисунках Г.3 – Г.4.

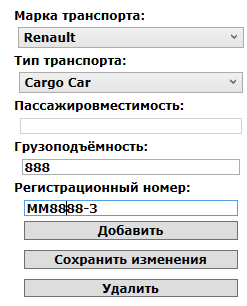


Рисунок Г.3 – Пример заполнения данных для транспорта



Рисунок Г.4 – Новая строка в таблице транспорта

На вкладке марок транспорта администратор может добавлять новую марку. Здесь достаточно ввести новое название и нажать «Добавить». Для редактирования сделать необходимые изменения и нажать «Сохранить изменения». Для удаления выбрать строку нажатием на неё и нажать «Удалить». Вкладка марок транспорта окна администратора представлена на рисунке Г.5.

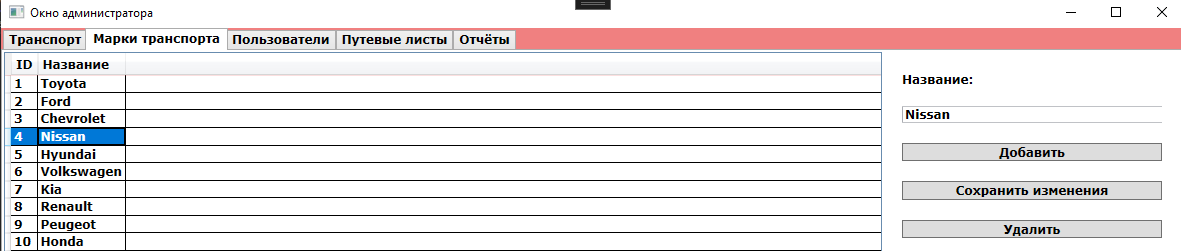


Рисунок Г.5 – Вкладка марок транспорта окна администратора

Пример правильного добавления или редактирования марки представлен на рисунке Г.6.

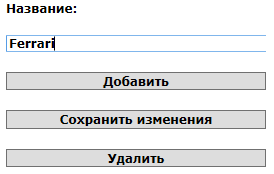


Рисунок Г.6 – Пример правильного заполнения поля для добавления или редактирования марки транспорта

Результат редактирования после нажатия кнопки «Сохранить изменения» представлен на рисунке Г.7.

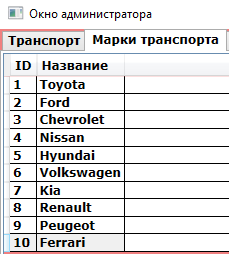


Рисунок Г.7 – Результат редактирования марки транспорта

Администратор имеет права добавлять новых пользователей или редактировать список существующих. Для этого на вкладке «Пользователи» администратору необходимо выбрать строку, которую он хочет редактировать и нажать соответствующие кнопки после изменений. На рисунке Г.8 представлена вкладка пользователей окна администратор.

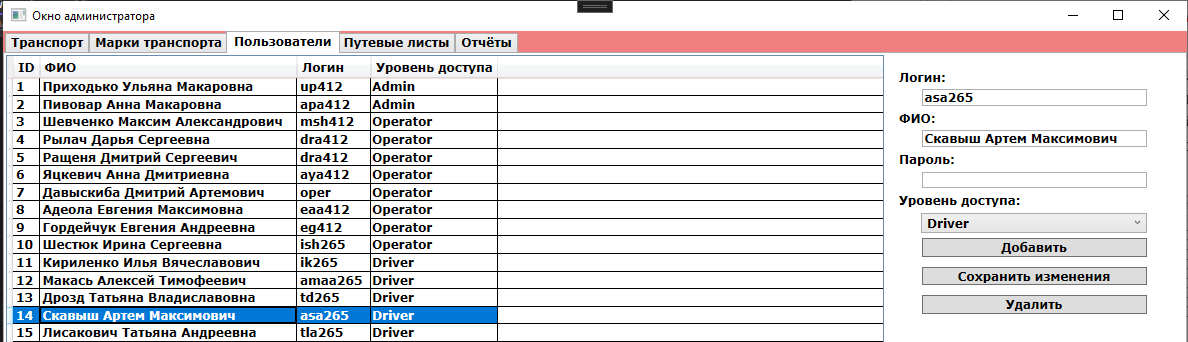


Рисунок Г.8 – Вкладка пользователей окна администратора

Пароль администратор не видит. Это сделано с точки зрения безопасности хранения данных. Но для добавления нового пользователя, администратору необходимо ввести пароль в поле «Пароль». В результате пароль хешируется. Пример правильно добавления нового пользователя представлен на рисунке Г.9.

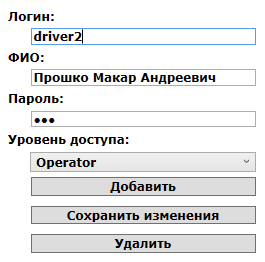


Рисунок Г.9 – Пример правильного добавления пользователя

Вкладка путевых листов позволяет добавлять новые записи, изменять или удалять уже существующие. Вкладка путевых листов окна администратора представлена на рисунке Г.10.

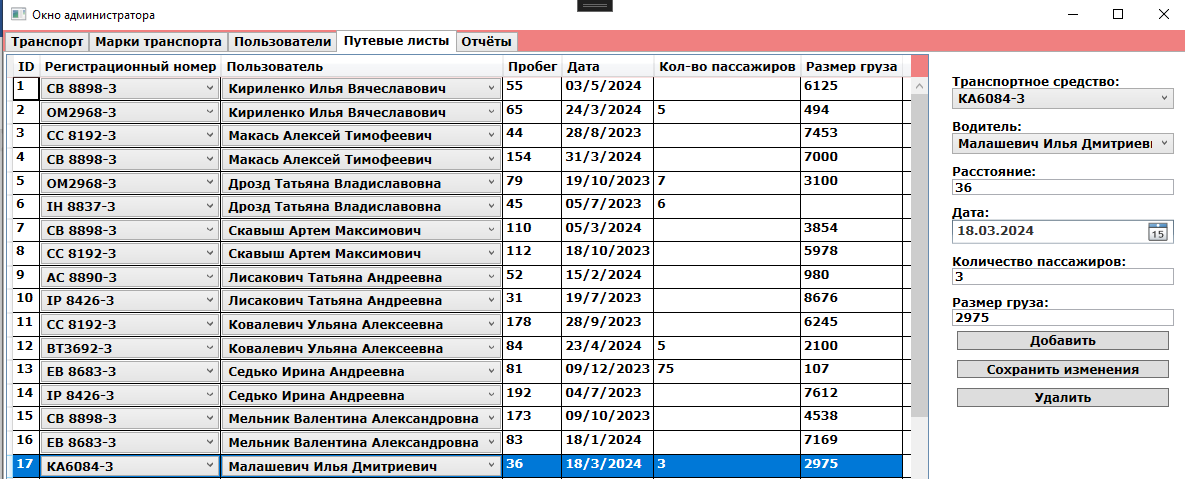


Рисунок Г.10 – Вкладка путевых листов окна администратора

Для успешного добавления нужно правильно ввести данные. Количество пассажиров и размер груза должен соответствовать выбранному типу транспорта. Пример правильного заполнения формы для добавления путевых листов представлен на рисунке Г.11.

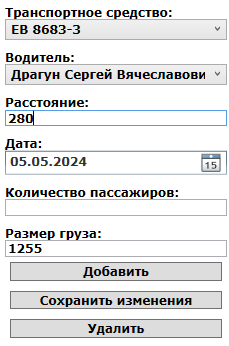


Рисунок Г.11 – Пример правильно заполнения для добавления записи в путевые листы

На рисунке Г.12 представлен результат добавления новой записи в путевые листы путём нажатия кнопки «Добавить».

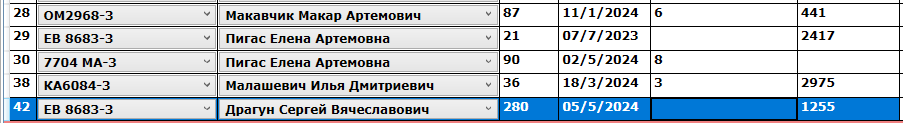


Рисунок Г.12 – Результат добавления новой записи в путевой лист

Для генерации отчётов используется вкладка «Отчёты». Для правильной генерации отчётов необходимо выбрать дату начала и дату окончания периода, за который сформировать отчёт. На рисунке Г.13 представлена вкладка отчётов окна администратора.

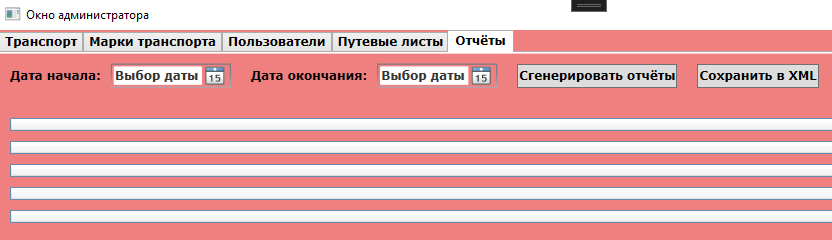


Рисунок Г.13 – Вкладка отчётов окна администратора

Чтобы получить отчёты, необходимо выбрать период и нажать кнопку «Сгенерировать отчёты». В результате появятся 5 отчётов. На рисунке Г.14 представлен результате генерации отчётов за небольшой период.



Рисунок Г.14 – Результат правильной генерации отчётов

Для сохранения отчётов в формате *XML* нужно нажать кнопку «Сохранить в *XML*», сгенерировав предварительно отчёты через кнопку «Сгенерировать отчёты». В результате они будут сохранены по пути *bin/debug*. На рисунке Г.15 представлен положительный результат сохранения отчётов в формате *XML*.

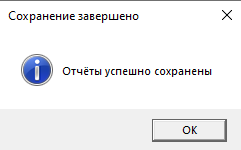


Рисунок Г.15 – Удачное сохранение отчётов в формате *XML*

Если роль пользователя оператор, и он ввёл верные данные при входе, то откроется окно *OperatorWindow* на вкладке «Транспорт». Оператор может изменять данные транспорта и путевых листов.

Все действия с вкладкой транспорта в окне администратора повторяются и в окне оператора. На рисунке Г.16 представлена вкладка транспорта окна оператора.

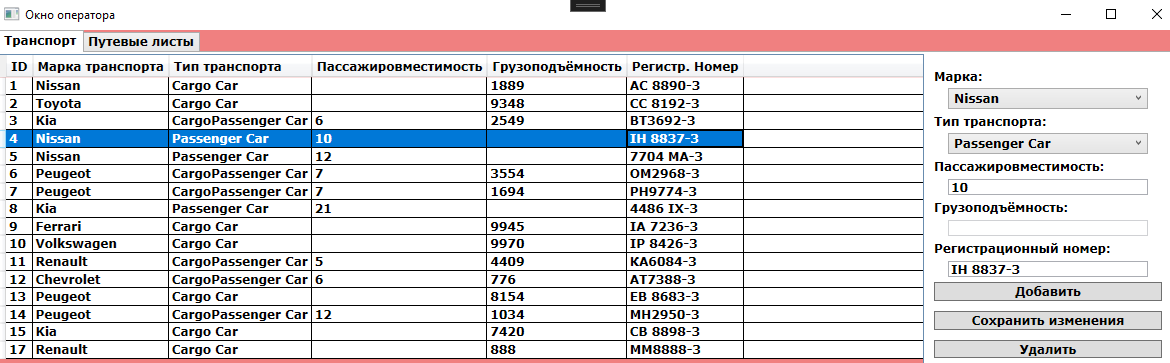


Рисунок Г.16 – Вкладка транспорта окна оператора

Все действия с вкладкой путевых листов в окне администратора повторяются и в окне оператора. На рисунке Г.17 представлена вкладка путевых листов окна оператора.

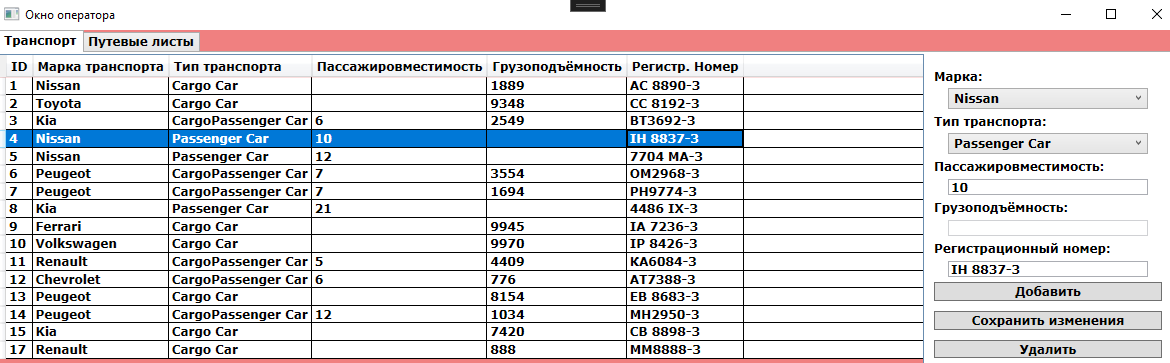


Рисунок Г.17 – Вкладка путевых листов окна оператора

Если роль пользователя оператор, и он ввёл верные данные при входе, то откроется окно *DriverWindow* на вкладке «Транспорт». Водитель может просматривать только информацию о себе. Поэтому он будет видеть только те записи, которые соответствуют его *Username*. На рисунке Г.18 представлена вкладка транспорта окна водителя.

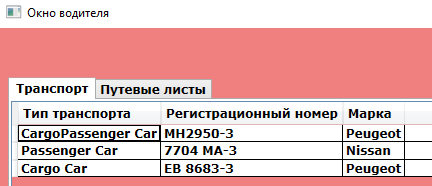


Рисунок Г.18 – Вкладка транспорта окна водителя

Редактировать ничего водитель не может. Добавлять и удалять тоже. На рисунке Г.19 представлена вкладка путевых листов окна водителя, где он видит только информацию о себе.



Рисунок Г.19 – Вкладка путевых листов окна водителя

Полностью закрыть систему можно нажав крестик в углу экрана.

**ПРИЛОЖЕНИЕ Ж**

(обязательное)

**РУКОВОДСТВО ПРОГРАММИСТА**

Созданное программное обеспечение обеспечивает эффективный доступ и управление данными через технологию *LINQ*. Это инструмент, который позволяет пользователю не только просматривать, но и добавлять, редактировать, а также фильтровать информацию в удобной форме, гарантируя высокий уровень интерактивности и контроля. Программа написана на языке программирования C#. Он обеспечивает эффективное, объектно-ориентированное приложения. Эта программа была специально разработана для мониторинга и управления автотранспортом компании, позволяя оптимизировать процессы использования и распределения транспортных средств.

Пользовательское приложение распространяется в виде исполняемого файла (.*exe*), который не требует дополнительных компонентов или сложных процедур установки для его функционирования на клиентских компьютерах, упрощая процесс начала работы.

В проектировании программы не предусматриваются специализированные требования или ограничения, что обеспечивает ее гибкость и доступность для разнообразных пользовательских сред. Встроенная система обратной связи информирует пользователя об ошибках ввода, способствуя коррекции и предотвращению потенциальных проблем в процессе взаимодействия с программой.

Исходные данные для работы приложения поступают непосредственно из базы данных предприятия, а также вносятся пользователями через интерфейс программы, когда это необходимо для выполнения определенной функции или задачи. Результатом работы программы могут быть не только успешно выполненные операции, но и уведомления о допущенных ошибках ввода, что значительно упрощает процесс выявления и устранения недочетов.

В процессе регистрации новые пользователи должны предоставить личную информацию, включая имя, логин и пароль. Приложение включает в себя систему проверки подлинности данных, обеспечивая высокий уровень безопасности и предотвращая доступ неавторизованных лиц.

В случае введения неверных учетных данных, программа выдает точные инструкции об ошибке, сопроводительные сообщения и указания по восстановлению доступа, поддерживая эффективное взаимодействие пользователя с системой.

**ПРИЛОЖЕНИЕ И**

(обязательное)

**РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА**

Приложение было специально создано как инструмент для эффективного учета использования автотранспорта на предприятии и предоставляет возможность его пользователям проводить авторизацию в системе и взаимодействовать с данными, связанными с потребителями, маршрутами, различными видами и марками транспортных средств. Оно также позволяет составлять всесторонние отчеты для любого необходимого отрезка времени.

Это программное решение имеет простоту в эксплуатации на стороне пользователей за счет необходимости наличия лишь основного исполняемого файла с расширением .*exe*, который позволяет доступно и оперативно начать работу с системой.

Данное программное обеспечение создано для стабильной работы на различных версиях операционной системы *Windows*, начиная от версии 7 и заканчивая последней редакцией *Windows* 11, однако поддержка более старых версий отсутствует, что может привести к техническим ошибкам в работе. Следует подчеркнуть, что несмотря на разработку в среде *Visual Studio*, для запуска и функционирования приложения не требуется установка дополнительного специализированного ПО, что делает его удобным и мобильным решением – оно может быть запущено из любого местоположения на жестком диске.

Инициирование работы с приложением происходит путем активации файла *TransportUse.exe*, что в свою очередь приводит к открытию его графического пользовательского интерфейса – интуитивно понятного и дружелюбного к пользователю.

Кроме того, в качестве тестовых данных для оценки и проверки работоспособности системы авторизации предусмотрена возможность входа в систему под учетной записью администратора, используя стандартные учетные данные: логин «*admin*» и пароль «123». В ходе эксплуатации, если возникнут сбои или ошибки, рекомендуется перезагрузить приложение.

В случаях, когда повторный запуск не ведет к решению проблемы, следует тщательно проверить корректность введенных данных и при необходимости скорректировать их или обратиться за помощью к службе технической поддержки.