МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированное проектирование и программирование»

Тема «Программное средство «Железнодорожный вокзал»»

Исполнитель

студент (ка) 3 курса группы 7 Тимошенко Д. В.

(Ф.И.О.)

Руководитель работы преп.-стажер Якунович А. В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель

(подпись)

Минск 2023

**Содержание**

[Введение 6](#_Toc154008215)

[1. Аналитический обзор литературы по теме проекта 7](#_Toc154008216)

[1.1. Сравнительный анализ теоретических и эвристических методов решения поставленных задач и существующих по данной тематике технических научных решений 7](#_Toc154008217)

[1.2. Аналитический обзор аналогов 8](#_Toc154008218)

[1.2.1. Аналог «Белорусская железная дорога» 8](#_Toc154008219)

[1.2.2. Аналог «Intercity.pl» 9](#_Toc154008220)

[1.3. Изучение требований, определение вариантов использования 10](#_Toc154008221)

[1.3.1. Определение основных функциональных требований к программному средству 10](#_Toc154008222)

[1.3.2. Определение вариантов использования 11](#_Toc154008223)

[1.4. Вывод 11](#_Toc154008224)

[2. Анализ и проектирование архитектуры программного средства 12](#_Toc154008225)

[2.1. Технические средства разработки 12](#_Toc154008226)

[2.1.1. Язык программирования C# и платформа .NET 12](#_Toc154008227)

[2.1.2. ODP.NET 13](#_Toc154008228)

[2.1.3. Windows Presentation Foundation (WPF) 13](#_Toc154008229)

[2.1.4. CommunityToolkit.MVVM 14](#_Toc154008230)

[2.1.5. MaterialDesign 14](#_Toc154008231)

[2.1.6. Oracle Database 19c 15](#_Toc154008232)

[2.2. Описание информационых объектов и ограничений целостности в базе данных 15](#_Toc154008233)

[2.2.1. Таблица Stations 16](#_Toc154008234)

[2.2.2. Таблица Routes 17](#_Toc154008235)

[2.2.3. Таблица Schedule 17](#_Toc154008236)

[2.2.4. Таблица Trains 17](#_Toc154008237)

[2.2.5. Таблица Vans 18](#_Toc154008238)

[2.2.6. Таблица Passengers 18](#_Toc154008239)

[2.2.7. Таблица Payments 18](#_Toc154008240)

[2.2.8. Таблица Tickets 18](#_Toc154008241)

[2.2.9. Таблица Stations\_routes 19](#_Toc154008242)

[2.3. Диаграммы UML, взаимосвязь всех компонентов 19](#_Toc154008243)

[2.4. Обобщенная архитектура программного средства 20](#_Toc154008244)

[2.5. Вывод 21](#_Toc154008245)

[3. Проектирование программного средства 22](#_Toc154008246)

[3.1. Общая структура 22](#_Toc154008247)

[3.2. Взаимоотношения между классами 25](#_Toc154008248)

[3.3. Проектирование архитектуры приложения 25](#_Toc154008249)

[3.4. Проектирование последовательностей проекта 27](#_Toc154008250)

[3.5. Вывод 27](#_Toc154008251)

[4. Реализация программного средства 28](#_Toc154008252)

[4.1. Основные классы программного средства 28](#_Toc154008253)

[4.2. Описание классов и методов программного средства 28](#_Toc154008254)

[4.2.1. Выполнение входа 28](#_Toc154008255)

[4.2.2. Изменение параметров входа 28](#_Toc154008256)

[4.2.3. Главная страница 29](#_Toc154008257)

[4.2.4. Страница «Home» 29](#_Toc154008258)

[4.2.5. Страница «Search» 29](#_Toc154008259)

[4.2.6. Страница «Board» 29](#_Toc154008260)

[4.2.7. Страница «Orders» 30](#_Toc154008261)

[4.2.8. Страница «More» 30](#_Toc154008262)

[4.2.9. Страница «Admin» 30](#_Toc154008263)

[4.2.10. Класс OracleContext 31](#_Toc154008264)

[4.3 Вывод 31](#_Toc154008265)

[5. Тестирование, проверка работоспособности и анализ полученных результатов. 32](#_Toc154008266)

[5.1. Тестирование авторизации 32](#_Toc154008267)

[5.2. Тестирование поиска 33](#_Toc154008268)

[5.3. Тестирование заказов билетов 34](#_Toc154008269)

[5.4. Тестирование панели администратора 40](#_Toc154008270)

[5.5. Вывод. 42](#_Toc154008271)

[6. Руководство по использованию программного продукта 43](#_Toc154008272)

[6.1. Вывод 46](#_Toc154008273)

[Заключение 47](#_Toc154008274)

[Список литературы 48](#_Toc154008275)

[Приложение А Диаграмма базы данных 49](#_Toc154008276)

[Приложение Б Use-case диаграмма 50](#_Toc154008277)

[Приложение В Диаграмма классов 51](#_Toc154008278)

[Приложение Г Диаграмма последовательностей 52](#_Toc154008279)

[Приложение Д Листинги кода 53](#_Toc154008280)

**ВВЕДЕНИЕ**

Объектно-ориентированное программирование (ООП) является одним из основных подходов в современной разработке программного обеспечения. ООП позволяет организовать код в виде объектов, которые взаимодействуют друг с другом, образуя сложные системы. Этот подход позволяет создавать гибкие, модульные и легко поддерживаемые программы.

Также в нашем современном мире имеется огромный спрос пользователей на приложения, для взаимодействия с данными в базе данных, которые могли бы постоянно работать и позволять удобно пользоваться хранимой информацией. Особенно важным становится использование баз данных в железнодорожной отрасли для обработки и хранения данных о состоянии железнодорожных вокзалов и поездов.

В рамках данного проекта рассматривается разработка программного средства "Железнодорожный вокзал". Железнодорожные вокзалы играют важную роль в организации пассажирских перевозок, обеспечивая комфорт и безопасность пассажиров. Целью программного средства является автоматизация процессов, связанных с управлением и организацией работы железнодорожного вокзала.

Программное средство "Железнодорожный вокзал" будет разработано с использованием принципов и методов объектно-ориентированного программирования. ООП позволит представить различные сущности вокзала, такие как поезда, расписание, билеты и пассажиры, в виде объектов с определенными свойствами и поведением. Благодаря этому подходу, разработчики смогут создать модульную систему, которая будет легко расширяемой и поддерживаемой.

1. Аналитический обзор литературы по теме проекта

1.1. Сравнительный анализ теоретических и эвристических методов решения поставленных задач и существующих по данной тематике технических научных решений

Существует множество методов решения этой задачи, которые могут быть классифицированы по следующим параметрам:

1. Производительность: скорость выполнения запросов пользователя и эффективность использования ресурсов.
2. Надежность: способность системы работать без сбоев, а также ее способность восстановиться после сбоя.
3. Гибкость: способность системы адаптироваться к изменениям требований и удовлетворять различным потребностям пользователей.
4. Расширяемость: возможность системы расширяться и добавлять новые функциональные возможности.

Существует также множество технических научных решений в контексте ООП, включая разработку программных инструментов, фреймворков и библиотек. Например, фреймворки Spring и Hibernate предоставляют инструменты для разработки и управления объектно-ориентированными приложениями, обеспечивая реализацию инверсии управления и персистентности данных.

Теоретические методы решения задач в ООП представляют собой формальные подходы и модели, разработанные для анализа и проектирования систем на основе ООП. В литературе были исследованы различные теоретические методы, такие как моделирование классов, анализ наследования и принципы SOLID (Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion).

Например, в работе "Object-Oriented Analysis and Design with Applications" Гради Буча описывает принципы и методы анализа и проектирования в ООП [1]. Автор предлагает использовать моделирование классов для выявления основных сущностей системы и определения их взаимодействия. Анализ наследования позволяет строить иерархии классов, обеспечивая повторное использование кода и гибкость системы.

Эвристические методы решения задач в ООП основаны на эмпирическом опыте и эвристических правилах, позволяющих разработчикам применять стандартные решения для типовых задач. В литературе были изучены различные эвристические методы, такие как шаблоны проектирования, архитектурные принципы и методологии разработки, такие как Agile и Scrum.

Например, в книге "Design Patterns: Elements of Reusable Object-Oriented Software" Гамма, Хелм, Джонсон и Влиссидес представляют каталог шаблонов проектирования, которые предлагают решения для типичных проблем проектирования в ООП [2]. Эти шаблоны, такие как "Фабричный метод", "Стратегия" и "Наблюдатель", позволяют разработчикам создавать гибкие и расширяемые системы.

1.2. Аналитический обзор аналогов

В современном мире, когда скорость и точность играют важную роль, транспортные компании становятся все более популярными. Одной из главных задач таких компаний является надежная доставка клиентов в места назначения.

1.2.1. Аналог «Белорусская железная дорога»

В качестве аналога рассмотрим приложение белорусской железной дороги[3], которое представлено на рисунке 1.1.

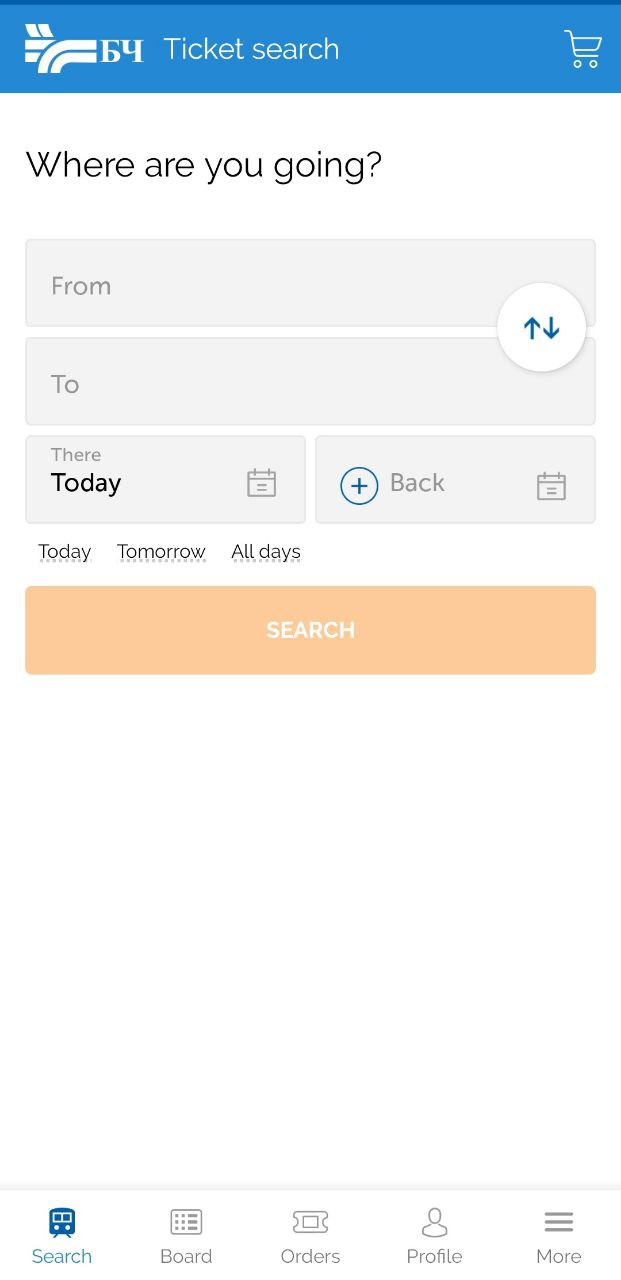


Рисунок 1.1 – Приложение «БЧ. Мой поезд»

Данное приложение позволяет из любой точки страны, где имеется доступ в интернет, забронировать билет на поезд. Пример представлен на рисунке 1.2.

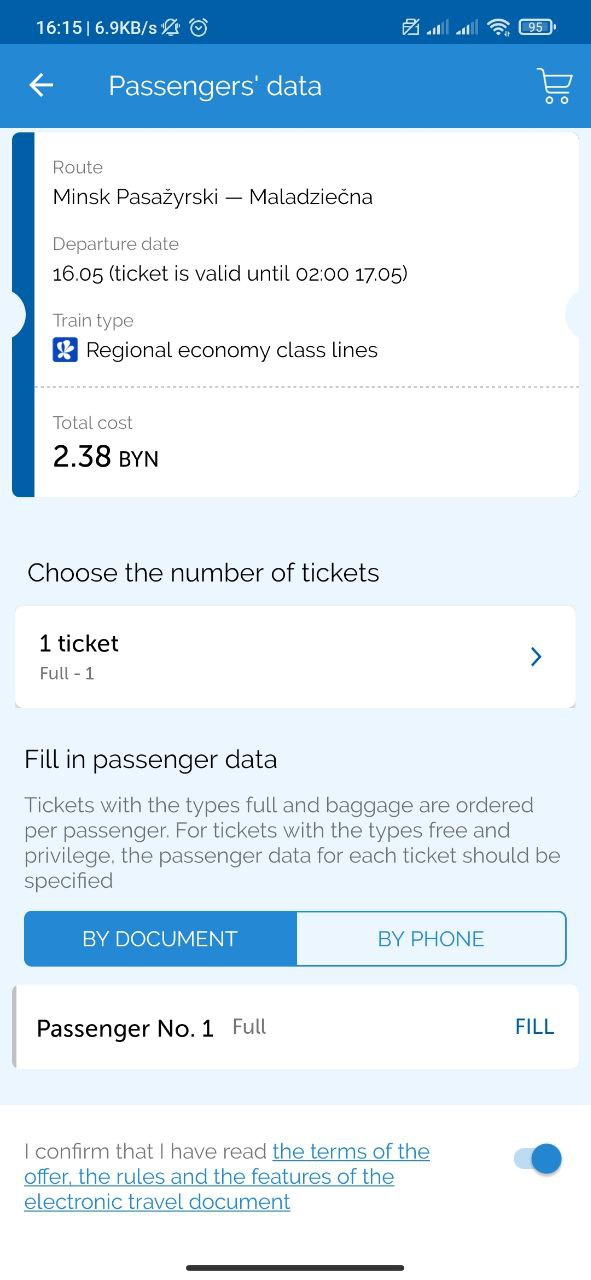


Рисунок 1.2 – Бронирование билета в приложении «БЧ. Мой поезд»

Также приложение позволяет просмотреть расписание рейсов онлайн. Пример на рисунке 1.3.

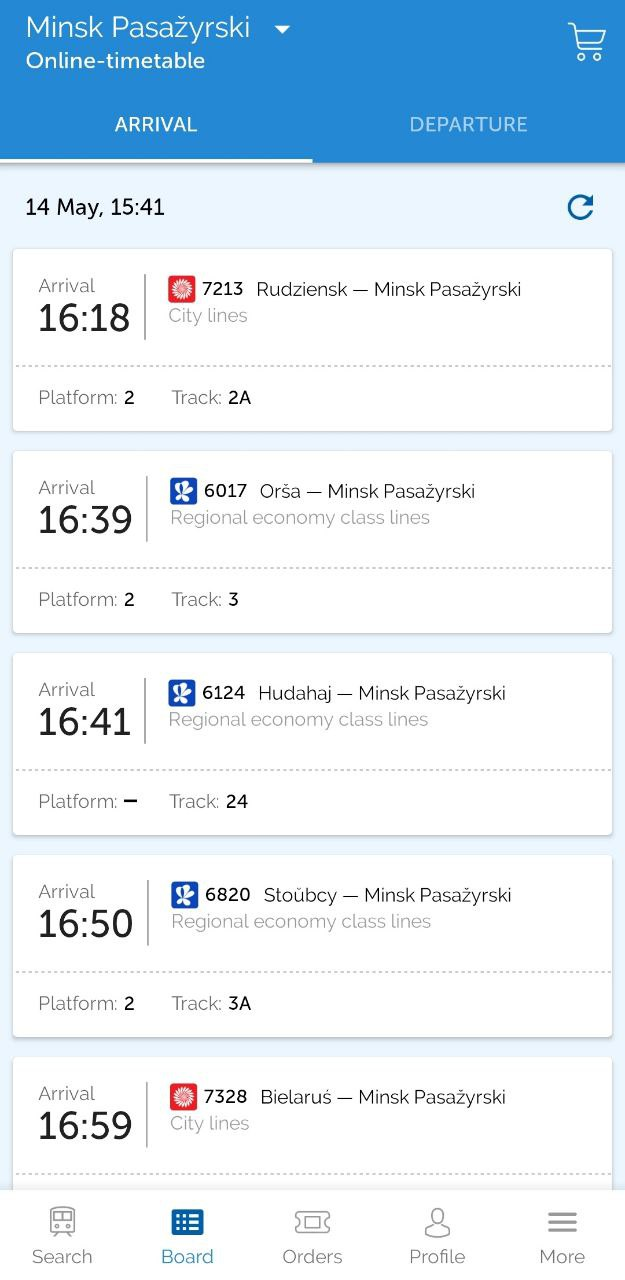


Рисунок 1.3 – Расписание рейсов в приложении «БЧ. Мой поезд»

Однако, как и любое подобное приложение, у него есть минусы. Одним из таких минусов можно назвать ошибки при бронировании билетов. У некоторых пользователей возникают проблемы с бронированием билетов через приложение. Например, система может выдавать ошибки при оплате или не отображать доступные для бронирования билеты.

1.2.2. Аналог «Intercity.pl»

Еще одним аналогом является сайт «Intercity.pl», предоставляющий услуги по бронированию билетов на территории Польши [4]. Сайт продемонстрирован на рисунке 1.4.

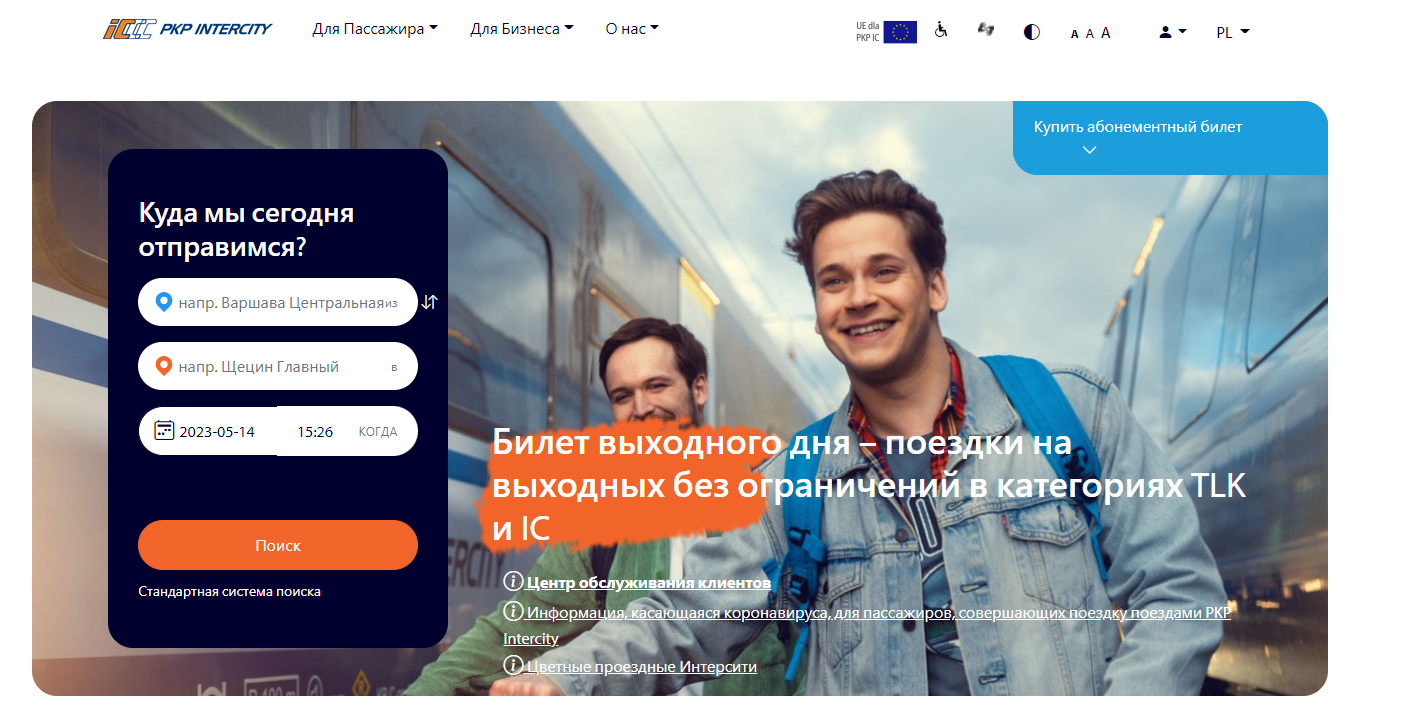


Рисунок 1.4 – Сайт «Intercity.pl»

Из минусов, можно отметить, наличие только возможности поиска билетов и бронирования билетов.

1.3. Изучение требований, определение вариантов использования

Важным этапом разработки проекта является изучение требований и определение возможных вариантов использования.

1.3.1. Определение основных функциональных требований к программному средству

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований можно рассматривать общие требования к разрабатываемому средству. К их числу относятся:

* простота и лёгкость интерфейса;
* использование принципов объектно-ориентированного программирования;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);

Весь дальнейший процесс проектирования и разработки программного средства должен находиться в очерченных бизнес-требованиями границах.

Следующими требованиями являются требования пользователей. Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы. Пользователь данного программного решения должен иметь возможность:

* входить в приложение, после ввода необходимых данных;
* просматривать доску расписания и существующие заказы;
* выполнять поиск необходимых рейсов;
* сортировать информацию по необходимому критерию;
* создавать новые заказы.

Менеджер имеет возможность:

* Поддерживать работу с базой данных;
* Изменять данных, хранящиеся в базе данных.

После проведения анализа были выявлены следующие функциональные требования:

* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем;
* приложение должно предоставлять возможность получения информации по различным критериям;
* приложение должно предоставлять информацию о расписании движения поездов пассажирам.
* приложение должно позволять создавать, изменять, удалять данных из базы данных.
* приложение должно отображать информацию о билетах на поезда, включая стоимость, маршрут, дату и время отправления, место в вагоне.
* приложение должно позволять выполнять поисковые запросы и фильтрацию.
* приложение должно предоставлять интерфейс для бронирования билетов.
* приложение должно обеспечивать защиту данных и доступ к базе данных, только авторизованных пользователей.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

1.3.2. Определение вариантов использования

Вариантами использования программного средства являются:

1. Авторизация пользователей или менеджеров.
2. Просмотр расписания поездов.
3. Просмотр информации о наличии мест на поездах.
4. Бронирование места на поезд.
5. Создание расписания со стороны менеджера.
6. Просмотр оформленных билетов.
7. Администрирование со стороны менеджера.

1.4. Вывод

В данном разделе проведен аналитический обзор литературы по теме проекта, а также осуществлен сравнительный анализ теоретических и эвристических методов решения поставленных задач и существующих технических научных решений.

Также был проведен аналитический обзор аналогов, включающий два конкретных примера аналогов: «Белорусская железная дорога» и «Intercity.pl». Это позволило определить преимущества и недостатки существующих решений и использовать эту информацию при разработке нового проекта.

2. Анализ и проектирование архитектуры программного средства

**2.1. Технические средства разработки**

Для создания программного средства "Железнодорожный вокзал" были выбраны передовые технологии и инструменты, чтобы обеспечить высокое качество и удобство использования. В процессе разработки был использован язык программирования C#, платформа .NET Framework, пакет CommunityToolkit.MVVM и инновационная технология Windows Presentation Foundation (WPF).

Организация работы с базой данных в приложении осуществляется с помощью ODP.NET, это ADO.NET драйвер, который обеспечивает быстрый доступ клиентов Microsoft .NET Core к базам данных Oracle. База данных разработана с использованием Oracle Database 19c, обеспечивая надежность и гибкость хранения данных.

Для создания привлекательного и интуитивно понятного пользовательского интерфейса программного средства "Железнодорожный вокзал" была использована библиотека MaterialDesign. Это позволило создать эстетически привлекательный и удобный интерфейс, соответствующий современным требованиям пользователей.

**2.1.1. Язык программирования C# и платформа .NET**

C# — это не только объектно-ориентированный язык программирования, но и сокровищница возможностей для разработки мощных и гибких приложений в различных областях. Рожденный в конце 90-х годов прошлого века благодаря талантливым разработчикам из Microsoft, C# привносит свежесть и инновации в мир программирования.

Один из главных вдохновителей и архитекторов C# - Андерс Хейлсберг, чье имя тесно связано с идеями Turbo Pascal и Delphi. Под его руководством C# стал языком, способным воплотить амбициозные проекты и преобразить их в реальность.

А что же платформа .NET? Она представляет собой еще одно грандиозное достижение Microsoft. Сочетая в себе набор уникальных особенностей, .NET становится мощной платформой для создания приложений. Во-первых, она поддерживает не только C#, но и другие языки программирования и их диалекты, что расширяет возможности разработчиков. Во-вторых, .NET является переносимой платформой, способной работать на различных операционных системах благодаря своей актуальной версии, доступной для большинства платформ. В-третьих, платформа предоставляет разработчикам мощную библиотеку классов, которая является их надежным партнером в создании сложных приложений. И наконец, автоматическая сборка мусора в .NET облегчает жизнь разработчикам, позволяя им фокусироваться на создании функциональности, не отвлекаясь на управление памятью.

Таким образом, C# и платформа .NET представляют собой мощный союз, обеспечивающий разработчикам средства и среду для творчества и инноваций. Они открывают новые горизонты в программировании и являются надежными инструментами для создания современных и высокоэффективных приложений.

**2.1.2. ODP.NET**

ODP.NET (Oracle Data Provider for .NET) — это набор инструментов, предоставляемых Oracle для взаимодействия с базами данных Oracle из приложений .NET. Он предоставляет высокопроизводительный и функциональный интерфейс для доступа и манипулирования данными в базах данных Oracle.

ODP.NET важен для разработчиков, поскольку он позволяет быстро начать разработку приложений, используя управляемые драйверы ODP.NET и Entity Framework. Это облегчает процесс разработки, поскольку действия по загрузке, установке и настройке комплектов разработки, связанных с ODP.NET, автоматизированы в Visual Studio.

Данный инструмент обеспечивает прямой доступ к базам данных Oracle, что позволяет разработчикам избегать использования общего интерфейса OLE DB или ODBC. Это увеличивает производительность и упрощает разработку, поскольку разработчики могут использовать специфические для Oracle функции.

ODP.NET поддерживает все основные функции Oracle, включая RAC, Performance Counters, Fast Connection Failover, Transaction Guard, Application Continuity, Sharding, и многие другие. Это делает его идеальным выбором для разработки высокопроизводительных и надежных приложений.

В заключение, ODP.NET — это мощный и гибкий инструмент для разработки .NET приложений, которые взаимодействуют с базами данных Oracle. Он предлагает множество функций и поддерживает все основные функции Oracle, что делает его идеальным выбором для разработки высокопроизводительных и надежных приложений.

**2.1.3. Windows Presentation Foundation (WPF)**

Windows Presentation Foundation (WPF) представляет собой изысканную структуру пользовательского интерфейса, которая вдохнула жизнь в настольные клиентские приложения. Эта платформа является истинным творческим пространством, предоставляющим разработчикам мощный инструментарий для создания впечатляющих и интерактивных пользовательских интерфейсов.

Одной из фундаментальных особенностей WPF является использование языка XAML - декларативного языка описания интерфейса, основанного на XML. XAML открывает двери в мир бесконечных возможностей, предоставляя панели, элементы управления, документы и графические фигуры. Это позволяет разработчикам создавать сложные макеты с несколькими страницами, воплощая свои идеи в прекрасные визуальные композиции.

WPF воплощает суть красоты и гибкости, предоставляя возможность создавать стили, темы и словари ресурсов для приложений. Благодаря этому, дизайнеры и разработчики могут сотворить неповторимые пользовательские элементы управления и преобразить шаблоны уже существующих. Каждый элемент интерфейса становится частью художественного произведения, где визуальные эффекты и анимации оживляют пользовательский опыт.

В итоге, WPF представляет собой гармоничное слияние технологии и искусства, где разработчики могут воплотить свои творческие видения в великолепные настольные приложения. Она подарит вам не только функциональность, но и эстетическое удовольствие, воплощая мир мультимедиа и анимации в пользовательском интерфейсе. WPF - это вдохновение для создания приложений, которые не только впечатляют функциональностью, но и приносят радость взаимодействия с пользователем.

**2.1.4. CommunityToolkit.MVVM**

CommunityToolkit.MVVM, ранее известный как MicrosoftToolkit.Mvvm — это современная, быстрая и модульная библиотека MVVM. Он является частью .NET Community Toolkit и построен на следующих принципах:

* Независимость от платформы и среды выполнения: .NET Standard 2.0, .NET Standard 2.1 и .NET 6.
* Простота получения и использования: нет строгих требований к структуре приложений или парадигмам кодирования.
* А-ля карт: свобода выбора компонентов для использования.
* Эталонная реализация: отказоустойчивая и производительная.

Community.Toolkit.MVVM поддерживается и публикуется Microsoft, а также является частью .NET Foundation. Он также используется в нескольких приложениях, встроенных в Windows, таких как Microsoft Store.

Этот пакет предназначен для .NET Standard, поэтому его можно использовать на любой платформе приложений: UWP, WinForms, WPF, Xamarin, Uno и многое другое. Он работает на всех из них.

Community.Toolkit.MVVM также имеет целевой объект .NET 6, который используется для включения дополнительных внутренних оптимизаций при запуске в .NET 6.

В общем, CommunityToolkit.MVVM – это мощный инструмент для создания современных приложений с использованием шаблона MVVM. Он предоставляет стандартные, автономные, упрощенные типы, которые обычно достаточны для многих пользователей для создания приложений без дополнительных внешних ссылок.

**2.1.5. MaterialDesign**

MaterialDesign в WPF C# — это библиотека стилей, которая позволяет разработчикам применять принципы Material Design к их приложениям WPF. Material Design — это система дизайна, разработанная Google, которая использует слои, анимацию и переходы для создания более интуитивно понятных и эстетически приятных интерфейсов.

Библиотека MaterialDesign в WPF C# предоставляет стили для всех основных элементов управления WPF, а также дополнительные элементы управления для поддержки темы, включая многофункциональные кнопки, карточки, диалоговые окна, часы и многое другое. Она также обеспечивает легкую настройку палитры (во время проектирования и выполнения), согласно руководствам Google.

Однако, стоит отметить, что при использовании этой библиотеки могут возникнуть некоторые проблемы. Например, программа может не запускаться, если стиль применяется неправильно. В таких случаях рекомендуется запустить проект без отладки.

В целом, MaterialDesign в WPF C# – это мощный инструмент для создания современных и эстетически приятных приложений. Он предлагает широкий спектр стилей и элементов управления, которые соответствуют принципам Material Design, делая интерфейсы более интуитивно понятными и приятными для глаз.

**2.1.6. Oracle Database 19c**

Oracle Database 19c — это выдающаяся система управления реляционными базами данных, созданная с особым вниманием к деталям корпорацией Oracle. В ее сердце заложена мощь и гибкость, которая обеспечивает высочайшую скорость работы, надежность и безопасность данных, а также простоту использования и администрирования.

С верным спутником – языком запросов SQL, Oracle Database 19c становится готовым рыцарем, способным исполнить все желания. SQL предлагает богатый набор операторов, охватывающих создание, изменение и удаление объектов баз данных, манипулирование данными, контроль доступа и управление транзакциями. Этот язык открывает двери в мир гибкости и функциональности, позволяя воплотить самые смелые идеи в работу с данными.

Oracle Database 19c не только предлагает выдающуюся производительность и надежность, но и обладает впечатляющей системой шифрования данных, обеспечивая безопасность и конфиденциальность важной информации. Она становится надежным союзником в сохранении ценных данных и защите их от нежелательных глаз.

В наше время Oracle Database 19c является великим лидером среди систем управления базами данных. Ее популярность и превосходство подтверждают ее путь к славе, ставя ее на передовом месте в мире баз данных. Благодаря своим превосходным возможностям и широкому применению, Oracle Database 19c становится надежным союзником в решении самых сложных задач управления данными.

2.2. Описание информационых объектов и ограничений целостности в базе данных

Для разработки веб-приложения в рамках курсового проекта понадобилась база данных с необходимой конфигурацией сущностей для хранения всей необходимой информации.

Структуру базу данных, ограничения целостности, связи и поля можно увидеть в приложении A и на рисунке 2.1.

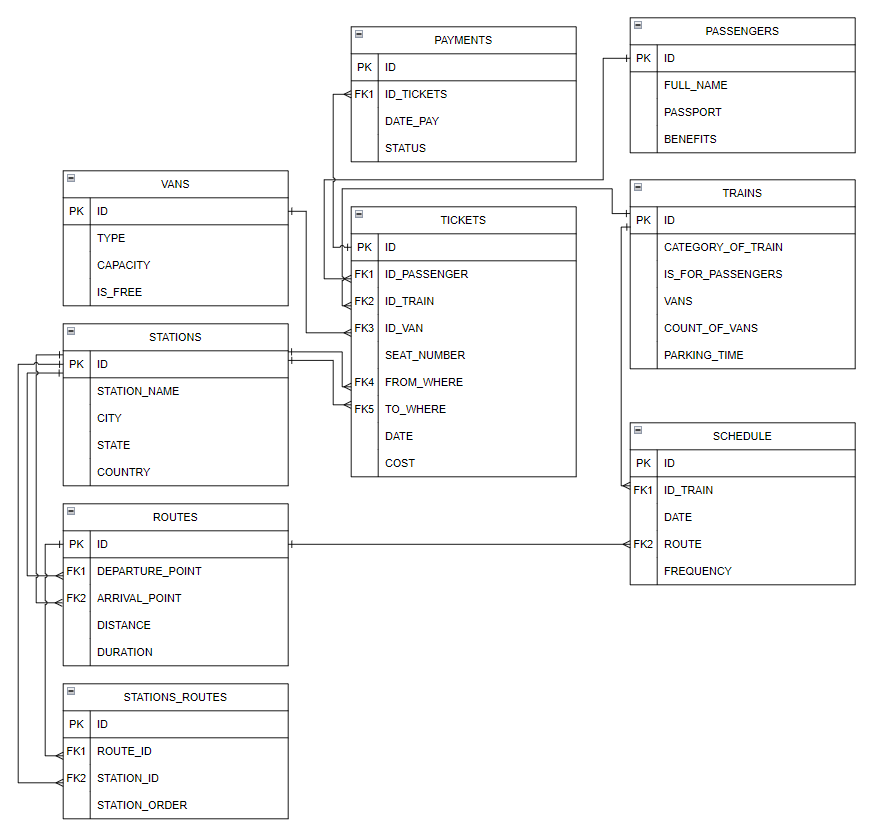


Рисунок 2.1 – Структура базы данных

В базе данных существует 9 таблиц, каждая из которых выполняет свою функцию.

2.2.1. Таблица Stations

Данная таблица используется для хранения информации о станциях остановки составов.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой точки остановки.
* Cтолбец STATION\_NAME. Хранит название станции.
* Cтолбец CITY. Хранит название города, в котором расположена станция.
* Cтолбец STATE. Хранит название региона, в котором расположена станция.
* Cтолбец COUNTRY. Хранит название страны, в котором расположена станция.

Данная таблица содержит ограничение первичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение.

2.2.2. Таблица Routes

Данная таблица используется для хранения информации о маршрутах передвижения составов.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого маршрута.
* Cтолбец DEPARTURE\_POINT. Хранит ID начальной точки.
* Cтолбец ARRIVAL\_POINT. Хранит ID конечной точки.
* Cтолбец DISTANCE. Хранит расстояние, проходящее составом.
* Cтолбец DURATION. Хранит время, проводимое в пути.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также для столбцов DEPARTURE\_POINT и ARRIVAL\_POINT существуют ограничения целостности по внешнему ключу к таблице STATIONS.

2.2.3. Таблица Schedule

Данная таблица используется для хранения информации о расписании передвижения составов.

В её состав входят следующие столбцы:

* Cтолбец ID. Хранит уникальный идентификатор каждого маршрута.
* Cтолбец ID\_TRAIN. Хранит ID поезда, зарезервированного для маршрута.
* Cтолбец DATE. Хранит дату начала выдвижения состава по маршруту.
* Cтолбец ROUTE. Хранит ID маршрута передвижения состава.
* Cтолбец FREQUENCY. Хранит частоту использования данной ячейки расписания.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также для столбцов ID\_TRAIN и ROUTE существуют ограничения целостности по внешнему ключу к таблице TRAINS и ROUTES соответсвенно.

Также таблица включает ограничение check на столбец FREQUENCY.

2.2.4. Таблица Trains

Данная таблица используется для хранения информации о поездах(составах).

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого состава.
* Cтолбец CATEGORY\_OF\_TRAIN. Хранит категорию данного состава.
* Cтолбец IS\_FOR\_PASSENGERS. Хранит число, обозначающее возможность использования данного состава для перевозки пассажиров.
* Cтолбец VANS. Хранит массив ID вагонов, из которых состоит поезд.
* Cтолбец COUNTS\_OF\_VANS. Хранит количество вагонов в составе.
* Cтолбец PARKING\_TIME. Хранит время стоянки состава.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также таблица включает ограничение check на столбец IS\_FOR\_PASSENGERS.

2.2.5. Таблица Vans

Данная таблица используется для хранения информации о вагонах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого вагона.
* Cтолбец TYPE. Хранит тип вагона.
* Cтолбец CAPACITY. Хранит вместимость вагона.
* Столбец IS\_FREE. Хранит число, указывающее, прикреплен ли вагон к составу.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также таблица включает ограничение check на столбец IS\_FREE.

2.2.6. Таблица Passengers

Данная таблица используется для хранения информации о пассажирах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого пассажира.
* Cтолбец FULLNAME. Хранит ФИО пассажира.
* Cтолбец PASSPORT. Хранит паспортные данные пассажира.
* Cтолбец BENEFITS. Хранит льготы, применяемые к пассажиру.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также таблица включает ограничение check на столбец BENEFITS.

2.2.7. Таблица Payments

Данная таблица используется для хранения информации о платежах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого платежа.
* Cтолбец ID\_TICKET. Хранит ID билета, по которому произошла оплата.
* Cтолбец DATE\_PAY. Хранит дату оплаты.
* Cтолбец STATUS. Хранит статус оплаты.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также для столбцов ID\_TICKET существует ограничение целостности по внешнему ключу к таблице TICKETS.

Также таблица включает ограничение check на столбец STATUS.

2.2.8. Таблица Tickets

Данная таблица используется для хранения информации о билетах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого билета.
* Cтолбец ID\_PASSENGER. Хранит ID пассажира, которому был выдан билет.
* Cтолбец ID\_TRAIN. Хранит ID поезда, на который забронирован билет.
* Cтолбец ID\_VAN. Хранит ID вагона, в котором находится арендованное место.
* Cтолбец SEAT\_NUMBER. Хранит номер арендованного места.
* Cтолбец FROM\_WHERE. Хранит место посадки на поезд.
* Cтолбец TO\_WHERE. Хранит место выхода из поезда.
* Cтолбец DATE. Хранит дату поездки.
* Cтолбец COST. Хранит стоимость билета.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также для столбцов ID\_PASSENGER, ID\_TRAIN и ID\_VAN существуют ограничения целостности по внешнему ключу к таблице PASSENGERS, TRAINS и VANS соответсвенно.

2.2.9. Таблица Stations\_routes

Данная таблица используется для хранения информации о порядке следования станций в маршрутах.

В её состав входят следующие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой станции в маршруте.
* Cтолбец ROUTE\_ID. Хранит ID маршрута, в котором привязана станция.
* Cтолбец STATION\_ID. Хранит ID станции, которая используется в маршруте.
* Cтолбец STATION\_ORDER. Хранит номер станции в маршруте.

Данная таблица содержит ограничение первичного ключа для столбца ID.

Также для столбцов ROUTE\_ID, STATION\_ID существуют ограничения целостности по внешнему ключу к таблице ROUTES, STATIONS соответственно.

2.3. Диаграммы UML, взаимосвязь всех компонентов

Диаграммы UML могут помочь визуализировать взаимосвязь всех компонентов в приложении, а также описать функциональность системы и ее взаимодействие с внешними пользователями или другими системами.

Диаграмма вариантов использования помогает команде разработки программного продукта понять, какие функциональности должны быть реализованы и как пользователи будут взаимодействовать с системой. Она также может использоваться в качестве средства коммуникации между разработчиками, заказчиками и другими заинтересованными сторонами для обсуждения требований и функциональности системы.

Преимущества использования диаграммы вариантов использования включают:

Визуализацию функциональности системы: Диаграмма вариантов использования помогает визуализировать все возможные сценарии использования системы, что помогает лучше понять ее функциональность и взаимодействие с пользователями.

Определение требований: Диаграмма вариантов использования помогает идентифицировать и описать функциональные требования к системе на основе сценариев использования.

Улучшение коммуникации: Диаграмма вариантов использования служит средством коммуникации между разработчиками, заказчиками и другими заинтересованными сторонами, позволяя им лучше понять ожидания от данного приложения.

Диаграмма вариантов использования была создана для описания функциональности программного средства «Железнодорожный вокзал» и ее взаимодействия с внешними пользователями. На диаграмме показаны актеры, варианты использования и связи между ними.

Актерами на диаграмме являются пользователи и менеджеры. Варианты использования включают бронирование билетов, просмотр расписания поездов, изменение информации о поездах и т.д.

Сама диаграмма представлена на рисунке 2.1, а также в приложении Б.

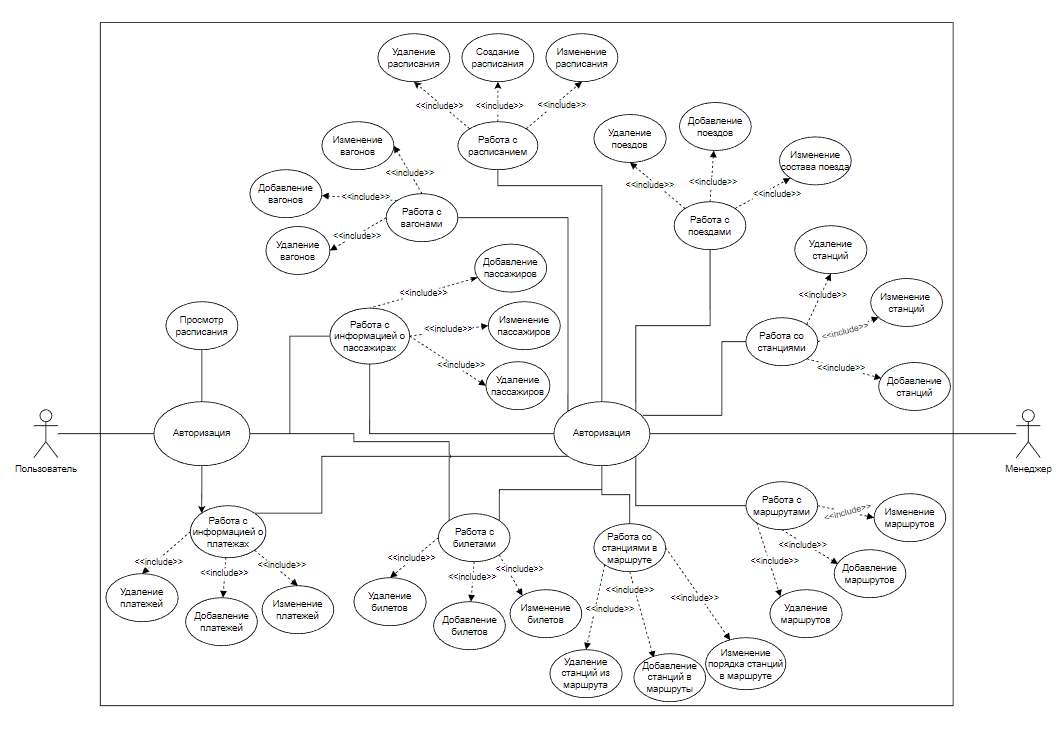


Рисунок 2.2 – Диаграмма вариантов использования

Диаграмма вариантов использования помогает лучше понять функциональность программного средства «Железнодорожный вокзал» и ее взаимодействие с внешними пользователями, что может быть полезным при проектировании и разработке системы.

2.4. Обобщенная архитектура программного средства

Для курсового проекта был использован способ разработки программных модулей, основанный на архитектуре клиентского приложения. Приложение состоит из двух частей:

* Клиентская часть. В этом модуле осуществляется взаимодействие пользователя с программным средством с помощью графического интерфейса. В клиентской части пользователь сможет получить полную информацию об объектах приложения, в зависимости от присвоенной ему роли (пользователь или менеджер).
* База данных. В базе данных описываемого приложения хранится необходимая для работы с клиентской частью приложения информация.

Требования, описанные выше, должны учитывать выбранную архитектуру приложения.

2.5. Вывод

В данном разделе были описаны используемые технические средства, разработана архитектура проекта, описано взаимодействие приложения с базой данных, составлена диаграмма вариантов использования. Также была описана структура базы данных вместе с ограничениями целостности.

3. Проектирование программного средства

**3.1. Общая структура**

Программное средство включает в себя 2 окна и 8 страниц и имеет следующую структуру, представленную на рисунке 3.1.

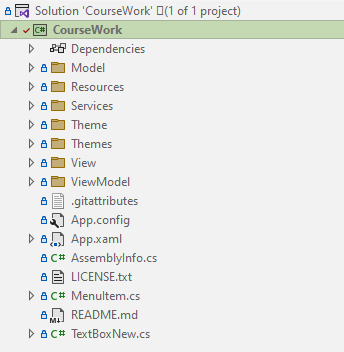


Рисунок 3.1 – Структура проекта

Описание структуры основных папок и файлов проекта представлено в таблице 3.1.

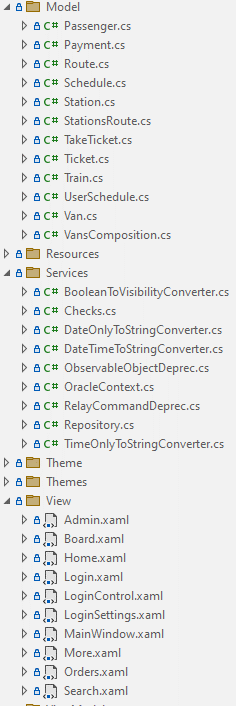
Таблица 3.1 – Описание структуры папок и файлов проекта

|  |  |
| --- | --- |
| Имя файла | Содержание |
| App.config | Файл с параметрами проекта. |
| Папка Services | Папка, содержащая вспомогательные классы проекта. |
| Папка Resources | Папка, содержащая все изображения. |
| Папка Model | Папка, содержащая все классы-модели проекта. |
| Папка Theme | Папка, содержащая, настройки стилей для встроенных элементов управления. |
| Папка Themes | Папка, содержащая, настройки стилей для пользовательских элементов управления. |
| Папка View | Папка, содержащая все имеющиеся в приложении графические интерфейсы пользователя. |
| Папка ViewModel | Папака, содержащая классы ViewModel для каждого View. |

Продолжение таблицы 3.1.

|  |  |
| --- | --- |
| App.xaml | Стартовое окно приложения. |
| AssemblyInfo.cs | Данные для сборки приложения. |
| LICENSE.txt | Файл лицензии для программного средства. |
| MenuItem.cs | Пользовательский элемент управления MenuItem. |
| TextBoxNew.cs | Пользовательский элемент управления TextBoxNew. |

Более подробная структура содержимого папок программного средства описана на рисунке 3.2 и в таблице 3.2.



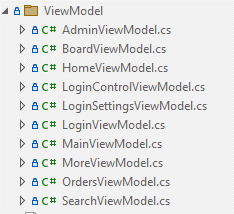


Рисунок 3.2 – Подробная структура проекта

Таблица 3.2 – Подробное описание структуры файлов проекта

|  |  |
| --- | --- |
| Имя файла | Содержание |
| Passenger.cs | Класс-модель для таблицы базы данных PASSENGERS. |
| Payment.cs | Класс-модель для таблицы базы данных PAYMENTS. |
| Route.cs | Класс-модель для таблицы базы данных ROUTES. |
| Schedule.cs | Класс-модель для таблицы базы данных SCHEDULE. |
| Station.cs | Класс-модель для таблицы базы данных STATIONS. |
| StationsRoute.cs | Класс-модель для таблицы базы данных STATIONS\_ROUTES. |
| TakeTicket.cs | Класс-модель для представления TAKE\_TICKET. |
| Ticket.cs | Класс-модель для таблицы базы данных TICKET. |
| Train.cs | Класс-модель для таблицы базы данных TRAINS. |
| UserSchedule.cs | Класс-модель для представления USER\_SCHEDULE. |
| Van.cs | Класс-модель для таблицы базы данных VAN. |
| VansComposition.cs | Класс-модель для объекта базы данных VANS\_COMPOSITION. |
| BooleanToVisibilityConverter.cs | Вспомогательный класс, для преобразования булевого типа данных в Visibility. |
| Checks.cs | Вспомогательный класс, содержащий все проверки и валидации для вводимых данных. |
| DateOnlyToStringConverter.cs | Вспомогательный класс, для преобразования типа данных DateOnly в тип данных String и наоборот. |
| DateTimeToStringConverter.cs | Вспомогательный класс, для преобразования типа данных DateTime в тип данных String и наоборот. |
| OracleContext.cs | Вспомогательный класс, для установления соединения с базой данных. |
| Repository.cs | Вспомогательный класс, для работы с базой данных. |
| TimeOnlyToStringConverter.cs | Вспомогательный класс, для преобразования типа данных TimeOnly в тип данных String и наоборот. |
| Admin.xaml | Страница с панелью администратора. |
| Board.xaml | Страница с доской расписания. |
| Home.xaml | Домашняя страница. |

Продолжение таблицы 3.2

|  |  |
| --- | --- |
| Login.xaml | Окно входа в приложение. |
| LoginControl.xaml | Страница входа в приложение. |
| LoginSettings.xaml | Страница настроек входа в приложение. |
| MainWindow.xaml | Главное окно приложения. |
| More.xaml | Страница «More». |
| Orders.xaml | Страница заказов. |
| Search.xaml | Страница поиска |
| AdminViewModel.cs | Класс ViewModel для страницы Admin.xaml |
| BoardViewModel.cs | Класс ViewModel для страницы Board.xaml |
| HomeViewModel.cs | Класс ViewModel для страницы Home.xaml |
| LoginControlViewModel.cs | Класс ViewModel для страницы LoginControl.xaml |
| LoginSettingsViewModel.cs | Класс ViewModel для страницы LoginSettings.xaml |
| LoginViewModel.cs | Класс ViewModel для страницы Login.xaml |
| MainViewModel.cs | Класс ViewModel для страницы MainWindow.xaml |
| MoreViewModel.cs | Класс ViewModel для страницы More.xaml |
| OrdersViewModel.cs | Класс ViewModel для страницы Orders.xaml |
| SearchViewModel.cs | Класс ViewModel для страницы Search.xaml |

# **3.2. Взаимоотношения между классами**

Для визуализации взаимосвязей между классами используется диаграмма UML – графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение спроектировано таким образом, что каждый класс выполняет свою функцию, все классы вместе образуют одно целое програмное средство. Диаграмма классов представлена в Приложении В.

**3.3. Проектирование архитектуры приложения**

Архитектура программного обеспечения представляет собой совокупность ключевых решений, касающихся организации программной системы. Она включает выбор структурных элементов и их интерфейсов, определение их взаимодействия, а также объединение этих элементов в более крупные системы. Однако, центральную роль играет архитектурный стиль, который определяет всю организацию системы, включая элементы, их интерфейсы, их сотрудничество и способы их соединения.

Для удовлетворения требований проектируемой системы с точки зрения различных атрибутов качества, применяются различные архитектурные шаблоны, так называемые паттерны. В разрабатываемом приложении был использован архитектурный шаблон Model-View-ViewModel (MVVM). Этот шаблон состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

На рисунке 3.8 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.



Рисунок 3.8 – Структура шаблона MVVM

Модель описывает данные, используемые в приложении. В моделях может содержаться логика, связанная с этими данными, например, валидация свойств модели. В то же время модель не должна содержать логику, связанную с отображением данных и взаимодействием с визуальными элементами управления. Часто модель реализует интерфейс INotifyPropertyChanged, который позволяет системе автоматически обнаруживать изменения свойств модели и облегчает привязку к представлению, хотя сама модель не взаимодействует напрямую с представлением.

Представление, или View, определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Важным аспектом является эффективное проектирование пользовательского интерфейса, чтобы создать приятный и удобный опыт для конечного пользователя. Представление не обрабатывает события, а в основном выполняет действия посредством команд.

ViewModel, или модель представления, связывает модель и представление через механизм привязки данных. Если значения свойств модели изменяются, с использованием интерфейса INotifyPropertyChanged, данные в представлении автоматически обновляются, хотя сама модель и представление не прямо связаны. ViewModel также содержит логику получения данных из модели и обновления данных в модели.

Кроме архитектурного паттерна, использовались паттерны для работы с базой данных: это Repository и Unit of Work, паттерн Command для работы с командами, паттерн Singleton для поддержки соединения с базой данных.

Repository и Unit of Work являются распространенными паттернами проектирования, применяемыми при разработке приложений с использованием базы данных.

Паттерн Repository (Репозиторий) предоставляет абстракцию для работы с данными. Он представляет собой слой абстракции между приложением и базой данных, скрывая детали взаимодействия с хранилищем данных.

Паттерн Unit of Work (Рабочая единица) обеспечивает контроль над транзакционной целостностью и согласованностью операций с базой данных. Он объединяет несколько операций чтения и записи в единую транзакцию, позволяя гарантировать, что все операции будут успешно выполнены или отменены целиком.

# **3.4. Проектирование последовательностей проекта**

В целях визуализации взаимодействия объектов системы между собой во времени в едином сценарии использования используется ещё одна UML-диаграмма – диаграмма последовательностей. Данная диаграмма иллюстрирует, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования.

Для отображения течения времени используется линия жизни объекта, которая изображается с помощью штриховой линии, которая проводится вертикально вниз. С помощью линии жизни показывается период, в течение которого объект существует в системе. Сами объекты изображаются в виде прямоугольников, а сообщения, которыми они обмениваются – в виде линий со стрелками.

Диаграмма последовательностей представлена в приложении Г.

**3.5. Вывод**

В данном разделе была рассмотрена архитектура проекта. Были описаны виды разработанных объектов. Также были описаны используемые архитектурный шаблон и паттерны проектирования. Была составлены UML диаграммы классов и последовательностей.

4. Реализация программного средства

# **Основные классы программного средства**

Для выполнения технических задач программного средства «Железнодорожный вокзал» должны быть реализованы следующие функции и соответствующие им классы и методы:

* Выполнение входа;
* Изменение параметров входа;
* Просмотр расписания;
* Просмотр и оформление заказов;
* Заполнение формы для поиска рейсов;
* Функционирование панели администратора.

Далее подробно рассмотрены каждые из необходимых для выполнения технических задач функции, а также созданные для их выполнения классы и методы, и их функционал, и реализация.

# **Описание классов и методов программного средства**

## **Выполнение входа**

Необходимый функционал для возможности входа пользователя в аккаунт, а также для проверки корректности данных, реализован в классе LoginControlViewModel.

Основным методом класса является метод SubmitButton\_Click(), выполняющийся при нажатия пользователя на кнопку «Войти». В данном методе вызывается метод OracleContext.Create(), пытающийся создать соединение с базой данных по введенным данным.

При успешном прохождении всех проверок пользователь перенаправляется на главную страницу, а противном случае – получает сообщение о соответствующей ошибке.

Для перехода к странице имзенения параметров входа, со страницы входа, реализована команда LoginSettingsCommand, выполняющаяся при нажатии пользователя на кнопку «Изменить настройки подключения к БД» на странице входа, и перенаправляющий пользователя со страницы входа на страницу изменения параметров входа.

Листинг реализации класса LoginControlViewModel представлена в приложении Д.

## **Изменение параметров входа**

Для изменения параметров входа реализован класс LoginSettingsViewModel.

При нажатии пользователя на кнопку «Изменить» вызывается основной метод класса – SubmitButton\_Click(). В начале все поля проверяются на пустоту.

После успешного прохождения проверки происходит запись данных в настройки приложения и перенаправление пользователя на страницу входа.

Листинг реализации класса LoginSettingsViewModel представлена в приложении Д.

## **Главная страница**

После успешной авторизации пользователь перенаправляется на основную страницу программного средства, представленную классом MainViewModel, состоящую из меню и основной части страницы, которая может меняться в зависимости от выбранного пункта меню.

Данный класс не содержит значимых методов, вся логика описана в конструкторе.

Сущность реализации данного класса состоит в том, что он хранит в себе экземпляры ViewModel других страниц, а также предоставляет логику для переключения между окнами. Логика реализована с помощью команд.

Листинг реализации класса MainViewModel представлена в приложении Д.

## **Страница «Home»**

Первая страница, куда попадает пользователь после входа – это страница «Home». Страница представлена классом HomeViewModel. В ней содержится форма поиска билетов. Для поиска билетов надо ввести данные и нажать на кнопку «Search», логика обработки нажатия на данную кнопку реализована с помощью команды SearchCommand. После этого произойдет переход на страницу «Search».

Листинг реализации класса HomeViewModel представлена в приложении Д.

## **Страница «Search»**

Отображение найденных, по указанным на странице «Home» данным, маршрутах происходит на странице «Search», представленной классом SearchViewModel. Главным методом этого класса является метод GetItems (), в котором происходит получение данных, по введенным условиям, из репозитория базы данных.

Данные, как и во всём приложении, отображаются в таблице, для удобного просмотра и редактирования.

Также в этом классе есть метод сортировки данных по столбцу, и команды, для осуществления пагинации по массиву данных.

Листинг реализации класса SearchViewModel представлена в приложении Д.

## **Страница «Board»**

Из главной страницы также можно попасть на страницу «Board», на которой содержится расписание всех рейсов. Страница представлена классом BoardViewModel.

Главным методом этого класса является метод GetItems (), в котором происходит получение данных из репозитория базы данных.

Также в этом классе, как и в классе других классах, где осуществяется работа с данными, есть метод сортировки данных по столбцу, и команды, для осуществления пагинации по массиву данных.

Листинг реализации класса BoardViewModel представлена в приложении Д.

## **Страница «Orders»**

Страница «Orders» является страницей, где можно оформить билет на рейс. Страница представлена классом OrdersViewModel.

Это первая страница, где любой сможет добавлять и удалять строки заказов.

Главным методом GetItems (), но, кроме него, важными методами являются методы: AddRows () и DeleteRows (). В этих методах содержится логика для добавления и удаления строк соответственно. Вызовы методов привязаны к нажатиям на клавиши «Enter» и «Delete» соответственно. В методе AddRows () происходит валидация вводимых данных и, при несоответствии выводится ошибка, а добавление не происходит, а при вызове DeleteRows () пользователю требуется подтвердить удаление строки.

Листинг реализации класса OrdersViewModel представлена в приложении Д.

## **Страница «More»**

Страница «More» представлена классом MoreViewModel. В ней содержится информация о разработчике приложения.

Листинг реализации класса MoreViewModel представлена в приложении Д.

## **Страница «Admin»**

Все необходимые менеджеру методы инкапсулированы в классе AdminViewModel, имеющем методы для добавления, удаления, и редактирования всех табличных данных из базы данных.

Данная страница содержит валидацию данных для каждого свойства моделей данных. Валидация выполняется различными способами, среди которых использование регулярных выражений или проверка на соответсвие строки условию.

Метод Selection\_Changed () запускается при изменении выбранного элемента в списке таблиц и выполняет отображение данных из таблицы, для работы с ними.

Метод GetItems () необходим для получения данных для отображения.

Метод UpdateRows () отвечает за изменение данных в строке.

Метод PreviewKeyDown () отвечает за определение нажатой клавиши и вызов нужного метода.

Метод AddRows () отвечает за добавление новых строк данных.

Метод DeleteRows () отвечает за удаление строк из таблицы.

Листинг реализации класса AdminViewModel представлен в приложении Д.

## **Класс OracleContext**

Класс OracleContext используется для подключения к базе данных. Данный класс является реализацией порождающего паттерна – Одиночка. Главным методом этого класса является метод Create (), позволяющий создать только один экземпляр класса. Вся же логика работы со строкой подключения описана в конструкторе.

Листинг реализации класса OracleContext представлена в приложении Д.

4.3 Вывод

В данном разделе были рассмотрены основные аспекты реализации классов и методов, отвечающих за корректное отображение и работы UI.

5. Тестирование, проверка работоспособности и анализ полученных результатов.

## **Тестирование авторизации**

В момент авторизации возможна ситуация, когда пользователь вводит некорректные данные, например, неверный пароль или логин, или вовсе забыл ввести данные, в этом случае авторизация выполнена не будет. Такие исключения обрабатываются программным средством с помощью показа пользователю всплывающих сообщений с текстом ошибки. Примеры обработки исключительной ситуаций представлен на рисунке 5.1.

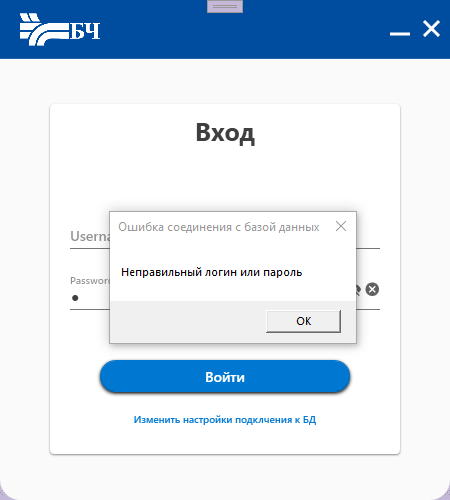


Рис 5.1 – Обработка ошибки авторизации

Также возможны исключительные ситуации при заполнении формы изменения параметров подключение к базе данных, такие как пустые поля или несоответствие поля Hostname необходимому шаблону. Пример обработки исключительной ситуации представлен на рисунке 5.2.

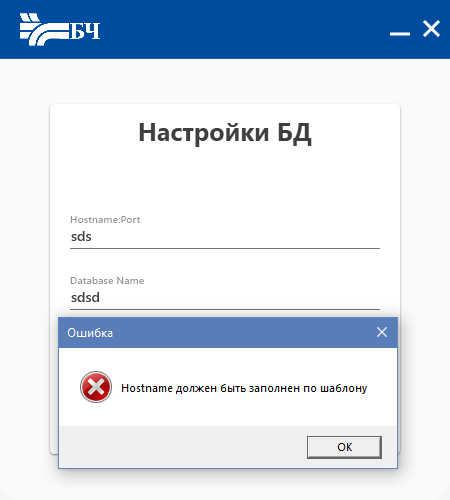


Рис 5.2 – Обработка неправильного заполнения поля Hostname

Продемострированные обработки ошибок справляются со 100% исключительных ситуаций, при работе с формами авторизации и изменения параметров подключения к базе данных.

## **Тестирование поиска**

При заполнении формы поиска возможно возникновение исключительных ситуаций. Входные данные в поле даты должны позже или равные текущей дате. При возникновении таких исключений пользователь уведомляется всплывающим окном. Пример обработки исключительной ситуаций, при вводе даты, представлен на рисунке 5.3.

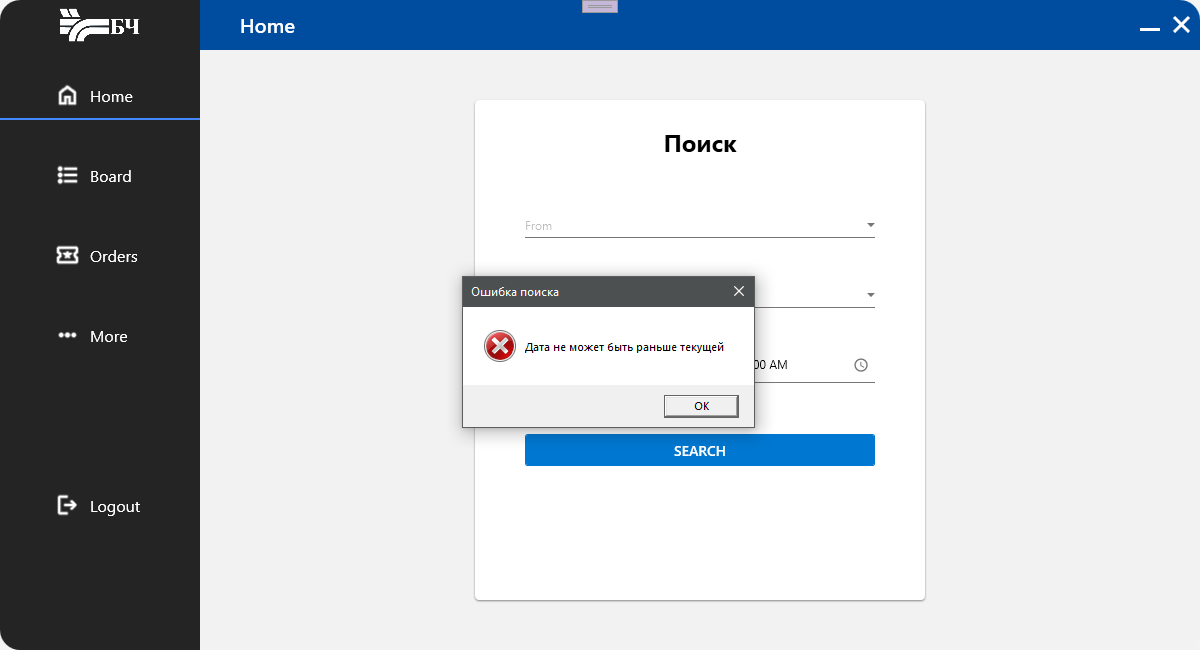


Рис 5.3 – Обработка даты в форме поиска

Обработка же других исключительных ситуаций изображена на рисунке 5.4.

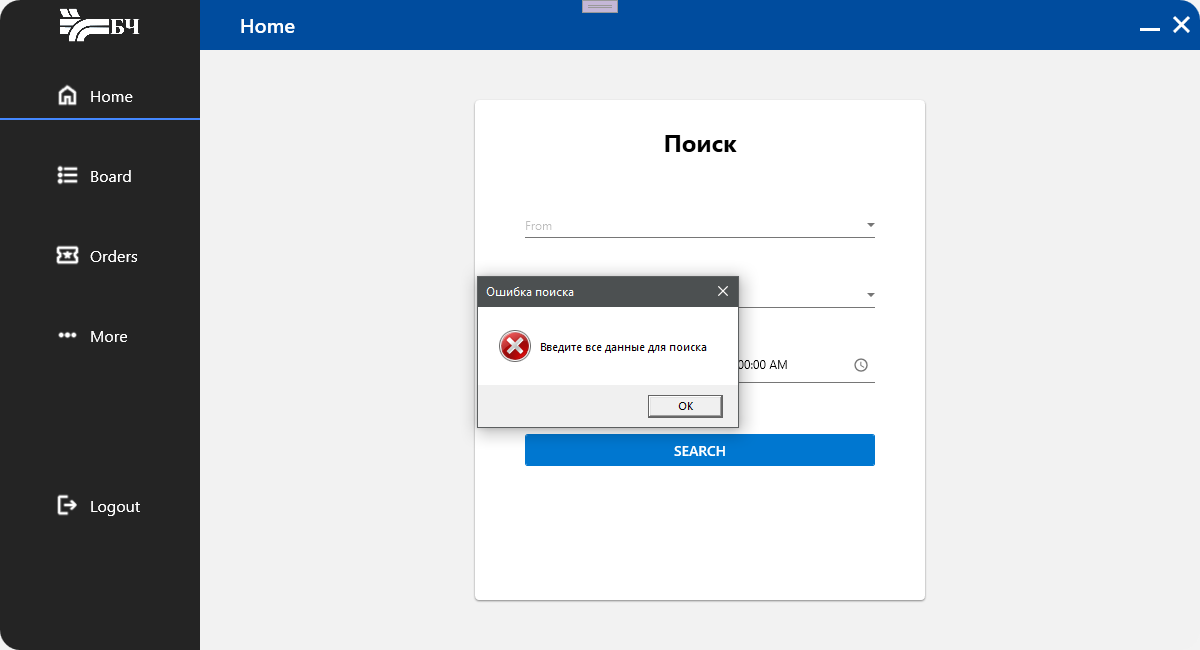


Рис 5.4 – Обработка остальных исключительных ситуаций

В ходе тестирования поиска были предствалены базовые ошибки пользователей при работе с поиском.

## **Тестирование заказов билетов**

Во время заполнения пользователем данных о заказе возможно возникновение исключительных ситуаций, таких как ввод неправильных значений, ввод значения, выходящего за границы установленных значений, или попытка отправить данные, до их полного заполнения. Данные исключения успешно отлавливаются и уведомляют пользователя о возникновении исключительной ситуации. Примеры обработки исключений представлены на рисунках 5.5 – 5.15.

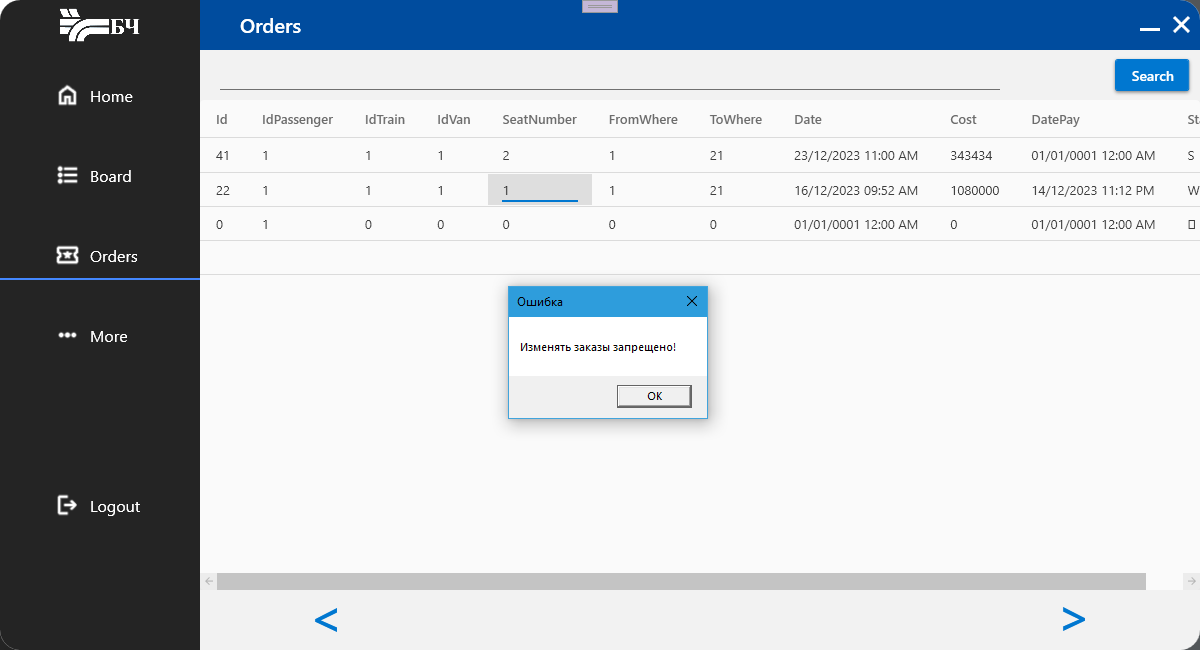


Рис 5.5 – Обработка попытки изменить существующий заказ

Обработка ввода неправильного IdPassenger изображена на рисунке 5.6.

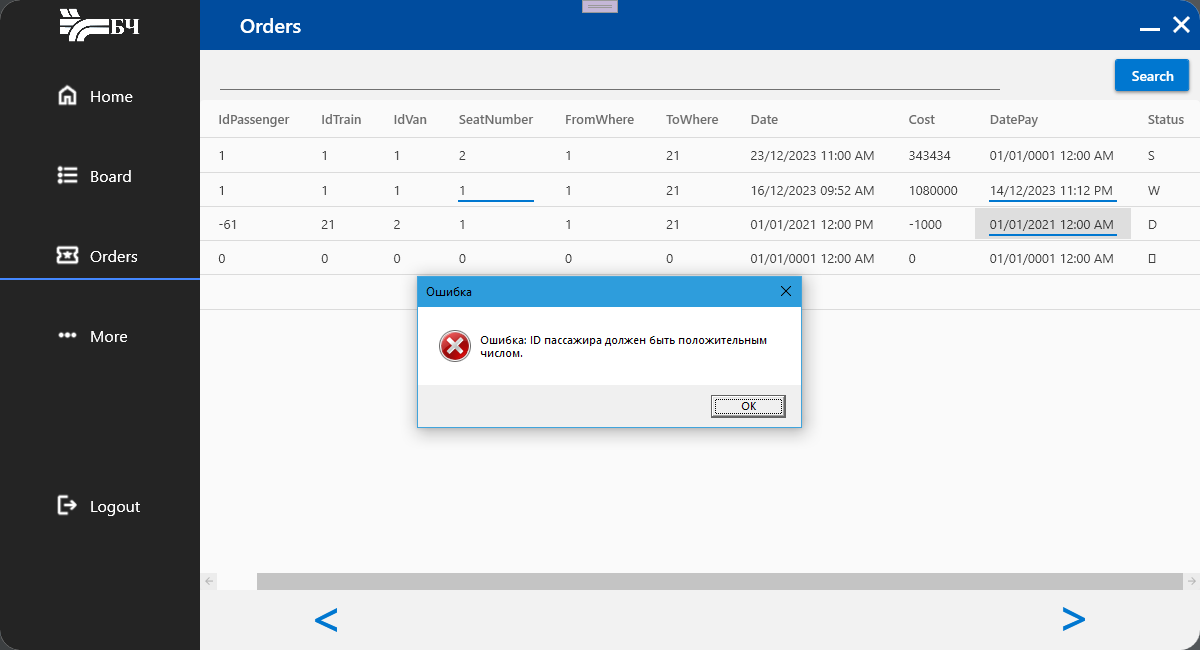


Рис 5.6 – Обработка ввода IdPassenger

Также, обработка IdTrain представлена на рисунке 5.7.

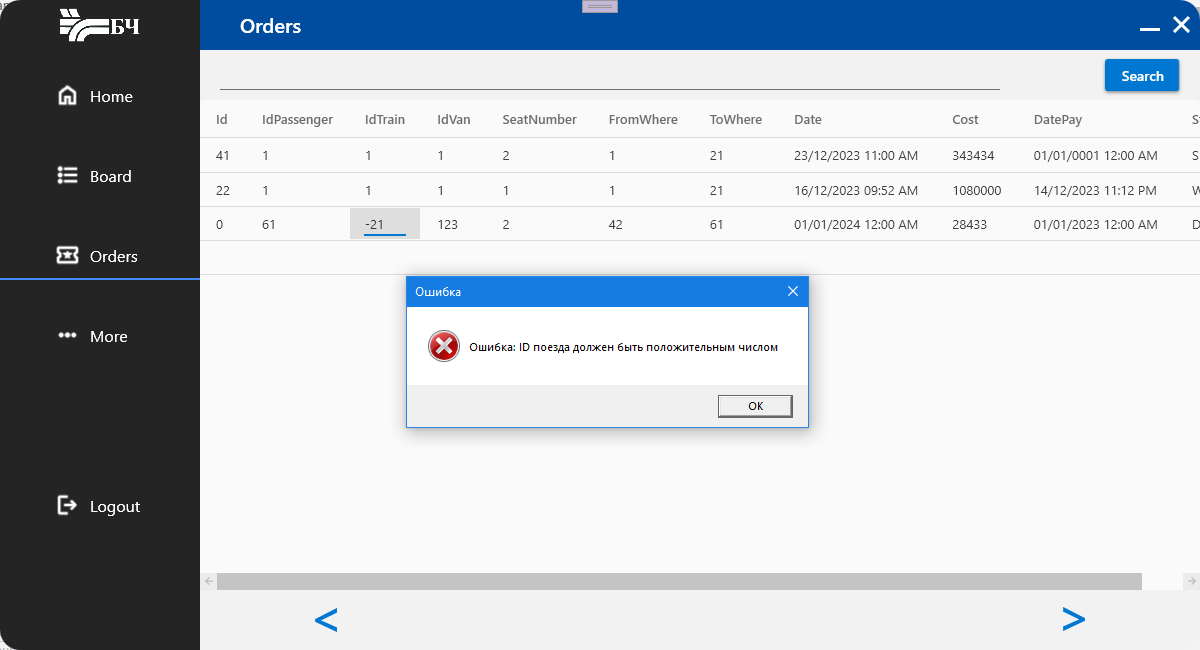


Рис 5.7 – Обработка ввода IdTrain

После обработки IdTrain идет валидация IdVan, изображенная на рисунке 5.8.

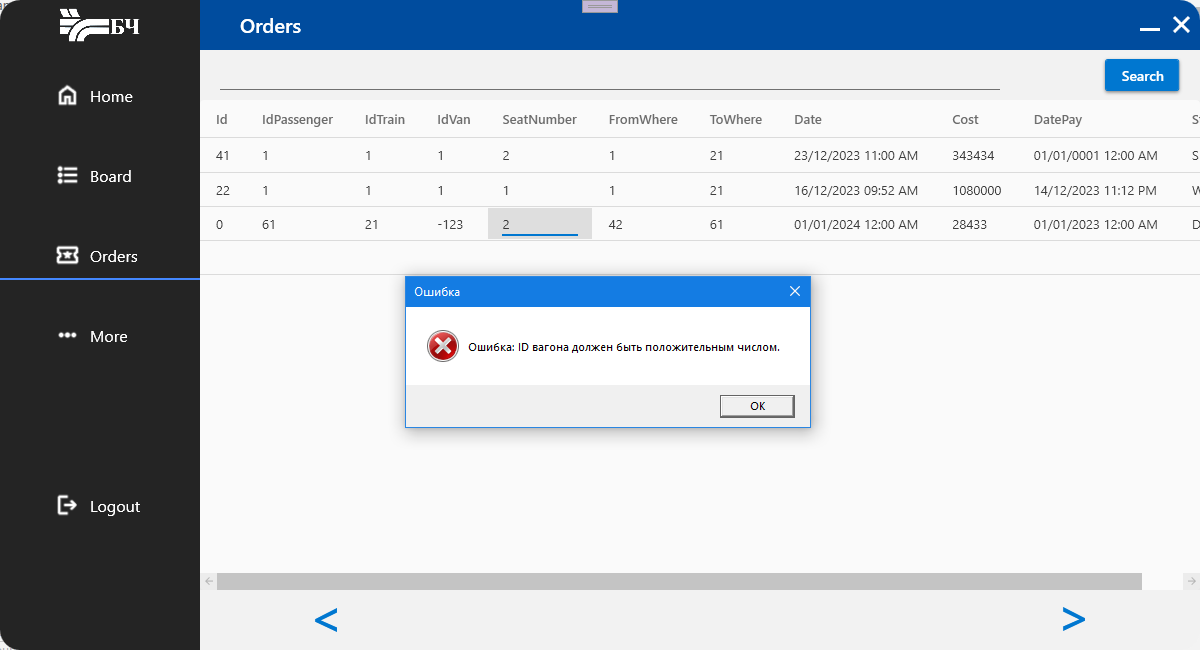


Рис 5.8 – Обработка ввода IdVan

Следующим полем для обработки является поле SeatNumber. Его обработка представлена на рисунке 5.9.

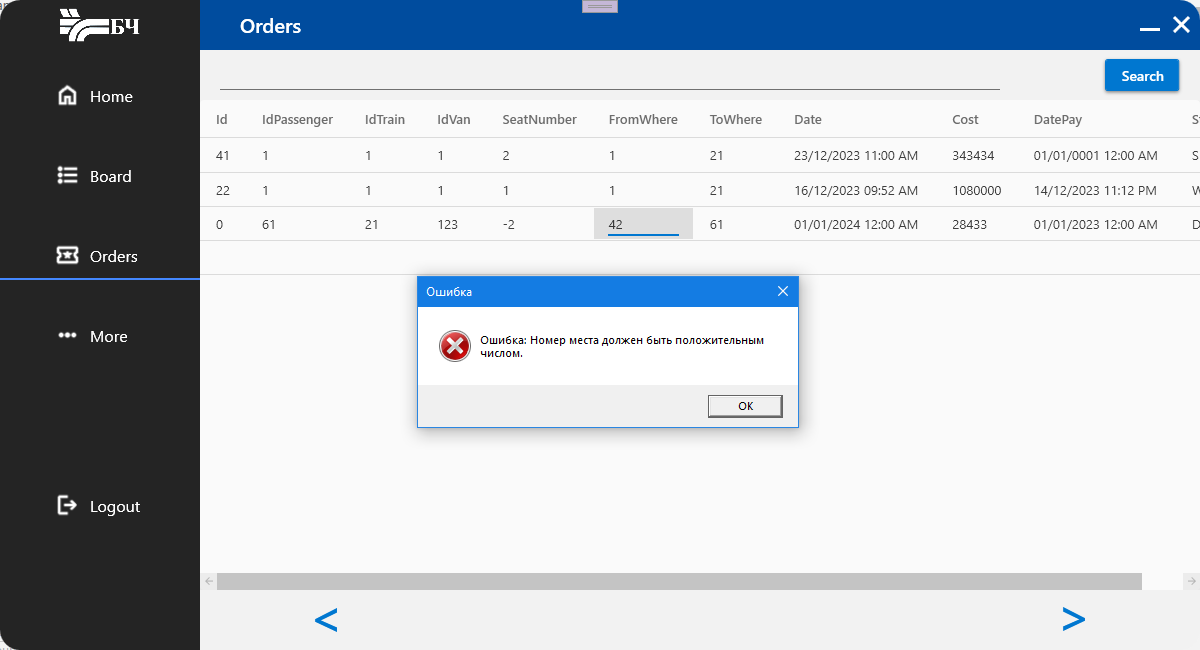


Рис 5.9 – Обработка SeatNumber

После обработки SeatNumber обрабатывается FromWhere, его обработка проилюстрирована на рисунке 5.10.

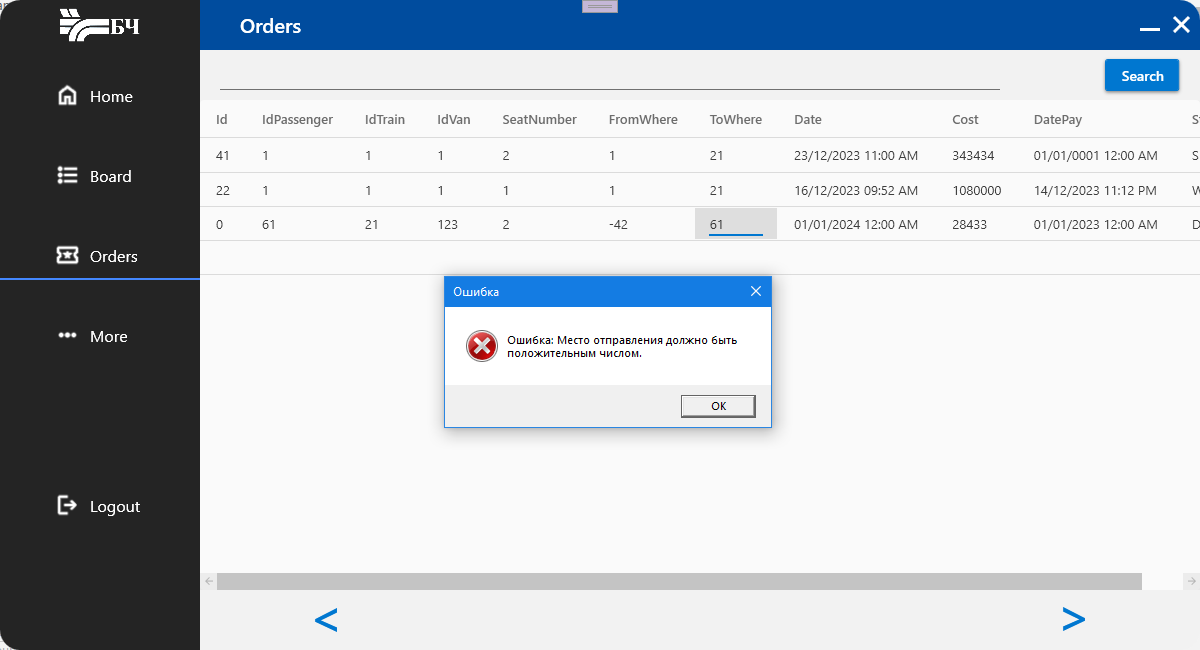


Рис 5.10 – Обработка ввода FromWhere

Далее требуется проверить введенные данные из поля ToWhere, его проверка продемонстрирована на рисунке 5.11.

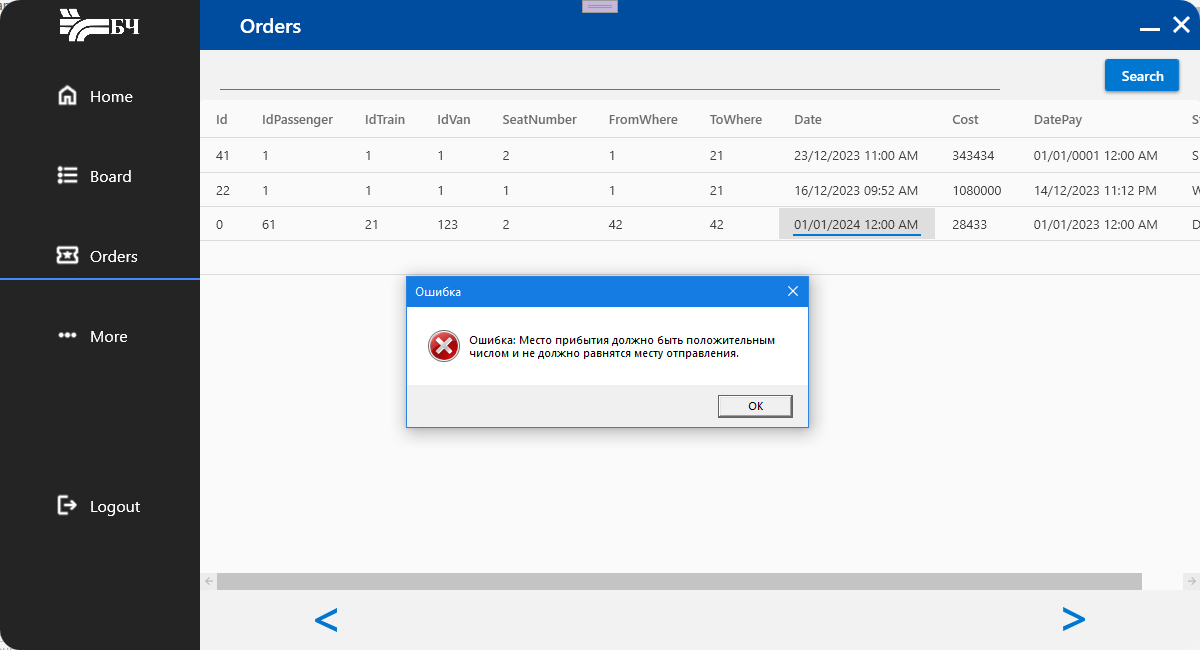


Рис 5.11 – Обработка ввода ToWhere

Далее надо проверять дату отправления, чтобы нельзя было забронировать билет на прошедший рейс, обработка представлена на рисунке 5.12.

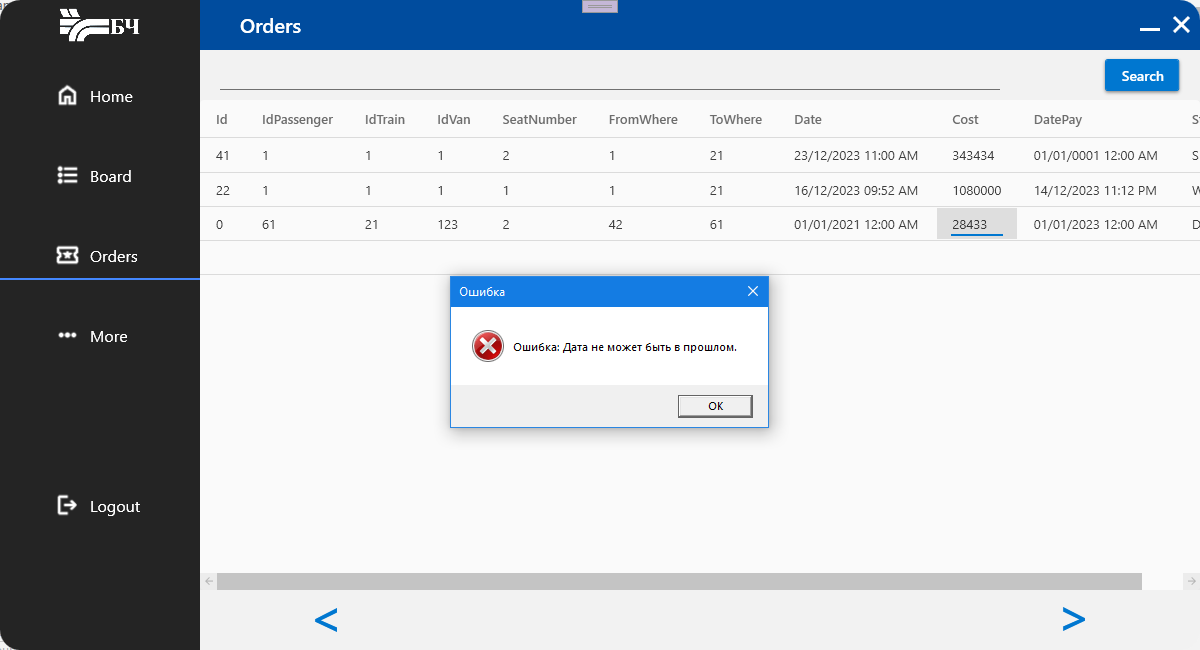


Рис 5.12 – Обработка ввода Date

Следующим полем надо проверять стоимость, на правильный диапазон значений, обработка продемонстрирована на рисунке 5.13.

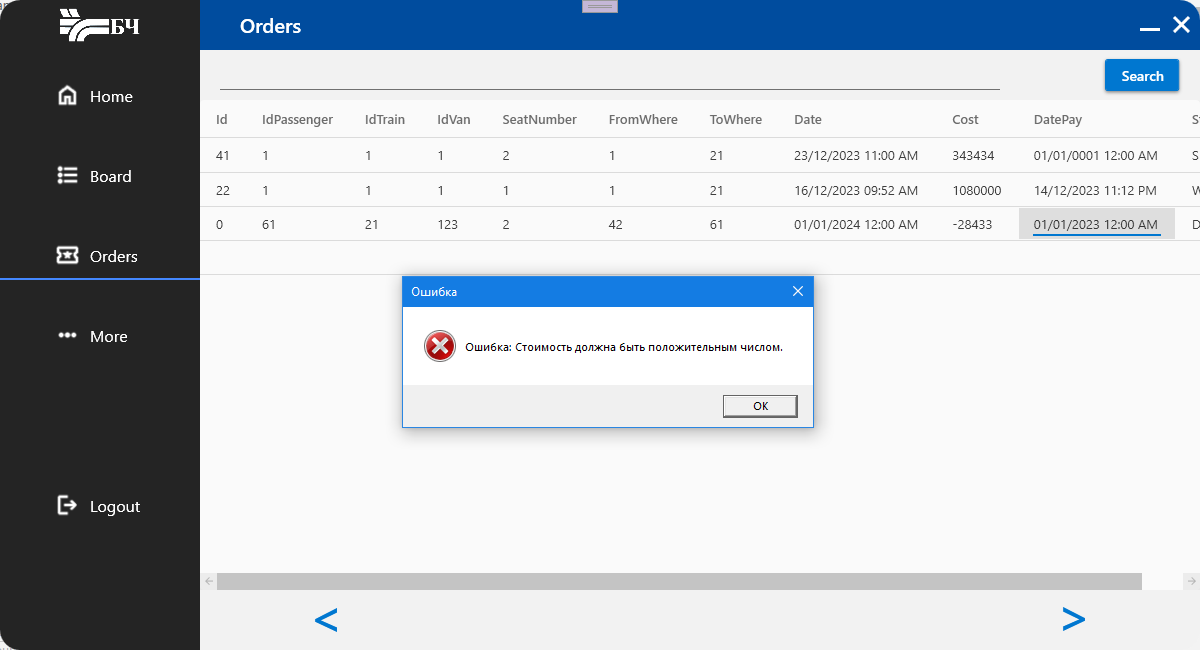


Рис 5.13 – Обработка ввода Cost

После проверки стоимости надо проверить дату оплаты, она не может быть позже текущей даты, обработка проилюстрирована на рисунке 5.14.

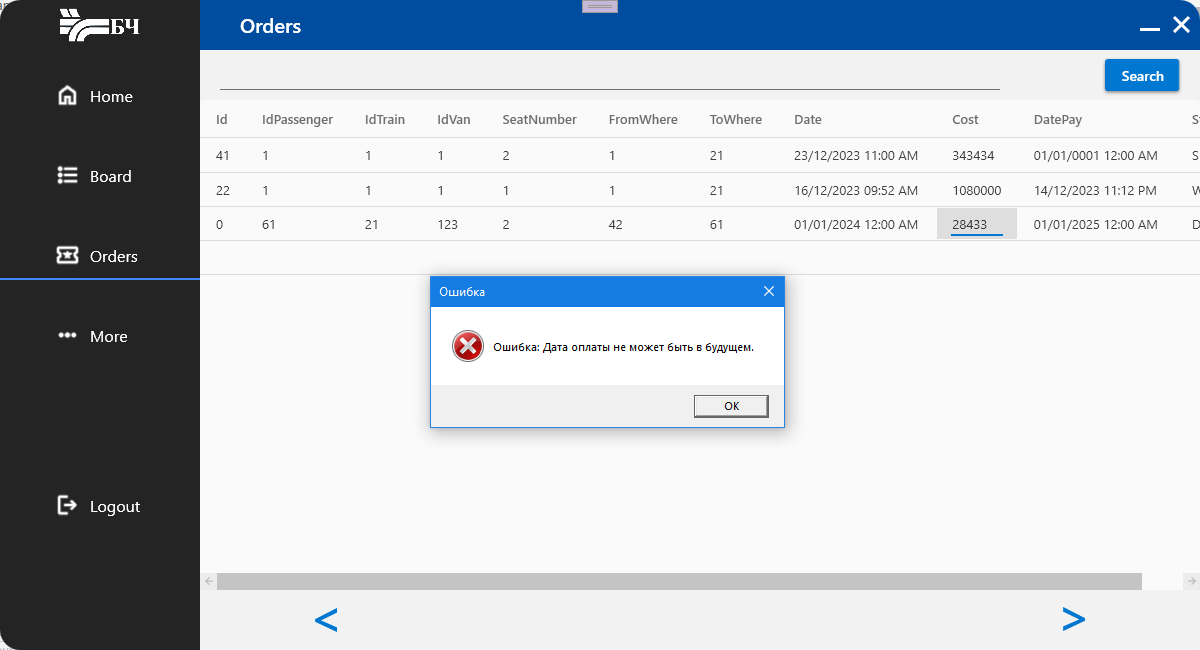


Рис 5.14 – Обработка ввода DatePay

Последним полем для проверки является поле Status, его проверка представлена на рисунке 5.15.

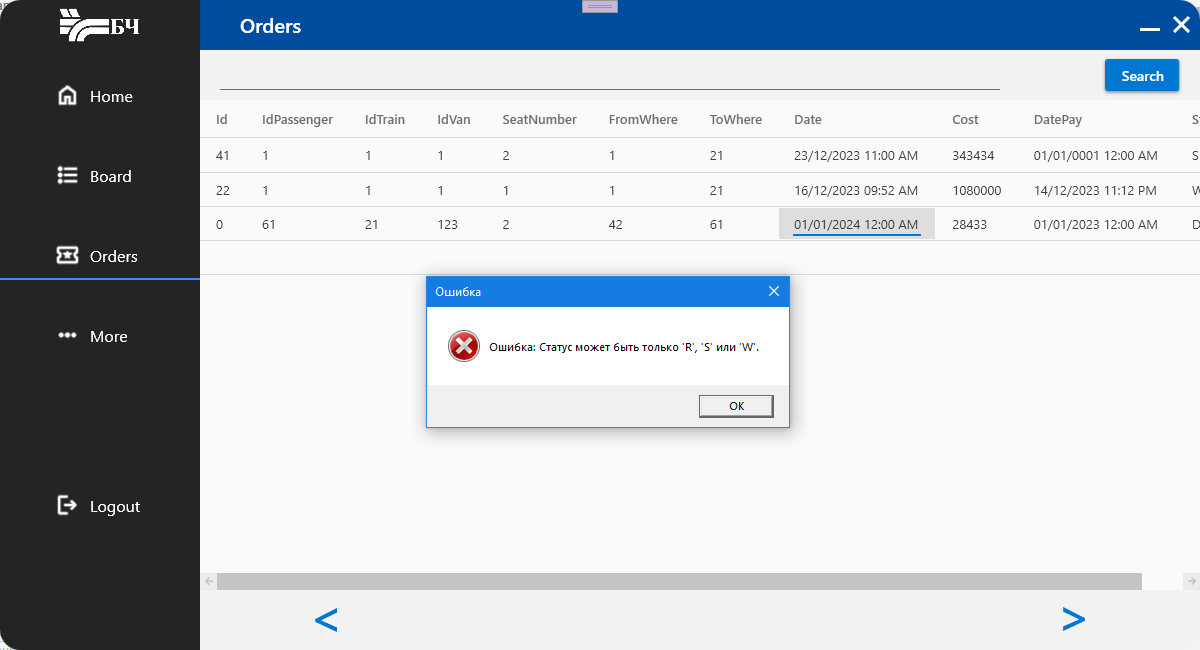


Рис 5.15 – Обработка ввода Status

Такие же проверки сделаны для всех полей в таблицах, на странице «Admin», поэтому указываться, далее в этом разделе, не будут.

## **Тестирование панели администратора**

В панели администратора возможно генерирование многих исключительных ситуаций, таких как: попытка добавления или редактирования строк при наличии незаполненных полей, ввод нецелочисленных либо выходящих за допустимые границы значений, ввод некорректного формата даты и времени. Все вышеописанные исключительные ситуации генерируют всплывающее окно с текстом ошибки, представленные на рисунках 5.16 – 5.20.

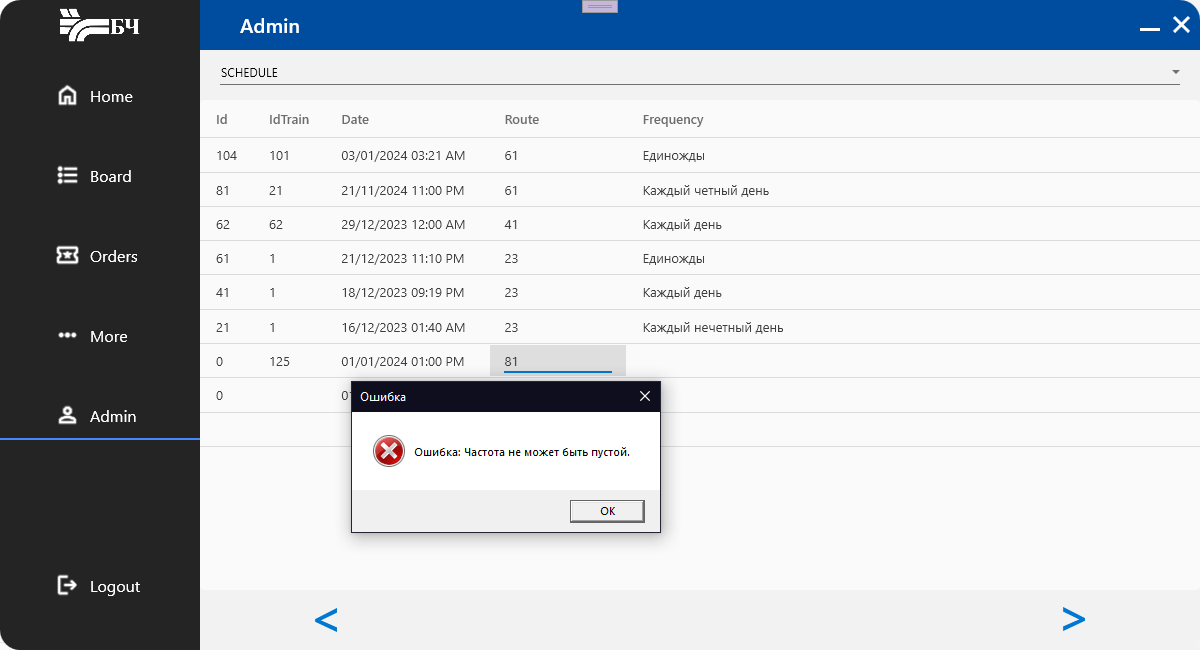


Рис 5.16 – Обработка попытки добавления строки при наличии незаполненных полей

Также, в обработке нуждается проверка на наличие незаполненных полей при редактировании, она показана на рисунке 5.17.

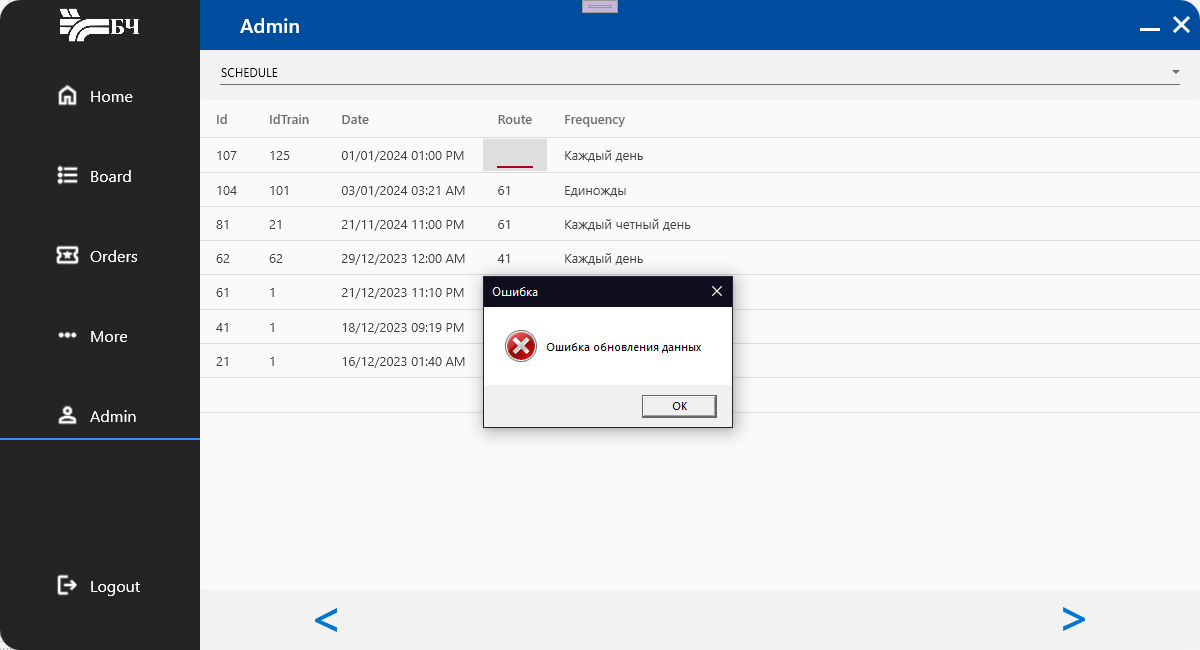


Рис 5.17 – Обработка попытки редактирования строки при наличии незаполненных полей

Кроме продемонстрированного, требуется обрабатывать ввод нецелочисленных значений, обработка представлена на рисунке 5.18.

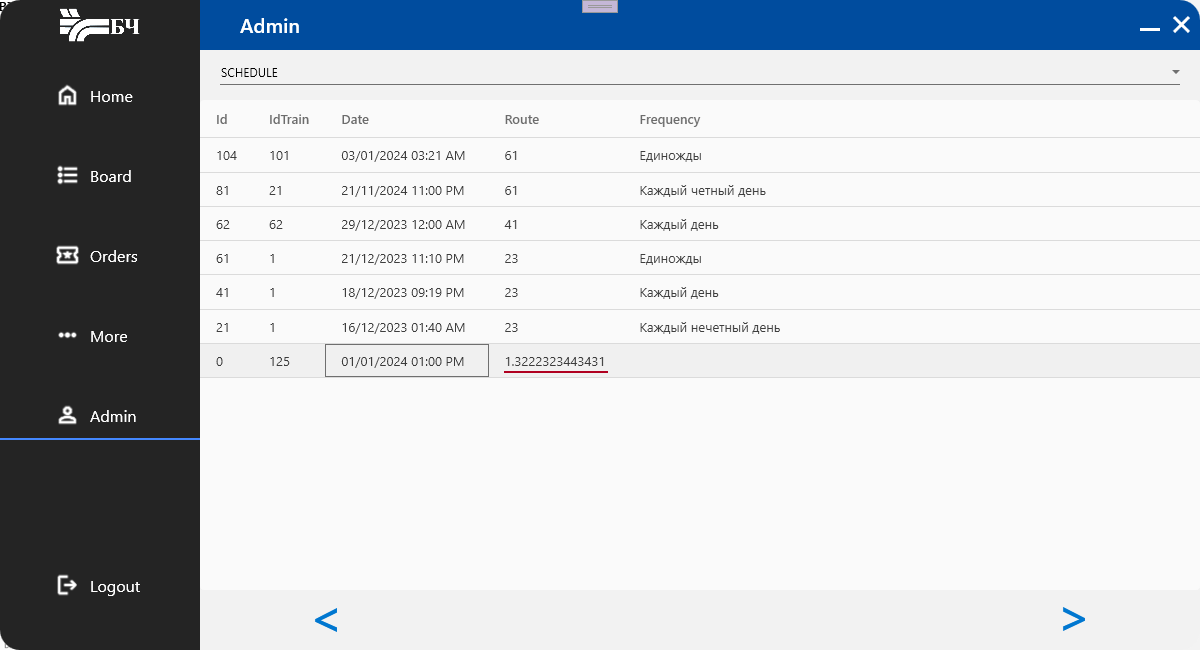


Рис 5.18 – Обработка ввода нецелочисленных значений

При наличии диапазона допустимых значений требуется обрабатывать значения, выходящие из диапазона, обработка продемонстрирована на рисунке 5.19.

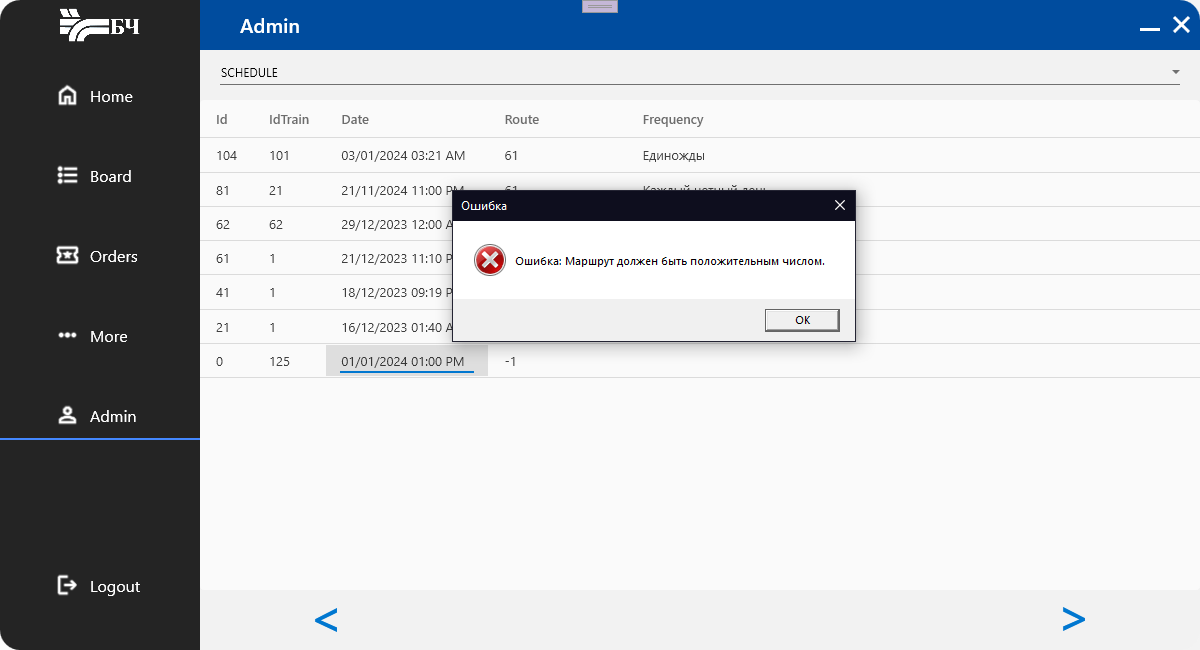


Рис 5.19 – Обработка ввода выходящих за допустимые границы значений

При вводе даты, требуется проверять её формат и при несоответствии обрабатывать, обработка показана на рисунке 5.20.

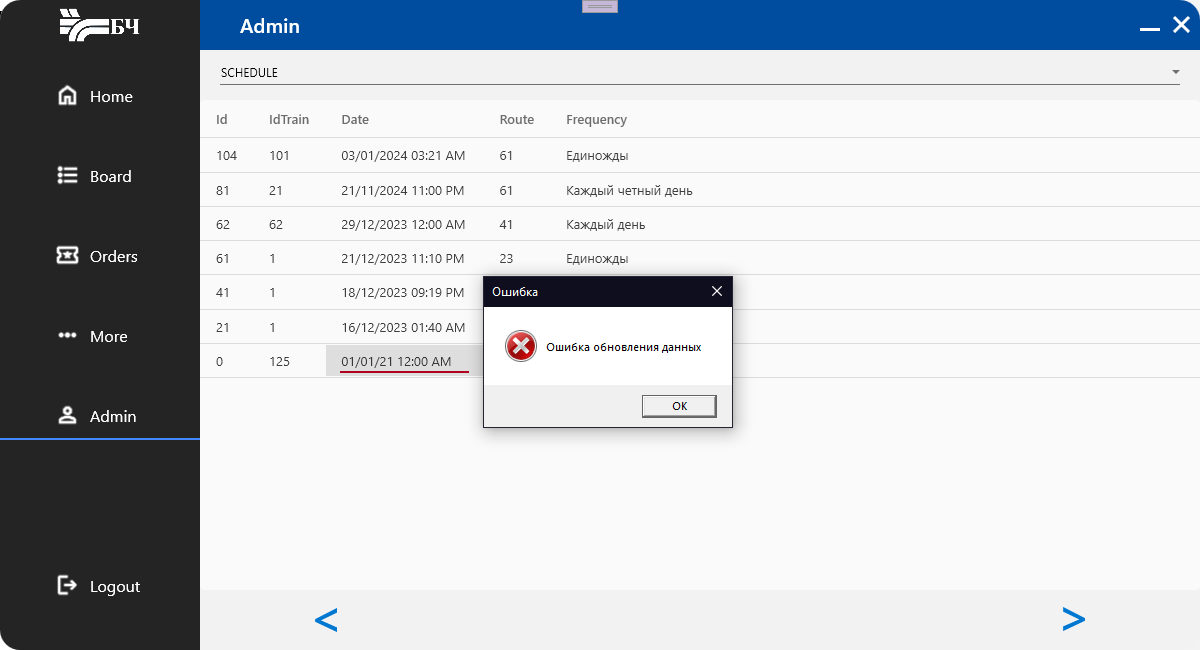


Рис 5.20 – Обработка ввода некорректного формата даты и времени

В ходе тестирования панели администратора были продемонстрированы самые частые ошибки, а также способы их обозначения.

5.5. Вывод.

В данном разделе были протестированы возможности клиентов и менеджеров. По итогам тестирования можно сказать, что все работает корректно.

6. Руководство по использованию программного продукта

При запуске программного средства «Железнодорожный вокзал» пользователь попадает на страницу авторизации, содержащую формы входа и изменения параметров подключения к базе данных.

Стартовая страница позволяет существующему пользователю совершить вход в аккаунт, а при необходимости в изменении параметров подключения к базе данных – перейти на страницу нажатием на кнопку «Изменение настройки подключения к БД» и изменить параметры на необходимые, после чего пользователь будет перенаправлен на страницу авторизации, с которой он может совершить вход в систему.

Окно авторизации представлено на рисунках 6.1.

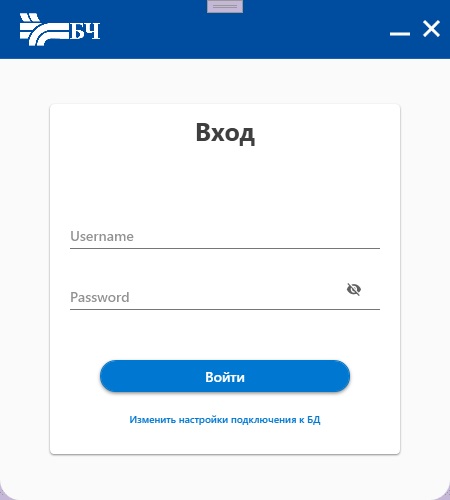


Рисунок 6.1 – Страница авторизации

Окно изменения настроек подключения к базе данных представлена на рисунке 6.2.

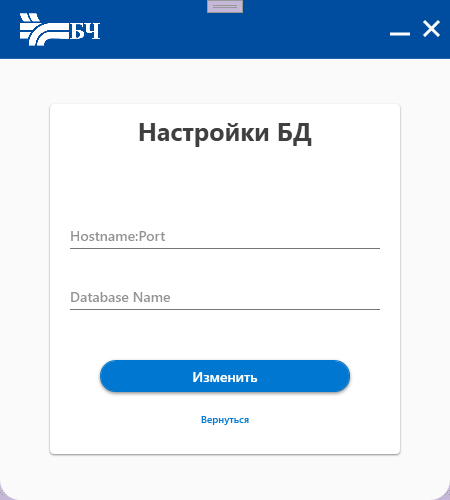


Рисунок 6.2 – Страница изменения параметров базы данных

После выполнения авторизации пользователь перенаправляется на основную страницу программного средства.

На основной странице расположено меню, позволяющее переключаться между страницами и контейнер для содержимого страницы. При переходе из формы регистрации, первой открывается страница поиска рейсов.

Пользователь, в любой момент, может переключиться на нужную страницу, выбрав её в меню.

Для осуществления поиска необходимо заполнить форму данными и нажать на кнопку «Search». Для отмены поиска и возвращения к странице поиска достаточно нажать на элемент меню «Home».

Основная страница изображена на рисунке 6.3.

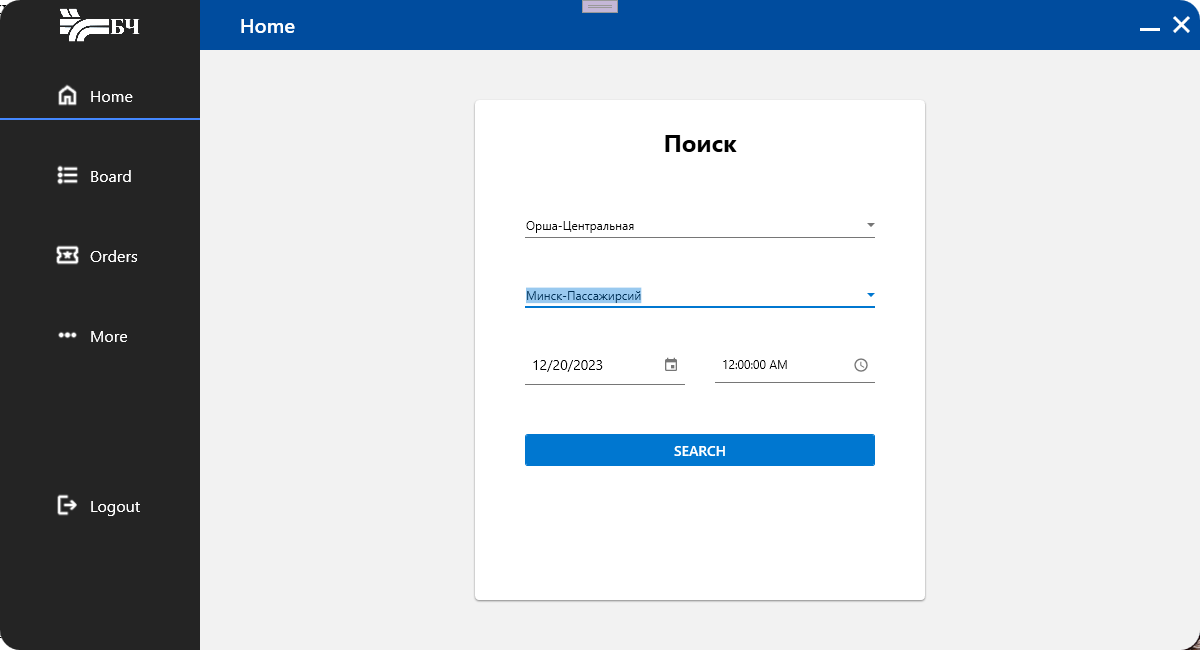


Рисунок 6.3 – Основная страница

Страница поиска представлена на рисунке 6.4.

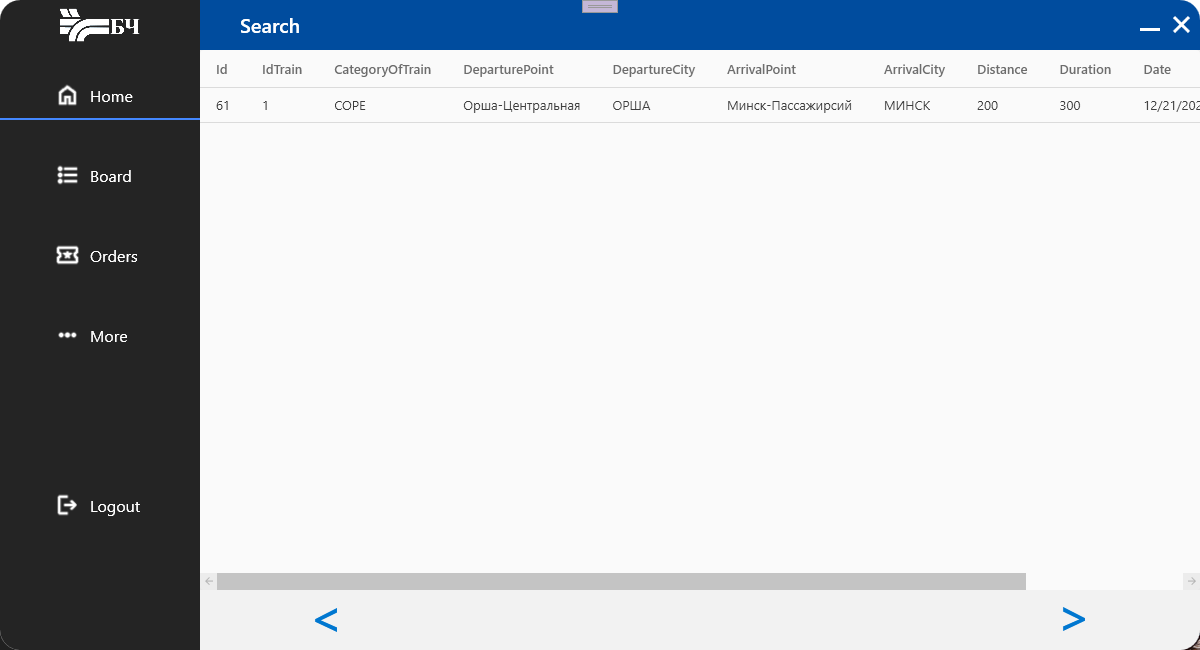


Рисунок 6.4 – Страница поиска

При нажатии на элемент меню «Board» пользователь перенаправляется на страницу доски расписания поездов. На этой странице пользователь может только простматривать информацию, без возможности редактирования. Также снизу расположены кнопки пагинации, позволяющие удобно получать доступ к расписанию, при наличии большого числа записей в таблице.

Также пользователь может выполнять сортировку по столбцу, путем нажатия на нужный столбец.

Страница «Board» проиллюстрирована на рисунке 6.5.

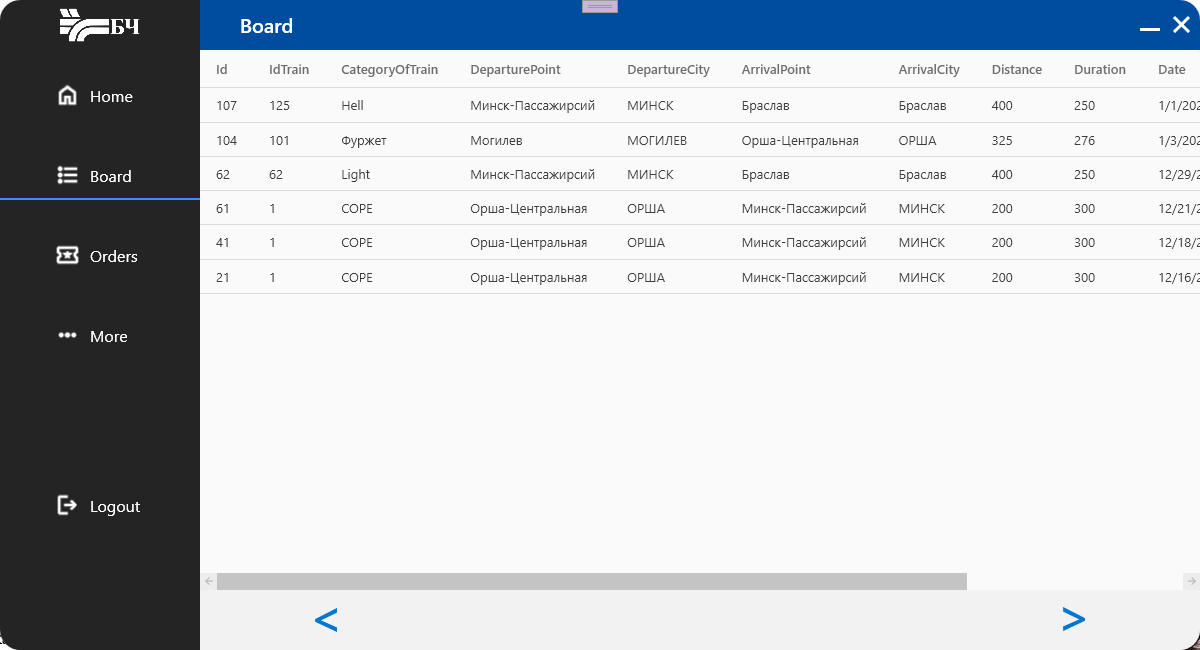


Рисунок 6.5 – Страница «Board»

Кроме просмотра расписания пользователь имеет возможность забронировать билет на странице, на которую можно перейти нажатием на элемент меню «Orders». На ней пользователь имеет возможность найти информацию о заказах определенного пассажира или просмотреть информацию о всех совершённых заказах, без возможности их редактирования, а также добавить новый заказ. Для этого необходимо добавить столбец и заполнить его валидными данными.

Страница «Orders» изображена на рисунке 6.6.

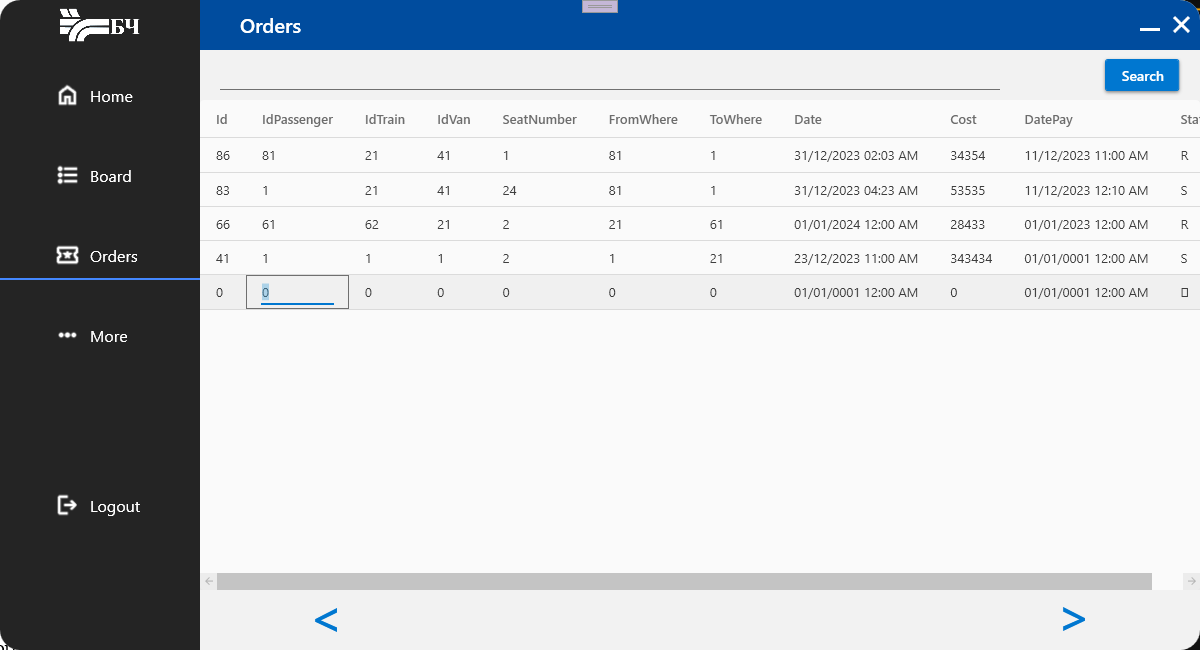


Рисунок 6.6 – Страница «Orders»

Последней доступной, для обычного пользователя, страницей, является страница «More». Страница содержит только информацию о создателе приложения.

Страница «More» представлена на рисунке 6.7.

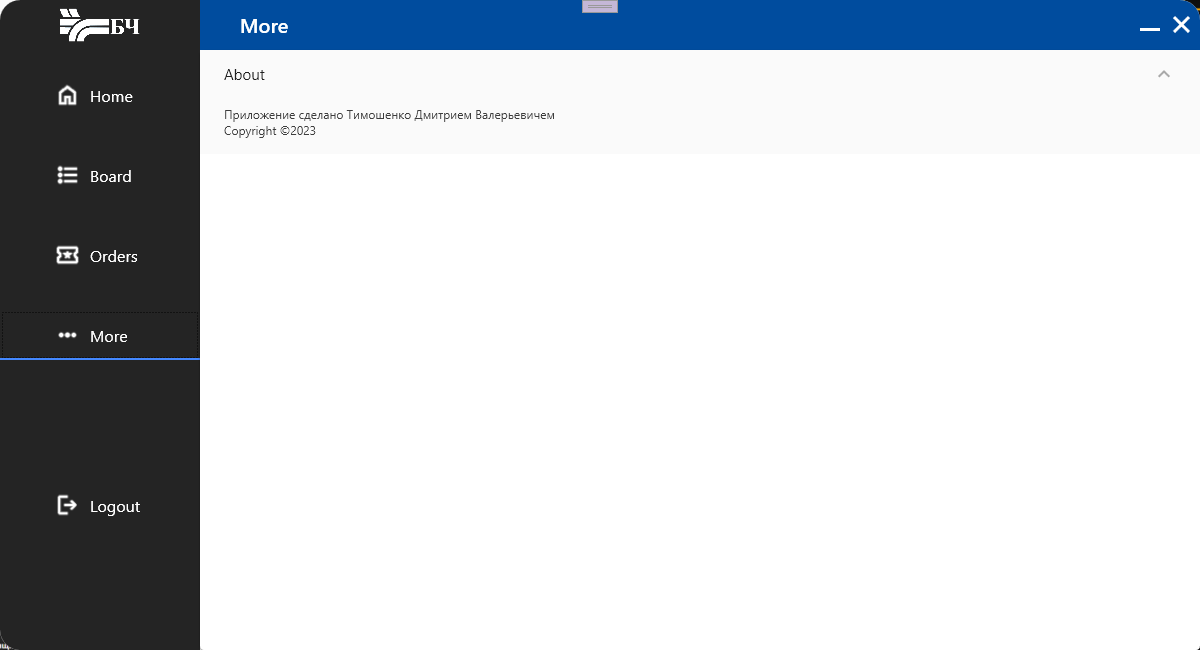


Рисунок 6.7 – Страница «More»

Для менеджера существует страница «Admin», предоставляющая стредства работы (добавления, удаления и изменения данных) со всеми таблицами в базе данных и открывающаяся при нажатии на одноимённый элемент меню, она представленна на рисунке 6.8.

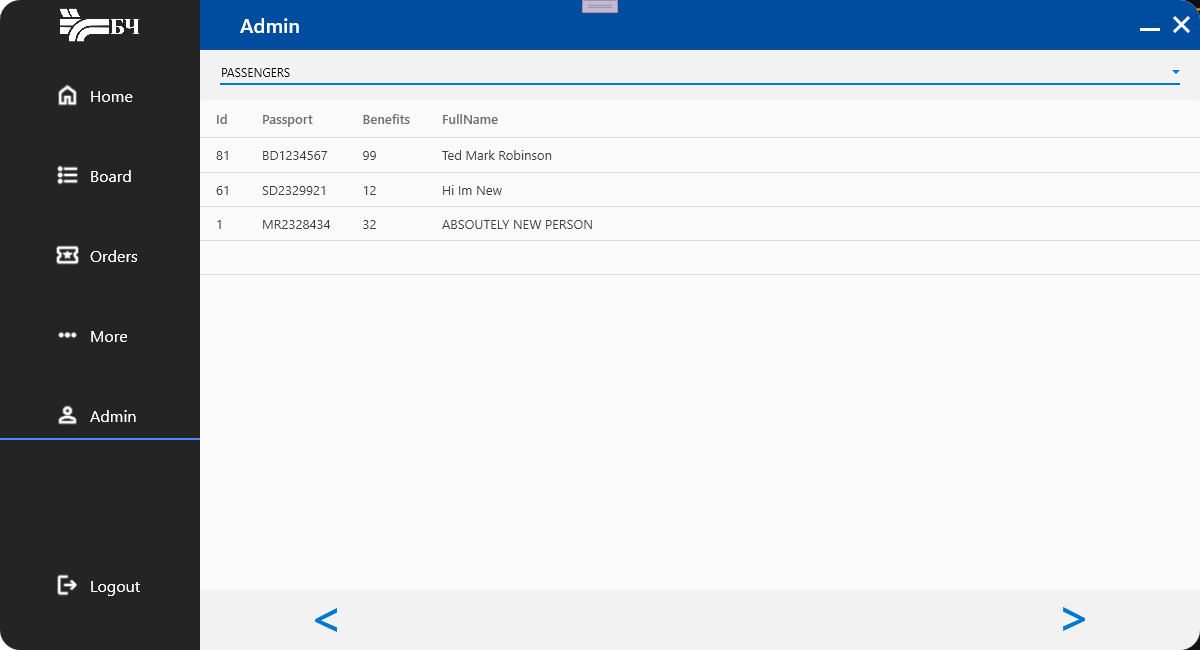


Рисунок 6.8 – Страница Admin

Также пользователь может выйти из приложения, нажав на кнопку «Logout», после чего он будет перенаправлен на страницу авторизации.

6.1. Вывод

Было составлено четкое и понятное руководство пользователя для работы с приложением. Были рассмотрены базовые функции приложения, а также предоставлены скриншоты.

ЗАКЛЮЧЕНИЕ

В данной работе было реализовано программное средство «Железнодорожный вокзал» с использованием C#, WPF и применением архитектурного паттерна MVVM. Целью работы было создание эффективного и надежного программного средства, которое позволит эффективно работать с данными о поездах, рейсах, билетах и пассажирах.

В рамках работы были поставлены следующие задачи: выбор технических средств разработки, проектирование программного средства, его реализация, тестирование и проверка работоспособности, и разработка руководства пользователя.

Были выбраны технические средства, полностью удовлетворяющие поставленным требованиям.

На основе выбранных технических средств были спроектированы и реализованы: база данных, структура и архитектура программного средства, а также были составлены UML диаграммы классов, use-case, последовательностей и базы данных.

Важным этапом работы было тестирование и проверка работоспособности. Были проведены необходимые тесты. Результаты тестирования показали полную работоспособность программного средства.

Было разработано руководство пользователя, которое содержит инструкции по использованию программным средством. В нем описаны основные функции и возможности приложения.

Как итог, работа позволила создать удобное в использовании и гибкое в управлении программное средство, которое может быть использовано для управления данными о железнодорожных рейсах и пассажирах, а также может быть адаптировано под конкретные потребности пользователей.

СПИСОК ЛИТЕРАТУРЫ

1. Object-Oriented Analysis and Design with Applications [Текст]: Гради Буч [и др.]. – 3-е изд., перераб. и доп. — М.: Издательский дом «Вильямс», 26.02.2020. — 720 с.
2. Design Patterns: Elements of Reusable Object-Oriented Software [Текст]: Э. Гамма [и др.]. — СПб.: Питер, 2001. — 368 с.
3. БЧ. Мой поезд [Электронный ресурс] / Справочник Режим доступа:

https://play.google.com/store/apps/details?id=by.rw.client/ – Дата доступа 28.09.2023.

1. Intercity.pl [Электронный ресурс] / Справочник Режим доступа:

https://www.intercity.pl/ – Дата доступа 01.10.2023.

1. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7 [Электронный ресурс] / Справочник Режим доступа:

https://metanit.com/sharp/tutorial/ – Дата доступа: 29.10.2023.

1. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
2. Руководство по WPF [Электронный ресурс] Справочник / Режим доступа:

https://metanit.com/sharp/wpf/ – Дата доступа: 06.11.2023.

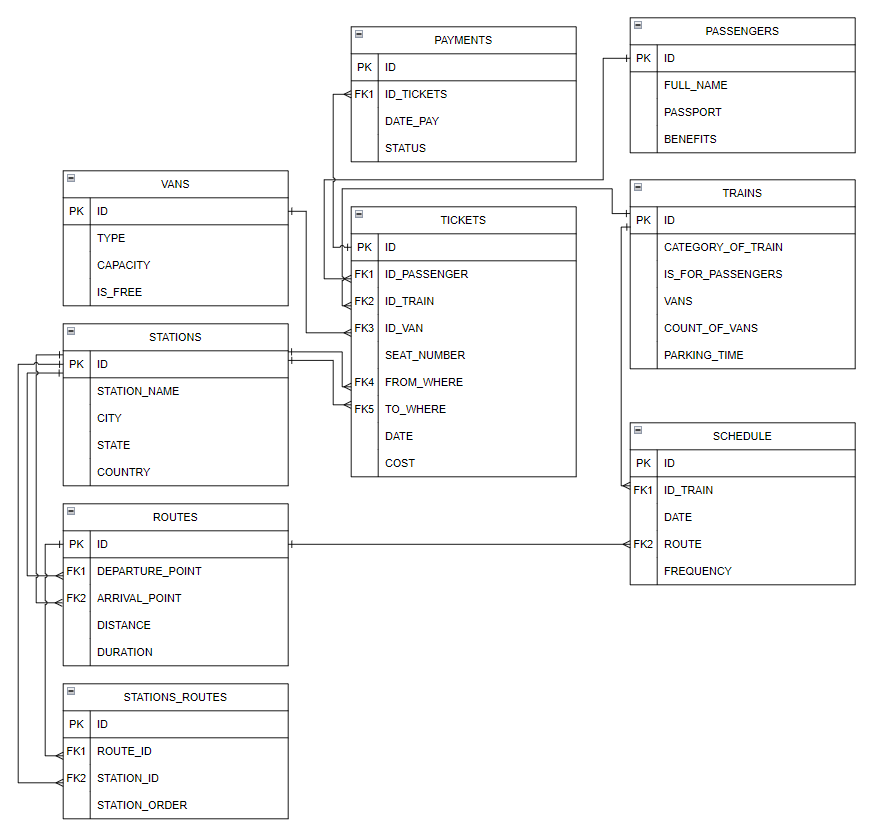
1. Форум StackOverflow [Электронный ресурс] / Справочник Режим доступа:

https://ru.stackoverflow.com/ – Дата доступа 03.12.2023.

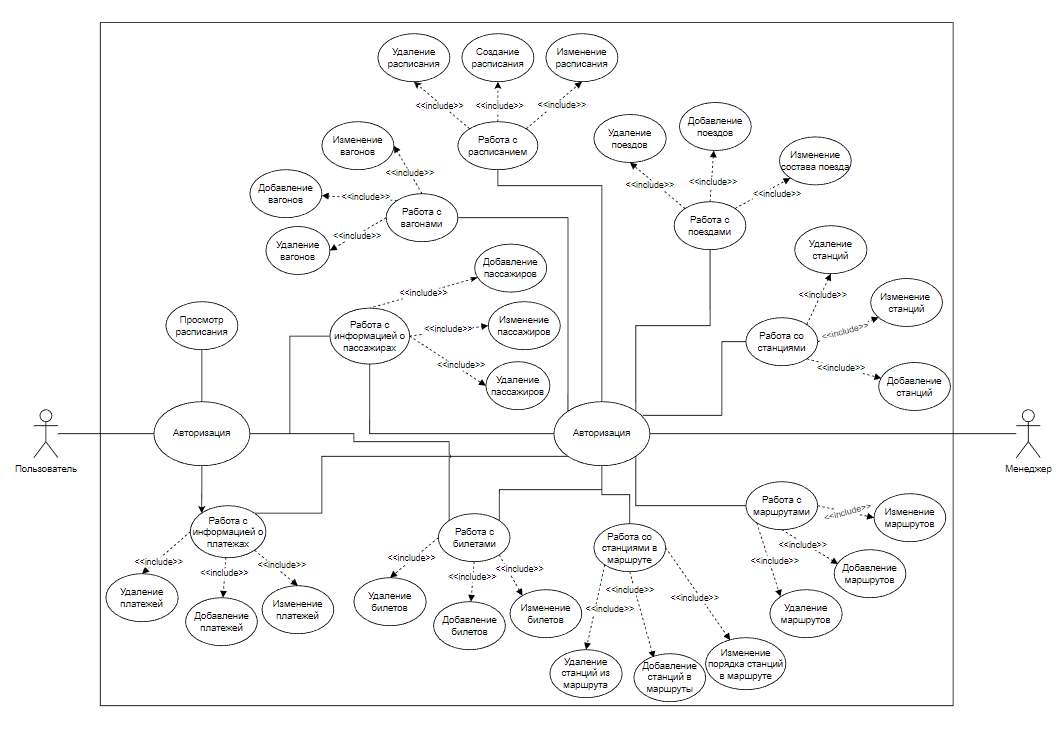
1. Оффициальная документация Oracle [Электронный ресурс] / Справочник Режим доступа:

https://docs.oracle.com/en/ – Дата доступа 11.12.2023.

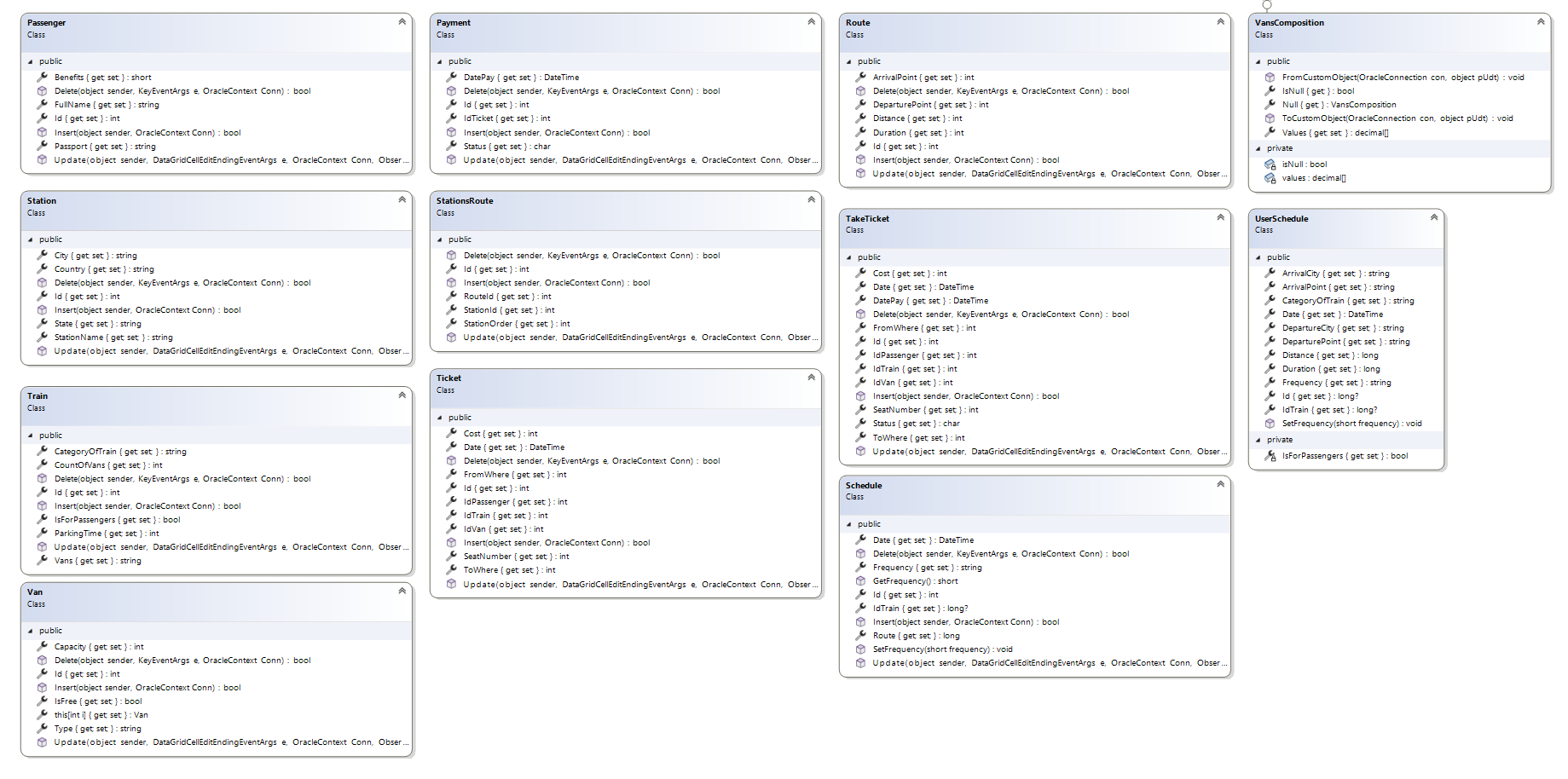
Приложение А Диаграмма базы данных



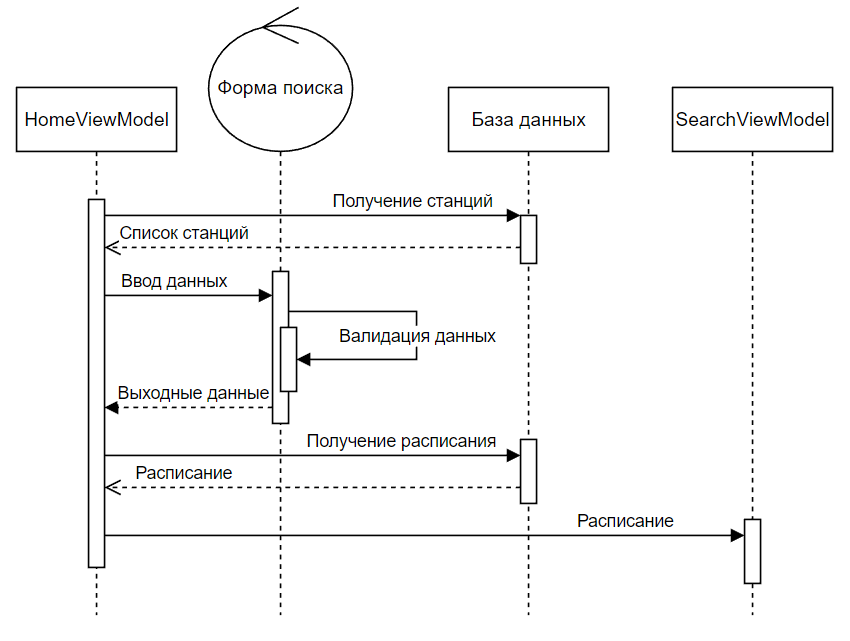
Приложение Б Use-case диаграмма



Приложение В Диаграмма классов



Приложение Г Диаграмма последовательностей



Приложение Д Листинги кода

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using CourseWork.View;  using System;  using System.Windows;  namespace CourseWork.ViewModel  {  class LoginControlViewModel : ObservableObject  {  public String LoginText { get; set; }  public String Username { get; set; }  public String Password { get; set; }  public RelayCommand SubmitButtonCommand { get; set; }  public RelayCommand ChangeSettingsCommand { get; set; }  public LoginControlViewModel(LoginViewModel parent)  {  LoginText = "Вход";  Username = "";  Password = "";  SubmitButtonCommand = new RelayCommand(SubmitButton\_Click);  ChangeSettingsCommand = new RelayCommand(() =>  {  parent.LoginSettingsCommand.Execute(this);  });  }  public void SubmitButton\_Click()  {  if (Username == "" && Password == "") { return; }  try  {  OracleContext conn = OracleContext.Create(Username, Password);  MainWindow w = new MainWindow();  w.Show();  Application.Current.MainWindow.Close();  Application.Current.MainWindow = w;  }  catch (Exception ex)  {  MessageBox.Show(ex.Message,  "Ошибка соединения с базой данных");  }  }  }  } |

Листинг 1 – Класс OracleContext

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using System.Collections.ObjectModel;  using System.ComponentModel;  using System.Text.RegularExpressions;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Input;  namespace CourseWork.ViewModel  {  public partial class AdminViewModel : ObservableObject  {  private int \_rowMin = 0;  private int \_rowMax = 50;  private string Order = "ID DESC";  public int RowMin  {  get { return \_rowMin; }  set { \_rowMin = value; }  }  public int RowMax  {  get => \_rowMax;  set { \_rowMax = value; }  }  public string[] Tables { get; set; } = { "PASSENGERS", "PAYMENTS", "ROUTES",  "SCHEDULE", "STATIONS", "STATIONS\_ROUTES",  "TICKETS", "TRAINS", "VANS"};  [ObservableProperty]  public ObservableCollection<Passenger> \_itemsPassenger = new();  [ObservableProperty]  public ObservableCollection<Payment> \_itemsPayment = new();  [ObservableProperty]  public ObservableCollection<Route> \_itemsRoute = new();  [ObservableProperty]  public ObservableCollection<Schedule> \_itemsSchedule = new();  [ObservableProperty]  public ObservableCollection<Station> \_itemsStation = new();  [ObservableProperty]  public ObservableCollection<StationsRoute> \_itemsStationsRoute = new();  [ObservableProperty]  public ObservableCollection<Ticket> \_itemsTicket = new();  [ObservableProperty]  public ObservableCollection<Train> \_itemsTrain = new();  [ObservableProperty]  public ObservableCollection<Van> \_itemsVan = new();  [ObservableProperty]  private string \_currentTable = "";  private OracleContext Conn { get; set; }  public RelayCommand PrevButtonCommand { get; set; }  public RelayCommand NextButtonCommand { get; set; }  public AdminViewModel()  {  try  {  Conn = OracleContext.Create();  //GetItems();  PrevButtonCommand = new RelayCommand(() =>  {  try  {  if (RowMin <= 0) return;  RowMin -= 50;  RowMax -= 50;  GetItems();  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  NextButtonCommand = new RelayCommand(() =>  {  try  {  RowMin += 50;  RowMax += 50;  GetItems();  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  }  catch  { }  }  private void GetItems(DataGrid dg = null)  {  try  {  switch (CurrentTable)  {  case "PASSENGERS":  Repository<Passenger> passengerRep = new PassengerRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsPassenger;  ItemsPassenger.Clear();  passengerRep.GetAll(RowMin, RowMax, Order, ItemsPassenger);  break;  case "PAYMENTS":  Repository<Payment> paymentRep = new PaymentRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsPayment;  ItemsPayment.Clear();  paymentRep.GetAll(RowMin, RowMax, Order, ItemsPayment);  break;  case "ROUTES":  Repository<Route> routeRep = new RouteRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsRoute;  ItemsRoute.Clear();  routeRep.GetAll(RowMin, RowMax, Order, ItemsRoute);  break;  case "SCHEDULE":  Repository<Schedule> scheduleRep = new ScheduleRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsSchedule;  ItemsSchedule.Clear();  scheduleRep.GetAll(RowMin, RowMax, Order, ItemsSchedule);  break;  case "STATIONS":  Repository<Station> stationRep = new StationRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsStation;  ItemsStation.Clear();  stationRep.GetAll(RowMin, RowMax, Order, ItemsStation);  break;  case "STATIONS\_ROUTES":  Repository<StationsRoute> stationsRouteRep = new StationsRouteRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsStationsRoute;  ItemsStationsRoute.Clear();  stationsRouteRep.GetAll(RowMin, RowMax, Order, ItemsStationsRoute);  break;  case "TICKETS":  Repository<Ticket> ticketRep = new TicketRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsTicket;  ItemsTicket.Clear();  ticketRep.GetAll(RowMin, RowMax, Order, ItemsTicket);  break;  case "TRAINS":  Repository<Train> trainRep = new TrainRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsTrain;  ItemsTrain.Clear();  trainRep.GetAll(RowMin, RowMax, Order, ItemsTrain);  break;  case "VANS":  Repository<Van> vanRep = new VanRepository(Conn);  if (dg != null) dg.ItemsSource = ItemsVan;  ItemsVan.Clear();  vanRep.GetAll(RowMin, RowMax, Order, ItemsVan);  break;  default:  MessageBox.Show("Не существует такой таблицы", "Ошибка", MessageBoxButton.OKCancel, MessageBoxImage.Error);  break;  }  }  catch  {  MessageBox.Show("Ошибка получения данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }    public void Selection\_Changed(object sender, SelectionChangedEventArgs e, DataGrid dg)  {  try  {  ComboBox box = sender as ComboBox;  CurrentTable = Tables[box.SelectedIndex];  Order = "ID DESC";  GetItems(dg);  }  catch  {  MessageBox.Show("Ошибка изменения выбранного элемента", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void Items\_Sorting(object sender, DataGridSortingEventArgs e)  {  try  {  e.Handled = true;  string columnNameUnusable = e.Column.SortMemberPath;  string sortDirection;  string[] substrings = Regex.Split(columnNameUnusable, @"(?<!^)(?=[A-Z])");  string columnName = string.Join("\_", substrings);  if (columnName.ToUpper() == "DATE") columnName = "\"DATE\"";  if (columnName.ToUpper() == "ID") columnName = "\"ID\"";  if (Order.Split(' ')[1].ToUpper() == "DESC")  {  e.Column.SortDirection = ListSortDirection.Ascending;  sortDirection = "ASC";  }  else  {  e.Column.SortDirection = ListSortDirection.Descending;  sortDirection = "DESC";  }  Order = $"{columnName}" + " " + sortDirection;  GetItems();  }  catch  {  MessageBox.Show("Ошибка сортировки", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void UpdateRows(object sender, DataGridCellEditEndingEventArgs e)  {  try  {  switch (CurrentTable)  {  case "PASSENGERS":  if (!Passenger.Update(sender, e, Conn, ItemsPassenger))  e.Cancel = true;  break;  case "PAYMENTS":  if (!Payment.Update(sender, e, Conn, ItemsPayment))  e.Cancel = true;  break;  case "ROUTES":  if (!Route.Update(sender, e, Conn, ItemsRoute))  e.Cancel = true;  break;  case "SCHEDULE":  if (!Schedule.Update(sender, e, Conn, ItemsSchedule))  e.Cancel = true;  break;  case "STATIONS":  if (!Station.Update(sender, e, Conn, ItemsStation))  e.Cancel = true;  break;  case "STATIONS\_ROUTES":  if (!StationsRoute.Update(sender, e, Conn, ItemsStationsRoute))  e.Cancel = true;  break;  case "TICKETS":  if (!Ticket.Update(sender, e, Conn, ItemsTicket))  e.Cancel = true;  break;  case "TRAINS":  if (!Train.Update(sender, e, Conn, ItemsTrain))  e.Cancel = true;  break;  case "VANS":  if (!Van.Update(sender, e, Conn, ItemsVan))  e.Cancel = true;  break;  default:  MessageBox.Show("Не существует такой таблицы", "Ошибка", MessageBoxButton.OKCancel, MessageBoxImage.Error);  break;  }  }  catch  {  e.Cancel = true;  MessageBox.Show("Ошибка обновления данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void PreviewKeyDown(object sender, KeyEventArgs e)  {  if (e.Key == Key.Delete) DeleteRows(sender, e);  if (e.Key == Key.Enter) AddRow(sender);  }    public void DeleteRows(object sender, KeyEventArgs e)  {  try  {  MessageBoxResult res = MessageBox.Show("Вы действительно хотите удалить строку?", "Подтвердите удаление",  MessageBoxButton.YesNo, MessageBoxImage.Question);  if (res != MessageBoxResult.Yes) return;  switch (CurrentTable)  {  case "PASSENGERS":  if (!Passenger.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsPassenger.Remove((Passenger)(sender as DataGrid).SelectedCells[0].Item);  break;  case "PAYMENTS":  if (!Payment.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsPayment.Remove((Payment)(sender as DataGrid).SelectedCells[0].Item);  break;  case "ROUTES":  if (!Route.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsRoute.Remove((Route)(sender as DataGrid).SelectedCells[0].Item);  break;  case "SCHEDULE":  if (!Schedule.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsSchedule.Remove((Schedule)(sender as DataGrid).SelectedCells[0].Item);  break;  case "STATIONS":  if (!Station.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsStation.Remove((Station)(sender as DataGrid).SelectedCells[0].Item);  break;  case "STATIONS\_ROUTES":  if (!StationsRoute.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsStationsRoute.Remove((StationsRoute)(sender as DataGrid).SelectedCells[0].Item);  break;  case "TICKETS":  if (!Ticket.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsTicket.Remove((Ticket)(sender as DataGrid).SelectedCells[0].Item);  break;  case "TRAINS":  if (!Train.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsTrain.Remove((Train)(sender as DataGrid).SelectedCells[0].Item);  break;  case "VANS":  if (!Van.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  ItemsVan.Remove((Van)(sender as DataGrid).SelectedCells[0].Item);  break;  default:  MessageBox.Show("Не существует такой таблицы", "Ошибка", MessageBoxButton.OKCancel, MessageBoxImage.Error);  break;  }  }  catch  {  MessageBox.Show("Ошибка удалений данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void AddRow(object sender)  {  try  {  if ((sender as DataGrid).SelectedItem != null) return;  switch (CurrentTable)  {  case "PASSENGERS":  if (!Passenger.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "PAYMENTS":  if (!Payment.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "ROUTES":  if (!Route.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "SCHEDULE":  if (!Schedule.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "STATIONS":  if (!Station.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "STATIONS\_ROUTES":  if (!StationsRoute.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "TICKETS":  if (!Ticket.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "TRAINS":  if (!Train.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  case "VANS":  if (!Van.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  break;  default:  MessageBox.Show("Не существует такой таблицы", "Ошибка", MessageBoxButton.OKCancel, MessageBoxImage.Error);  break;  }  }  catch  {  MessageBox.Show("Ошибка добавления данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  }  } |

Листинг 2 – Класс AdminViewModel

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using System.Collections.ObjectModel;  using System.ComponentModel;  using System.Text.RegularExpressions;  using System.Windows;  using System.Windows.Controls;  namespace CourseWork.ViewModel  {  public partial class BoardViewModel : ObservableObject  {  private int \_rowMin = 0;  private int \_rowMax = 50;  public int RowMin  {  get { return \_rowMin; }  set { \_rowMin = value; }  }  public int RowMax  {  get => \_rowMax;  set { \_rowMax = value; }  }  [ObservableProperty]  public ObservableCollection<UserSchedule> \_items = new ObservableCollection<UserSchedule>();  private OracleContext conn;  private ScheduleRepository rep;  public RelayCommand PrevButtonCommand { get; set; }  public RelayCommand NextButtonCommand { get; set; }  public BoardViewModel()  {  try  {    conn = OracleContext.Create();  rep = new ScheduleRepository(conn);  GetItems();  PrevButtonCommand = new RelayCommand(() =>  {  if (RowMin <= 0) return;  try  {  RowMin -= 50;  RowMax -= 50;  GetItems();  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  NextButtonCommand = new RelayCommand(() =>  {  try  {  RowMin += 50;  RowMax += 50;  GetItems();  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  }  catch  { }  }  private void GetItems(string Order = "ID desc")  {  try  {  Items.Clear();  rep.TakeSchedule\_User(RowMin, RowMax, Order, Items);  }  catch  {  MessageBox.Show("Ошибка получения данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void Items\_Sorting(object sender, DataGridSortingEventArgs e)  {  try  {  e.Handled = true;  string columnNameUnusable = e.Column.SortMemberPath;  string sortDirection;  string[] substrings = Regex.Split(columnNameUnusable, @"(?<!^)(?=[A-Z])");  string columnName = string.Join("\_", substrings);  if (columnName.ToUpper() == "DATE") columnName = "\"DATE\"";  if (columnName.ToUpper() == "ID") columnName = "\"ID\"";  if (e.Column.SortDirection == ListSortDirection.Descending)  {  e.Column.SortDirection = ListSortDirection.Ascending;  sortDirection = "ASC";  }else  {  e.Column.SortDirection = ListSortDirection.Descending;  sortDirection = "DESC";  }  GetItems($"{columnName}" + " " + sortDirection);  }  catch  {  MessageBox.Show("Ошибка сортировки", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  }  } |

Листинг 3 – Класс BoardViewModel

|  |
| --- |
| using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using Oracle.ManagedDataAccess.Client;  using System;  using System.Collections.Generic;  using System.Text;  using System.Windows;  namespace CourseWork.ViewModel  {  internal class HomeViewModel  {  private readonly MainViewModel MainVM;  public List<string> Stations { get; set; }  public string FromStation { get; set; } = string.Empty;  public string ToStation { get; set; } = string.Empty;  public DateOnly DateBegin { get; set; } = DateOnly.FromDateTime(DateTime.Today);  public TimeOnly DateEnd { get; set; }  public RelayCommand SearchCommand { get; set; }  private OracleContext conn;  public HomeViewModel(MainViewModel mvm)  {  MainVM = mvm;  Stations = new List<string>();  conn = OracleContext.Create();  string sql = "SELECT STATION\_NAME FROM MANAGER.STATIONS";  try  {  OracleCommand cmd = new OracleCommand(sql);  using (OracleDataReader reader = conn.SelectQuery(cmd))  {  while (reader.Read())  {  Stations.Add(reader.GetString(0));  }  conn.conn.Close();  }  }  catch  {  MessageBox.Show("Ошибка чтения данных из базы данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  SearchCommand = new RelayCommand(() =>  {  try  {  if (ToStation.Trim() == ""  && FromStation.Trim() == ""  && (DateBegin <= DateOnly.FromDateTime(DateTime.Today)  && DateEnd == TimeOnly.MinValue))  {  if (DateBegin < DateOnly.FromDateTime(DateTime.Today))  throw new Exception("Дата не может быть раньше текущей");  throw new Exception("Введите все данные для поиска");  }  StringBuilder Where = new StringBuilder();  Where.Append("ARRIVAL\_POINT = '" + ToStation + "'");  Where.Append(" AND ");  Where.Append("DEPARTURE\_POINT = '" + FromStation + "'");  Where.Append(" AND ");  if (!(DateBegin == DateOnly.MinValue  && DateEnd == TimeOnly.MinValue))  Where.Append("\"DATE\" >= " +  "TO\_DATE('" + DateBegin + " " + DateEnd.ToString("HH:mm") + "', 'MM/DD/YYYY HH24:MI')");  else if (DateBegin == DateOnly.MinValue)  Where.Append("\"DATE\" >= " + "TO\_DATE('" + DateEnd.ToString("HH:mm") + "', 'HH24:MI')");  else  Where.Append("\"DATE\" >= " + "TO\_DATE('" + DateBegin + "', 'MM/DD/YYYY')");  MainVM.SearchVM = new SearchViewModel(Where.ToString());  MainVM.CurrentView = MainVM.SearchVM;  MainVM.PageName = "Search";  }  catch (Exception ex)  {  MessageBox.Show(ex.Message, "Ошибка поиска", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  }  }  } |

Листинг 4 – Класс HomeViewModel

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using CourseWork.View;  using System;  using System.Windows;  namespace CourseWork.ViewModel  {  class LoginControlViewModel : ObservableObject  {  public String LoginText { get; set; }  public String Username { get; set; }  public String Password { get; set; }  public RelayCommand SubmitButtonCommand { get; set; }  public RelayCommand ChangeSettingsCommand { get; set; }  public LoginControlViewModel(LoginViewModel parent)  {  LoginText = "Вход";  Username = "";  Password = "";  SubmitButtonCommand = new RelayCommand(SubmitButton\_Click);  ChangeSettingsCommand = new RelayCommand(() =>  {  parent.LoginSettingsCommand.Execute(this);  });  }  public void SubmitButton\_Click()  {  if (Username == "" && Password == "") { return; }  try  {  OracleContext conn = OracleContext.Create(Username, Password);  MainWindow w = new MainWindow();  w.Show();  Application.Current.MainWindow.Close();  Application.Current.MainWindow = w;  }  catch (Exception ex)  {  MessageBox.Show(ex.Message,  "Ошибка соединения с базой данных");  }  }  }  } |

Листинг 5 – Класс LoginControlViewModel

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using System;  using System.Text.RegularExpressions;  using System.Windows;  namespace CourseWork.ViewModel  {  class LoginSettingsViewModel : ObservableObject  {  public String LoginText { get; set; }  public String Hostname { get; set; }  public String DbName { get; set; }  public RelayCommand SubmitButtonCommand { get; set; }  public RelayCommand ChangeSettingsCommand { get; set; }  public LoginSettingsViewModel(LoginViewModel parent)  {  LoginText = "Настройки БД";  Hostname = "";  DbName = "";  SubmitButtonCommand = new RelayCommand(SubmitButton\_Click);  ChangeSettingsCommand = new RelayCommand(() =>  {  parent.LoginControlCommand.Execute(this);  });  }  public void SubmitButton\_Click()  {  if (Hostname == "" && DbName == "") { return; }  try  {  Regex regex = new Regex(@"([A-Za-z1-9.])+:[1-9]{4,5}", RegexOptions.IgnoreCase);  if (!regex.IsMatch(Hostname))  {  MessageBox.Show("Hostname должен быть заполнен по шаблону", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  return;  }  Application.Current.Resources["connectionString"] = "Data Source = " + Hostname + "/" + DbName  + ";Persist Security Info=True;";  ChangeSettingsCommand.Execute(this);  }  catch (Exception ex)  {  MessageBox.Show(ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  }  } |

Листинг 6 – Класс LoginSettingsViewModel

|  |
| --- |
| using System.Windows;  using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  namespace CourseWork.ViewModel  {  partial class LoginViewModel : ObservableObject  {  [ObservableProperty]  private object \_loginView;  public RelayCommand LoginControlCommand { get; set; }  public RelayCommand LoginSettingsCommand { get; set; }  public LoginControlViewModel LoginControlVM { get; set; }  public LoginSettingsViewModel LoginSettingsVM { get; set; }  public RelayCommand CloseButtonCommand { get; set; }  public RelayCommand HideButtonCommand { get; set; }  public LoginViewModel()  {  LoginControlVM = new LoginControlViewModel(this);  LoginSettingsVM = new LoginSettingsViewModel(this);    LoginView = LoginControlVM;  LoginControlCommand = new RelayCommand(() =>  {  LoginView = LoginControlVM;  });  LoginSettingsCommand = new RelayCommand(() =>  {  LoginView = LoginSettingsVM;  });  CloseButtonCommand = new RelayCommand(CloseButton\_Click);  HideButtonCommand = new RelayCommand(HideButton\_Click);  }  public void CloseButton\_Click()  {  Application.Current.Shutdown();  }  public void HideButton\_Click()  {  Application.Current.MainWindow.WindowState  = WindowState.Minimized;  }  }  } |

Листинг 7 – Класс LoginViewModel

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using CourseWork.View;  using System;  using System.Windows;  namespace CourseWork.ViewModel  {  class MainViewModel : ObservableObject  {  private object \_currentView;  private string \_pageName;  public bool IsAdmin { get; set; }  public string PageName  {  get { return \_pageName; }  set  {  \_pageName = value;  OnPropertyChanged();  }  }  public RelayCommand LoginViewCommand { get; set; }  public RelayCommand HomeViewCommand { get; set; }  public RelayCommand BoardViewCommand { get; set; }  public RelayCommand AdminViewCommand { get; set; }  public RelayCommand OrdersViewCommand { get; set; }  public RelayCommand MoreViewCommand { get; set; }  public RelayCommand CloseButtonCommand { get; set; }  public RelayCommand HideButtonCommand { get; set; }  public HomeViewModel HomeVM { get; set; }  public BoardViewModel BoardVM { get; set; }  public SearchViewModel SearchVM { get; set; }  public AdminViewModel AdminVM { get; set; }  public OrdersViewModel OrdersVM { get; set; }  public MoreViewModel MoreVM { get; set; }  public object CurrentView  {  get { return \_currentView; }  set  {  \_currentView = value;  OnPropertyChanged();  }  }  public MainViewModel()  {  PageName = "Home";  HomeVM = new HomeViewModel(this);  BoardVM = new BoardViewModel();  OrdersVM = new OrdersViewModel();  MoreVM = new MoreViewModel();  AdminVM = new AdminViewModel();  SearchVM = null;  CurrentView = HomeVM;  IsAdmin = OracleContext.Create().IsAdmin;  LoginViewCommand = new RelayCommand(() =>  {  try  {  CurrentView = HomeVM;  PageName = "Home";  OracleContext.Delete();  Login w = new Login();  w.Show();  Application.Current.MainWindow.Close();  Application.Current.MainWindow = w;  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  });  HomeViewCommand = new RelayCommand(() =>  {  try  {  HomeVM = new HomeViewModel(this);  CurrentView = HomeVM;  PageName = "Home";  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  });  BoardViewCommand = new RelayCommand(() =>  {  try  {  BoardVM = new BoardViewModel();  CurrentView = BoardVM;  PageName = "Board";  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  });  OrdersViewCommand = new RelayCommand(() =>  {  try  {  OrdersVM = new OrdersViewModel();  CurrentView = OrdersVM;  PageName = "Orders";  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  });  MoreViewCommand = new RelayCommand(() =>  {  try  {  MoreVM = new MoreViewModel();  CurrentView = MoreVM;  PageName = "More";  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  });  AdminViewCommand = new RelayCommand(() =>  {  try  {  CurrentView = AdminVM;  PageName = "Admin";  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  });  CloseButtonCommand = new RelayCommand(CloseButton\_Click);  HideButtonCommand = new RelayCommand(HideButton\_Click);  }  public void CloseButton\_Click()  {  Application.Current.Shutdown();  }  public void HideButton\_Click()  {  Application.Current.MainWindow.WindowState  = WindowState.Minimized;  }  }  } |

Листинг 8 – Класс MainViewModel

|  |
| --- |
| using System;  using CommunityToolkit.Mvvm.ComponentModel;  namespace CourseWork.ViewModel  {  class MoreViewModel : ObservableObject  {  public string Info { get; set; }  public MoreViewModel()  {  Info = "Приложение сделано Тимошенко Дмитрием Валерьевичем\n" +  $"Copyright ©{DateTime.Now.ToString("yyyy")}";  }  }  } |

Листинг 9 – Класс MoreViewModel

|  |
| --- |
| using System;  using System.Collections.ObjectModel;  using System.ComponentModel;  using System.Text.RegularExpressions;  using System.Windows.Controls;  using System.Windows.Input;  using System.Windows;  using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  namespace CourseWork.ViewModel  {  public partial class OrdersViewModel : ObservableObject  {  private int \_rowMin = 0;  private int \_rowMax = 50;  private string Order = "ID DESC";  public int RowMin  {  get { return \_rowMin; }  set { \_rowMin = value; }  }  public int RowMax  {  get => \_rowMax;  set { \_rowMax = value; }  }  public string SearchPassenger { get; set; } = "";  [ObservableProperty]  public ObservableCollection<TakeTicket> \_items = new();  private OracleContext Conn { get; set; }  public RelayCommand PrevButtonCommand { get; set; }  public RelayCommand NextButtonCommand { get; set; }  public RelayCommand SearchButtonCommand { get; set; }  public OrdersViewModel()  {  try  {  Conn = OracleContext.Create();  GetItems();  PrevButtonCommand = new RelayCommand(() =>  {  try  {  if (RowMin <= 0) return;  RowMin -= 50;  RowMax -= 50;  GetItems();  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  NextButtonCommand = new RelayCommand(() =>  {  try  {  RowMin += 50;  RowMax += 50;  GetItems();  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  SearchButtonCommand = new RelayCommand(() =>  {  try  {  if (SearchPassenger == "") GetItems();  string IdPas = Convert.ToInt64(SearchPassenger).ToString();  GetItems(IdPas);  }  catch (Exception ex)  {  MessageBox.Show(ex.Message, "Ошибка");  }  });  }  catch  { }  }  private void GetItems(string \_SearchPassenger = "")  {  try  {  OrdersRepository orderRep = new OrdersRepository(Conn);  Items.Clear();  orderRep.GetAll(RowMin, RowMax, \_SearchPassenger, Order, Items);  }  catch  {  MessageBox.Show("Ошибка получения данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void Items\_Sorting(object sender, DataGridSortingEventArgs e)  {  try  {  e.Handled = true;  string columnNameUnusable = e.Column.SortMemberPath;  string sortDirection;  string[] substrings = Regex.Split(columnNameUnusable, @"(?<!^)(?=[A-Z])");  string columnName = string.Join("\_", substrings);  if (columnName.ToUpper() == "DATE") columnName = "\"DATE\"";  if (columnName.ToUpper() == "ID") columnName = "\"ID\"";  if (Order.Split(' ')[1].ToUpper() == "DESC")  {  e.Column.SortDirection = ListSortDirection.Ascending;  sortDirection = "ASC";  }  else  {  e.Column.SortDirection = ListSortDirection.Descending;  sortDirection = "DESC";  }  Order = $"{columnName}" + " " + sortDirection;  GetItems();  }  catch  {  MessageBox.Show("Ошибка сортировки", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void UpdateRows(object sender, DataGridCellEditEndingEventArgs e)  {  try  {  if (!TakeTicket.Update(sender, e, Conn, Items))  e.Cancel = true;  }  catch  {  e.Cancel = true;  MessageBox.Show("Ошибка обновления данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void PreviewKeyDown(object sender, KeyEventArgs e)  {  if (e.Key == Key.Delete) DeleteRows(sender, e);  if (e.Key == Key.Enter) AddRow(sender);  }  public void DeleteRows(object sender, KeyEventArgs e)  {  try  {  MessageBoxResult res = MessageBox.Show("Вы действительно хотите удалить строку?", "Подтвердите удаление",  MessageBoxButton.YesNo, MessageBoxImage.Question);  if (res != MessageBoxResult.Yes) return;  if (!TakeTicket.Delete(sender, e, Conn))  MessageBox.Show("Ошибка при удалении строки");  else  Items.Remove((TakeTicket)(sender as DataGrid).SelectedCells[0].Item);  }  catch  {  MessageBox.Show("Ошибка удаления данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void AddRow(object sender)  {  try  {  if ((sender as DataGrid).SelectedItem != null) return;  if (!TakeTicket.Insert(sender, Conn))  MessageBox.Show("Ошибка при добавлении строки");  }  catch  {  MessageBox.Show("Ошибка добавления строки", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  }  } |

Листинг 10 – Класс OrdersViewModel

|  |
| --- |
| using CommunityToolkit.Mvvm.ComponentModel;  using CommunityToolkit.Mvvm.Input;  using CourseWork.Model;  using System.Collections.ObjectModel;  using System.ComponentModel;  using System.Text.RegularExpressions;  using System.Windows;  using System.Windows.Controls;  namespace CourseWork.ViewModel  {  public partial class SearchViewModel : ObservableObject  {  private int \_rowMin = 0;  private int \_rowMax = 50;  private string Where;  private string Order;  public int RowMin  {  get { return \_rowMin; }  set { \_rowMin = value; }  }  public int RowMax  {  get => \_rowMax;  set { \_rowMax = value; }  }  [ObservableProperty]  public ObservableCollection<UserSchedule> \_items = new ObservableCollection<UserSchedule>();  private OracleContext conn;  private ScheduleRepository rep;  public RelayCommand PrevButtonCommand { get; set; }  public RelayCommand NextButtonCommand { get; set; }  public SearchViewModel(string \_Where = "ID > 0")  {  try  {  Where = \_Where;  Order = "ID desc";  conn = OracleContext.Create();  rep = new ScheduleRepository(conn);  GetItems(Where, Order);  PrevButtonCommand = new RelayCommand(() =>  {  try  {  if (RowMin <= 0) return;  RowMin -= 50;  RowMax -= 50;  GetItems(Where, Order);  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  NextButtonCommand = new RelayCommand(() =>  {  try  {  RowMin += 50;  RowMax += 50;  GetItems(Where, Order);  }  catch  {  MessageBox.Show("Ошибка перехода на другую страницу", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  });  }  catch  { }  }  private void GetItems(string Where, string Order = "ID desc")  {  try  {  Items.Clear();  rep.TakeSchedule\_User(RowMin, RowMax, Where, Order, Items);  }  catch  {  MessageBox.Show("Ошибка получения данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  public void Items\_Sorting(object sender, DataGridSortingEventArgs e)  {  try  {  e.Handled = true;  string columnNameUnusable = e.Column.SortMemberPath;  string sortDirection;  string[] substrings = Regex.Split(columnNameUnusable, @"(?<!^)(?=[A-Z])");  string columnName = string.Join("\_", substrings);  if (columnName.ToUpper() == "DATE") columnName = "\"DATE\"";  if (columnName.ToUpper() == "ID") columnName = "\"ID\"";  if (e.Column.SortDirection == ListSortDirection.Descending)  {  e.Column.SortDirection = ListSortDirection.Ascending;  sortDirection = "ASC";  }  else  {  e.Column.SortDirection = ListSortDirection.Descending;  sortDirection = "DESC";  }  Order = $"{columnName}" + " " + sortDirection;  GetItems(Where, Order);  }  catch  {  MessageBox.Show("Ошибка сортировки", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);  }  }  }  } |

Листинг 11 – Класс SearchViewModel