Частное учреждение образования

«Колледж бизнеса и права»

ТЕМА ПОЛНОСТЬЮ КАК НАПИСАНО В ПОСТАНОВКЕ ЗАДАЧИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по учебному предмету «Конструирование программ  
 и языки программирования»

КП Т.094023.401

Руководитель проекта (А.В. Кривошеина)

Обучающийся ( А. И. Шпак )

2023

**СОДЕРЖАНИЕ**

Изм.

Изм.

Изм.

Изм.

Лист

Лист

Лист

Лист

№ докум.

№ докум.

№ докум.

№ докум.

Подпись

Подпись

Подпись

Подпись

Дата

Дата

Дата

Дата

Лист

Лист

Лист

Лист

3

3

3

3

КП Т.094023.401 ПЗ

КП Т.716007.401 ПЗ

КП Т.716009.401 ПЗ

КП Т.716007.401 ПЗ

Разраб.

Разраб.

Разраб.

Разраб.

*Шпак А.И.*

*Голубев И.В.*

*Гринь А.М.*

*Голубев И.В.*

Провер.

Провер.

Провер.

Провер.

Кривошеина А. В.

Банцевич. С.В

Банцевич. С.В

Банцевич. С.В

Т. контр.

Т. контр.

Т. контр.

Т. контр.

Н. контр.

Н. контр.

Н. контр.

Н. контр.

Утверд.

Утверд.

Утверд.

Утверд.

ТЕМА КАК В ПОСТАНОВКЕ

Лит.

У

Лит.

У

Лит.

У

Лит.

У

Листов

Листов

Листов

Листов

60

50

55

50

КБП

КБП

КБП

КБП

[Введение 3](#_Toc132640062)

[1 Описание задачи 5](#_Toc132640063)

[1.1 Анализ предметной области 5](#_Toc132640064)

[1.2 Постановка задачи 6](#_Toc132640065)

[2 Проектирование системы 7](#_Toc132640066)

[2.1 Требования к приложению 7](#_Toc132640067)

[2.2 Проектирование модели 7](#_Toc132640068)

[2.3 Организация данных 9](#_Toc132640069)

[2.4 Концептуальный прототип 10](#_Toc132640070)

[3 Описание реализации программного средства 13](#_Toc132640071)

[3.1 Инструменты разработки и применяемые технологии 13](#_Toc132640072)

[3.2 Порядок авторизации пользователей 14](#_Toc132640073)

[3.3 Организация данных 14](#_Toc132640074)

[3.4 Функции: логическая и физическая организация 17](#_Toc132640075)

[3.5 Входные и выходные данные 22](#_Toc132640076)

[3.6 Функциональное тестирование 23](#_Toc132640077)

[3.7 Описание справочной системы 26](#_Toc132640078)

[4 Применение 27](#_Toc132640079)

[4.1 Назначение программного средства 27](#_Toc132640080)

[4.2 Условия применения 27](#_Toc132640081)

[Заключение 29](#_Toc132640082)

[Список использованных источников 30](#_Toc132640083)

[Приложение А Текст программных модулей 31](#_Toc132640084)

[Приложение Б Формы выходных документов 32](#_Toc132640085)

[Приложение В Результаты работы программы 33](#_Toc132640086)

**Введение**

Мессенджер – это программа, мобильное приложение или веб-сервис для мгновенного обмена сообщениями.

Нужно сказать, что понятие мессенджера уже давно не связывают только с обменом текстовыми сообщениями. Современные мессенджеры уже стали полноценными коммуникационными центрами, которые помимо обмена сообщениями реализуют голосовую и видеосвязь, обмен файлами, веб-конференции.

С помощью мессенджеров бизнес отправляет рассылки, создаёт каналы для привлечения клиентов или распространяет рекламу. Бизнес использует мессенджеры, чтобы распространять контент, общаться с покупателями, уведомлять о заказах и доставке, рассылать информацию об акциях и скидках, привлекать клиентов с помощью розыгрышей и викторин, продавать товары и услуги.

Целью курсового проекта на тему «Приложение для покупки билетов на электропоезда» является разработка программы, с помощью которой пользователи могли бы приобретать билеты на маршруты электропоездов. Для того, чтобы пользователь мог приобретать билеты на электропоезда, ему надо завести аккаунт. Для этого необходимо зарегистрироваться, указав уникальное имя пользователя, электронный адрес и пароль. После успешной регистрации, пользователям будут доступны функции просмотра доступных маршрутов, покупка билетов, просмотр и печать купленных билетов.

Решение поставленных задач отражено в пояснительной записке, которая состоит из четырех разделов и содержит необходимую и достаточную информацию по использованию данного программного средства.

В первом разделе «Описание задачи» рассматривается сущность и актуальность поставленной задачи, описание существующих аналогов, описание основных функций для разработки программного средства.

Второй раздел «Проектирование системы» содержит проектирование модели на основе выбранных диаграмм, требования к аппаратным и операционным ресурсам, средства защиты, организация данных и описание концептуального прототипа.

Третий раздел «Описание реализации программного средства» включает описание избранной технологии для реализации программы и избранные инструменты разработки, физическая организация данных, способы реализации функции пользователя, формат входных и выходных данных, функциональное тестирование, описание справочной системы.

Четвертый раздел «Применение» содержит информацию, необходимую в процессе эксплуатации программного средства: его назначение и условия применения.

В заключении описывается выполнение поставленной задачи, степень соответствия проектных решений задания, причины несоответствия, если таковые имеются.

В приложении А представлен текст программных модулей, в приложении Б – формы выходных документов, в приложении В – результаты работы программы.

Графическая часть представлена диаграммами вариантов использования, классов, последовательности, деятельности.

**1 Описание задачи**

**1.1 Анализ предметной области**

Требуется разработать программное средство для возможности приобретения билетов на электропоезда.

Приложение для покупки билетов на электропоезда – это программное средство, предназначенное для приобретения и печати билетов пользователями.

К основным функциям приложений для покупки билетов относятся:

* поиск маршрутов по станциям;
* покупка билетов на существующие маршруты;
* просмотр корзины купленных билетов;
* добавление/удаление станций;
* добавление/редактирование/удаление маршрутов.

Приложение для покупки билетов на электропоезда является программным средством, которое предоставляет возможность покупки билетов на электропоезда через интернет. Пользователи могут использовать клиентское приложение для поиска, покупки и печати билетов на желаемый маршрут электропоездов.

Программное средство будет предназначено для людей, которые желают приобрести билеты дистанционно.

Приложение будет включать в себя две части: пользовательская и админская. Пользователи будут видеть только интерфейс для поиска, приобретения и печати билетов.

Администратору будет доступно добавление и удаление станций и маршрутов.

**1.2 Постановка задачи**

Исходя из анализа предметной области можно выделить следующие задачи:

* ведение базы данных, в которой будет храниться вся информация;
* авторизация пользователей;
* регистрация пользователей;
* поиск, покупка и печать купленных билетов;
* добавление и редактирование маршрутов;

Существует довольно много аналогов для данного приложения, такие как RZD Pass, [TUTU.ru](http://tutu.ru/), Яндекс.Расписания и другие. Разрабатываемое приложение направлено только на пользователей Windows 10. Программное средство сильно упрощено и не имеет больших массивов данных.

**2 Проектирование системы**

**2.1 Требования к приложению**

Разрабатываемое программное средство должно иметь понятный и удобный в использовании интерфейс, чтобы взаимодействие между программой и пользователем было максимально упрощено.

Согласно общим требованиям, графический интерфейс разрабатываемого программного средства должен:

* ориентироваться на пользователя, который общается с программой на внешнем уровне взаимодействия;
* сохранять стандартизированное назначение и местоположение на экране графических объектов, работающих в среде Windows.

Интерфейс программного средства будет разрабатываться с учетом общих требований к пользовательскому интерфейсу.

Существуют общие требования, которые предъявляются к программным приложениям:

* соответствие стандартам организации интерфейса: использование многооконного подхода, реализация управления работой программного приложения с помощью элементов управления;
* требования к выбранной цветовой схеме рабочего экрана;
* выполнение одной функции с помощью разных элементов управления.

Для обучения пользователя необходимо разработать справочную систему, в которой должны быть раскрыты все аспекты работы с программой, возможные трудности, возникшие во время работы и пути их решения.

Для удобства работы пользователя с программным средством необходимо при разработке форм придерживаться единого стиля оформления. Формы не должны быть перегружены излишней информацией или содержать информацию, не относящуюся к данной форме. Также необходимо предусмотреть защиту данных от удаления и изменения, а также от ввода некорректных данных.

**2.2 Проектирование модели**

Цель моделирования данных состоит в обеспечении разработчика информационной системы концептуальной схемой базы данных в форме одной или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему без данных.

Наиболее распространенным средством моделирования данных является диаграмма «Сущность-связь» (ERD). С их помощью определяется важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных.

Диаграмма «Сущность-связь» в нотации Чена представлена на рисунке 2.1.

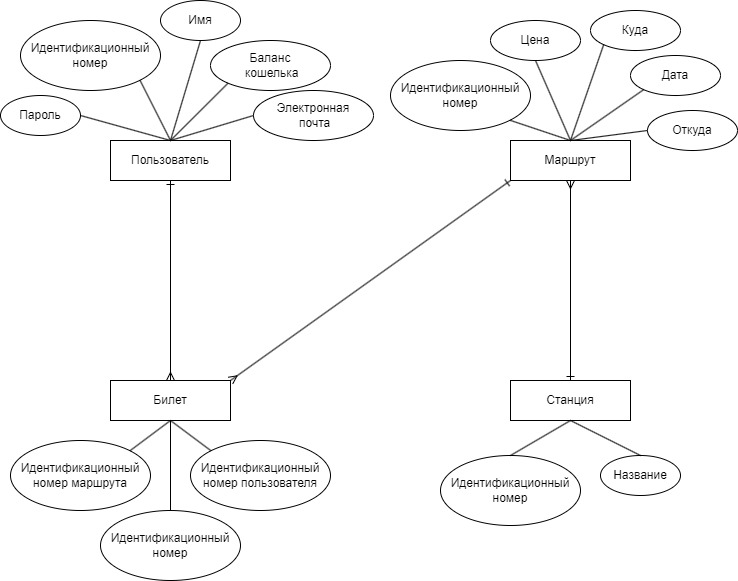


Рисунок 2.1

Исходя из предметной области можно выделить следующие сущности разработки: «Пользователи», «Билеты», «Станции», «Маршруты».

Для сущности «Пользователи» атрибутами будут являться:

* идентификационный номер;
* пароль;
* имя;
* электронная почта;
* имя;
* баланс кошелька;

Для сущности «Билеты» атрибутами будут являться:

* идентификационный номер;
* идентификационный номер маршрута;
* идентификационный номер пользователя.

Для сущности «Станции» атрибутами будут являться:

* идентификационный номер;
* номер.

Для сущности «Маршруты» атрибутами будут являться:

* идентификационный номер;
* откуда (Станция);
* куда (Станция);
* дата;
* цена.

Суть диаграммы вариантов использования заключается в том, что проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования.

Диаграмма вариантов использования представлена на листе 1 графической части.

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

При реализации программного средства были реализованы следующие основные классы:

* Авторизация;
* Регистрация;
* ФормаГлавнаяАдмин;
* ФормаГлавнаяПользователь;
* ФормаДобавлениеСтанции;
* ФормаУдалениеСтанции;
* ФормаДобавлениМаршрута;
* ФормаУдалениеРедактированиеМаршрута;
* ФормаПополнениеБаланса;
* ФормаПокупкаБилета;
* ФормаБилетыПользователя.

Класс «Авторизация» отвечает за включение формы входа, а также за ее логику. Класс «Регистрация» включает форму регистрации нового пользователя. Класс «ФормаГлавнаяАдмин» необходим для отображения доступных функций для администратора, таких как добавление или удаление маршрута, станции. Класс «ФормаГлавнаяПользователь» необходим для отображения доступных функций для пользователя: поиск и покупка билетов, а также просмотр купленных билетов. Класс «ФормаДобавлениеСтанции» позволяет администратору создать новую станцию. Класс «ФормаУдалениеСтанции» позволяет администратору удалить существующую станцию. Класс «ФормаДобавлениМаршрута» позволяет администратору создать новый маршрут, построенный на основе существующих станций. Класс «ФормаУдалениеРедактированиеМаршрута» дает возможность администратору изменить или удалить существующий маршрут. Класс «ФормаПополнениеБаланса» открывает форму пополнения баланса пользователя. Класс «ФормаПокупкаБилета» открывает форму, с помощью которой пользователь может найти интересующий его маршрут по станциям, а также приобрести его. Класс «ФормаБилетыПользователя» отображает все купленный пользователем билеты и позволяет распечатать выбранный.

Диаграмма классов представлена на листе 2 графической части.

При моделировании поведения проектируемой или анализируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности.

Диаграммы деятельности – частный случай диаграмм состояний. Основная цель использования таких диаграмм – визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. Диаграмма деятельности представлена на листе 3 графической части.

Временной аспект поведения имеет существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели и используются диаграммы последовательности, в которых ключевым моментом является динамика взаимодействия объектов во времени. Например, сотрудник переходит с вкладки на вкладку и добавляет данные, что является временным аспектом поведения сотрудника. Он включает программу, вводит данные, сохраняет данные – последовательность действий. Диаграмма последовательности представлена на листе 4 графической части.

**2.3 Концептуальный прототип**

Концептуальный прототип состоит из описания внешнего пользовательского интерфейса, а именно, элементов управления.

При создании данного приложения важную роль играют формы, так как они являются основным средством работы с пользователем. Разрабатываемое приложение будет содержать несколько форм.

Все формы будут содержать стандартные пользовательские элементы управления.

В рабочем режиме программы, пользователю, для удобной навигации, будет предоставлено меню.

Первая форма, которую увидит пользователь будет форма Авторизации, с помощью которой пользователь сможет авторизоваться в приложения. Данная форма будет иметь два текстовых поля для ввода логина пользователя и пароля, а также кнопку «Зарегистрироваться». Кнопка «Зарегистрироваться» будет открывать страницу для регистрации нового аккаунта. Для входа в аккаунт пользователь должен будет заполнить поля верными данными и нажать кнопку «Войти». Макет формы представлен на рисунке 2.2. Макет формы для регистрации аналогичен макету, изображенному на рисунке 2.2.

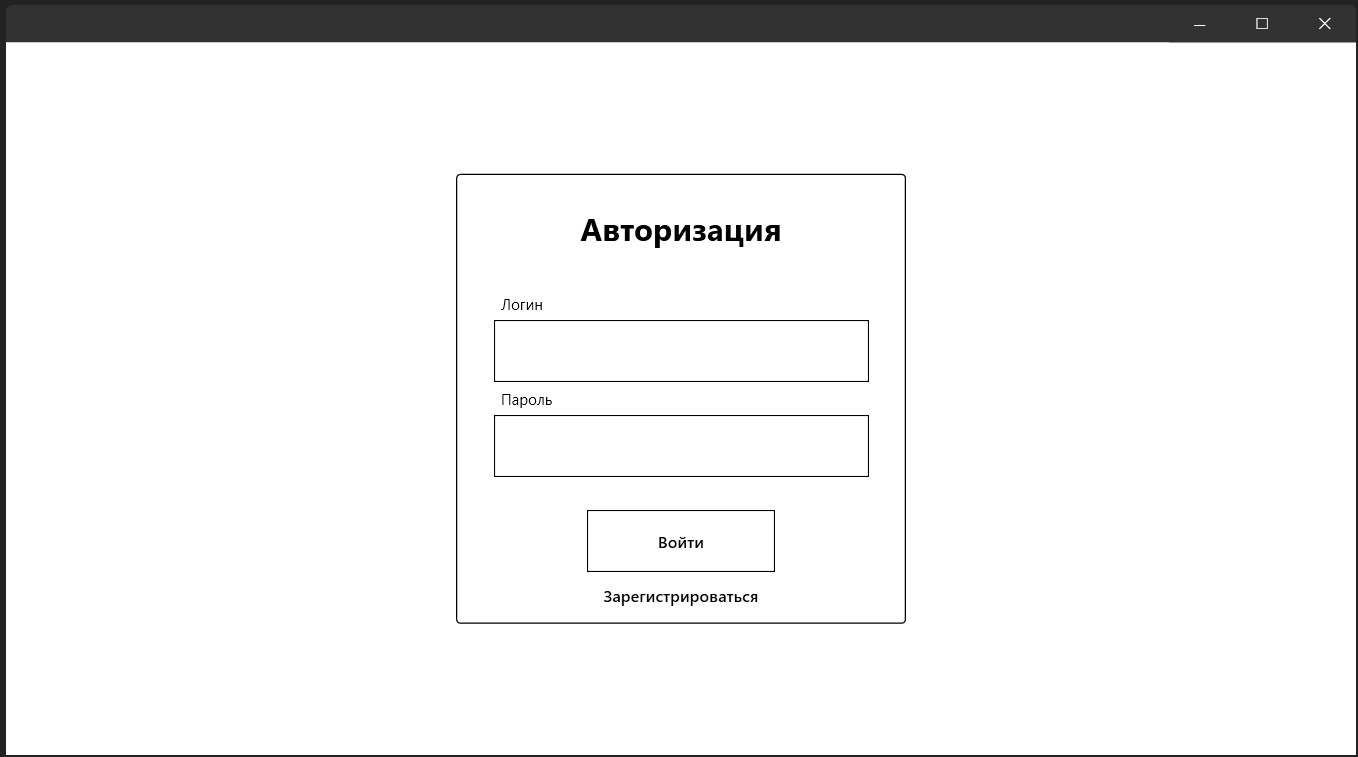


Рисунок 2.2

На форме Авторизации пользователь может авторизироваться как пользователь или администратор. Если введенные пользователем данные будут соответствовать данным администратора, откроется форма «ГлавнаяАдмин», иначе форма «ГлавнаяПользователь». Форма «ГлавнаяПользователь» будет иметь главное меню с двумя вкладками: «Купить билет» и «Мои билеты». Вкладка «Купить билет» будет иметь список существующих маршрутов, которые пользователь может купить или найти интересующий. Форма «Мои билеты» будет иметь список всех приобретенных пользователем билетов, где пользователь сможет распечатать любой купленный билет. Макет формы «Купить билет» предоставлен на рисунке 2.3.

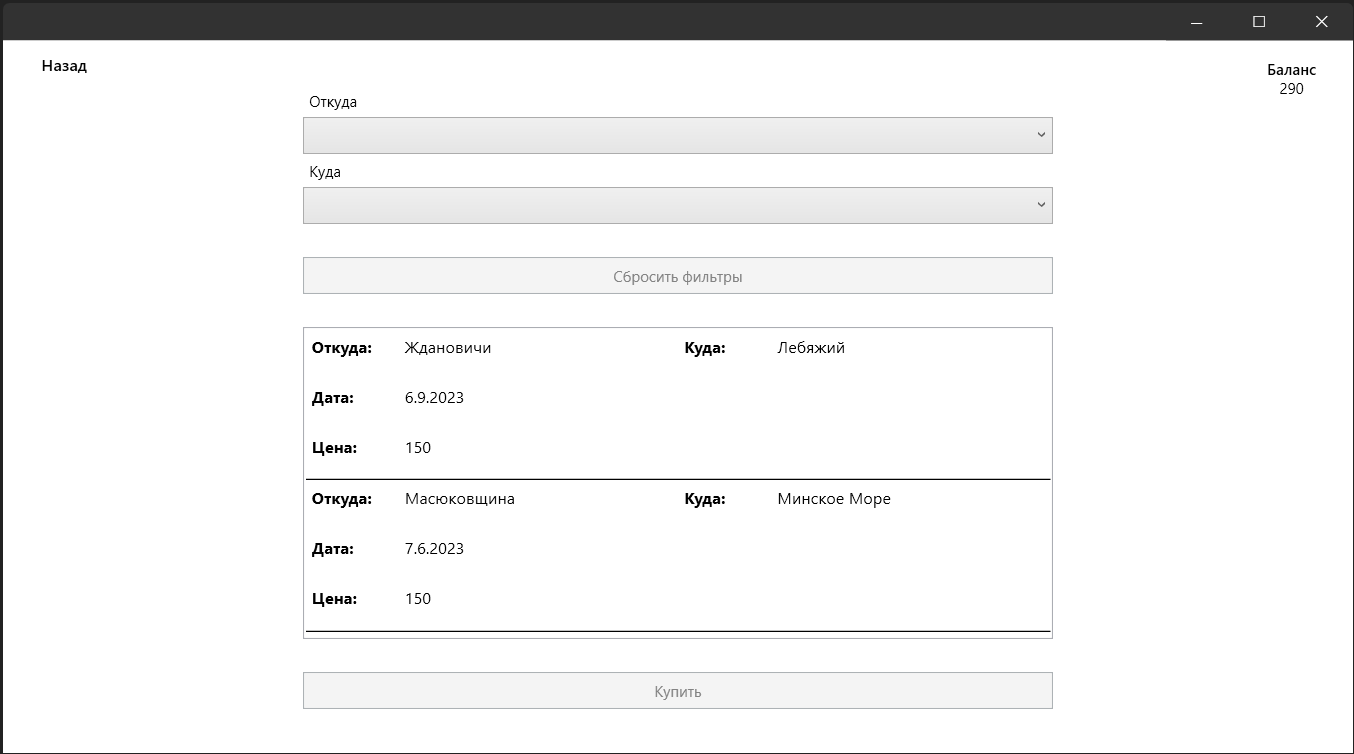


Рисунок 2.3

Макет вкладки «Мои билеты» представлен на рисунке 2.4.

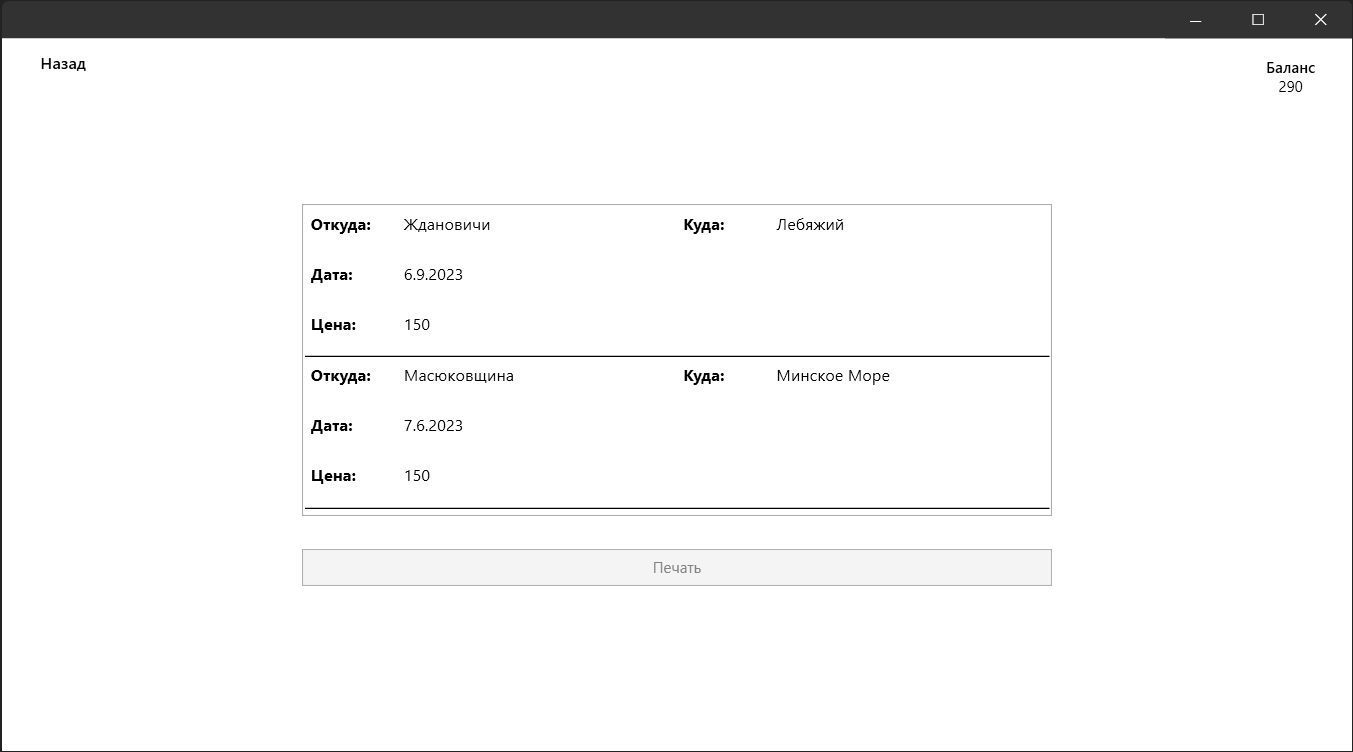


Рисунок 2.4

Форма «ГлавнаяАдмин» будет иметь главное меню с четырьмя вкладками: «Добавить станцию», «Удалить станцию», «Добавить маршрут», «Редактировать маршрут». Вкладка «Добавить станцию» будет иметь поле для ввода названия станции и кнопку добавления. Макет формы «Добавить станцию» представлен на рисунке 2.5. Форма «Удалить станцию» аналогичная форме «Добавить станцию», только вместо ввода названия – выбор существующей. Макет формы «Добавить маршрут» предоставлен на рисунке 2.6. Форма будет иметь поля для заполнения, такие как: станция откуда, станция куда, дата и цена маршрута. Макет формы «Редактировать маршрут» представлен на рисунке 2.7.

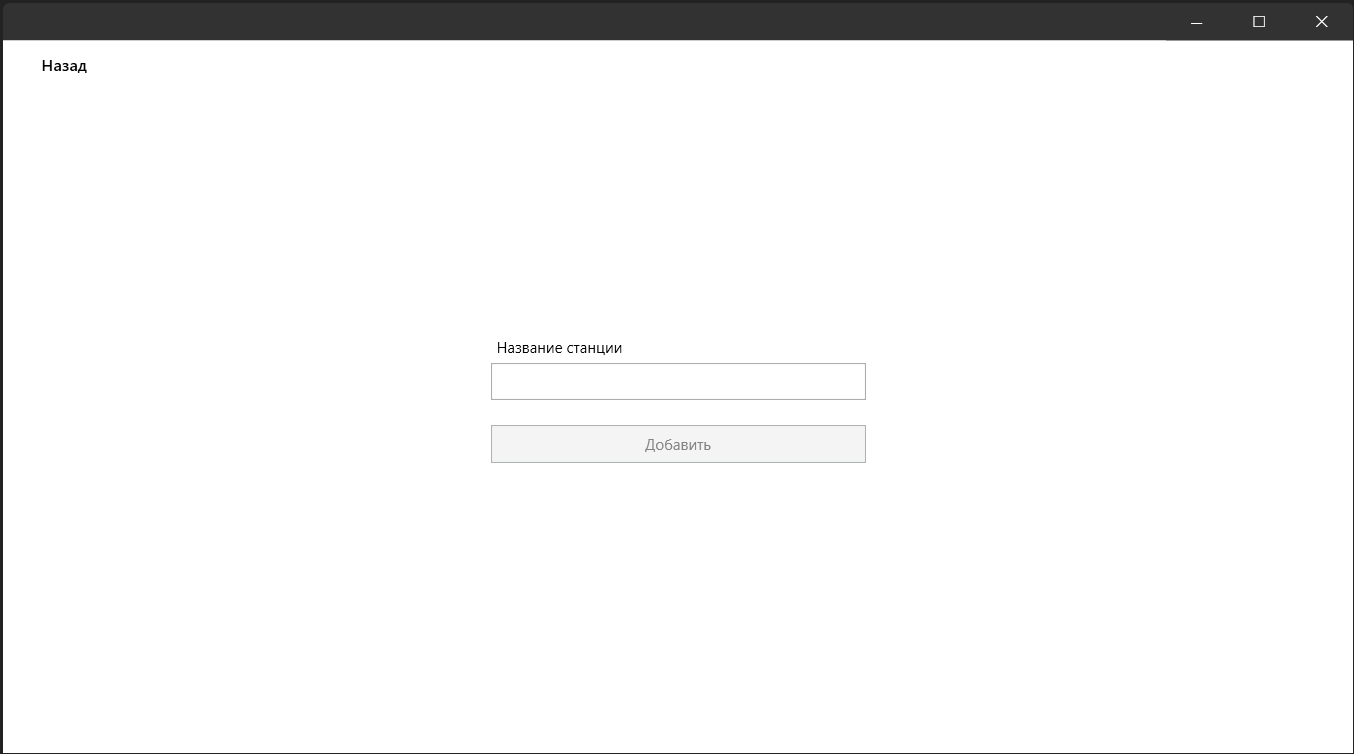


Рисунок 2.5

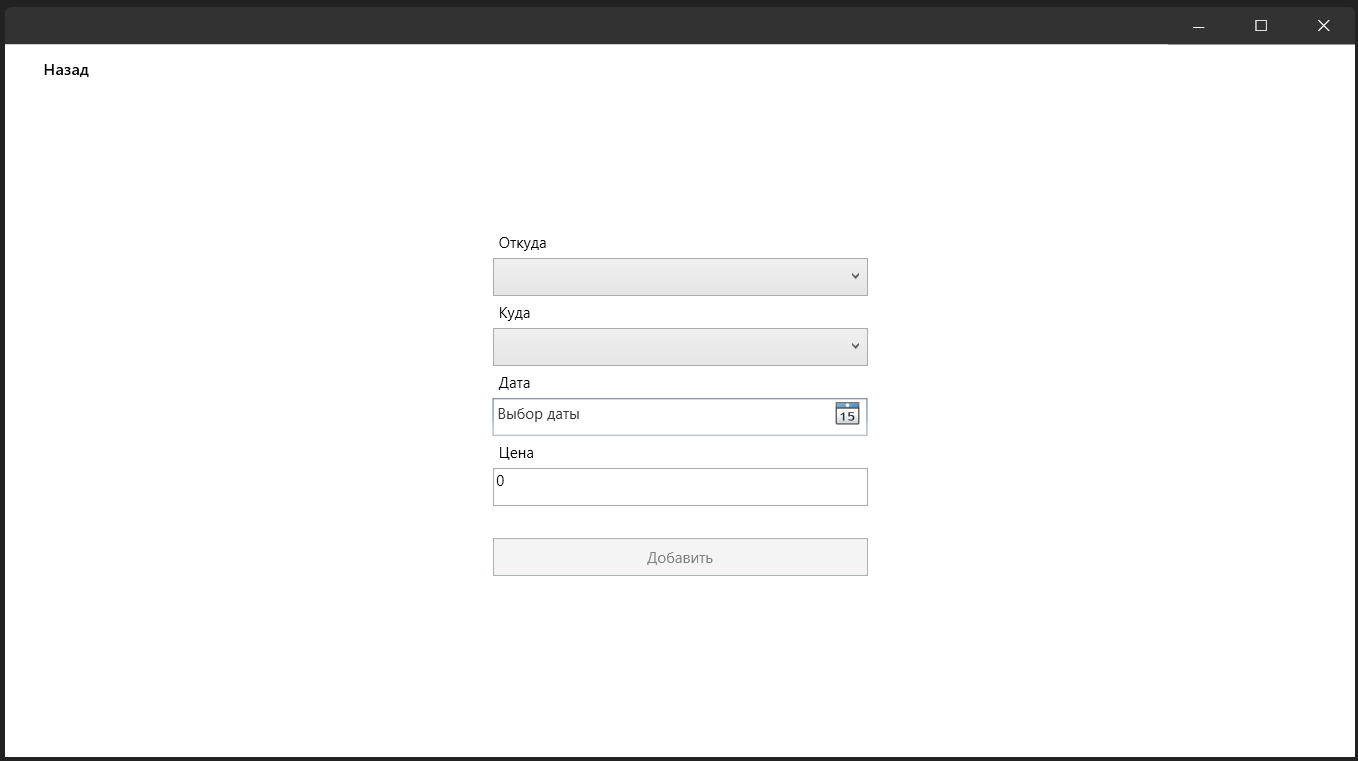


Рисунок 2.6

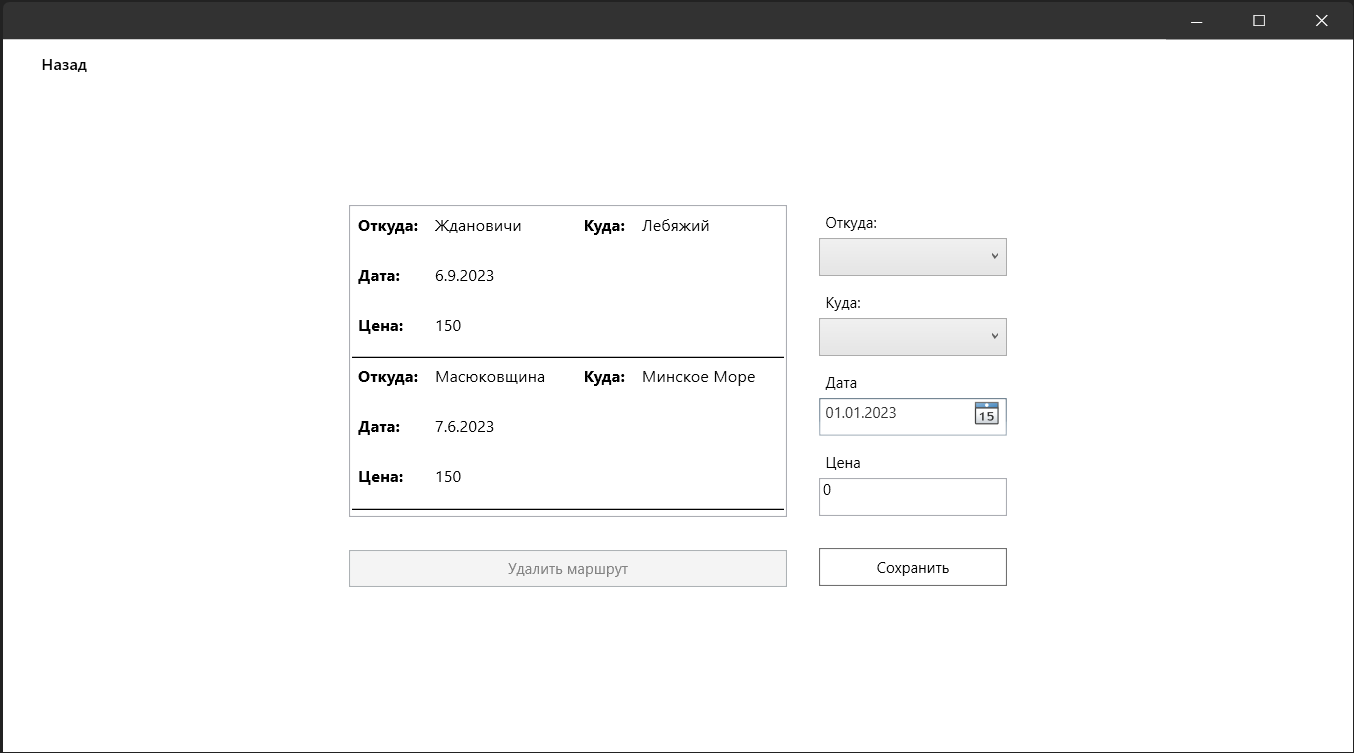


Рисунок 2.7

**3 Описание реализации программного средства**

**3.1 Инструменты разработки и применяемые технологии**

Инструментами для разработки данного программного средства будут являться:

* операционная система Windows 10;
* среда программирования Visual Studio;
* язык программирования C#.
* база данных MS SQL;
* менеджер базами данных SQL Server Management Studio;
* система контроля версий GitHub;
* средство для разработки диаграмм draw.io.

Для разработки программного средства выбрана операционная система Windows 10.

Интегрированная среда разработки Visual Studio – это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки [7].

Язык программирования C# – это объектно- и **компонентно-ориентированный** язык программирования. Он предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО [2].

Microsoft SQL Server - это реляционная система управления базами данных (СУБД), разработанная корпорацией Microsoft. Она является клиент-серверной базой данных, в которой движок SQL Server работает как отдельный процесс, а клиенты используются для подключения к SQL Server и взаимодействия с базой данных.

SQL Server может хранить базы данных в нескольких файлах на сервере, а также обеспечивает возможность масштабирования и резервного копирования данных. ACID функции в SQL Server достигаются за счет использования транзакций и журнала транзакций, которые позволяют сохранять целостность данных и восстанавливать их в случае сбоев.

Microsoft SQL Server Management Studio (также известный как SSMS) - это бесплатный инструмент Microsoft для управления базами данных SQL Server. Он имеет русский интерфейс и может быть загружен с официального сайта Microsoft. Этот менеджер баз данных является кроссплатформенным и работает на ОС Windows. MS SQL Management Studio предоставляет широкий спектр функций для управления базами данных SQL Server, включая управление объектами базы данных, создание запросов, управление безопасностью, отладку и многое другое. Он также позволяет подключаться к удаленным базам данных и работать с ними.

GitHub  – крупнейший [веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для [хостинга](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3) [IT-проектов](https://ru.wikipedia.org/wiki/IT) и их совместной разработки. Веб-сервис основан на системе контроля версий [Git](https://ru.wikipedia.org/wiki/Git" \o "Git) и разработан на [Ruby on Rails](https://ru.wikipedia.org/wiki/Ruby_on_Rails) и [Erlang](https://ru.wikipedia.org/wiki/Erlang" \o "Erlang) компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая [SSL](https://ru.wikipedia.org/wiki/SSL)), а для крупных корпоративных проектов предлагаются различные платные тарифные планы [11].

Draw.io — это бесплатный онлайн-сервис для создания диаграмм и схем. С помощью draw.io пользователи могут создавать различные типы диаграмм, такие как блок-схемы, ER-диаграммы, UML-диаграммы, схемы баз данных и многое другое.

**3.2 Порядок авторизации пользователей**

В данной программе существует разделение ролей: администратор и пользователь. Для авторизации как администратор существует единственный логин и пароль, которые знает только администратор. Для авторизации как пользователь необходимо зарегистрировать новый аккаунт или войти в ранее зарегистрированный.

Для администратора будут доступны следующие функции: добавление/удаление станций, добавление/редактирование/удаление маршрутов.

Для пользователя будут доступны следующие функции: поиск, покупка билетов, печать приобретенных билетов, а также функция пополнения баланса пользователя.

**3.3 Организация данных**

Организация данных подразумевает создание модели данных, главными элементами которой являются сущности и их связи.

Реляционная модель основана на математическом понятии отношения, представлением которого является таблица. В реляционной модели отношения используются для хранения информации об объектах, представленных в базе данных. Отношение имеет вид двухмерной таблицы, в которой строки соответствуют записям, а столбцы – атрибутам. Каждая запись должна однозначно характеризоваться в таблице. Для этого используют первичные и внешние ключи. Достоинством реляционной модели является простота и удобство физической реализации.

Структура базы данных разрабатываемого программного средства включает шесть таблиц. Описание таблиц приводится в таблицах 1-5.

Таблица «Users» хранит информацию о пользователях, структура которой представлена в таблице 3.1.

Таблица 3.1 – Структура таблицы «Users»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля, байт | Описание поля |
| id | int | 4 | Идентификатор пользователя |
| name | nvarchar | 50 | Уникальное имя пользователя |
| password | nvarchar | 50 | Пароль пользователя |
| email | nvarchar | 50 | Электронная почта пользователя |
| walletBalance | int | 4 | Баланс пользователя |

Таблица «Stations» хранит информацию о станциях, структура которой представлена в таблице 3.2.

Таблица 3.2 – Структура таблицы «Stations»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля, байт | Описание поля |
| id | int | 4 | Идентификатор станции |
| name | nvarchar | 50 | Название станции |

Таблица «Tickets» хранит информацию о билетах. Структура таблицы представлена в таблице 3.3.

Таблица 3.3 – Структура таблицы «Tickets»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля, байт | Описание поля |
| id | int | 4 | Идентификатор билета |
| userId | int | 4 | Внешний ключ, ссылающийся на идентификатор пользователя |
| routeId | int | 4 | Внешний ключ, ссылающийся на идентификатор маршрута |

Таблица «Routes» хранит информацию о маршрутах, структура таблицы представлена в таблице 3.4.

Таблица 3.4 – Структура таблицы «Routes»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля, байт | Описание поля |
| id | int | 4 | Идентификатор маршрута |
| fromStation | nvarchar | 50 | Внешний ключ, ссылающийся на название станции |
| toStation | nvarchar | 50 | Внешний ключ, ссылающийся на название станции |
| date | datetime | 4 | Дата отправки электропоезда |
| price | int | 4 | Цена билета за маршрут |

Схема базы данных представлена в виде ERD диаграммы в нотации Баркера на рисунке 3.1.

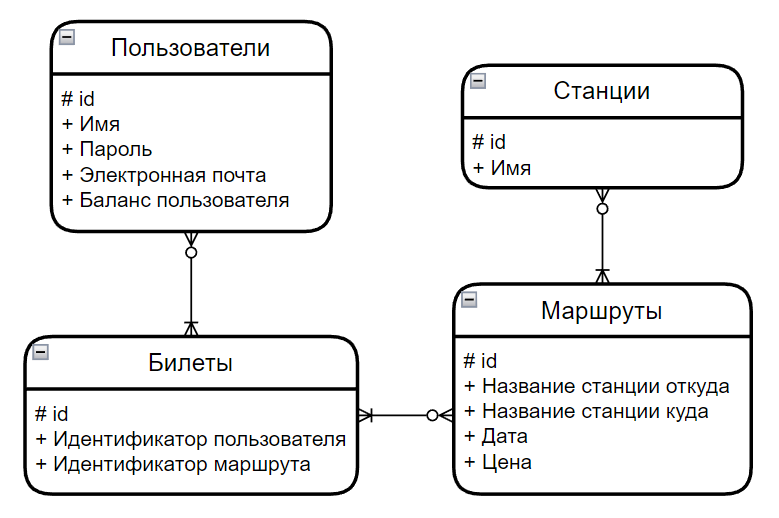


Рисунок 3.1

**3.4 Функции: логическая и физическая организация**

На основании диаграммы вариантов использования в программном средстве реализованы функции авторизации и регистрации.

Так как модель приложения имеет только клиентскую структуру, код функций будет приводится с клиента.

Рассмотрим данные функции программы.

Для того, чтобы авторизоваться в приложении, пользователю необходимо ввести имя пользователя и пароль. Предварительно пользователь должен быть зарегистрирован в системе. После нажатия кнопки «Войти» производится обработка данных. Если в результате проверки данные не были найдены в базе данных, то вход в приложение отменяется и на клиенте отображается сообщение об ошибке.

Код функции авторизации на клиенте предоставлен ниже.

private void ExecuteSignInCommand(object obj)

{

var admin = JsonConvert.DeserializeObject<User>(File.ReadAllText("admin.json"))!;

var user = \_context.Users.FirstOrDefault(i => i.Name == Name && i.Password == Password)!;

if (user == null)

{

ErrorMessage = "Пароль или логин некорректный";

}

else if (user.Name == admin.Name && user.Password == admin.Password && user.Email == admin.Email && user.Id == admin.Id)

{

NavigationService.NavigateTo<AdminHomeViewModel>(true);

}

else if (user != null)

{

using (StreamWriter file = File.CreateText("user.json"))

{

JsonSerializer serializer = new JsonSerializer()

{

Formatting = Formatting.Indented

};

serializer.Serialize(file, user);

}

NavigationService.NavigateTo<HomeViewModel>(true);

}

}

Для регистрации, пользователю необходимо перейти на форму регистрации, нажав кнопку «Зарегистрироваться». Далее пользователь должен заполнить все текстовые поля. Для успешной регистрации пользователю должен ввести уникальное имя пользователя, электронный адрес, пароль. После нажатия кнопки «Зарегистрироваться», начинается обработка данных.

Код функции регистрации на клиенте представлен ниже.

private bool CanExecuteSignUpCommand(object obj)

{

if (string.IsNullOrWhiteSpace(Email) || string.IsNullOrWhiteSpace(Password) ||

Password.Length < 8 || !Email.Contains("@") || !Email.Contains(".") || Email.Length == 1)

{

return false;

}

return true;

}

private void ExecuteSignUpCommand(object obj)

{

var user = new User

{

Name = Name,

Email = Email,

Password = Password,

};

\_context.Users.Add(user);

\_context.SaveChanges();

NavigationService.NavigateTo<SignInViewModel>(true);

}

Полный текст программы представлен в приложении А.

**3.5 Входные и выходные данные**

Входными являются данные, введённые пользователем в таблицы базы данных: «Users», «Routes», «Stations».

Входными данным при добавлении данных о новом пользователе будут являться:

* имя пользователя;
* пароль;
* адрес электронной почты.

Входными данными при добавлении данных о новой станции являются:

* название станции.

Входными данными при добавлении маршрутов являются:

* Название станции откуда;
* Название станции куда;
* Дата отправки;
* Цена билета.

Вся вносимая в программное средство информация хранится в базе данных.

Выходные данные – это Microsoft Word документ. Пример выходных данных представлен на рисунке Б.1.

**3.6 Функциональное тестирование**

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям и проводится для выявления неполадок и недочетов программы на этапе ее сдачи в эксплуатацию.

Проведем тестирование проверки каждого пункта меню с целью проверки всех функций.

Тестирование программы будет производиться последовательно, переходя из одной части программы в другую. Во время теста будут проверяться все действия с программой, навигация пунктам меню, которые может произвести пользователь.

В таблице 3.7 представлены тест-кейсы, подготовленные для проведения функционального тестирования.

Таблица 3.7 – Тест-кейсы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Модуль / Функция | Шаги выполнения | Ожидаемый результат | Фактический результат |
| 1 | Регистрация пользователя | 1. Перейти на вкладку «Регистрация».  2. В поле «Логин» ввести строку «Hinary».  3. В поле «Почта» ввести строку «yourmail@gmail.com»  4. В поле «Пароль» ввести строку «passs123».  5. Нажать кнопку «Зарегистрироваться». | Пользователь регистрируется, данные добавляются в базу данных, открывается форма авторизации. | Пользователь зарегистрировался, данные занеслись в базу данных. Открылась форма авторизации Результат представлен на рисунке В.1 |
| 2 | Авторизация  пользователя | 1. Перейти на вкладку «Авторизация».  2. В поле «Логин» ввести строку «Hinary».  3. В поле «Пароль» ввести строку «passs123».  4. Нажать кнопку «Войти». | Данные о пользователе записываются в json файл. Открывается главная форма. | Данные о пользователе записались в json файл. Открылась главная форма. Результат представлен на рисунке В.2 |

Продолжение таблицы 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Модуль / Функция | Шаги выполнения | Ожидаемый результат | Фактический результат |
| 3 | Покупка билета | 1. Войти в аккаунт.  2. Перейти на вкладку «Купить билет».  3. В списке выбрать нужный маршрут.  4. Нажать на кнопку «Купить». | Со счета списывается сумма, указанная в маршруте. Билет добавляется в список купленных. | Со счета списалась сумма. Билет добавился в список купленных.  Результат представлен на рисунке В.3 |
| 4 | Печать билета | 1. Войти в аккаунт.  2. Перейти на вкладку «Мои билеты».  3. Выбрать билет.  4. Нажать на кнопку «Печать». | Формируется Word файл, в котором указана информация о билете. | Сформировался Word файл, в котором указана информация о билете. Результат представлен на рисунке В.4 |
| 6 | Выход из аккаунта | 1. Войти в аккаунт.  2. Перейти на вкладку «Главная».  3. Нажать кнопку «Выйти». | Главная форма сменяется формой входа. | Главная форма сменилась формой входа. Результат представлен на рисунке В.6 |

**3.7 Описание справочной системы**

Для эффективной работы пользователя с приложением необходимо обеспечить его качественной справочной системой, в которой должны быть приведены методы и приемы работы с приложением, включающие данные о том, что произойдет после нажатия на определенную кнопку или при выборе пункта меню, сведения о том, какую информацию и в каком виде следует вводить в соответствующие поля, каким образом можно работать.

Справочная система будет представлена в виде встроенного файла в приложении, содержащая информацию по эксплуатации программного средства. Файл будет разработан в программе Dr. Explain.

Система справки данного программного средства будет содержать раздел «Руководство по использованию». Он будет подразделяться на два подраздела:

«Покупка билетов»;

«Печать билетов».

Структура справочной системы приведена на рисунке 3.2.

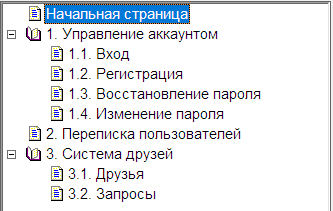


Рисунок 3.2

**4** **Применение**

**4.1 Назначение программного средства**

Программное средство для покупки билетов на электропоезда разрабатывалось с целью создания приложения, с помощью которого люди смогут покупать билеты на электропоезда дистанционно. В приложении присутствует поиск и покупка билетов, а также печать купленных билетов. Данная программа была разработана на ноутбуке Honor MagicBook Pro 16 с операционной системой Windows 10 Home, со следующей конфигурацией:

* процессор AMD Ryzen 5 4600H;
* интегрированная видеокарта Radeon Vega Mobile Gfx 2.30 GHz;
* ОЗУ 16,00 ГБ.

Программа создана в средстве разработки Microsoft Visual Studio 2022 на языке программирования С#. Она может работать в средах операционных систем семейства Microsoft Windows, начиная с Windows 7.

Для работы программы, необходима база данных «TrainTickets.db». Программа не требовательна к системным ресурсам и не имеет больших массивов данных, также проста в использовании и не требует специальных навыков при работе. Для работы данного программного средства необходима предварительная установка и настройка следующих программных продуктов:

* платформа Microsoft Net Framework версии 4.5.

В состав программы входят:

* программа «TrainTickets.exe»;
* база данных «TrainTickets.db»;
* файл справочной системы «help.chm».

Объем установленного приложения не превышает 140 МБ, файлов – 60 МБ.

Приложение «TrainTickets.exe предназначено для покупки билетов. Программное средство предоставляет пользователю возможности для пополнения баланса, поиска маршрутов, покупки билетов, а также печати купленных билетов.

**4.2 Условия применения**

Для работы с программой необходимо наличие программного обеспечения:

* операционная система, начиная с Windows 7;
* библиотека Microsoft .NET Framework 4.5.

На случай редактирования проекта программы необходимо наличие программного обеспечения:

* менеджер базами данных SQL Server Management Studio.

**Заключение**

В рамках курсового проекта на тему «Приложения для покупки билетов на электропоезда» было разработано программное средство «TrainTickets.exe».

Основными функциями приложения являются:

* ведение базы данных на клиенте, в которой будет храниться вся информация;
* авторизация пользователей;
* регистрация пользователей;
* поиск билетов;
* покупка билетов;
* печать купленных билетов.

Для достижения целей курсового проекта были решены следующие задачи:

* изучена предметная область;
* разработана физическая и логическая модель данных;
* разработано программное средство;
* описана область применения созданного программного средства.

Разработка имеет интуитивно понятный графический интерфейс, позволяющий даже с минимальным знанием компьютера использовать данное программного средство.

Программа реализована в полном объеме и в соответствии с заданными требованиями, полностью отлажена и протестирована. Поставленные задачи выполнены.

К достоинствам программного средства можно отнести простоту использования, малые массивы данных. К недостаткам можно отнести …

В процессе разработки данного программного средства были применены и закреплены знания по уже изученному материалу, были отработаны навыки владения методами надежного программирования и эффективности разработки программного обеспечения в Microsoft Visual Studio 2022 с использованием языка программирования C#, разработана база данных средствами системы управления базами данных SQL Server Management Studio.

**Список использованных источников**

1. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск : КБП, 2023.
2. Михалевич В.Ю. Методические указания к курсовому проектированию по учебному предмету «Конструирование программ и языки программирования» / В.Ю.Михалевич. – Минск : КБП, 2023.
3. Программа и методика испытаний. Требования к содержанию, оформлению и контролю качества : ГОСТ 19.301-2000. – Введ. 01.09.2001. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 14 с
4. .NET Framework [Электронный ресурс]. – Microsoft, 2020. – Режим доступа : https://docs.microsoft.com/en-us/dotnet/framework/. – Дата доступа : 01.09.2020.
5. Dr.Explain [Электронный ресурс]. – Dr.Explain, 2021. – Режим доступа : https://www.drexplain.ru/. – Дата доступа : 04.05.2021.
6. MS Office [Электронный ресурс]. – Microsoft, 2021. – Режим доступа : https://officeapplications.net/microsoft-word/. – Дата доступа : 03.05.2021.
7. Smart Install Maker [Электронный ресурс]. – InstallBuilders, 2020. – Режим доступа : https://www.ixbt.com/news/soft/index.shtml?10/52/79/. – Дата доступа : 05.05.2021.
8. Diagrams [Электронный ресурс]. – UML, 2021. – Режим доступа : https://www.diagrams.net/. – Дата доступа : 10.05.2021.
9. Visual Studio [Электронный ресурс]. – Microsoft, 2020. – Режим доступа : https://visualstudio.microsoft.com/vs/. – Дата доступа : 01.09.2020.

**Приложение А**

(обязательное)

**Текст программных модулей**

Код файла TicketPurchaseViewModel.cs

using Microsoft.EntityFrameworkCore.ChangeTracking;

using Microsoft.IdentityModel.Tokens;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using TrainTickets.Interfaces;

using TrainTickets.Model;

using TrainTickets.Persistence;

namespace TrainTickets.ViewModel

{

class TicketPurchaseViewModel : ViewModelBase

{

private ApplicationDbContext \_context;

private INavigationService \_navigationService;

private string \_fromStation;

private string \_toStation;

private List<string> \_stations;

private List<Route> \_routes;

private Route \_selectedRoute;

private int \_balance;

private User \_user;

public Route SelectedRoute

{

get => \_selectedRoute;

set

{

\_selectedRoute = value;

OnPropertyChanged();

}

}

public List<string> Stations

{

get => \_stations;

set

{

\_stations = value;

OnPropertyChanged(nameof(Stations));

}

}

public string FromStation

{

get => \_fromStation;

set

{

\_fromStation = value;

if(FromStation != null && ToStation == null)

Routes = \_context.Routes.Where(i => i.FromStation == FromStation).ToList();

else if(FromStation != null && ToStation != null)

Routes = \_context.Routes.Where(i => i.FromStation == FromStation && i.ToStation == ToStation).ToList();

OnPropertyChanged(nameof(FromStation));

}

}

public string ToStation

{

get => \_toStation;

set

{

\_toStation = value;

if (ToStation != null && FromStation == null)

Routes = \_context.Routes.Where(i => i.ToStation == ToStation).ToList();

else if (ToStation != null && FromStation != null)

Routes = \_context.Routes.Where(i => i.FromStation == FromStation && i.ToStation == ToStation).ToList();

OnPropertyChanged(nameof(ToStation));

}

}

public INavigationService NavigationService

{

get => \_navigationService;

set

{

\_navigationService = value;

OnPropertyChanged();

}

}

public ICommand NavigationToHomePageCommand { get; }

public ICommand ResetFiltersCommand { get; }

public ICommand BuyTicketCommand { get; }

public List<Route> Routes

{

get => \_routes;

set

{

\_routes = value;

OnPropertyChanged();

}

}

public int Balance

{

get => \_balance;

set

{

\_balance = value;

OnPropertyChanged();

}

}

public User { get => \_user; set => \_user = value; }

public TicketPurchaseViewModel(ApplicationDbContext context, INavigationService navigationService)

{

\_context = context;

\_navigationService = navigationService;

NavigationToHomePageCommand = new ViewModelCommand(i => NavigationService.NavigateTo<HomeViewModel>(true));

ResetFiltersCommand = new ViewModelCommand(ExecuteResetFiltersCommand, CanExecuteResetFiltersCommand);

BuyTicketCommand = new ViewModelCommand(ExecuteBuyTicketCommand, CanExecuteBuyTicketCommand);

Stations = \_context.Stations.Select(i => i.Name).ToList();

Stations.Sort();

Routes = \_context.Routes.ToList();

Routes.OrderByDescending(i => i.Date);

var user = JsonConvert.DeserializeObject<User>(File.ReadAllText("user.json"))!;

User = \_context.Users.FirstOrDefault(i => i.Id == user.Id)!;

Balance = User.WalletBalance;

//SelectedRoute = Routes[0];

}

private bool CanExecuteBuyTicketCommand(object obj)

{

if (SelectedRoute == null)

return false;

return (!Routes.IsNullOrEmpty() && Balance >= SelectedRoute.Price);

}

private void ExecuteBuyTicketCommand(object obj)

{

User.WalletBalance -= SelectedRoute.Price;

\_context.Users.Update(User);

Balance = User.WalletBalance;

var ticket = new Ticket

{

User = User,

Route = SelectedRoute

};

\_context.Tickets.Add(ticket);

\_context.SaveChanges();

MessageBox.Show("Билет успешно куплен, со счета списано " + SelectedRoute.Price + " рублив");

}

private bool CanExecuteResetFiltersCommand(object obj)

{

return !string.IsNullOrEmpty(ToStation)

|| !string.IsNullOrEmpty(FromStation);

}

private void ExecuteResetFiltersCommand(object obj)

{

Routes = \_context.Routes.ToList();

SelectedRoute = null;

FromStation = null;

ToStation = null;

//if(!Routes.IsNullOrEmpty())

// SelectedRoute = Routes[0];

}

}

}

Код файла UserTicketsViewModel.cs

using Microsoft.Office.Interop.Word;

using Microsoft.Win32;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading.Tasks.Dataflow;

using System.Windows.Input;

using TrainTickets.Interfaces;

using TrainTickets.Model;

using TrainTickets.Persistence;

using static System.Runtime.InteropServices.JavaScript.JSType;

namespace TrainTickets.ViewModel

{

public class UserTicketsViewModel : ViewModelBase

{

private ApplicationDbContext \_context;

private INavigationService \_navigationService;

private string \_fromStation;

private string \_toStation;

private Route \_selectedRoute;

private int \_balance;

private List<Route> \_routes;

private List<Ticket> \_tickets;

public string FromStation

{

get => \_fromStation;

set

{

\_fromStation = value;

OnPropertyChanged(nameof(FromStation));

}

}

public string ToStation

{

get => \_toStation;

set

{

\_toStation = value;

OnPropertyChanged(nameof(ToStation));

}

}

public Route SelectedRoute

{

get => \_selectedRoute;

set

{

\_selectedRoute = value;

OnPropertyChanged();

}

}

public int Balance

{

get => \_balance;

set

{

\_balance = value;

OnPropertyChanged(nameof(Balance));

}

}

public INavigationService NavigationService

{

get => \_navigationService;

set

{

\_navigationService = value;

OnPropertyChanged();

}

}

public ICommand NavigationToHomePageCommand { get; }

public ICommand PrintTicketCommand { get; }

public List<Route> Routes { get => \_routes; set => \_routes = value; }

public List<Ticket> Tickets { get => \_tickets; set => \_tickets = value; }

public UserTicketsViewModel(ApplicationDbContext context, INavigationService navigationService)

{

\_context = context;

\_navigationService = navigationService;

NavigationToHomePageCommand = new ViewModelCommand(i => NavigationService.NavigateTo<HomeViewModel>(true));

PrintTicketCommand = new ViewModelCommand(ExecutePrintTicketCommand, CanExecutePrintTicketCommand);

var user = JsonConvert.DeserializeObject<User>(File.ReadAllText("user.json"))!;

user = \_context.Users.FirstOrDefault(i => i.Id == user.Id)!;

Balance = user.WalletBalance;

var r = \_context.Routes.ToList();

Tickets = \_context.Tickets.Where(i => i.User.Id == user.Id).ToList();

Routes = Tickets.Select(i => i.Route).ToList();

}

private bool CanExecutePrintTicketCommand(object obj)

{

return SelectedRoute != null;

}

private void ExecutePrintTicketCommand(object obj)

{

var saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Документ Word (\*.docx)|\*.docx";

saveFileDialog.Title = "Сохранить документ Word";

saveFileDialog.FileName = SelectedRoute.FromStation + "-" + SelectedRoute.ToStation + " " + SelectedRoute.Date.ToShortDateString();

if (saveFileDialog.FileName != "" && saveFileDialog.ShowDialog() == true)

{

Application word = new Application();

Document doc = word.Documents.Add();

Border = doc.Content.Borders[WdBorderType.wdBorderTop];

border.LineStyle = WdLineStyle.wdLineStyleSingle;

border.Color = WdColor.wdColorBlack;

border.LineWidth = WdLineWidth.wdLineWidth225pt;

Microsoft.Office.Interop.Word.Range range = doc.Content;

range.Text = "\*\*\*";

range.Font.Size = 24;

range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphCenter;

range.InsertParagraphAfter();

Paragraph title = doc.Content.Paragraphs.Add();

title.Range.Text = "Билет на электричку";

title.Range.Font.Size = 18;

title.Range.Font.Bold = 1;

title.Range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphCenter;

title.Range.InsertParagraphAfter();

Paragraph fromStation = doc.Content.Paragraphs.Add();

fromStation.Range.Text = "Станция отправления: \t.\t.\t.\t.\t.\t.\t.\t" + SelectedRoute.FromStation;

fromStation.Range.Font.Size = 14;

fromStation.Range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphLeft;

fromStation.Range.InsertParagraphAfter();

Paragraph toStation = doc.Content.Paragraphs.Add();

toStation.Range.Text = "Станция прибытия: \t.\t.\t.\t.\t.\t.\t.\t" + SelectedRoute.ToStation;

toStation.Range.Font.Size = 14;

toStation.Range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphLeft;

toStation.Range.InsertParagraphAfter();

Paragraph date = doc.Content.Paragraphs.Add();

date.Range.Text = "Дата и время отправления: \t.\t.\t.\t.\t.\t.\t" + SelectedRoute.Date.ToShortDateString();

date.Range.Font.Size = 14;

date.Range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphLeft;

date.Range.InsertParagraphAfter();

Paragraph price = doc.Content.Paragraphs.Add();

price.Range.Text = "Цена билета: \t.\t.\t.\t.\t.\t.\t.\t.\t" + SelectedRoute.Price;

price.Range.Font.Size = 14;

price.Range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphLeft;

price.Range.InsertParagraphAfter();

doc.SaveAs2(saveFileDialog.FileName);

doc.Close();

word.Quit();

}

}

}

}

Код файла StationAddingViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

using System.Windows;

using TrainTickets.Interfaces;

using TrainTickets.Model;

using TrainTickets.Persistence;

namespace TrainTickets.ViewModel

{

class StationAddingViewModel : ViewModelBase

{

private ApplicationDbContext \_context;

private INavigationService \_navigationService;

private string \_name;

public INavigationService NavigationService

{

get => \_navigationService;

set

{

\_navigationService = value;

OnPropertyChanged();

}

}

public ICommand AddStationCommand { get; set; }

public ICommand NavigateToAdminHomePageCommand { get; }

public string Name

{

get => \_name;

set

{

\_name = value;

OnPropertyChanged(nameof(Name));

}

}

public StationAddingViewModel(ApplicationDbContext context, INavigationService navigationService)

{

\_context = context;

\_navigationService = navigationService;

NavigateToAdminHomePageCommand = new ViewModelCommand(i => NavigationService.NavigateTo<AdminHomeViewModel>(true));

AddStationCommand = new ViewModelCommand(ExecuteAddStationCommand, CanExecuteAddStationCommand);

}

private bool CanExecuteAddStationCommand(object obj)

{

return !string.IsNullOrEmpty(Name);

}

private void ExecuteAddStationCommand(object obj)

{

var station = new Station

{

Name = Name,

};

\_context.Stations.Add(station);

\_context.SaveChanges();

MessageBox.Show("Станция успешно добавлена");

Name = "";

}

}

}

Код файла StationAddingViewModel.cs

using Microsoft.EntityFrameworkCore.Metadata.Internal;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using TrainTickets.Persistence;

using TrainTickets.Interfaces;

using System.Windows.Input;

using TrainTickets.Model;

using System.Windows;

using System.Collections.ObjectModel;

using System.Net.WebSockets;

using System.Runtime.CompilerServices;

using TrainTickets.Services;

namespace TrainTickets.ViewModel

{

class RouteAddingViewModel : ViewModelBase

{

private ApplicationDbContext \_context;

private INavigationService \_navigationService;

private string \_fromStation;

private string \_toStation;

private DateTime \_date = DateTime.Today;

private int \_price;

private List<string> \_stations;

public List<string> Stations

{

get => \_stations;

set

{

\_stations = value;

OnPropertyChanged(nameof(Stations));

}

}

public INavigationService NavigationService

{

get => \_navigationService;

set

{

\_navigationService = value;

OnPropertyChanged();

}

}

public string FromStation

{

get => \_fromStation;

set

{

\_fromStation = value;

OnPropertyChanged(nameof(FromStation));

}

}

public string ToStation

{

get => \_toStation;

set

{

\_toStation = value;

OnPropertyChanged(nameof(ToStation));

}

}

public DateTime Date

{

get => \_date;

set

{

\_date = value;

OnPropertyChanged(nameof(Date));

}

}

public int Price

{

get => \_price;

set

{

\_price = value;

OnPropertyChanged(nameof(Price));

}

}

public ICommand AddRouteCommand { get; set; }

public ICommand NavigateToAdminHomePageCommand { get; }

public RouteAddingViewModel(ApplicationDbContext context, INavigationService navigationService)

{

\_context = context;

\_navigationService = navigationService;

NavigateToAdminHomePageCommand = new ViewModelCommand(i => NavigationService.NavigateTo<AdminHomeViewModel>(true));

AddRouteCommand = new ViewModelCommand(ExecuteAddRouteCommand, CanExecuteAddRouteCommand);

Stations = \_context.Stations.Select(i => i.Name).ToList();

Stations.Sort();

}

private bool CanExecuteAddRouteCommand(object obj)

{

return !string.IsNullOrEmpty(FromStation)

&& !string.IsNullOrEmpty(ToStation)

&& !string.IsNullOrEmpty(Date.ToString())

&& !string.IsNullOrEmpty(Price.ToString())

&& !(Price == 0)

&& !(FromStation == ToStation);

}

private void ExecuteAddRouteCommand(object obj)

{

var route = new Route

{

FromStation = FromStation,

ToStation = ToStation,

Date = Date,

Price = Price

};

\_context.Routes.Add(route);

\_context.SaveChanges();

MessageBox.Show("Маршрут успешно добавлен");

FromStation = "";

ToStation = "";

Date = DateTime.Today;

Price = 0;

}

}

}

Код файла RouteDeletingViewModel.cs

using Microsoft.IdentityModel.Tokens;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Input;

using TrainTickets.Interfaces;

using TrainTickets.Model;

using TrainTickets.Persistence;

namespace TrainTickets.ViewModel

{

class RouteDeletingViewModel : ViewModelBase

{

private ApplicationDbContext \_context;

private INavigationService \_navigationService;

private Route \_selectedRoute;

private List<Route> \_routes;

private List<string> \_stations;

private string \_fromStation;

private string \_toStation;

private DateTime \_date = new DateTime(2023, 01, 01);

private int \_price;

public Route SelectedRoute

{

get => \_selectedRoute;

set

{

\_selectedRoute = value;

FromStation = \_selectedRoute.FromStation;

ToStation = \_selectedRoute.ToStation;

Date = \_selectedRoute.Date;

Price = \_selectedRoute.Price;

OnPropertyChanged();

}

}

public INavigationService NavigationService

{

get => \_navigationService;

set

{

\_navigationService = value;

OnPropertyChanged();

}

}

public ICommand NavigationToAdminHomePageCommand { get; }

public ICommand DeleteRouteCommand { get; }

public ICommand SaveEditedRouteCommand { get; }

public List<Route> Routes

{

get => \_routes;

set

{

\_routes = value;

OnPropertyChanged(nameof(Routes));

}

}

public List<string> Stations

{

get => \_stations;

set

{

\_stations = value;

OnPropertyChanged(nameof(Stations));

}

}

public string FromStation

{

get => \_fromStation;

set

{

\_fromStation = value;

OnPropertyChanged(nameof(FromStation));

}

}

public string ToStation

{

get => \_toStation;

set

{

\_toStation = value;

OnPropertyChanged(nameof(ToStation));

}

}

public DateTime Date

{

get => \_date;

set

{

\_date = value;

OnPropertyChanged(nameof(Date));

}

}

public int Price

{

get => \_price;

set

{

\_price = value;

OnPropertyChanged(nameof(Price));

}

}

public RouteDeletingViewModel(ApplicationDbContext context, INavigationService navigationService)

{

\_context = context;

\_navigationService = navigationService;

NavigationToAdminHomePageCommand = new ViewModelCommand(i => NavigationService.NavigateTo<AdminHomeViewModel>(true));

DeleteRouteCommand = new ViewModelCommand(ExecuteDeleteRouteCommand, CanExecuteDeleteRouteCommand);

SaveEditedRouteCommand = new ViewModelCommand(ExecuteSaveEditedRouteCommand, CanExecuteSaveEditedRouteCommand);

Routes = \_context.Routes.ToList();

Stations = \_context.Stations.Select(i => i.Name).ToList();

Stations.Sort();

}

private bool CanExecuteSaveEditedRouteCommand(object obj)

{

return true;

}

private void ExecuteSaveEditedRouteCommand(object obj)

{

if(SelectedRoute != null)

{

SelectedRoute.FromStation = FromStation;

SelectedRoute.ToStation = ToStation;

SelectedRoute.Date = Date;

SelectedRoute.Price = Price;

\_context.Update(SelectedRoute);

\_context.SaveChanges();

Routes = \_context.Routes.ToList();

}

}

private bool CanExecuteDeleteRouteCommand(object obj)

{

return SelectedRoute != null;

}

private void ExecuteDeleteRouteCommand(object obj)

{

var tickets = \_context.Tickets.Where(i => i.Route == SelectedRoute).ToList();

foreach (var ticket in tickets)

\_context.Tickets.Remove(ticket);

\_context.SaveChanges();

if (SelectedRoute != null)

\_context.Routes.Remove(SelectedRoute);

\_context.SaveChanges();

Routes = \_context.Routes.ToList();

}

}

}

Код файла StationDeletingViewModel.cs

using Microsoft.EntityFrameworkCore.Query.SqlExpressions;

using Microsoft.Office.Interop.Word;

using Microsoft.Win32;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

using TrainTickets.Interfaces;

using TrainTickets.Model;

using TrainTickets.Persistence;

namespace TrainTickets.ViewModel

{

class StationDeletingViewModel : ViewModelBase

{

private ApplicationDbContext \_context;

private INavigationService \_navigationService;

private string \_selectedStation;

private List<string> \_stations;

private List<Ticket> \_tickets;

public string SelectedStation

{

get => \_selectedStation;

set

{

\_selectedStation = value;

OnPropertyChanged();

}

}

public INavigationService NavigationService

{

get => \_navigationService;

set

{

\_navigationService = value;

OnPropertyChanged();

}

}

public ICommand NavigationToAdminHomePageCommand { get; }

public ICommand DeleteStationCommand { get; }

public List<string> Stations

{

get => \_stations;

set

{

\_stations = value;

OnPropertyChanged(nameof(Stations));

}

}

public List<Ticket> Tickets { get => \_tickets; set => \_tickets = value; }

public StationDeletingViewModel(ApplicationDbContext context, INavigationService navigationService)

{

\_context = context;

\_navigationService = navigationService;

NavigationToAdminHomePageCommand = new ViewModelCommand(i => NavigationService.NavigateTo<AdminHomeViewModel>(true));

DeleteStationCommand = new ViewModelCommand(ExecuteDeleteStationCommand, CanExecuteDeleteStationCommand);

Stations = \_context.Stations.Select(i => i.Name).ToList();

Stations.Sort();

}

private bool CanExecuteDeleteStationCommand(object obj)

{

return SelectedStation != null;

}

private void ExecuteDeleteStationCommand(object obj)

{

var selStation = \_context.Stations.FirstOrDefault(i => i.Name == SelectedStation)!;

var routes = \_context.Routes.Where(i => i.FromStation == SelectedStation || i.ToStation == SelectedStation).ToList();

var tickets = \_context.Tickets.Where(i => i.Route.ToStation == SelectedStation || i.Route.FromStation == SelectedStation).ToList();

foreach (var ticket in tickets)

\_context.Tickets.Remove(ticket);

\_context.SaveChanges();

foreach (var route in routes)

\_context.Routes.Remove(route);

\_context.SaveChanges();

\_context.Stations.Remove(selStation);

\_context.SaveChanges();

Stations = \_context.Stations.Select(i => i.Name).ToList();

Stations.Sort();

}

}

}

**Приложение Б**

(обязательное)

**Формы выходных документов**

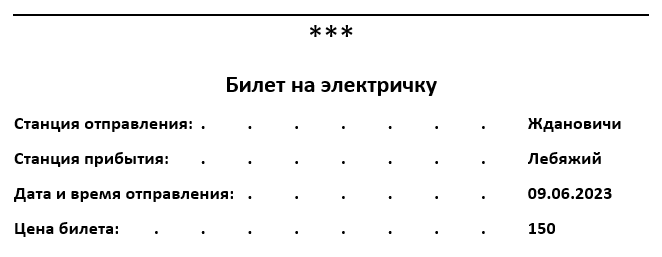


Рисунок Б.1 – Распечатанный билет

**Приложение В**

(обязательное)

**Результаты работы программы**

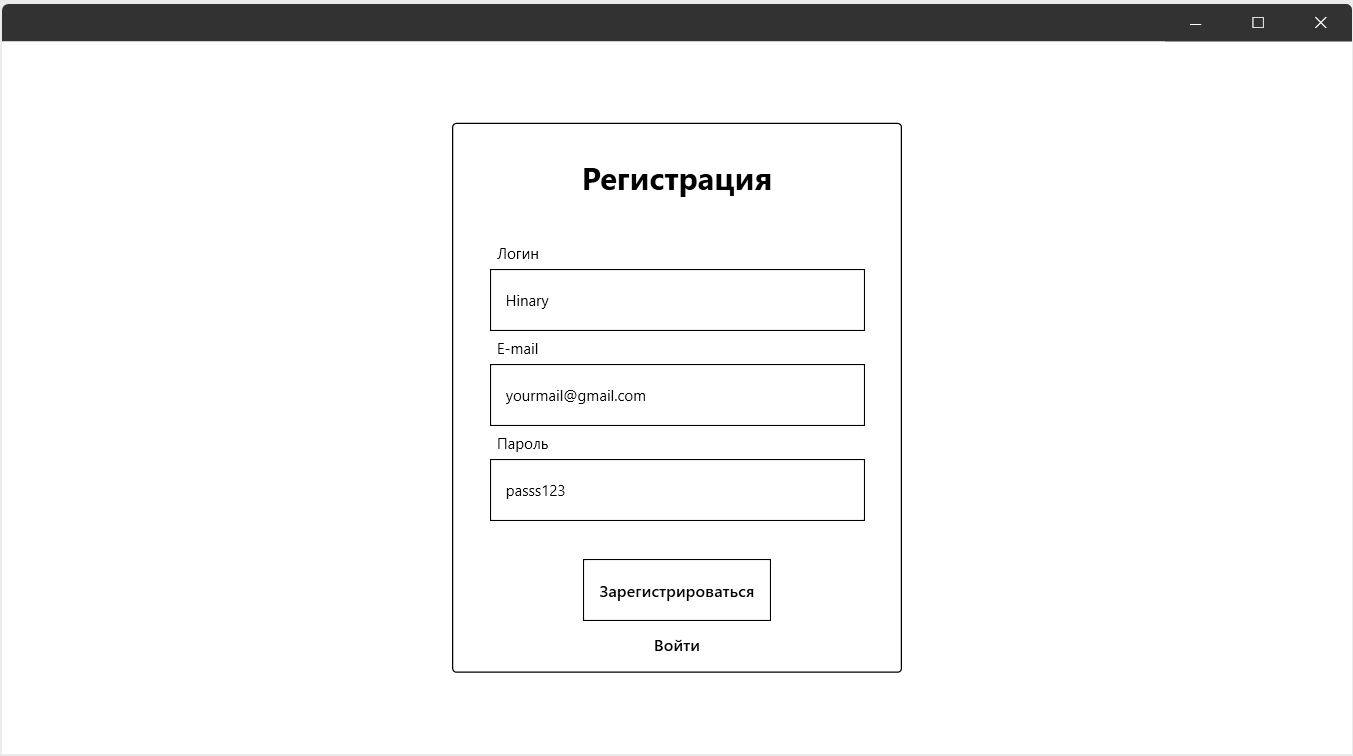
****

Рисунок В.1 – Регистрация пользователя

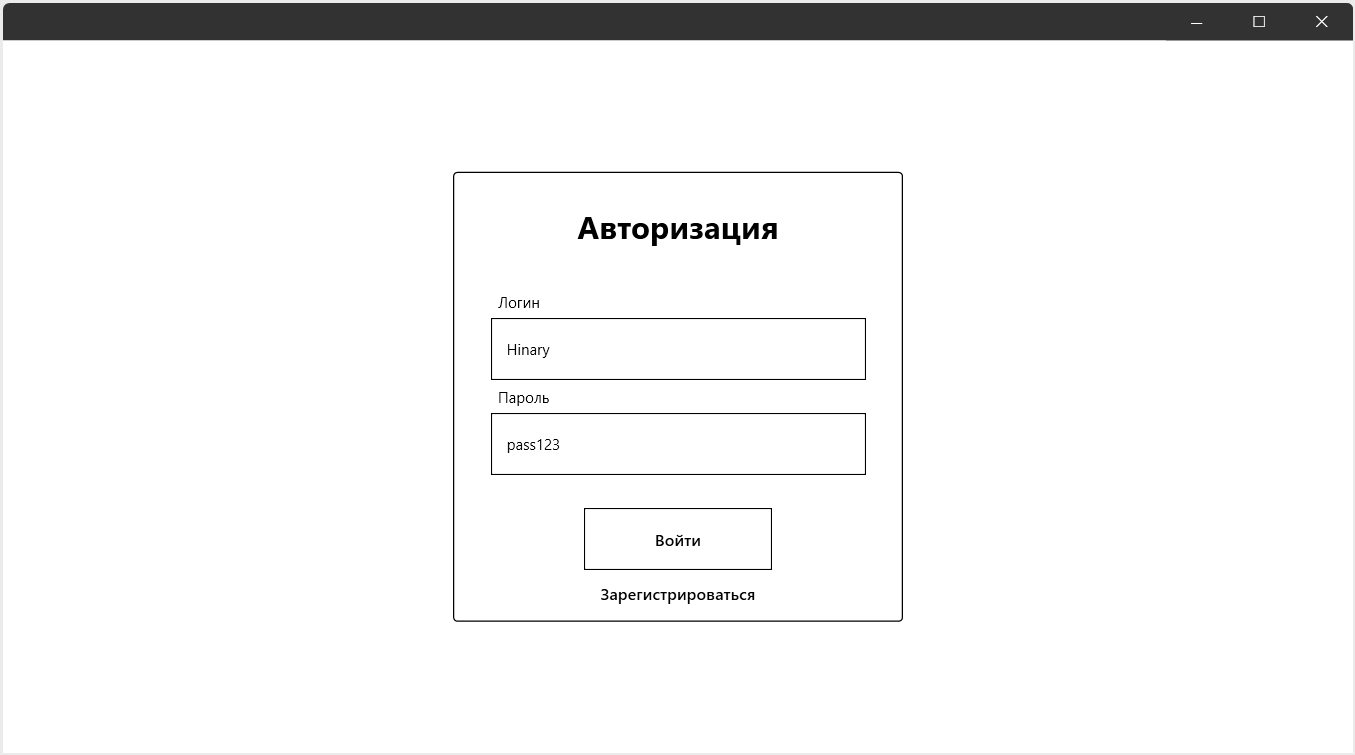


Рисунок В.2 – Авторизация пользователя

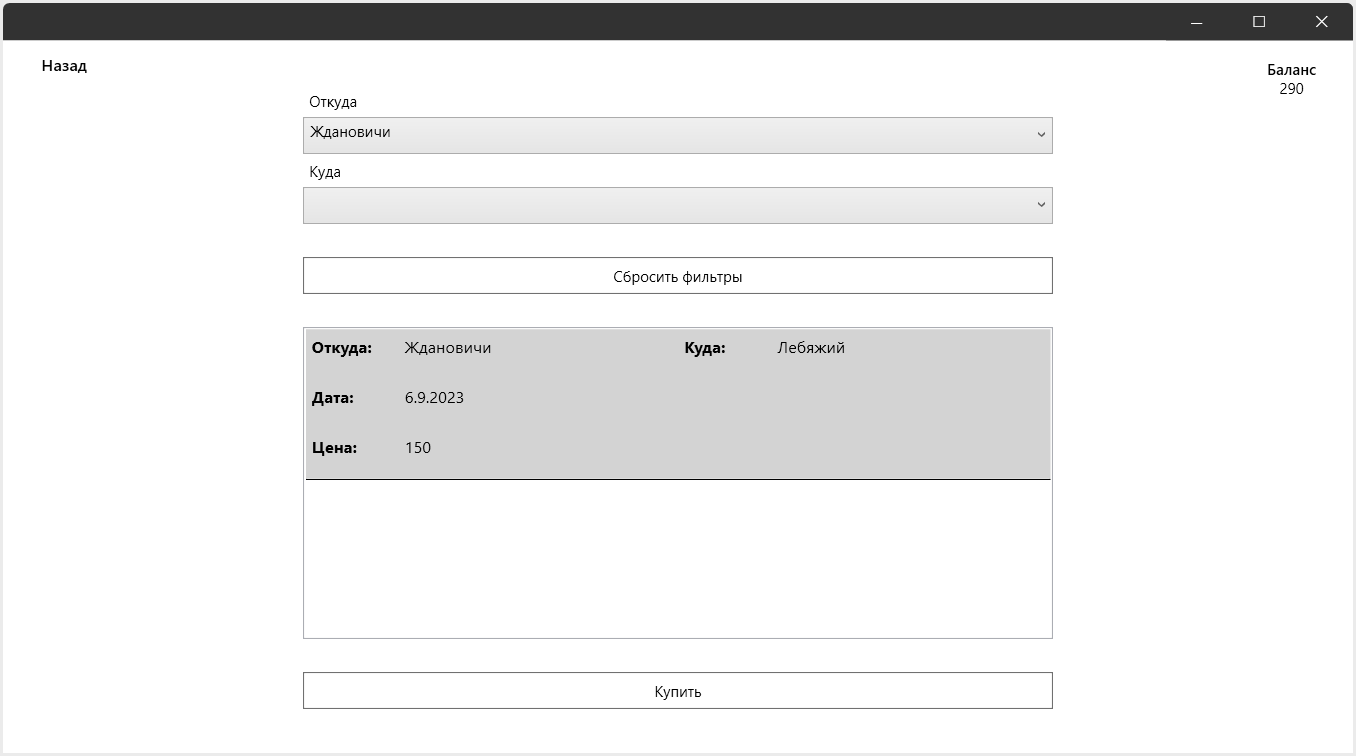


Рисунок В.3 – Покупка билета

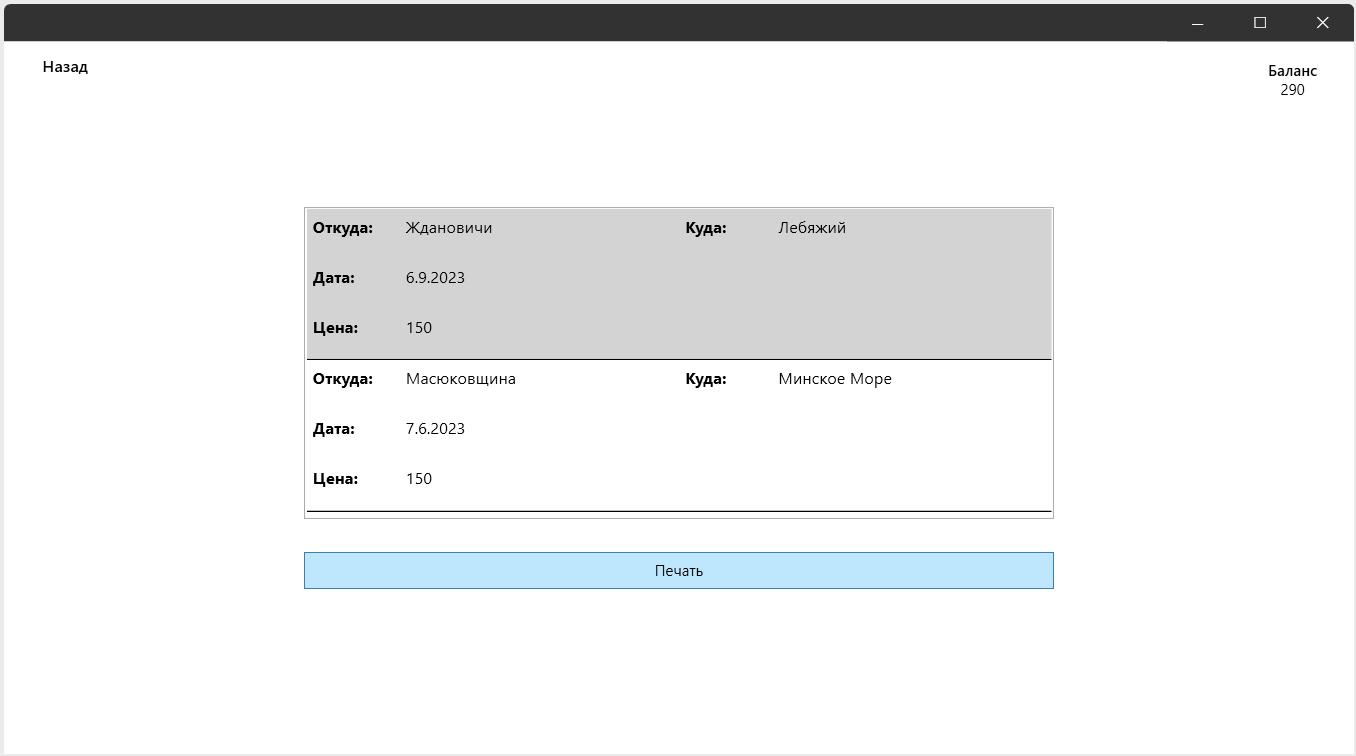


Рисунок В.4 – Печать билета

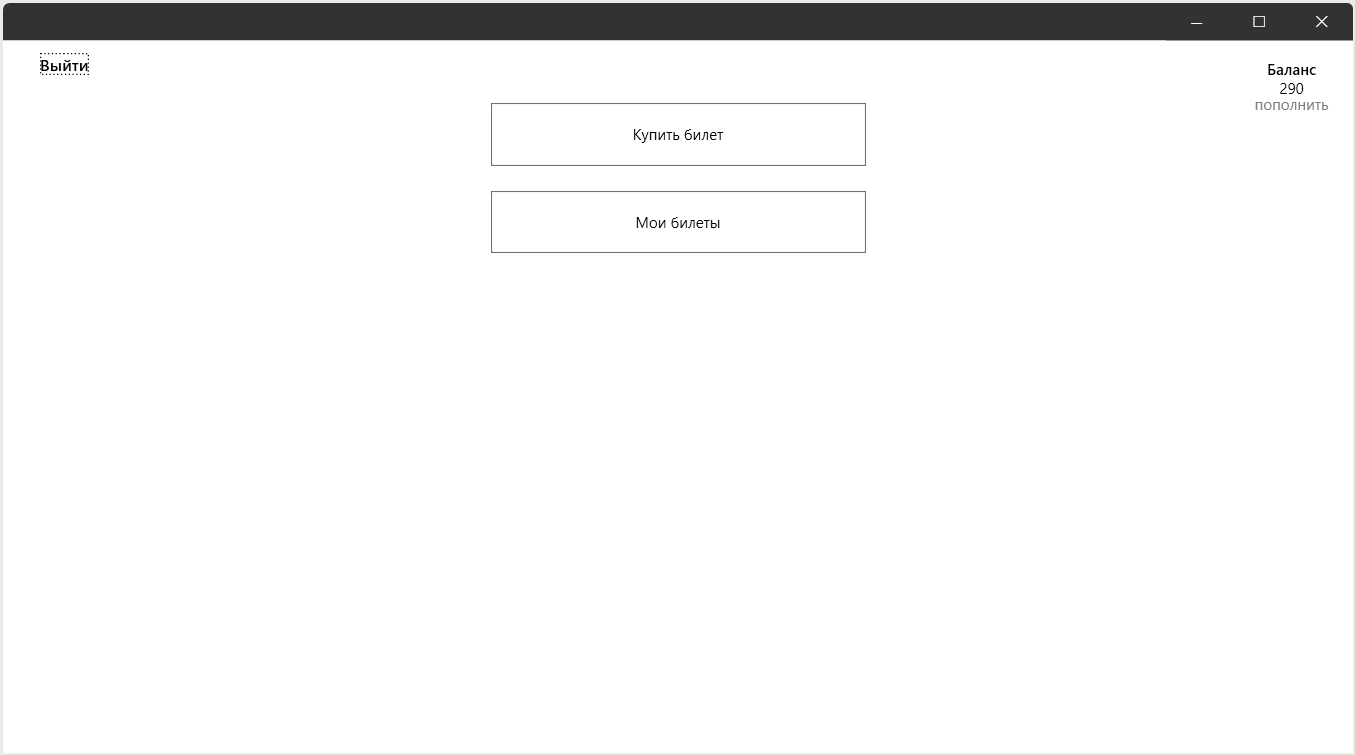


Рисунок В.5 – Выход из аккаунта