Содержание

Введение 4

1 Техническое задание 5

1.1 Вариант задания 5

1.2 Исходные данные 5

1.3 Функциональные требования 5

2 Описание программы 8

2.1 Общие сведения 8

2.2 Функциональное назначение 8

2.3 Описание логической структуры 8

2.4 Используемые технические средства 11

2.5 Вызов и загрузка 11

2.6 Входные данные 11

2.7 Выходные данные 12

3 Руководство оператора 13

3.1 Назначение программы 13

3.2 Условия выполнения программы 13

3.3 Выполнение программы 13

3.4 Сообщения оператору 22

4 Тестирование 24

Заключение 26

Список использованных источников 27

Приложение А 28

# Введение

**За последние десятилетия, программное обеспечение претерпело огромные изменения, пройдя путь от программ, которые способны выполнять лишь простейшие логические и арифметические операции, до сложных систем управления организациями. На сегодняшний день управление организацией без компьютера просто немыслимо. Для принятия какого-либо решения в условиях неопределенности и риска нужно регулярно держать под контролем разные аспекты финансово-хозяйственной деятельности. На современном этапе развития информатизации и автоматизации характерно применение распределенной обработки информации. В качестве более эффективной и перспективной сферы применения концепции распределенной обработки информации выступает автоматизация планово-управленческих функций на основе персональных компьютеров, которые установлены непосредственно на рабочем месте того или иного специалиста. Системы, которые выполняют данные функции, наделены широким распространением, которые называются автоматизированные рабочие места (АРМ).**

Программа автоматизации работы автопарка автобусов представляет собой программу управления базой данных. Реализация базы данных возможна несколькими способами:

– использование существующих систем управления базами данных (СУБД);

– создание собственной структурированной базы данных.

К преимуществу СУБД можно отнести централизованное хранение всей информации о базе данных в самой базе данных (БД). В системе изначально созданы и реализованы алгоритмы обработки массивов данных без внешнего вмешательства. Все операции по изменению, сохранению информации выполняются самой СУБД, следовательно, нет необходимости программисту самому реализовывать методы сохранения и открытия файла данных. К недостаткам СУБД можно отнести избыточность хранимой информации в файле БД, что существенно увеличивает размер файла БД. А также необходимость наличия СУБД на компьютере пользователя, что существенно усложняет поставку программ конечному пользователю.

**Цель данного курсового проекта - создание программы для автоматизации работы автопарка автобусов, удовлетворяющий перечисленным ранее требованиям, а также реализованный с использованием соответствующих СУБД и программного обеспечения.**

# 1. Техническое задание

## 1.1 Вариант задания

Разработка автоматизированной системы управления автобусным парком и продажей билетов "Автопарк". Система предназначена для управления маршрутами, расписаниями движения автобусов, ведения учета транспортных средств, сотрудников, продаж билетов и формирования отчетности. Основные сущности системы включают:

Маршруты (Marshut): Начальная и конечная точки, список остановок.

Расписания маршрутов (RouteSchedules): Привязка к маршруту, время отправления и прибытия, цена билета, доступные места, информация об автобусе, дни курсирования, активность, повторяемость.

Автобусы (Bus): Информация о транспортных средствах парка.

Сотрудники (Employee): Данные о водителях и другом персонале.

Должности (Job): Справочник должностей.

Техническое обслуживание (Maintenance): Учет работ по обслуживанию автобусов.

Продажи (Sales): Информация о проданных билетах.

Билеты (Ticket): Данные конкретных билетов.

Пользователи (User): Учетные записи для доступа к системе.

Ключевые задачи:

Управление данными маршрутов и их расписаний (добавление, редактирование, удаление, просмотр).

Фильтрация и поиск расписаний по маршруту и дате.

Управление данными об автобусах, сотрудниках, должностях, ТО.

Учет и просмотр продаж билетов.

Формирование отчетов (например, отчет о доходах frmIncomeReport, статистика продаж frmSalesStatistics).

Реализация ролевой модели доступа (Администратор, Пользователь).

Обеспечение авторизации для входа в систему.

1.2 Исходные данные и Технологии

Требуется разработать клиент-серверное приложение.

Бэкенд (TicketSalesApp.AdminServer): Разрабатывается на ASP.NET Core (C#), предоставляет REST API для взаимодействия с данными.

Клиентские приложения:

Основное приложение для администрирования

(TicketSalesApp.UI.LegacyForms.DX.Windows): Приложение Windows Forms с использованием компонентов DevExpress, разработанное в Microsoft Visual Studio на C#.

Дополнительные клиентские приложения

(TicketSalesApp.UI.Administration.Avalonia, TicketSalesApp.UI.Avalonia): Разрабатываются с использованием кроссплатформенного фреймворка Avalonia UI.

Хранение данных: Основные данные хранятся в централизованной базе данных (тип СУБД определяется конфигурацией бэкенда), доступ к которой осуществляется через API бэкенда. Локальные файлы могут использоваться для хранения настроек (например, данных для "Запомнить меня").

Взаимодействие: Клиентские приложения взаимодействуют с бэкендом по протоколу HTTP(S).

Структура: Логика разделена на слои: представление (UI), сервисы (TicketSalesApp.Services), ядро (TicketSalesApp.Core), сервер API (TicketSalesApp.AdminServer). Каждая логически завершенная задача реализуется в виде методов в соответствующих классах (контроллеры API, сервисы, классы форм/ViewModel).

Интерфейс: Интерфейс пользователя должен быть интуитивно понятным и защищенным от ошибок ввода и сбоев.

Ввод данных: Осуществляется через пользовательский интерфейс клиентских приложений.

Вывод данных: Результаты работы (например, отчеты) должны сохраняться в файлы (например, текстовые или Excel) по запросу пользователя.

1.3 Функциональные требования

Авторизация: Работа с системой начинается с окна входа (frmLogin), где пользователь вводит логин и пароль. Данные проверяются путем вызова соответствующего метода API бэкенда. При успешной аутентификации пользователь получает доступ к функционалу в соответствии со своей ролью (например, Администратор или Пользователь). Реализована возможность "Запомнить меня" путем сохранения учетных данных локально (с сохранением в файл).

Управление пользователями: Регистрация новых пользователей не предусмотрена в окне входа. Добавление и управление учетными записями осуществляется Администратором через соответствующий интерфейс (например, frmEmployeeManagement включает управление данными сотрудников, сохраненными в базе данных).

Ролевая модель:

Администратор:

Полный доступ ко всем данным системы: просмотр, добавление, редактирование, удаление записей во всех управляемых сущностях (Маршруты, Расписания, Автобусы, Сотрудники, Должности, ТО, Продажи, Билеты, Пользователи).

Поиск и фильтрация данных (например, фильтрация расписаний по маршруту и дате в frmRouteSchedulesManagement). Сортировка данных (часто предоставляется компонентами UI, например, DevExpress GridView).

Формирование отчетов (frmIncomeReport, frmSalesStatistics) и их сохранение в файл.

Пользователь (предполагаемые возможности):

Просмотр данных, необходимых для работы (например, расписания, маршруты), за исключением управления учетными записями и, возможно, некоторых административных данных.

Поиск, фильтрация и сортировка доступных данных.

Формирование доступных отчетов и их сохранение в файл.

Пользовательский интерфейс: Доступ к различным функциям (формам управления) осуществляется через главное меню или панель инструментов, элементы которых активны в зависимости от роли авторизованного пользователя. Система должна содержать раздел помощи или справочную информацию.

Надежность и Безопасность:

Обработка исключительных ситуаций (try-catch) на клиенте и сервере для предотвращения аварийного завершения работы. Использование логирования (Serilog) для диагностики проблем.

Валидация вводимых данных на стороне клиента (в формах) и/или сервера (в API) для предотвращения ввода некорректных данных.

Запросы на подтверждение перед выполнением необратимых действий (например, удаление записи btnDelete\_Click).

Предоставление обратной связи пользователю через информационные сообщения, предупреждения и сообщения об ошибках (XtraMessageBox).

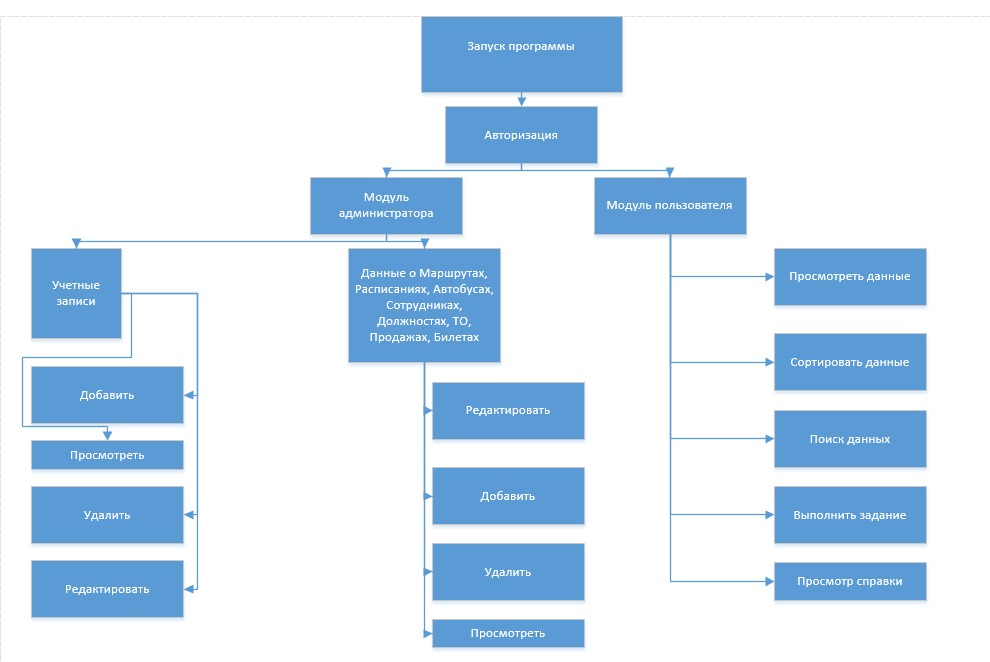


Рисунок 1 – Структура меню

2. Описание программы

2.1 Общие сведения

Программа "Автоматизированная система управления автобусным парком и продажей билетов 'Автопарк'" предназначена для комплексного управления деятельностью автобусного парка, включая управление маршрутами, расписаниями, транспортными средствами, персоналом, а также для учета продаж билетов и формирования отчетности.Система разработана с использованием следующих технологий:

Серверная часть (TicketSalesApp.AdminServer): Платформа ASP.NET Core, язык C#.

Клиентские приложения:

Основное приложение администрирования (TicketSalesApp.UI.LegacyForms.DX.Windows): Технология Windows Forms с использованием компонентов DevExpress, язык C#. Ведется работа по обеспечению совместимости с ОС Windows XP.

Современные клиентские приложения (TicketSalesApp.UI.Administration.Avalonia, TicketSalesApp.UI.Avalonia): Кроссплатформенный фреймворк Avalonia UI (требующий .NET 9.0 или выше), язык C#.

Среда разработки: Microsoft Visual Studio 2022.

Автором программы является Савич Андрей Олегович, учащийся 1 курса группы АСОИСЗ-241.Для работы системы требуется установка соответствующей версии .NET Runtime для клиентских приложений (версия зависит от конкретного клиента, включая .NET Framework для совместимости с Windows XP и .NET 9.0+ для Avalonia) и ASP.NET Core Runtime для серверной части.Программа должна удовлетворять следующим требованиям:

Надежности функционирования.

Безопасности данных.

Защиты информации от несанкционированного доступа через систему ролей и авторизацию.

Минимальные системные требования к программе (могут отличаться для разных компонентов):

Операционная система: Windows XP (для

 TicketSalesApp.UI.LegacyForms.DX.Windows),

Windows 7-11 (для всех компонентов),

Linux/macOS (для Avalonia-клиентов).

Процессор: Тактовая частота - от 1 ГГц.

Свободная оперативная память: от 512 МБ (рекомендуется 1 ГБ и выше, особенно для сервера).

Свободное место на жёстком диске: от 100 МБ (без учета базы данных).

Наличие сетевого подключения для взаимодействия клиентов с сервером API. Для работы с клиентскими приложениями используется клавиатура и манипулятор типа «мышь».

2.2 Функциональное назначение

Программа представляет собой клиент-серверную систему для управления базой данных автобусного парка. Она позволяет авторизованным пользователям (в зависимости от их роли – Администратор или Пользователь) взаимодействовать с данными через различные клиентские приложения (Windows Forms или Avalonia UI).Основные функции включают:

Управление маршрутами и расписаниями (просмотр, добавление, редактирование, удаление).

Управление данными об автобусах, сотрудниках, должностях, техническом обслуживании.

Учет и анализ продаж билетов.

Поиск, фильтрация и сортировка данных по различным критериям.

Формирование отчетов (например, отчет о доходах, статистика продаж) и их экспорт в файлы.

2.3 Описание логической структуры

Работа пользователя с системой начинается с окна авторизации (frmLogin в WinForms-клиенте), где требуется ввод логина и пароля для доступа к системе.После успешной авторизации пользователь получает доступ к главному интерфейсу приложения (например, главному окну с меню или панели инструментов в WinForms), откуда можно перейти к различным модулям системы.Основные модули представлены в виде отдельных окон (форм) или представлений (view), каждое из которых отвечает за управление определенной сущностью или выполнение конкретной задачи:

Управление расписаниями (frmRouteSchedulesManagement).

Управление маршрутами (frmRouteManagement).

Управление автобусами (frmBusManagement).

Управление сотрудниками (frmEmployeeManagement).

Управление должностями (frmJobManagement).

Управление ТО (frmMaintenanceManagement).

Управление продажами (frmSalesManagement).

Формирование отчетов (frmIncomeReport, frmSalesStatistics).

(Предположительно) Управление пользователями (может быть частью frmEmployeeManagement или отдельной формой).

Доступ к этим модулям и их функциональным возможностям (просмотр, добавление, редактирование, удаление) ограничен правами авторизованного пользователя (Администратор/Пользователь).Взаимодействие между клиентскими приложениями и базой данных происходит через сервер API (TicketSalesApp.AdminServer), который реализует бизнес-логику и обеспечивает доступ к данным.

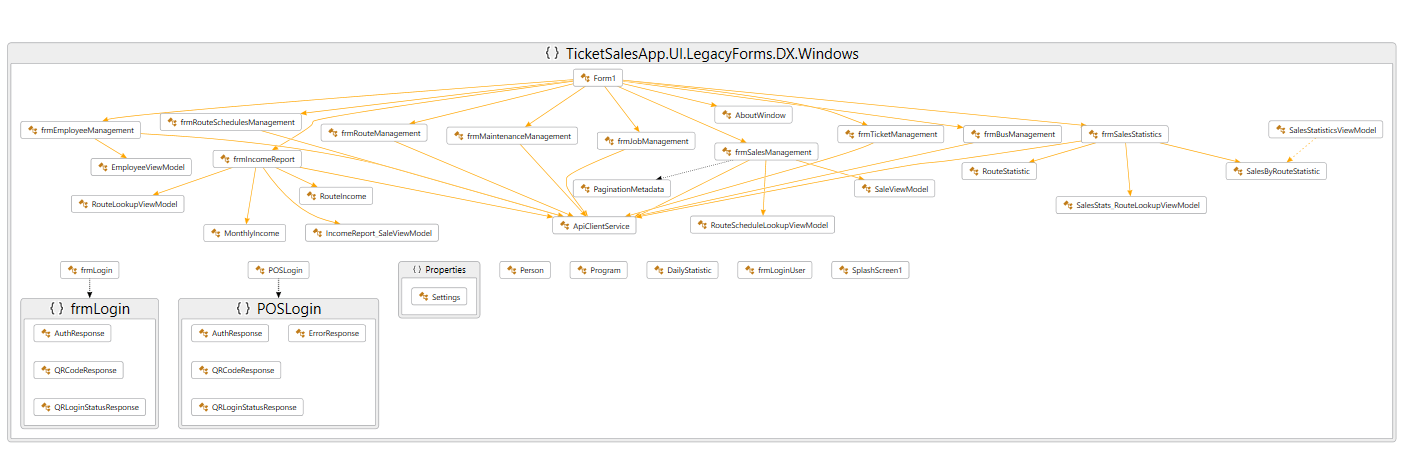


Рисунок 2 – Диаграмма классов для проекта Windows forms

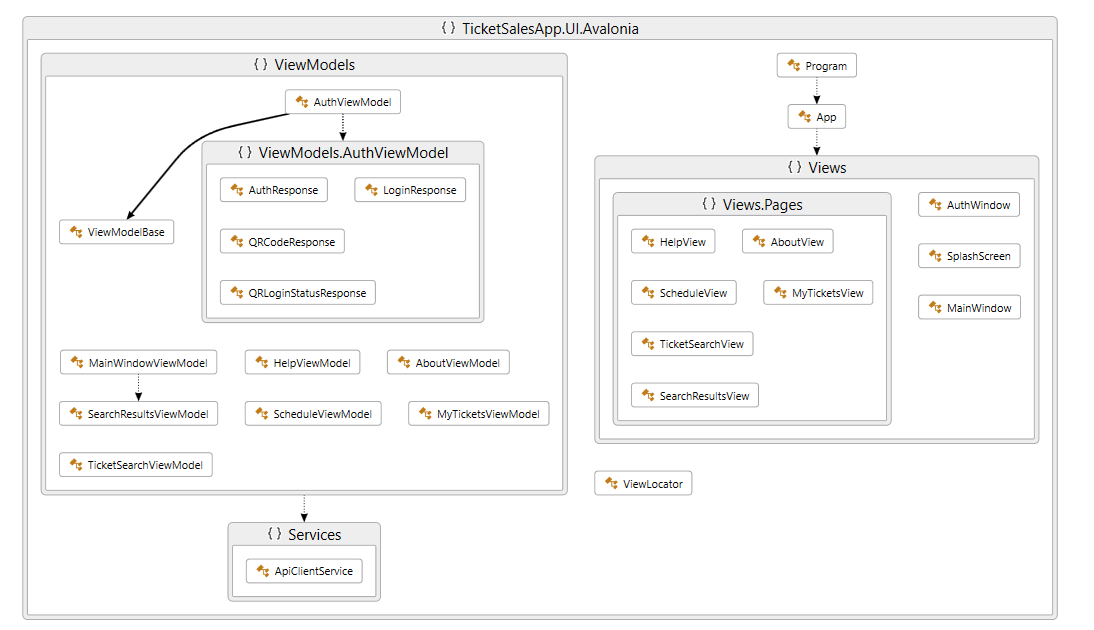


Рисунок 2.1 Диаграмма классов для проекта TicketSalesApp.UI.Avalonia

В таблице 1 описаны методы программы описанием выполняемых им действий.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя формы | Имя элемента управления | Метод | Краткое описание |
| frmLogin | (Форма) | frmLogin\_Load | Загружает сохраненные данные пользователя из XML при запуске |
| frmLogin | btnContinueWithUsername | btnContinueWithUsername\_Click | Переключает интерфейс на ввод пароля после ввода имени пользователя |
| frmLogin | btnLogin | btnLogin\_Click | Выполняет аутентификацию по логину и паролю |
| frmLogin | btnSwitchToPassword | btnSwitchToPassword\_Click | Переключает интерфейс на режим входа по логину/паролю |
| frmLogin | btnSwitchToQR | btnSwitchToQR\_Click | Переключает интерфейс на режим входа по QR-коду |
| frmLogin | btnRefreshQR | btnRefreshQR\_Click | Запрашивает и отображает новый QR-код для входа |
| frmLogin | (Таймер) | CheckQRLoginStatus | Периодически проверяет статус входа по QR-коду |
| Form1 (Главное меню) | BaseItem1 (Автобусы) | BaseItem1\_Click | Открывает форму управления автобусами (frmBusManagement) |
| Form1 (Главное меню) | BaseItem2 (Должности) | BaseItem2\_Click | Открывает форму управления должностями (frmJobManagement) |
| Form1 (Главное меню) | BaseItem3 (Обслуживание) | BaseItem3\_Click | Открывает форму управления техническим обслуживанием (frmMaintenanceManagement) |
| Form1 (Главное меню) | BaseItem4 (Маршруты) | BaseItem4\_Click | Открывает форму управления маршрутами (frmRouteManagement) |
| Form1 (Главное меню) | BaseItem5 (Сотрудники) | BaseItem5\_Click | Открывает форму управления сотрудниками (frmEmployeeManagement) |
| Form1 (Главное меню) | BaseItem6 (Расписания) | BaseItem6\_Click | Открывает форму управления расписаниями (frmRouteSchedulesManagement) |
| Form1 (Главное меню) | продажиБилетовToolStripMenuItem | MenuItemSalesManagement\_Click | Открывает форму управления продажами (frmSalesManagement) |
| Form1 (Главное меню) | menuItemTicketManagement | MenuItemTicketManagement\_Click | Открывает форму управления билетами (frmTicketManagement) |
| Form1 (Главное меню) | menuItemIncomeReport | MenuItemIncomeReport\_Click | Открывает форму отчета о доходах (frmIncomeReport) |
| Form1 (Главное меню) | menuItemSalesStatistics | MenuItemSalesStatistics\_Click | Открывает форму статистики продаж (frmSalesStatistics) |
| Form1 (Главное меню) | оПрограммеToolStripMenuItem | оПрограммеToolStripMenuItem\_Click | Открывает окно «О программе» (AboutWindow) |
| frmBusManagement | (Форма) | frmBusManagement\_Load | Загружает список автобусов при открытии формы |
| frmBusManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления нового автобуса |
| frmBusManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранного автобуса |
| frmBusManagement | btnDelete | btnDelete\_Click | Удаляет выбранный автобус после подтверждения |
| frmBusManagement | btnRefresh | btnRefresh\_Click | Обновляет список автобусов с сервера |
| frmBusManagement | txtSearch | txtSearch\_EditValueChanged | Фильтрует список автобусов по мере ввода текста в поле поиска |
| frmRouteManagement | (Форма) | frmRouteManagement\_Load | Загружает список маршрутов, автобусов и водителей при открытии формы |
| frmRouteManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления нового маршрута |
| frmRouteManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранного маршрута |
| frmRouteManagement | btnDelete | btnDelete\_Click | Удаляет выбранный маршрут после подтверждения |
| frmRouteManagement | btnRefresh | btnRefresh\_Click | Обновляет список маршрутов, автобусов и водителей с сервера |
| frmRouteManagement | txtSearch | txtSearch\_EditValueChanged | Фильтрует список маршрутов по мере ввода текста |
| frmRouteSchedulesManagement | (Форма) | frmRouteSchedulesManagement\_Load | Загружает список маршрутов для фильтра при открытии формы |
| frmRouteSchedulesManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления нового расписания |
| frmRouteSchedulesManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранного расписания |
| frmRouteSchedulesManagement | btnDelete | btnDelete\_Click | Удаляет выбранное расписание после подтверждения |
| frmRouteSchedulesManagement | btnRefresh | btnRefresh\_Click | Загружает/обновляет список расписаний с учетом выбранных фильтров |
| frmEmployeeManagement | (Форма) | frmEmployeeManagement\_Load | Загружает список сотрудников и должностей при открытии формы |
| frmEmployeeManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления нового сотрудника |
| frmEmployeeManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранного сотрудника |
| frmEmployeeManagement | btnDelete | btnDelete\_Click | Удаляет выбранного сотрудника после подтверждения |
| frmEmployeeManagement | btnRefresh | btnRefresh\_Click | Обновляет список сотрудников и должностей с сервера |
| frmEmployeeManagement | txtSearch | txtSearch\_EditValueChanged | Фильтрует список сотрудников по мере ввода текста |
| frmJobManagement | (Форма) | frmJobManagement\_Load | Загружает список должностей при открытии формы |
| frmJobManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления новой должности |
| frmJobManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранной должности |
| frmJobManagement | btnDelete | btnDelete\_Click | Удаляет выбранную должность после подтверждения |
| frmJobManagement | btnRefresh | btnRefresh\_Click | Обновляет список должностей с сервера |
| frmJobManagement | txtSearch | txtSearch\_EditValueChanged | Фильтрует список должностей по мере ввода текста |
| frmMaintenanceManagement | (Форма) | frmMaintenanceManagement\_Load | Загружает список записей ТО и автобусов при открытии формы |
| frmMaintenanceManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления новой записи ТО |
| frmMaintenanceManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранной записи ТО |
| frmMaintenanceManagement | btnDelete | btnDelete\_Click | Удаляет выбранную запись ТО после подтверждения |
| frmMaintenanceManagement | btnRefresh | btnRefresh\_Click | Обновляет список записей ТО и автобусов с сервера |
| frmMaintenanceManagement | txtSearch | txtSearch\_EditValueChanged | Фильтрует список записей ТО по мере ввода текста |
| frmSalesManagement | (Форма) | frmSalesManagement\_Load | Загружает список продаж и доступных расписаний при открытии формы |
| frmSalesManagement | btnAdd | btnAdd\_Click | Открывает диалог для добавления новой продажи |
| frmSalesManagement | btnEdit | btnEdit\_Click | Открывает диалог для редактирования выбранной продажи |
| frmSalesManagement | btnDelete | btnDelete\_Click | Удаляет выбранную продажу после подтверждения |
| frmSalesManagement | btnRefresh | btnRefresh\_Click | Обновляет список продаж и расписаний с сервера |
| frmSalesManagement | btnApplyFilter | btnApplyFilter\_Click | Применяет фильтры по дате к списку продаж |
| frmSalesManagement | btnExport | btnExport\_Click | Экспортирует текущие данные продаж в файл (например, CSV/Excel) |
| frmSalesManagement | gridViewSales | gridViewSales\_DoubleClick | Открывает диалог редактирования для строки, по которой дважды кликнули |
| frmTicketManagement | (Форма) | frmTicketManagement\_Load | Загружает список билетов и расписаний для фильтра при открытии формы |
| frmTicketManagement | btnRefresh | btnRefresh\_Click | Обновляет список билетов с сервера |
| frmTicketManagement | btnApplyFilter | btnApplyFilter\_Click | Применяет фильтры (рейс, статус) к списку билетов |
| frmTicketManagement | btnViewDetails | btnViewDetails\_Click | Отображает подробную информацию о выбранном билете |
| frmTicketManagement | btnCancelTicket | btnCancelTicket\_Click | Отменяет (удаляет) выбранный, еще не проданный билет после подтверждения |
| frmIncomeReport | (Форма) | frmIncomeReport\_Load | Загружает список маршрутов и данные о продажах при открытии формы |
| frmIncomeReport | btnApplyFilter | btnApplyFilter\_Click | Применяет фильтры по дате и маршруту к данным отчета |
| frmIncomeReport | btnExport | btnExport\_Click | Экспортирует данные отчета в файл |
| frmSalesStatistics | (Форма) | frmSalesStatistics\_Load | Загружает статистические данные при открытии формы |
| frmSalesStatistics | btnApplyFilter | btnApplyFilter\_Click | Загружает статистику за выбранный период дат |
| frmLoginUser | (Форма) | frmLoginUser\_Load | Загружает данные (пользователей) при открытии формы (реализация неполная) |
| frmLoginUser | tsbtnSave | tsbtnSave\_Click | Сохраняет изменения в данных пользователей (реализация неполная) |
| frmLoginUser | tsBtnDel | tsBtnDel\_Click | Удаляет выбранного пользователя (реализация неполная) |
| frmLoginUser | tsBtnExit | tsBtnExit\_Click | Закрывает форму управления пользователями |
| AboutWindow | (Форма) | – | Отображает информацию о программе |

2.4 Используемые технические средства

Для разработки клиентской части администратора (`TicketSalesApp.UI.LegacyForms.DX.Windows`) была выбрана среда разработки Microsoft Visual Studio 2022, платформа .NET Framework (с целью обеспечения совместимости с Windows XP), язык C# и набор компонентов пользовательского интерфейса DevExpress WinForms. Серверная часть приложения (`TicketSalesApp.AdminServer`) разработана на платформе ASP.NET Core.

Данный выбор языка C# объясняется тем, что он предусматривает следующие преимущества: строгую типизацию, сохранение концепций объектно-ориентированного программирования, функциональность, достаточно мощный инструментарий, стабильную работу через Visual Studio, компактный и легко читаемый код. Для пользовательского интерфейса основного клиента администрирования используются компоненты DevExpress, расширяющие стандартные возможности Windows Forms.

Взаимодействие с базой данных осуществляется не напрямую из клиента, а через API, предоставляемый серверной частью (TicketSalesApp.AdminServer). Тип используемой СУБД определяется конфигурацией сервера.

2.5 Вызов и загрузка

Запуск программы осуществляется путем выполнения соответствующего .exe файла клиентского приложения (TicketSalesApp.UI.LegacyForms.DX.Windows.exe). При запуске отображается окно-заставка (`SplashScreen1`), после чего открывается окно авторизации (`frmLogin`). Установка программы может включать развертывание необходимых файлов и создание ярлыков.

2.6 Входные данные

Основными входными данными для клиентского приложения являются учетные данные пользователя (логин и пароль или QR-код), вводимые в окне авторизации (`frmLogin`). Эти данные отправляются на сервер API для проверки. Программа также может считывать локально сохраненные учетные данные для функции "Запомнить меня" (методы `GetLoginUnitXML` и `WriteLoginUnitXML` в `frmLogin`). Все основные данные (маршруты, расписания, сотрудники и т.д.) загружаются динамически с сервера API по мере необходимости. Успешное подключение к серверу API является необходимым условием для работы программы.

2.7 Выходные данные

Выходными данными для программы являются данные, отображаемые пользователю в табличном виде (с использованием `DevExpress.XtraGrid.GridControl`) в различных формах управления (например, `frmBusManagement`, `frmRouteSchedulesManagement`, `frmSalesManagement` и др.). Пользователь может взаимодействовать с этими данными (просматривать, фильтровать, сортировать).

Программа также формирует отчеты (`frmIncomeReport`, `frmSalesStatistics`), которые могут быть экспортированы в файлы (например, CSV/Excel) по запросу пользователя (кнопки `btnExport`). Система информирует пользователя об успешном выполнении действий или возникших ошибках путем вывода сообщений на экран (с использованием `XtraMessageBox`).

3 Руководство оператора

3.1 Назначение программы

Программа TicketSalesApp предназначена для управления системой продажи билетов и операционной деятельностью автотранспортного предприятия. Она предназначена для использования на рабочих местах диспетчеров, кассиров, администраторов и других сотрудников, отвечающих за управление перевозками и продажу билетов. Пользователями являются сотрудники с уверенным уровнем владения компьютером. Программа предоставляет возможность выполнения следующих функций:

Управление билетами: Просмотр информации о билетах, фильтрация по различным критериям (например, по рейсу), отмена билетов (если они не проданы), просмотр деталей билета.

Управление расписаниями рейсов: Добавление, редактирование и удаление записей о расписаниях конкретных рейсов автобусов, включая время отправления/прибытия, цены, количество мест.

Управление маршрутами: Добавление, редактирование и удаление автобусных маршрутов (начальный и конечный пункты).

Управление автобусным парком: Ведение учета автобусов предприятия.

Управление данными сотрудников: Ведение базы данных сотрудников, включая водителей, и их должностей.

Управление продажами: Просмотр и управление записями о продажах билетов.

Управление техническим обслуживанием: Учет записей о прохождении ТО автобусами.

Управление должностями: Ведение справочника должностей сотрудников.

Поиск и Фильтрация: Поиск записей в таблицах по текстовому вводу, фильтрация данных по различным параметрам (например, датам, маршрутам).

Обновление данных: Загрузка актуальных данных с сервера.

Формирование отчетов: Генерация отчетов (например, отчет о доходах, статистика продаж).

Экспорт данных: Возможность экспорта табличных данных (например, в формат Excel).

3.2 Условия выполнения программы

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять стандартным требованиям для офисных помещений: оптимальная температура 22—24 °C в холодное время года и 23—25 °C в теплое; относительная влажность воздуха 15—75%; скорость движения воздуха 0,1—0,2 м/с.

Существуют две версии программы с разными системными требованиями:

А) Первая версия (Legacy): Данная версия предназначена для работы на устаревшем оборудовании и операционных системах. В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя: Операционная система: Windows XP (SP3) или следующие версии.

Требуемое ПО: Установленный .NET Framework 4.0 (или совместимая версия 4.x). Также требуется наличие соответствующих библиотек DevExpress v15.2, которые устанавливаются вместе с программой.

Процессор: Pentium 4 / Athlon XP или совместимый, с тактовой частотой от 1 ГГц.

Оперативная память: от 512 МБ (рекомендуется 1 ГБ или больше).

Видеокарта: видеокарта, поддерживающая разрешение экрана 1024x768 или выше.

Свободное место на жёстком диске: от 100 МБ (для установки программы и кэширования данных).

Б) Современная версия:

Для организаций с более современным парком компьютеров разработана версия программы на платформе .NET 9.0. Для ее работы требуются следующие условия:

Операционная система: Windows 10 (64-bit, версия 1809 или новее), Windows 11.

Требуемое ПО: Установленный .NET 9 Desktop Runtime (или выше).

Процессор: 64-битный процессор (рекомендуется многоядерный).

Оперативная память: от 4 ГБ (рекомендуется 8 ГБ или больше).

Видеокарта: с поддержкой DirectX 11 или выше.

Свободное место на жёстком диске: от 200 МБ (зависит от объема кэшируемых данных).

3.3 Выполнение программы

После установки программы запуск осуществляется двойным кликом по ярлыку программы на рабочем столе или через меню "Пуск". В случае успешного запуска на рабочем столе будет представлено окно авторизации (frmLogin). Необходимо ввести логин и пароль пользователя, предоставленные администратором системы, после чего нажать на кнопку "Войти". Если пользователь не желает каждый раз вводить свои данные вручную, можно выбрать пункт «Запомнить меня» (если данная опция доступна в окне входа). В таком случае программа сохранит введенные данные и при последующей загрузке будет автоматически подставлять их в поля логина и пароля.

При успешной авторизации откроется главное окно приложения (Form1) с доступом к основным функциям системы.

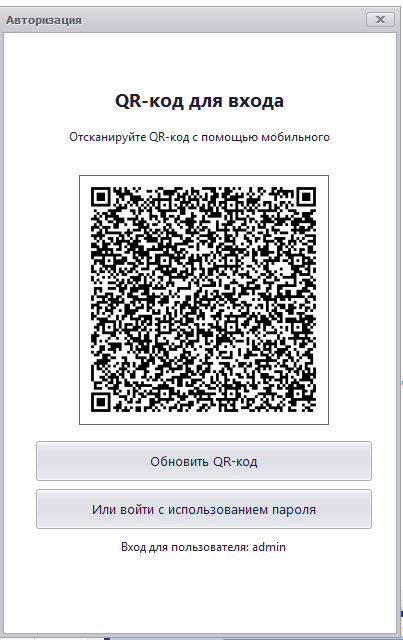


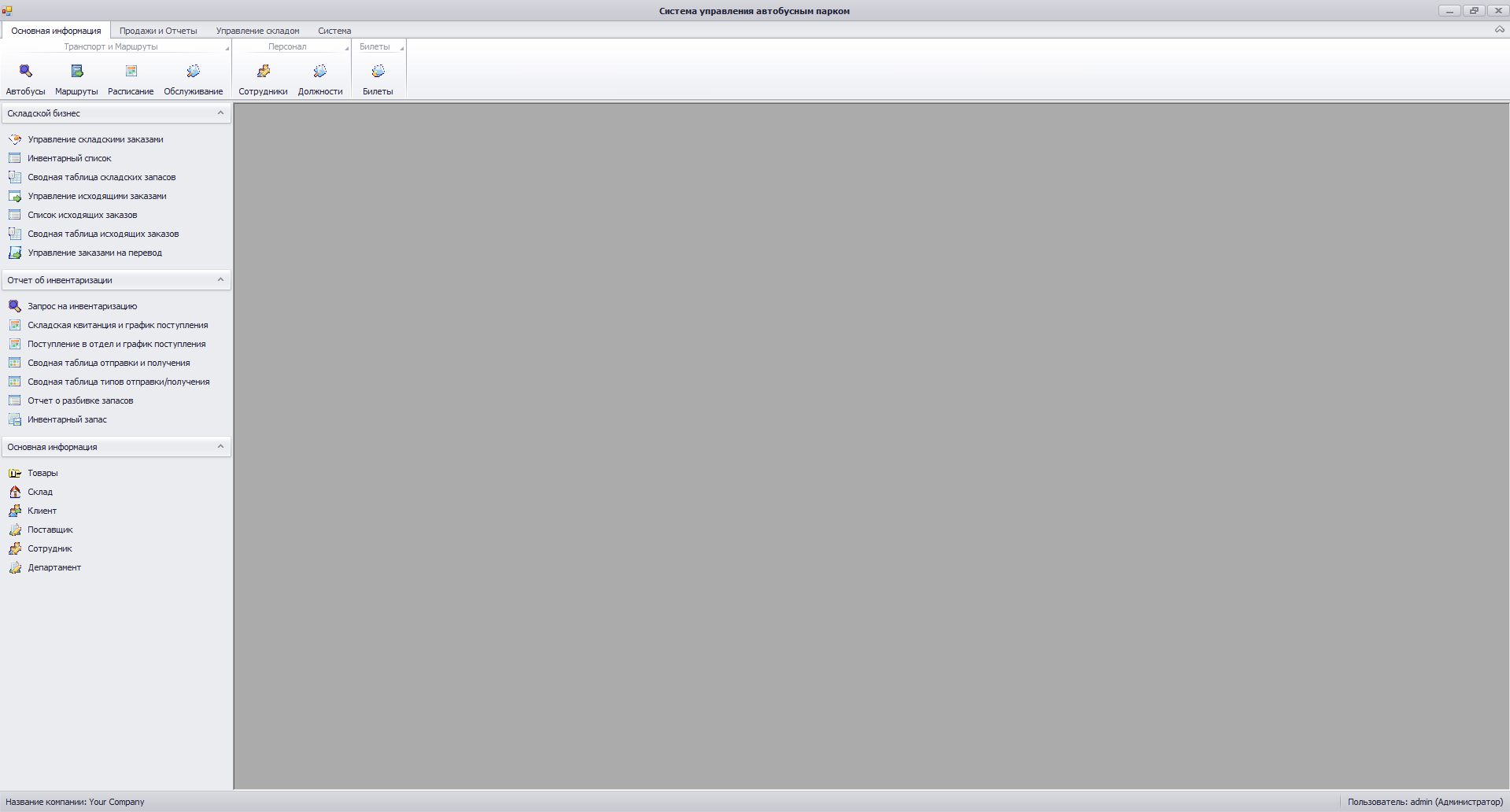
Рисунок 3 – Форма авторизации

Если логин и пароль введены неправильно, пользователь получит сообщение об ошибке (рисунок 4).



Рисунок 4 – Сообщение о неверно введенных данных

После успешной авторизации программа определяет права доступа пользователя и в зависимости от них ограничивает доступные возможности путем блокирования кнопок. Форма меню, открывающаяся после успешной авторизации обычного пользователя, представлена на рисунке 5.

 Рисунок 5 – Основная форма

После успешной авторизации открывается главное окно приложения (Form1) с ленточным интерфейсом (Ribbon). Программа анализирует JWT-токен, полученный при авторизации, и определяет роль пользователя (Администратор или Обычный пользователь). В зависимости от роли, некоторые кнопки на ленте или целые разделы меню могут быть неактивны (заблокированы) или скрыты. Например, разделы администрирования системы, управления должностями, и, возможно, некоторая отчетность доступны только администраторам. Доступ к основным функциям программы осуществляется через кнопки на ленте меню:

Вкладка "Основная информация":

"Автобусы": Открывает форму frmBusManagement для управления парком автобусов.

"Маршруты": Открывает форму frmRouteManagement для управления маршрутами.

"Расписание": Открывает форму frmRouteSchedulesManagement для управления расписаниями рейсов.

"Обслуживание": Открывает форму frmMaintenanceManagement для учета ТО.

"Сотрудники": Открывает форму frmEmployeeManagement для управления персоналом.

"Должности": Открывает форму frmJobManagement (доступно администратору).

"Билеты": Открывает форму frmTicketManagement. При открытии этой формы сначала запрашивается выбор основного маршрута, для которого будут загружены данные. Затем отображаются билеты, относящиеся к выбранному маршруту, с возможностью фильтрации по конкретному расписанию (рейсу) этого маршрута. Доступны кнопки "Детали" для просмотра информации о билете и "Отменить Билет" для отмены (только если билет не продан).

Вкладка "Продажи и Отчеты":

"Продажи билетов": Открывает форму frmSalesManagement для управления записями о продажах.

"Отчет по Доходам": Открывает форму frmIncomeReport.

"Статистика Продаж": Открывает форму frmSalesStatistics (может быть доступно администратору).

Вкладка "Управление складом": (Если функциональность склада реализована и доступна) Содержит кнопки для управления складскими операциями (Приход, Расход, Перемещение, Инвентаризация). Доступ может быть ограничен.

Вкладка "Система": (Доступно администратору)

"Пользователи": Открывает форму управления учетными записями пользователей.

"Права доступа": Открывает форму настройки прав доступа.

"Помощь": Открывает справочную информацию или руководство пользователя.

Меню приложения (кнопка слева вверху):

"О программе": Открывает окно AboutWindow с информацией о программе.

"Выйти из системы": Завершает текущий сеанс и возвращает к окну авторизации.

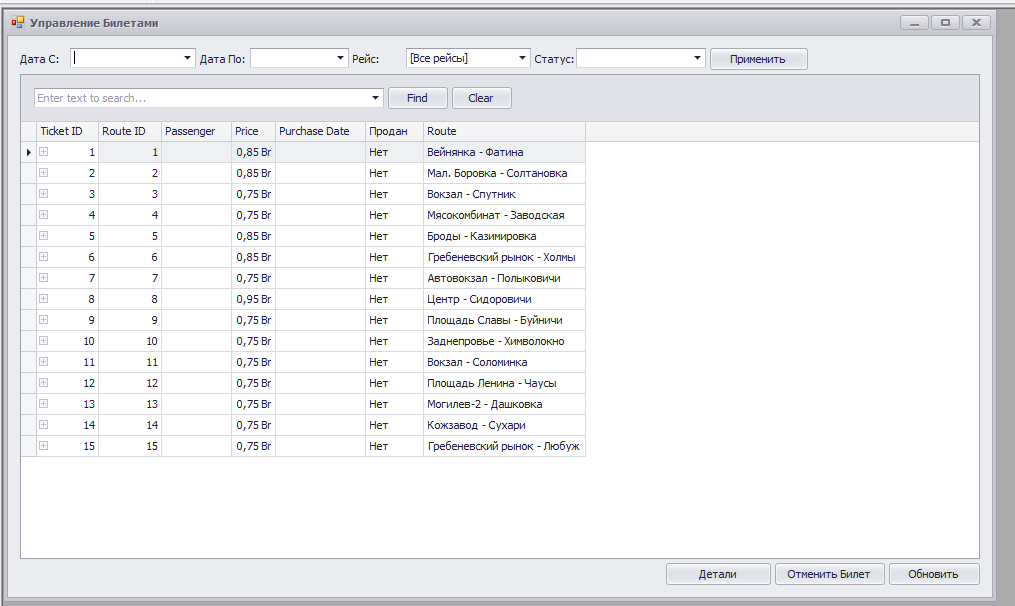


Рисунок 6 – Форма Управление билетами

Описание формы "Управление Билетами": Данная форма предназначена для просмотра и управления билетами, связанными с конкретным маршрутом, выбранным пользователем при открытии формы или при обновлении данных.

Предварительный выбор маршрута: При первом открытии формы или при нажатии "Обновить" сначала появляется диалоговое окно для выбора основного маршрута (Marshut). Только после выбора маршрута загружаются данные для этой формы.

Фильтры:

Рейс: (Выпадающий список lueRouteScheduleFilter): Позволяет выбрать конкретное расписание (рейс), относящееся к ранее выбранному основному маршруту. По умолчанию установлено "[Все рейсы]", что отображает билеты для всех расписаний выбранного маршрута.

Статус: (Выпадающий список cboStatusFilter): Позволяет фильтровать билеты по их статусу (например, "Продан", "Доступен"). *Примечание: Реализация этого фильтра может потребовать доработки.*

Дата С: / Дата По: (Поля выбора даты dateFromFilter, dateToFilter): Позволяют задать временной диапазон для фильтрации (например, по дате покупки или дате рейса). *Примечание: Фильтрация по дате в текущей реализации кода может отсутствовать или требовать уточнения.*

Кнопка Применить: Применяет выбранные значения фильтров к отображаемым данным.

Таблица (gridControlTickets): Отображает список билетов, соответствующих выбранным фильтрам. Колонки включают:

Ticket ID: Уникальный идентификатор билета.

Route ID: Идентификатор маршрута (Marshut), к которому относится билет.

Passenger: Имя пассажира (если билет продан, извлекается из данных продажи).

Price: Цена билета (форматируется как валюта).

Purchase Date: Дата и время покупки билета (если продан, извлекается из данных продажи).

Продан: Статус билета ("Да" или "Нет"), определяемый наличием связанных записей о продажах.

Route: Отображение маршрута (Начальный пункт - Конечный пункт), связанного с билетом.

Поиск: Поле "Enter text to search..." позволяет выполнять быстрый поиск по видимым данным в таблице.

Кнопки управления (внизу):

Детали: Открывает окно с подробной информацией о выбранном в таблице билете.

Отменить Билет: Позволяет отменить выбранный билет. Кнопка активна только если билет не имеет связанных записей о продажах (т.е., статус "Продан" - "Нет"). Перед отменой запрашивается подтверждение.

Обновить: Перезагружает данные для формы, снова запрашивая выбор основного маршрута, а затем загружая связанные расписания и все билеты.

При необходимости удалить запись (также доступна только администратору), необходимо нажать на неё и затем на кнопку Удалить. Программа попросит подтверждение необратимого действия (рисунок 7). Если пользователь согласен, выбранная запись будет удалена и пропадет из таблицы.

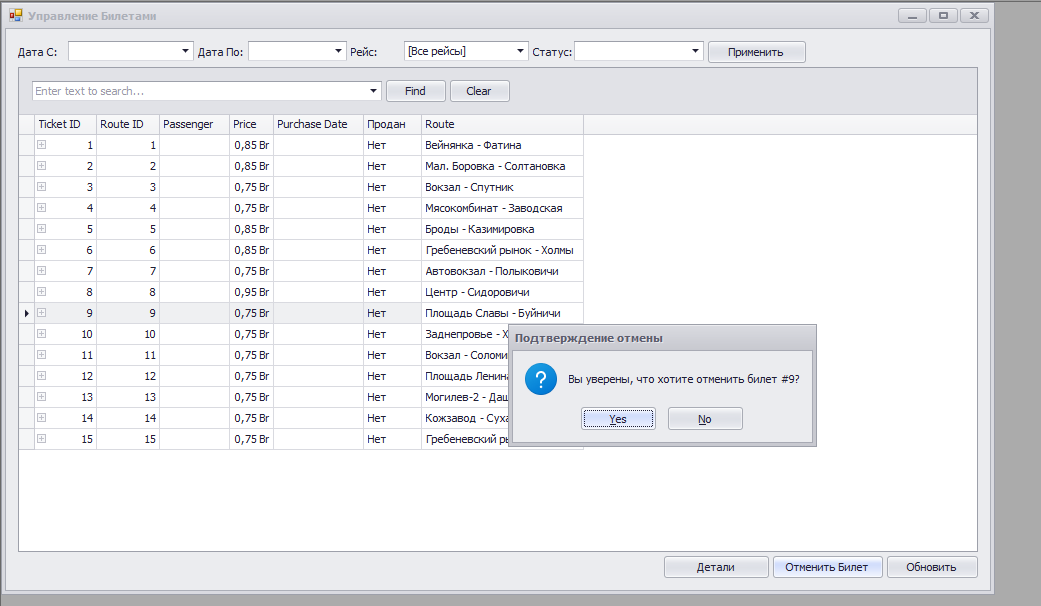


Рисунок 7 – Запрос о отмене билета

Описание формы "Управление расписаниями рейсов": Эта форма позволяет управлять расписаниями (конкретными рейсами) для различных маршрутов.

Фильтры:

Маршрут: (Выпадающий список lueRouteFilter): Обязательный фильтр для выбора основного маршрута, расписания которого будут отображены. Загрузка данных происходит только для выбранного маршрута.

Дата: (Поле выбора даты dateFilter): Позволяет фильтровать расписания по конкретной дате отправления. Можно оставить пустым для отображения расписаний на все даты для выбранного маршрута.

Таблица (gridControlSchedules): Отображает список расписаний, соответствующих выбранному маршруту и дате. Колонки включают ID расписания, ID маршрута, Пункты отправления/назначения, Время отправления/прибытия, Цену, Доступные места и другую релевантную информацию.

Кнопки управления:

Добавить: Открывает диалоговое окно для создания нового расписания рейса. Доступно администратору.

Редактировать: Открывает диалоговое окно для изменения выбранного в таблице расписания. Доступно администратору.

Удалить: Удаляет выбранное расписание после подтверждения. Доступно администратору.

Обновить: Перезагружает список расписаний для текущего выбранного маршрута и даты.

Диалог Добавления/Редактирования:

Позволяет выбрать Маршрут (из загруженного списка), указать Дату и Время отправления/прибытия, Цену билета, Количество доступных мест, а также статус активности расписания.

При сохранении выполняется валидация данных (например, время отправления должно быть раньше времени прибытия).

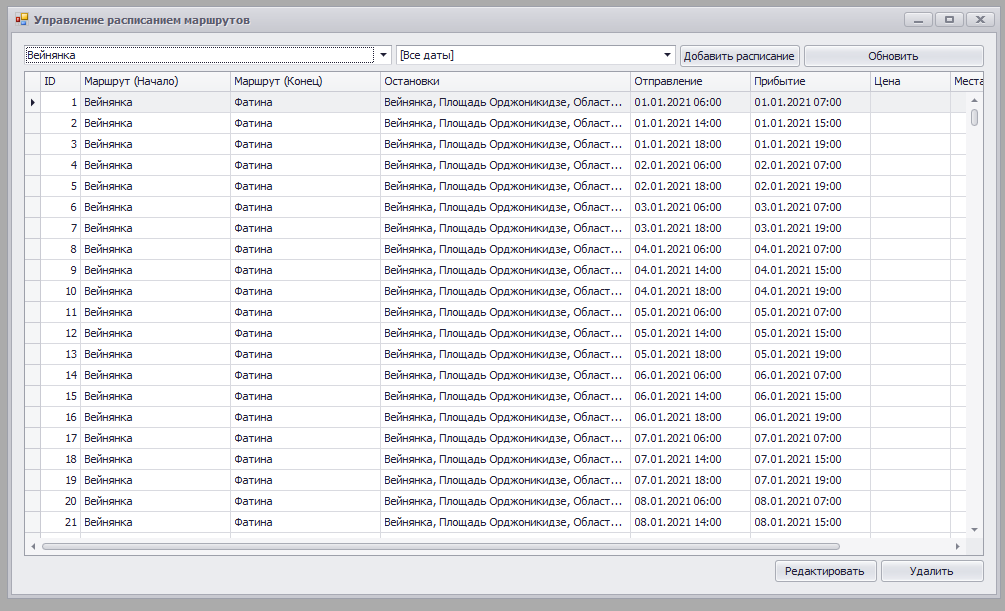


Рисунок 8 – Форма Расписание маршрутов

Кнопки Удалить, Сохранить и Показать все работают аналогично кнопке на форме Управление Билетами

Описание формы "Управление Маршрутами":Форма предназначена для ведения справочника автобусных маршрутов.

Таблица: Отображает список маршрутов с колонками ID Маршрута, Начальный пункт, Конечный пункт.

Кнопки управления: "Добавить", "Редактировать", "Удалить" (доступны администратору), "Обновить".

Диалог Добавления/Редактирования: Позволяет ввести или изменить Начальный и Конечный пункты для маршрута.

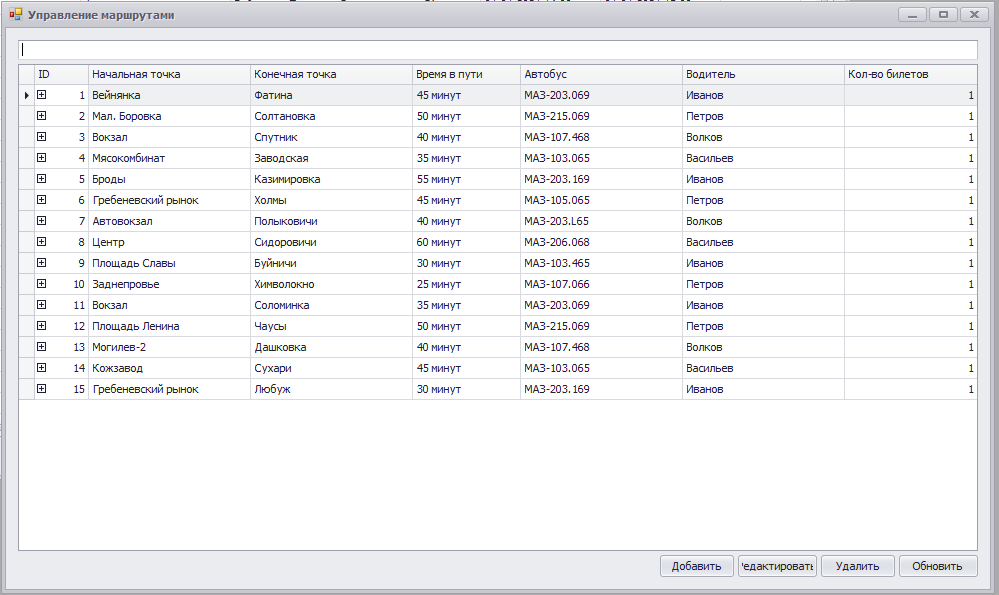


Рисунок 9 – Форма Управление Маршрутами

Описание формы "Управление Должностями": Форма доступна администратору и предназначена для ведения справочника должностей.

Таблица: Отображает список должностей с колонками ID Должности, Название должности.

Кнопки управления: "Добавить", "Редактировать", "Удалить" (доступны администратору), "Обновить".

Диалог Добавления/Редактирования: Позволяет ввести или изменить Название должности.

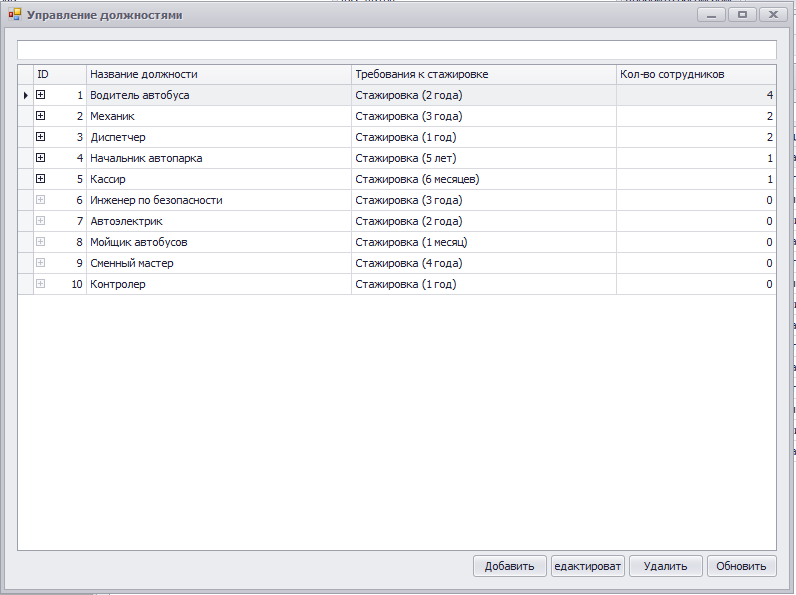


Рисунок 10 – Форма Управление Должностями

Описание формы "Управление Персоналом": Форма для управления данными сотрудников предприятия.

Поле поиска (txtSearch): Позволяет искать сотрудников по ФИО или должности.

Таблица (gridControlEmployees): Отображает список сотрудников с колонками ID, Фамилия, Имя, Отчество, Должность, Дата приема.

Кнопки управления: "Добавить", "Редактировать", "Удалить" (доступны администратору), "Обновить".

Диалог Добавления/Редактирования: Позволяет ввести или изменить Фамилию, Имя, Отчество сотрудника, Дату приема на работу и выбрать Должность из выпадающего списка (загруженного из справочника должностей).

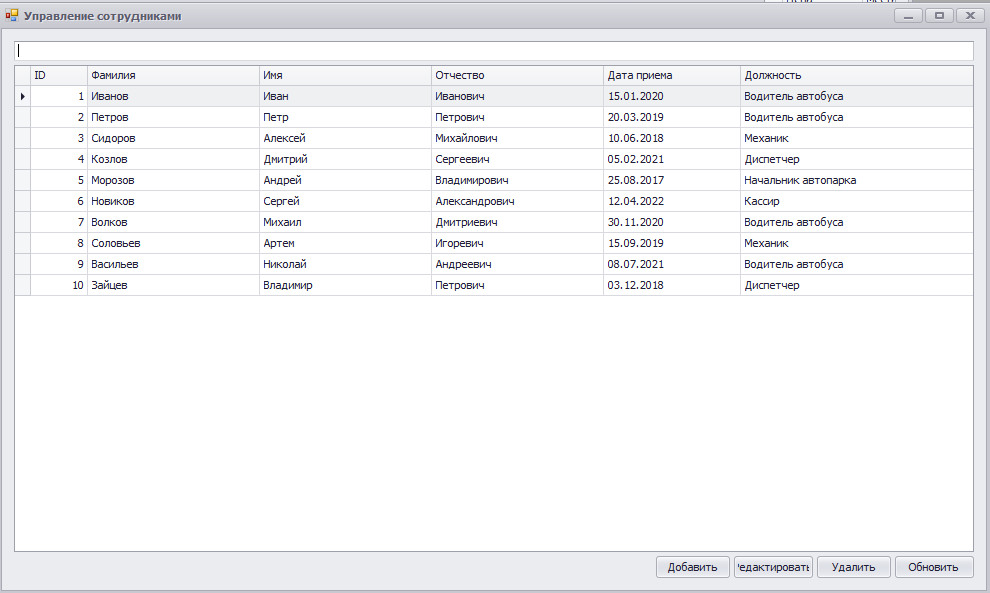


Рисунок 11 – Управление Персоналом

Форма Продажа билетов служит для показа проданных билетов и служит для показа проданных билетов в выбранную дату и по выбранному маршруту. Вид формы представлен на рисунке 12

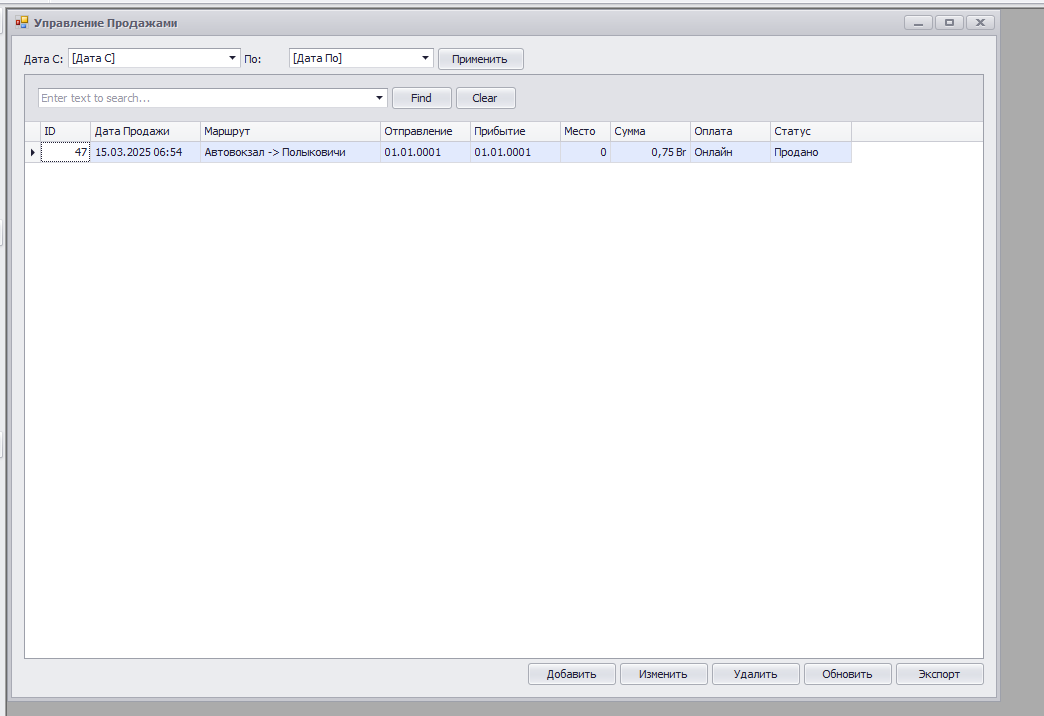


Рисунок 12 – Форма Продажа билетов

Также в программе присутствует формы отчета о доходах в которой показывается общий доход за определенный период времени

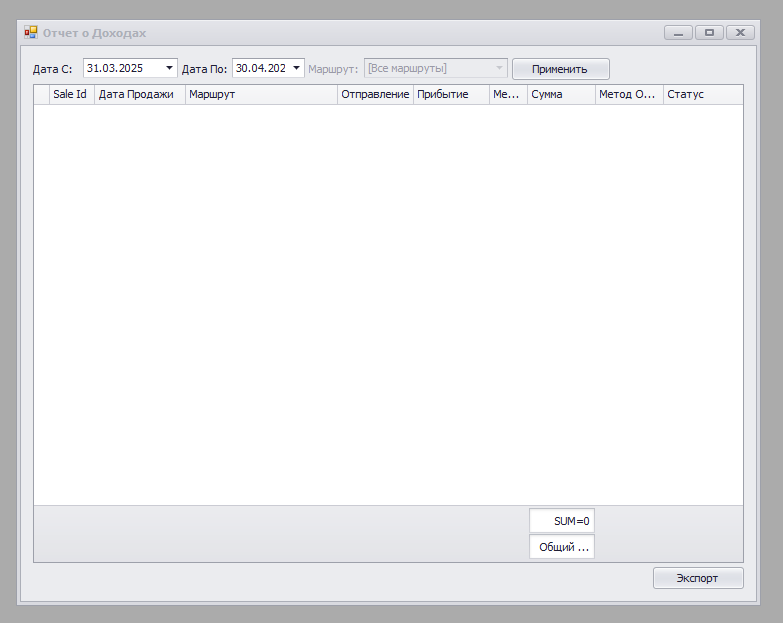


Рисунок 13 – Форма отчета о доходах

Также в программе присутствует форма статистики продаж, показывающая общую статистику продаж билетов

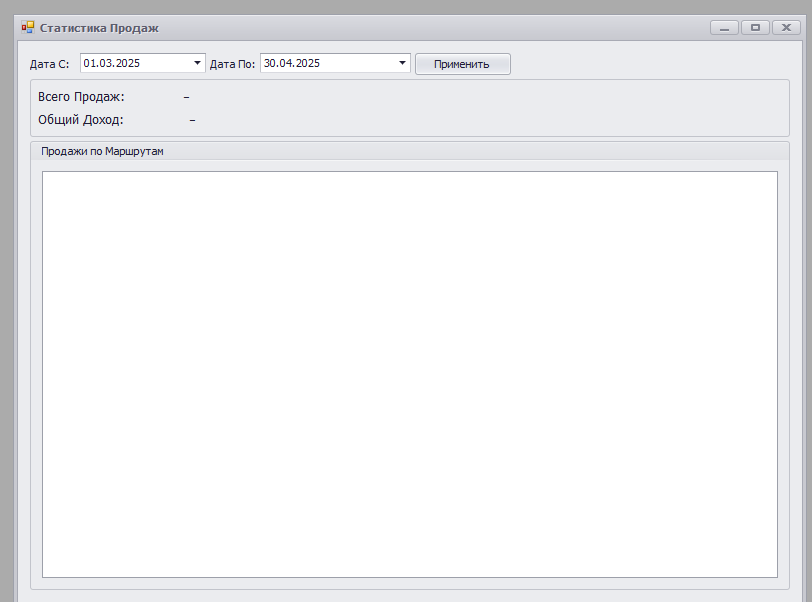


Рисунок 13 – Форма статистики продаж

Завершение работы в приложение возможно после закрытия формы Меню. Это позволяет приложению вернуться к авторизации. Закрытие приложения возможно либо нажатием кнопки Выход, либо нажатием на крестик в верхнем правом углу.

## 3.4 Сообщения оператору

При установке галочки Запомнить меня и последующего нажатия кнопки Войти на форме авторизации, пользователю будет выведено сообщение об успешном сохранении его данных (рисунок 14).

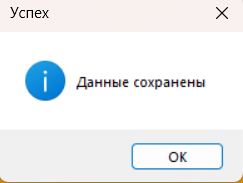


Рисунок 14 – Сообщение о сохранении данных пользователя

Если введенные пользователем данные неверны и такой ученой записи не существует ему выводить сообщение об ошибке (рисунок 15).

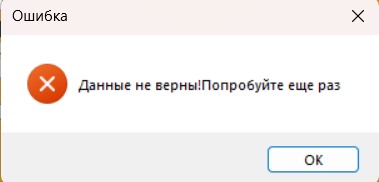


Рисунок 15 – Сообщение о ошибочных данных

При добавлении и сохранении данных пользователь получает сообщения о успешном сохранении (рисунок 16).

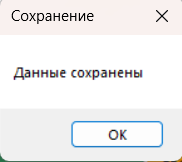


Рисунок 16 – Сообщение успешном сохранении

При нажатии кнопки Удалить пользователь получает предупреждение с запросом на подтверждение удаления (рисунок 17). При нажатии на кнопку ОК удаление происходит, а при нажатии на кнопку Отмена, отменяется.

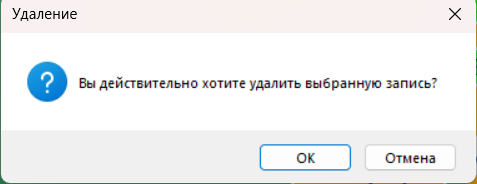


Рисунок 17 – Запрос подтверждения удаления

В случае возникновения исключительных ситуаций пользователю выводятся соответствующие сообщения в зависимости от ошибки.

# 4. Тестирование

Для проверки функциональности разработанного программного обеспечения были подготовлены и проведены тесты, охватывающие все ключевые сценарии взаимодействия пользователя с системой управления автобусным парком (TicketSalesApp.UI.LegacyForms.DX.Windows). В ходе тестирования было установлено, что приложение соответствует заявленным требованиям и функционирует корректно. Результаты проведенного тестирования сведены в «Журнал тестирования программного продукта» (таблица 2).

Таблица 2 – Журнал тестирования программного продукта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Объект | Команда пользователя | Ожидаемая реакция | Отметка об отклике системы |
| 1 | Окно входа (frmLogin) | Ввести имя пользователя | Отображение имени пользователя и переход к вводу пароля или генерации QR-кода | Работает корректно |
| 2 | Окно входа (frmLogin) | Ввести пароль и нажать "Войти" | Проверка учетных данных через API (AuthController), авторизация пользователя, открытие главной формы (Form1) | Работает корректно |
| 3 | Окно входа (frmLogin) | Нажать "Обновить QR" | Генерация нового QR-кода для входа через API (AuthController) | Работает корректно |
| 4 | Окно входа (frmLogin) | Сканирование QR-кода (симуляция) | Проверка QR-токена через API (AuthController), авторизация, открытие главной формы (Form1) | Работает корректно |
| 5 | Окно входа (frmLogin) | Авторизация | Сохранение имени пользователя для последующих входов (WriteLoginUnitXML) | Данные записываются в файл |
| 6 | Окно входа (frmLogin) | Авторизация | При неправильном вводе имени пользователя или пароля выдается ошибка и возврат к вводу имени пользователя | Неправильный  ввод |
| 7 | Главная форма (Form1) | Нажатие "Выход" (в меню) | Закрытие приложения | Работает корректно |
| 8 | Главная форма (Form1) | Нажатие "О программе" (в меню) | Появление модального окна AboutWindow с информацией о программе | Работает корректно |
| 9 | Главная форма (Form1) | Нажатие кнопок управления | Открытие соответствующей формы управления (Автобусы, Маршруты, Сотрудники и т.д.) с учетом прав доступа | Формы открываются корректно |
| 10 | Формы управления (общие) | Нажатие "Добавить" | Открытие формы/диалога для ввода данных, добавление записи через API | Данные добавляются в базу |
| 11 | Формы управления (общие) | Нажатие "Редактировать" | Открытие формы/диалога с данными выбранной записи, обновление записи через API | Данные обновляются в базе |
| 12 | Формы управления (общие) | Нажатие "Удалить" | Вывод запроса на подтверждение, удаление записи через API | Работает корректно |
| 13 | Формы управления (общие) | Нажатие "Обновить" | Перезагрузка данных из API и обновление таблицы | Работает корректно |
| 14 | Формы управления (общие) | Ввод текста в поле поиска/фильтра | Фильтрация отображаемых данных в таблице по введенному тексту | Работает корректно |
| 15 | Управление билетами (frmTicketManagement) | Нажатие "Просмотр деталей" | (Действие не реализовано) Открытие окна с детальной информацией о билете | - |
| 16 | Управление билетами (frmTicketManagement) | Нажатие "Отменить билет" | (Действие не реализовано) Отправка запроса на отмену билета через API | - |
| 17 | Отчет по доходам (frmIncomeReport) | Выбор фильтров и нажатие "Применить" | Загрузка и отображение отфильтрованных данных о продажах | Работает корректно |
| 18 | Отчет по доходам (frmIncomeReport) | Нажатие "Экспорт" | Открытие диалога сохранения файла и экспорт данных из таблицы | Работает корректно |
| 19 | Статистика продаж (frmSalesStatistics) | Выбор фильтров и нажатие "Применить" | Загрузка данных статистики через API (TicketSalesController), отображение диаграмм и итоговых сумм | Работает корректно |
| 20 | Формы управления/отчетов | Нажатие "Закрыть" (крестик окна) | Закрытие текущей формы | Работает корректно |
| 21 | Все формы просмотра данных | Загрузка данных | При непредвиденной ситуации (например резкое отключение сервера) загрузка данных будет автоматически остановлена и выведено сообщение о ошибке | Работает корректно |
| 22 | При критеческой ошибке в пользовательской части | Перезапуск программы в режиме обработки ошибок | При непредвиденной ситуации или ошибке в коде пользовательской части программа автоматически перезапуститься в режиме обработки ошибок и покажет диагностическую ииформацию | Ошибка пользовательской части |

# Заключение

В результате выполнения курсовой работы была разработана система управления автобусным парком (TicketSalesApp). Она предоставляет администраторам и сотрудникам функционал, необходимый для эффективного управления ресурсами и операциями автопарка. Программа в удобной форме обеспечивает возможность управления автобусами, маршрутами, расписаниями, сотрудниками, должностями, записями о техническом обслуживании и продажами билетов. Реализованы функции добавления, редактирования и удаления соответствующей информации. Данные системы хранятся в централизованной базе данных, доступ к которой осуществляется через разработанный API (TicketSalesApp.AdminServer). Тестированием была подтверждена корректная работа как серверной части, так и клиентских приложений (TicketSalesApp.UI.LegacyForms.DX.Windows,

TicketSalesApp.UI.Administration.Avalonia, TicketSalesApp.UI.Avalonia).

При выполнении курсовой работы были применены знания современных подходов к разработке программного обеспечения, включая:

Проектирование и использование баз данных (с применением Entity Framework Core).

Построение многоуровневых приложений (API бэкенд, несколько UI фронтендов).

Разработку RESTful API с использованием ASP.NET Core.

Объектно-ориентированное программирование на языке C#.

Навыки создания пользовательских интерфейсов для Windows с использованием Windows Forms (DevExpress) и кроссплатформенных интерфейсов с использованием Avalonia UI.

Реализацию механизмов аутентификации и авторизации (JWT, ролевая модель).

В ходе выполнения курсовой работы были закреплены основы и углублены знания приемов программирования на языке C# и платформе .NET. При реализации проекта получены практические навыки на всех этапах создания программного продукта: от постановки задачи и проектирования архитектуры до практической реализации API, клиентских приложений и документирования.

В результате выполнения курсовой работы были выбраны подходящая среда разработки (Visual Studio / Rider), СУБД (MS SQL Server), приведены используемые библиотеки и фреймворки (ASP.NET Core, Entity Framework Core, Avalonia UI, DevExpress WinForms).

Разработаны функциональные пользовательские интерфейсы для различных платформ. Разработанная система может использоваться автотранспортными предприятиями для автоматизации учета и управления автобусным парком, маршрутами, расписаниями и продажами билетов.

В дальнейшем система может быть дополнена:

Интеграцией с платежными системами для онлайн-продажи билетов.

Разработкой мобильных приложений для пассажиров и водителей.

Реализацией функционала отслеживания автобусов в реальном времени.

Расширением системы отчетности и аналитики.

Внедрением более гранулярной системы прав доступа на основе разрешений (PermissionsController, RolesController).

# Список использованных источников

1. Волков О. И. Экономика предприятия: Учебное пособие / О.И. Волков, В.К. Скляренко. — 4-e изд. — М.: НИЦ ИНФРА-М, 2017. — 264 с.
2. Васильев, А. Н. C#. Объектно-ориентированное программирование. Учебный курс / А. Н. Васильев. – СПб. : Питер, 2012. –320 с.: ил.
3. Биллиг, В. А. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008) / В.А. Биллиг. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2017. - 584 c.
4. Герман, О.В Программирование на Java и C# для студентов / О.В Герман. - СПб.: BHV, 2005. - 512 c.
5. Голицына, О.Л. Базы данных : учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М. : Форум : Инфра-М, 2005. - 352с.

# Приложение А

(обязательное)

Текст программы

using DevExpress.XtraEditors;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TicketSalesApp.UI.LegacyForms.DX.Windows

{

    public partial class AboutWindow : DevExpress.XtraEditors.XtraForm

    {

        public AboutWindow()

        {

            InitializeComponent();

        }

        private void label2\_Click(object sender, EventArgs e)

        {

        }

    }

}

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Security.Cryptography;

using System.Text;

using System.Threading;

using Serilog;

using NLog;

using Newtonsoft.Json.Linq;

using Newtonsoft.Json;

namespace TicketSalesApp.UI.LegacyForms.DX.Windows

{

    public class ApiClientService

    {

        private static ApiClientService \_instance;

        private static readonly object \_lock = new object();

        private static readonly Logger Log = LogManager.GetCurrentClassLogger();

        private string \_authToken;

        private int? \_userRole;

        private string \_userName;

        private Timer \_refreshTimer;

        private const int RefreshIntervalMinutes = 25; // Token refresh interval

        public static ApiClientService Instance

        {

            get

            {

                if (\_instance == null)

                {

                    lock (\_lock)

                    {

                        if (\_instance == null)

                        {

                           \_instance = new ApiClientService();

                        }

                    }

                }

                return \_instance;

            }

        }

        private ApiClientService()

        {

        }

        public string AuthToken

        {

            get { return \_authToken; }

            set

            {

                \_authToken = value;

                ParseTokenAndStoreInfo(value);

                var handler = OnAuthTokenChanged;

                if (handler != null)

                {

                    handler(this, value);

                }

            }

        }

        public event Action<object, string> OnAuthTokenChanged;

        public HttpClient CreateClient()

        {

            var client = new HttpClient

            {

                BaseAddress = new Uri("http://localhost:5000/api/")

            };

            if (!string.IsNullOrEmpty(\_authToken))

            {

                client.DefaultRequestHeaders.Authorization =

                    new AuthenticationHeaderValue("Bearer", \_authToken);

            }

            // Set the default request headers to accept UTF-8 encoded responses

            client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json", 1.0));

            client.DefaultRequestHeaders.AcceptCharset.Add(new StringWithQualityHeaderValue("utf-8"));

            return client;

        }

        public int? UserRole

        {

            get { return \_userRole; }

        }

        public string UserName

        {

            get { return \_userName; }

        }

        private void ParseTokenAndStoreInfo(string token)

        {

            \_userRole = null;

            \_userName = null;

            if (string.IsNullOrEmpty(token))

            {

                Log.Debug("Token is null or empty, clearing role and username.");

                return;

            }

            try

            {

                var parts = token.Split('.');

                if (parts.Length < 2)

                {

                    Log.Warn("JWT token does not contain enough parts.");

                    return;

                }

                string payload = parts[1];

                payload = payload.Replace('-', '+').Replace('\_', '/');

                switch (payload.Length % 4)

                {

                    case 2: payload += "=="; break;

                    case 3: payload += "="; break;

                }

                byte[] data = Convert.FromBase64String(payload);

                string decodedPayload = Encoding.UTF8.GetString(data);

                Log.Debug("Decoded JWT Payload: {0}", decodedPayload);

                JObject jsonPayload = JObject.Parse(decodedPayload);

                JToken roleToken;

                if (jsonPayload.TryGetValue("role", StringComparison.OrdinalIgnoreCase, out roleToken))

                {

                    try

                    {

                        \_userRole = roleToken.Value<int>();

                        Log.Info("User role parsed from token: {0}", \_userRole);

                    }

                    catch (Exception exConv)

                    {

                         string roleValueStr = roleToken.ToString();

                         string errorMsg = string.Format("Failed to convert role claim '{0}' to integer. Exception: {1}", roleValueStr, exConv.ToString());

                         Log.Error(errorMsg);

                        \_userRole = null;

                    }

                }

                else

                {

                    Log.Warn("JWT payload does not contain 'role' claim.");

                    \_userRole = null;

                }

                JToken nameToken;

                if (jsonPayload.TryGetValue("unique\_name", StringComparison.OrdinalIgnoreCase, out nameToken) ||

                    jsonPayload.TryGetValue("name", StringComparison.OrdinalIgnoreCase, out nameToken))

                {

                     \_userName = nameToken.Value<string>();

                     Log.Info("Username parsed from token: {0}", \_userName);

                }

                else

                {

                     Log.Warn("JWT payload does not contain standard username claim ('unique\_name' or 'name').");

                     \_userName = null;

                }

            }

            catch (FormatException exFormat)

            {

                string errorMsg = string.Format("Failed to decode Base64 payload. Exception: {0}", exFormat.ToString());

                Log.Error(errorMsg);

            }

            catch (JsonReaderException exJson)

            {

                 string errorMsg = string.Format("Failed to parse JSON payload. Exception: {0}", exJson.ToString());

                 Log.Error(errorMsg);

            }

            catch (Exception ex)

            {

                string errorMsg = string.Format("An unexpected error occurred while parsing JWT token. Exception: {0}", ex.ToString());

                Log.Error(errorMsg);

            }

        }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using DevExpress.XtraBars; // Required for BarItem, ItemClickEventArgs

using DevExpress.XtraBars.Ribbon; // Required for RibbonForm

using NLog;

namespace TicketSalesApp.UI.LegacyForms.DX.Windows

{

    // Inherit from RibbonForm instead of XtraForm

    public partial class Form1 : RibbonForm

    {

        // Add static logger field

        private static readonly Logger Log = LogManager.GetCurrentClassLogger();

        public Form1(bool loginresult)

        {

            InitializeComponent();

            if (!loginresult)

            {

                // Optional: Handle unsuccessful login, maybe close the app or show a message

                // For now, just preventing menu handlers from being attached if login failed.

                 this.Load += (s, e) => Close();

                return;

            }

            // Wire up menu item clicks programmatically

            WireUpEventHandlers();

            // Set status bar info from ApiClientService

            bsiCompanyName.Caption = "Название компании: Your Company";

            // Apply role-based permissions

            ApplyPermissions();

            // Set user info in status bar

            UpdateStatusBarUserInfo();

        }

        private void WireUpEventHandlers()

        {

             // --- Wire up Ribbon BarButtonItems ---

             if (this.bbiBuses != null)

                 this.bbiBuses.ItemClick += BaseItem1\_Click; // Buses

             if (this.bbiRoutes != null)

                 this.bbiRoutes.ItemClick += BaseItem4\_Click; // Routes

             if (this.bbiEmployees != null)

                 this.bbiEmployees.ItemClick += BaseItem5\_Click; // Employees

             if (this.bbiJobs != null)

                 this.bbiJobs.ItemClick += BaseItem2\_Click; // Jobs

             if (this.bbiMaintenance != null)

                 this.bbiMaintenance.ItemClick += BaseItem3\_Click; // Maintenance

             if (this.bbiRouteSchedules != null)

                 this.bbiRouteSchedules.ItemClick += BaseItem6\_Click; // Route Schedules

             if (this.bbiTickets != null)

                 this.bbiTickets.ItemClick += MenuItemTicketManagement\_Click; // Tickets

             if (this.bbiSales != null)

                 this.bbiSales.ItemClick += MenuItemSalesManagement\_Click; // Sales

             if (this.bbiIncomeReport != null)

                 this.bbiIncomeReport.ItemClick += MenuItemIncomeReport\_Click; // Income Report

             if (this.bbiSalesStatistics != null)

                 this.bbiSalesStatistics.ItemClick += MenuItemSalesStatistics\_Click; // Sales Statistics

             // --- Wire up Application Menu Items ---

             if (this.bbiAbout != null)

                 this.bbiAbout.ItemClick += оПрограммеToolStripMenuItem\_Click; // About

             if (this.bbiExit != null)

                 this.bbiExit.ItemClick += ExitApplication\_Click; // Exit

            // --- Wire up NavBar Items (Optional but good for consistency) ---

            // Note: These often mirror Ribbon items. Ensure names match designer.

            // Example: If navBarItem\_BusManagement exists and should open Bus Management:

            // if (this.navBarItem\_BusManagement != null)

            //    this.navBarItem\_BusManagement.LinkClicked += (s, e) => ShowMdiChildForm(new frmBusManagement());

            // Add similar handlers for other relevant NavBarItems if you keep the NavBarControl fully functional.

            // --- Existing Inventory/System Admin (Map if needed) ---

            // Example: Map User Management if a corresponding BarButtonItem 'bbiUserManagement' exists

            // if (this.bbiUserManagement != null)

            //     this.bbiUserManagement.ItemClick += TsbUserManage\_Click;

            // Add handlers for Inventory Items if corresponding BarButtonItems exist

            // if (this.bbiStockIn != null) this.bbiStockIn.ItemClick += YourStockInHandler;

            // if (this.bbiStockOut != null) this.bbiStockOut.ItemClick += YourStockOutHandler;

            // ... etc ...

        }

        // --- Event Handler Methods (Reused from original) ---

        private void BaseItem1\_Click(object sender, ItemClickEventArgs e) // Buses (Triggered by bbiBuses)

        {

            ShowMdiChildForm(new frmBusManagement());

        }

        private void BaseItem4\_Click(object sender, ItemClickEventArgs e) // Routes (Triggered by bbiRoutes)

        {

            ShowMdiChildForm(new frmRouteManagement());

        }

        private void BaseItem5\_Click(object sender, ItemClickEventArgs e) // Employees (Triggered by bbiEmployees)

        {

            ShowMdiChildForm(new frmEmployeeManagement());

        }

        private void BaseItem2\_Click(object sender, ItemClickEventArgs e) // Jobs (Triggered by bbiJobs)

        {

            ShowMdiChildForm(new frmJobManagement());

        }

        private void BaseItem3\_Click(object sender, ItemClickEventArgs e) // Maintenance (Triggered by bbiMaintenance)

        {

            ShowMdiChildForm(new frmMaintenanceManagement());

        }

        private void BaseItem6\_Click(object sender, ItemClickEventArgs e) // Route Schedules (Triggered by bbiRouteSchedules)

        {

            ShowMdiChildForm(new frmRouteSchedulesManagement());

        }

        private void MenuItemTicketManagement\_Click(object sender, ItemClickEventArgs e) // Ticket Management (Triggered by bbiTickets)

        {

            ShowMdiChildForm(new frmTicketManagement());

        }

        private void MenuItemSalesManagement\_Click(object sender, ItemClickEventArgs e) // Sales (Triggered by bbiSales)

        {

             ShowMdiChildForm(new frmSalesManagement());

        }

        private void MenuItemSalesStatistics\_Click(object sender, ItemClickEventArgs e) // Sales Statistics (Triggered by bbiSalesStatistics)

        {

            ShowMdiChildForm(new frmSalesStatistics());

        }

        private void MenuItemIncomeReport\_Click(object sender, ItemClickEventArgs e) // Income Report (Triggered by bbiIncomeReport)

        {

            ShowMdiChildForm(new frmIncomeReport());

        }

        private void оПрограммеToolStripMenuItem\_Click(object sender, ItemClickEventArgs e) // About (Triggered by bbiAbout)

        {

            ShowMdiChildForm(new AboutWindow());

        }

        private void ExitApplication\_Click(object sender, ItemClickEventArgs e) // Exit (Triggered by bbiExit)

        {

             Application.Exit();

        }

        // Placeholder for User Management click (if implemented)

        private void TsbUserManage\_Click(object sender, ItemClickEventArgs e)

        {

             // Example: ShowMdiChildForm(new frmUserManagement());

             MessageBox.Show("User Management Clicked (Not Implemented)");

        }

        // Placeholder for Inventory clicks (if implemented)

        private void 库存业务ToolStripMenuItem\_Click(object sender, EventArgs e)

        {

            // This handler might become obsolete if using Ribbon items directly

             MessageBox.Show("Inventory Business Clicked (Not Implemented via Menu)");

        }

        // --- Helper Method to Show MDI Child Forms (Prevents Duplicates) ---

        private void ShowMdiChildForm(Form childForm)

        {

            // Check if a form of the same type is already open

            foreach (Form openForm in this.MdiChildren)

            {

                if (openForm.GetType() == childForm.GetType())

                {

                    openForm.MdiParent = this; // Ensure it's still parented correctly

                    openForm.Activate(); // Bring existing form to front

                    childForm.Dispose(); // Dispose the new instance we were about to show

                    return;

                }

            }

            // If not open, set MdiParent and show

            childForm.MdiParent = this;

            childForm.Show();

        }

        // --- Apply Permissions Based on Role ---

        private void ApplyPermissions()

        {

            var role = ApiClientService.Instance.UserRole;

            Log.Debug("Applying permissions for role: {0}", role.HasValue ? role.Value.ToString() : "None");

            bool isAdmin = role.HasValue && role.Value == 1;

            // --- Ribbon Button Visibility/Enabled State ---

            // Assumed Admin-Only functions

            if (bbiUserManagement != null) bbiUserManagement.Enabled = isAdmin;

            if (bbiPermissions != null) bbiPermissions.Enabled = isAdmin;

            if (bbiJobs != null) bbiJobs.Enabled = isAdmin; // Example: Only admins manage job titles

            // Add other admin-only buttons here...

            // Example: Potentially Maintenance, Sales Statistics etc might be admin only

            if (bbiMaintenance != null) bbiMaintenance.Enabled = isAdmin;

            if (bbiSalesStatistics != null) bbiSalesStatistics.Enabled = isAdmin;

            // --- Potentially Hide Entire Ribbon Pages/Groups for non-admins ---

            // Example: Hide System Admin Page for non-admins

            if (ribbonPage4 != null) ribbonPage4.Visible = isAdmin;

            // Example: Hide Inventory Management Page (if permissions require)

            if (ribbonPage3 != null) ribbonPage3.Visible = isAdmin; // Or based on specific role

            // Example: Maybe regular users can only view tickets/schedules but not manage them

            // You might need finer control than just enabling/disabling top-level buttons.

            // This might involve passing the role to the child forms themselves.

            // --- NavBar Control Item Visibility/Enabled State (Example) ---

            /\*

            if (navBarControl2 != null)

            {

               // Example: Hide System Admin group

               // Find NavBarGroup by Caption or Name

               var adminGroup = navBarControl2.Groups.FirstOrDefault(g => g.Caption == "System Administration");

               if (adminGroup != null) adminGroup.Visible = isAdmin;

               // Example: Disable specific items

               var jobItem = navBarControl2.Items.FirstOrDefault(i => i.Name == "navBarItem\_Jobs"); // Assuming Name property is set in designer

               if (jobItem != null) jobItem.Enabled = isAdmin;

            }

            \*/

            Log.Debug("Permissions applied.");

        }

        // --- Update Status Bar ---

        private void UpdateStatusBarUserInfo()

        {

           string userName = ApiClientService.Instance.UserName;

           string roleDescription = "Пользователь"; // Default

           if (ApiClientService.Instance.UserRole.HasValue)

           {

               if (ApiClientService.Instance.UserRole.Value == 1) {

                   roleDescription = "Администратор";

               } // Add other roles if needed

           }

           bsiUserID.Caption = string.Format("Пользователь: {0} ({1})",

               string.IsNullOrEmpty(userName) ? "[Неизвестно]" : userName,

               roleDescription);

        }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using DevExpress.XtraEditors;

using DevExpress.XtraGrid.Views.Base;

using TicketSalesApp.Core.Models;

using Serilog;

using Newtonsoft.Json;

using Newtonsoft.Json.Serialization;

using DevExpress.XtraLayout;

namespace TicketSalesApp.UI.LegacyForms.DX.Windows

{

    public partial class frmBusManagement : DevExpress.XtraEditors.XtraForm

    {

        private readonly ApiClientService \_apiClient;

        private List<Avtobus> \_allBuses = new List<Avtobus>();

        private readonly string \_baseUrl = "http://localhost:5000/api";

        private readonly JsonSerializerSettings \_jsonSettings = new JsonSerializerSettings

        {

            PreserveReferencesHandling = PreserveReferencesHandling.Objects,

            ReferenceLoopHandling = ReferenceLoopHandling.Ignore

        };

        #region Helper: Input Dialog

        private static string ShowInputDialog(string caption, string prompt, string defaultValue = "")

        {

            using (var form = new XtraForm())

            {

                form.Text = caption;

                form.StartPosition = FormStartPosition.CenterParent;

                form.FormBorderStyle = FormBorderStyle.FixedDialog;

                form.MinimizeBox = false;

                form.MaximizeBox = false;

                form.ClientSize = new Size(350, 120);

                var lc = new LayoutControl() { Dock = DockStyle.Fill };

                form.Controls.Add(lc);

                var txtInput = new TextEdit();

                txtInput.Text = defaultValue;

                var btnOK = new SimpleButton() { Text = "OK" };

                var btnCancel = new SimpleButton() { Text = "Отмена" };

                btnOK.DialogResult = DialogResult.OK;

                btnCancel.DialogResult = DialogResult.Cancel;

                lc.Root.AddItem(prompt, txtInput).TextLocation = DevExpress.Utils.Locations.Top;

                var itemOK = lc.Root.AddItem(string.Empty, btnOK);

                var itemCancel = lc.Root.AddItem(string.Empty, btnCancel);

                // Basic horizontal button layout (adjust spacing/alignment as needed)

                lc.Root.AddItem(itemCancel).Move(itemOK, DevExpress.XtraLayout.Utils.InsertType.Right);

                form.AcceptButton = btnOK;

                form.CancelButton = btnCancel;

                if (form.ShowDialog() == DialogResult.OK)

                {

                    return txtInput.Text;

                }

                else

                {

                    return null; // Indicate cancellation

                }

            }

        }

        #endregion

        public frmBusManagement()

        {

            InitializeComponent();

            \_apiClient = ApiClientService.Instance;

            gridViewBuses.CustomUnboundColumnData += GridViewBuses\_CustomUnboundColumnData;

            UpdateButtonStates();

            \_apiClient.OnAuthTokenChanged += async delegate(object sender, string token) {

                if (this.Visible) {

                    await LoadBusesAsync();

                }

            };

        }

        private async void frmBusManagement\_Load(object sender, EventArgs e)

        {

            await LoadBusesAsync();

        }

        private async Task LoadBusesAsync()

        {

            HttpClient client = null;

            try

            {

                client = \_apiClient.CreateClient();

                var apiUrl = string.Format("{0}/Buses?includeRoutes=true&includeMaintenance=true", \_baseUrl);

                var response = await client.GetAsync(apiUrl);

                if (response.IsSuccessStatusCode)

                {

                    string jsonResponse = await response.Content.ReadAsStringAsync();

                    Log.Debug("Raw JSON response from /api/buses: {JsonResponse}", jsonResponse);

                    \_allBuses = JsonConvert.DeserializeObject<List<Avtobus>>(jsonResponse, \_jsonSettings);

                    if (\_allBuses == null) { \_allBuses = new List<Avtobus>(); }

                    FilterAndBindBuses();

                }

                else

                {

                    var error = await response.Content.ReadAsStringAsync();

                    Log.Error("Failed to load buses. Status: {StatusCode}, Error: {Error}", response.StatusCode, error);

                    XtraMessageBox.Show(string.Format("Не удалось загрузить автобусы: {0}", response.ReasonPhrase), "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                    \_allBuses = new List<Avtobus>();

                    FilterAndBindBuses();

                }

            }

            catch (Exception ex)

            {

                Log.Error(ex, "Exception loading buses");

                XtraMessageBox.Show(string.Format("Произошла ошибка при загрузке автобусов: {0}", ex.Message), "Критическая ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                \_allBuses = new List<Avtobus>();

                FilterAndBindBuses();

            }

            finally

            {

                 if (client != null) client.Dispose();

                 UpdateButtonStates();

            }

        }

        private void FilterAndBindBuses()

        {

            var searchText = txtSearch.Text.Trim();

            List<Avtobus> filteredBuses;

            if (string.IsNullOrWhiteSpace(searchText))

            {

                filteredBuses = \_allBuses;

            }

            else

            {

                filteredBuses = \_allBuses

                    .Where(delegate(Avtobus b) {

                        return b.Model != null && b.Model.IndexOf(searchText, StringComparison.OrdinalIgnoreCase) >= 0;

                     })

                    .ToList();

            }

            busBindingSource.DataSource = filteredBuses;

            gridControlBuses.RefreshDataSource();

        }

        private void GridViewBuses\_CustomUnboundColumnData(object sender, CustomColumnDataEventArgs e)

        {

            if (e.Column.FieldName == "Routes.Count" && e.IsGetData)

            {

                if (e.Row is Avtobus)

                {

                    Avtobus bus = (Avtobus)e.Row;

                    e.Value = (bus.Routes != null) ? bus.Routes.Count : 0;

                }

            }

            else if (e.Column.FieldName == "Obsluzhivanies.Count" && e.IsGetData)

            {

                if (e.Row is Avtobus)

                {

                    Avtobus bus = (Avtobus)e.Row;

                    e.Value = (bus.Obsluzhivanies != null) ? bus.Obsluzhivanies.Count : 0;

                }

            }

        }

        private async void btnAdd\_Click(object sender, EventArgs e)

        {

            string busModel = ShowInputDialog("Добавить автобус", "Введите модель нового автобуса:");

            if (!string.IsNullOrWhiteSpace(busModel))

            {

                var newBus = new Avtobus { Model = busModel };

                HttpClient client = null;

                try

                {

                    client = \_apiClient.CreateClient();

                    var json = JsonConvert.SerializeObject(newBus, \_jsonSettings);

                    var content = new StringContent(json, Encoding.UTF8, "application/json");

                    var apiUrl = string.Format("{0}/Buses", \_baseUrl);

                    var response = await client.PostAsync(apiUrl, content);

                    if (response.IsSuccessStatusCode)

                    {

                        Log.Information("Bus added successfully: {Model}", busModel);

                        XtraMessageBox.Show("Автобус успешно добавлен.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

                        await LoadBusesAsync();

                    }

                    else

                    {

                        var error = await response.Content.ReadAsStringAsync();

                        Log.Error("Failed to add bus. Status: {StatusCode}, Error: {Error}", response.StatusCode, error);

                        XtraMessageBox.Show(string.Format("Не удалось добавить автобус: {0}\n{1}", response.ReasonPhrase, error), "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                    }

                }

                catch (Exception ex)

                {

                    Log.Error(ex, "Exception adding bus");

                    XtraMessageBox.Show(string.Format("Произошла ошибка при добавлении автобуса: {0}", ex.Message), "Критическая ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                }

                finally { if (client != null) client.Dispose(); }

            }

        }

        private async void btnEdit\_Click(object sender, EventArgs e)

        {

            var selectedBus = gridViewBuses.GetFocusedRow() as Avtobus;

            if (selectedBus == null) return;

            string newBusModel = ShowInputDialog("Редактировать автобус", "Введите новую модель для автобуса:", selectedBus.Model);

            if (!string.IsNullOrWhiteSpace(newBusModel) && newBusModel != selectedBus.Model)

            {

                var updatedBusData = new Avtobus { BusId = selectedBus.BusId, Model = newBusModel };

                HttpClient client = null;

                try

                {

                    client = \_apiClient.CreateClient();

                    var json = JsonConvert.SerializeObject(updatedBusData, \_jsonSettings);

                    var content = new StringContent(json, Encoding.UTF8, "application/json");

                    var apiUrl = string.Format("{0}/Buses/{1}", \_baseUrl, selectedBus.BusId);

                    var response = await client.PutAsync(apiUrl, content);

                    if (response.IsSuccessStatusCode)

                    {

                        Log.Information("Bus updated successfully: ID {BusId}, New Model {Model}", selectedBus.BusId, newBusModel);

                        XtraMessageBox.Show("Автобус успешно обновлен.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

                        await LoadBusesAsync();

                    }

                    else

                    {

                        var error = await response.Content.ReadAsStringAsync();

                        Log.Error("Failed to update bus. Status: {StatusCode}, Error: {Error}", response.StatusCode, error);

                        XtraMessageBox.Show(string.Format("Не удалось обновить автобус: {0}\n{1}", response.ReasonPhrase, error), "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                    }

                }

                catch (Exception ex)

                {

                    Log.Error(ex, "Exception updating bus");

                    XtraMessageBox.Show(string.Format("Произошла ошибка при обновлении автобуса: {0}", ex.Message), "Критическая ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                }

                finally { if (client != null) client.Dispose(); }

            }

        }

        private async void btnDelete\_Click(object sender, EventArgs e)

        {

            var selectedBus = gridViewBuses.GetFocusedRow() as Avtobus;

            if (selectedBus == null) return;

            var result = XtraMessageBox.Show(string.Format("Вы уверены, что хотите удалить автобус '{0}' (ID: {1})?", selectedBus.Model, selectedBus.BusId),

                                              "Подтверждение удаления",

                                              MessageBoxButtons.YesNo,

                                              MessageBoxIcon.Warning);

            if (result == DialogResult.Yes)

            {

                HttpClient client = null;

                try

                {

                    client = \_apiClient.CreateClient();

                    var apiUrl = string.Format("{0}/Buses/{1}", \_baseUrl, selectedBus.BusId);

                    var response = await client.DeleteAsync(apiUrl);

                    if (response.IsSuccessStatusCode)

                    {

                        Log.Information("Bus deleted successfully: ID {BusId}, Model {Model}", selectedBus.BusId, selectedBus.Model);

                        XtraMessageBox.Show("Автобус успешно удален.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

                        await LoadBusesAsync();

                    }

                    else

                    {

                        var error = await response.Content.ReadAsStringAsync();

                        Log.Error("Failed to delete bus. Status: {StatusCode}, Error: {Error}", response.StatusCode, error);

                        if (response.StatusCode == System.Net.HttpStatusCode.Conflict)

                        {

                            XtraMessageBox.Show("Не удалось удалить автобус: он используется в маршрутах или обслуживании.", "Ошибка удаления", MessageBoxButtons.OK, MessageBoxIcon.Error);

                        }

                        else

                        {

                            XtraMessageBox.Show(string.Format("Не удалось удалить автобус: {0}\n{1}", response.ReasonPhrase, error), "Ошибка удаления", MessageBoxButtons.OK, MessageBoxIcon.Error);

                        }

                    }

                }

                catch (Exception ex)

                {

                    Log.Error(ex, "Exception deleting bus");

                    XtraMessageBox.Show(string.Format("Произошла ошибка при удалении автобуса: {0}", ex.Message), "Критическая ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

                }

                finally { if (client != null) client.Dispose(); }

            }

        }

        private async void btnRefresh\_Click(object sender, EventArgs e)

        {

            txtSearch.Text = string.Empty;

            await LoadBusesAsync();

        }

        private void txtSearch\_EditValueChanged(object sender, EventArgs e)

        {

            FilterAndBindBuses();

        }

        private void gridViewBuses\_FocusedRowChanged(object sender, FocusedRowChangedEventArgs e)

        {

             UpdateButtonStates();

        }

        private void UpdateButtonStates()

        {

             bool isRowSelected = gridViewBuses.GetFocusedRow() is Avtobus;

             btnEdit.Enabled = isRowSelected;

             btnDelete.Enabled = isRowSelected;

        }

    }

}