**СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 1](#_Toc154492332)

[**ВВЕДЕНИЕ** 2](#_Toc154492333)

[**1.** **МОДЕЛИ И СПЕЦИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 4](#_Toc154492334)

[1.1. Основные принципы и понятия ООП 4](#_Toc154492335)

[1.2. Платформа .NET 5](#_Toc154492336)

[1.3. Выбор компилятора для выполнения работы 7](#_Toc154492337)

[**2.** **РЕАЛИЗАЦИЯ ПРОЕКТА** 10](#_Toc154492338)

[2.1. Постановка задач 10](#_Toc154492339)

[2.2. Проектный план 12](#_Toc154492340)

[2.3. Руководство программиста 13](#_Toc154492341)

[2.4. Руководство пользователя 18](#_Toc154492342)

[**ЗАКЛЮЧЕНИЕ** 23](#_Toc154492343)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 24](#_Toc154492344)

[**ПРИЛОЖЕНИЕ A** 25](#_Toc154492345)

[**ПРИЛОЖЕНИЕ Б** 25](#_Toc154492346)

[**ПРИЛОЖЕНИЕ В** 28](#_Toc154492347)

**ВВЕДЕНИЕ**

В современном мире авиационная индустрия сталкивается с постоянными вызовами, требуя внедрения эффективных технологических решений для улучшения качества обслуживания и оптимизации бизнес-процессов. Данная курсовая работа посвящена созданию кассовой системы для авиакомпании "Аэрофлот" с использованием языка программирования C# и технологии Windows Forms. Разработка данного приложения направлена на улучшение процессов продажи билетов и обеспечение более удобного взаимодействия с клиентами.

Целью курсовой работы является: Исследование, проектирование и реализация кассовой системы для авиакомпании "Аэрофлот" с использованием C# и Windows Forms.

Для достижения цели в работе были решены следующие задачи:

* Исследован рынок авиаперевозок и выявлены основные требования к кассовой системе.
* Изучены и анализированы современные технологии программирования на C# и возможности, предоставляемые Windows Forms для создания графического пользовательского интерфейса.
* Разработана кассовая система, включая функционал оформления билетов, учета проданных билетов и обеспечения удобного взаимодействия с клиентами.

Таким образом, данная курсовая работа не только представляет техническое решение в виде программного продукта, но и включает в себя анализ современных технологий и методов, используемых при разработке программного обеспечения для авиакомпаний.

При разработке большинства программных проектов приоритетным стал объектно-ориентированный подход. Технология объектно-ориентированного программирования (ООП) — это методология, основанная на организации программного кода в виде объектов, которые взаимодействуют между собой для реализации функциональности программы.

В качестве технологий для разработки прототипа приложения была выбрана платформа .NET и язык программирования для неё - C#.

Данная курсовая работа состоит из 2 глав, заключения и списка использованных источников.

В первой главе рассмотрены модели спецификации программного обеспечения. Даны объяснения, что из себя представляет то или иное программное обеспечение, используемое в работе над курсовой работой и почему наш выбор пал именно на него.

Во второй главе описан процесс реализации курсовой работы. Поставлены задачи, составлен проектный план, руководство программиста и руководство пользователя.

В заключении приводятся результаты проделанной нами работы, дальнейшие планы в отношении продукта этой курсовой работы, а также описано, какие навыки мы получили в ходе решения задачи этой работы.

В списке использованных источников указаны учебные пособия, которые использовались нами при выполнении этой курсовой работы, а также ссылка на репозиторий проекта курсовой работы на хостинге для хранения проектов GitHub.

1. **МОДЕЛИ И СПЕЦИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

# 1.1. Основные принципы и понятия ООП

Основные понятия объектно-ориентированного программирования (ООП) включают в себя следующие ключевые элементы:

1. Объект (Object):

* Объект представляет экземпляр класса и является базовым строительным блоком в ООП.
* Объекты имеют свойства (члены данных) и методы (функции или процедуры), которые могут выполнять действия или возвращать значения.

1. Класс (Class):

* Класс — это шаблон или формальное описание, определяющее структуру и поведение объектов.
* Он содержит члены данных (поля) и методы, которые определяют поведение объектов этого класса.

Основные принципы ООП включают в себя:

1. Инкапсуляция (Encapsulation):

* Объединение данных и методов, обрабатывающих эти данные, в единый объект.
* Сокрытие внутренних деталей реализации объекта от внешнего мира, предоставление только интерфейса для взаимодействия с объектом.

1. Наследование (Inheritance):

* Возможность создания новых классов на основе существующих.
* Позволяет классу наследовать свойства и методы другого класса, обеспечивая повторное использование кода.

1. Полиморфизм (Polymorphism):

* Возможность объектов разных типов использовать общий интерфейс.
* Объекты могут вести себя по-разному в зависимости от контекста выполнения.

1. Абстракция (Abstraction):

* Выделение общих характеристик объектов и игнорирование ненужных деталей.
* Способ представления объекта с точки зрения его основных характеристик, опуская детали реализации.

ООП предоставляет эффективные средства для моделирования реального мира в виде программных объектов, что облегчает разработку, поддержку и масштабирование программных систем. ООП способствует повышению читаемости кода, его модульности и уровню абстракции, что обеспечивает более гибкое и понятное программирование. Важными языками программирования, поддерживающими ООП, являются C++, Java, C#, Python и другие.

# Платформа .NET

.NET (произносится "дотнет") — это комплекс технологий, разработанный корпорацией Microsoft, предназначенный для создания и выполнения приложений на платформе Windows. Этот комплекс включает в себя не только среду выполнения (Common Language Runtime - CLR), но и множество библиотек классов, языков программирования, инструментов разработки и других технологий.

Вот основные компоненты .NET:

1. Common Language Runtime (CLR):

* CLR является виртуальной машиной, обеспечивающей выполнение программ, написанных на различных языках программирования, в том числе C#, VB.NET, F# и других.
* Он управляет выполнением кода, обеспечивает сборку мусора, управление памятью и обеспечивает другие службы, необходимые для выполнения программ.

1. Библиотеки классов .NET (NET Framework Class Library):

* Библиотеки классов представляют собой обширный набор предопределенных классов, содержащих множество функций для обработки файлов, работы с сетью, базами данных, создания графического интерфейса и многих других задач.
* Обеспечивают высокоуровневый интерфейс для многих операций, что упрощает разработку и повышает производительность.

1. Языки программирования:

* Одним из ключевых преимуществ .NET является возможность использования различных языков программирования, таких как C#, VB.NET, F#, IronPython, IronRuby и др., при этом все они могут взаимодействовать между собой на уровне байт-кода CLR.

1. ASP.NET:

* ASP.NET представляет собой фреймворк для разработки веб-приложений и веб-служб.
* Обеспечивает разработку динамических веб-сайтов, используя языки программирования, такие как C# и VB.NET.

1. Windows Presentation Foundation (WPF):

* WPF предоставляет средства для создания графического пользовательского интерфейса (GUI) с использованием векторной графики.
* Позволяет создавать богатые по функционалу и визуально привлекательные приложения с поддержкой анимаций и стилей.

1. Windows Communication Foundation (WCF):

* WCF предоставляет средства для создания распределенных приложений, включая разработку служб и клиентов для обмена данными между приложениями.

1. Entity Framework:

* Entity Framework представляет собой ORM (Object-Relational Mapping), который упрощает взаимодействие с базами данных через объекты в коде.

.NET предоставляет разработчикам средства для создания широкого спектра приложений — от десктопных и веб-приложений до мобильных и облачных сервисов. Важно отметить, что с течением времени появляются новые версии .NET (например, .NET Core и .NET 5), расширяя его функциональность и обеспечивая кроссплатформенность для разработки.

# Выбор компилятора для выполнения работы

Компилятор для C# — это программный инструмент, который преобразует исходный код, написанный на языке программирования C#, в исполняемый код, который может быть выполнен на целевой платформе, как правило, с использованием Common Language Runtime (CLR). Процесс преобразования исходного кода в исполняемый код называется компиляцией.

В контексте C# и .NET, компиляция происходит в два этапа:

1. Компиляция в промежуточный язык (Intermediate Language - IL):

* Когда вы пишете программу на C#, исходный код компилируется в промежуточный язык, который называется Common Intermediate Language (CIL) или просто IL.
* IL является абстрактным представлением кода и независим от конкретной архитектуры процессора. Это позволяет одному и тому же исполняемому файлу работать на разных платформах.

1. JIT-компиляция (Just-In-Time Compilation):

* Во время выполнения программы на .NET, IL код JIT-компилируется в нативный машинный код для конкретной архитектуры процессора.
* Это происходит "на лету" (just-in-time), когда приложение запускается. JIT-компиляция обеспечивает оптимизацию кода для конкретной платформы и улучшает производительность приложения.

Компилятор обеспечивает не только создание исполняемого кода, но и включает в себя ряд дополнительных функций:

* Проверка типов: Компилятор осуществляет проверку типов в процессе компиляции, что помогает выявить ошибки до момента выполнения программы.
* Генерация метаданных: Компилятор создает метаданные, которые содержат информацию о типах, используемых в программе. Это необходимо, например, для возможности использования Reflection.
* Оптимизация кода: Некоторые компиляторы проводят оптимизацию IL кода для улучшения производительности при выполнении программы.

В среде .NET основным компилятором для C# является компилятор Roslyn (полное название — .NET Compiler Platform), который предоставляет открытый и модульный подход к компиляции и анализу кода. Roslyn позволяет разработчикам создавать инструменты для анализа кода, рефакторинга и других сценариев.

Однако в качестве компилятора для выполнения данной курсовой работы я выбрал Microsoft Visual Studio.

Использование компилятора в составе Microsoft Visual Studio для создания приложения на C# с использованием .NET и Windows Forms обладает рядом преимуществ:

1. Интегрированная среда разработки (IDE):

* Microsoft Visual Studio предоставляет мощную и удобную среду разработки, обеспечивая поддержку C#, .NET и Windows Forms "из коробки".
* Интегрированная среда разработки включает в себя редактор кода, отладчик, дизайнер форм, инструменты анализа кода и множество других функций, упрощающих процесс разработки.

1. Дизайнер форм (Windows Forms Designer):

* Визуальный дизайнер форм в Visual Studio облегчает создание графического пользовательского интерфейса приложения с использованием Windows Forms.
* Разработчик может перетаскивать и редактировать элементы управления прямо на форме, что упрощает процесс создания интерфейса.

1. Интеграция с .NET и Windows Forms:

* Visual Studio тесно интегрирован с технологиями .NET и Windows Forms, что обеспечивает легкость создания, сборки и отладки приложений под управлением CLR.
* Обширная библиотека классов .NET Framework доступна для использования в проекте, предоставляя готовые решения для множества задач.

1. Богатые возможности отладки:

* Visual Studio предоставляет расширенные средства отладки, позволяя легко отслеживать ошибки, анализировать переменные, использовать точки останова и другие инструменты для эффективной отладки приложений.

1. Поддержка различных языков и технологий:

* Visual Studio поддерживает не только C#, но и другие языки программирования, такие как VB.NET, F#, и дополнительные технологии, что упрощает совместную работу и внедрение различных компонентов в проект.

1. Обновления и новые возможности:

* Microsoft регулярно обновляет Visual Studio, предоставляя новые возможности, улучшения и исправления ошибок, что поддерживает разработчика в актуальном и продуктивном состоянии.

Выбор Microsoft Visual Studio для разработки приложения на C# с использованием .NET и Windows Forms обеспечивает разработчикам широкий инструментарий, удобство в работе, высокую производительность и качественную поддержку со стороны разработчиков Microsoft.

## **РЕАЛИЗАЦИЯ ПРОЕКТА**

# Постановка задач

Требуется разработать средствами языка C# и технологии .Net модель кассы Аэрофлота. Программа должна содержать: Расписание: номер рейса, маршрут, пункты промежуточной посадки, время отправления, дни полета. Программа должна определять количество свободных мест на каждом рейсе, выбирать ближайший рейс до заданного пункта (при наличии свободных мест), оформление заданного числа билетов по согласованию с пассажиром (с уменьшением числа свободных мест).

Постановка задач:

1. Ознакомиться с языком программирования C#
2. Ознакомиться с .Net framework и научиться устанавливать взаимодействие C# и .Net
3. Реализовать средствами языка C# и технологии .Net модель кассы Аэрофлота.
4. Научиться работать с базами данных MySql
5. Закрепить навыки программирования с использованием принципов ООП.

Примерный внешний вид приложения изображен на Рис. 1

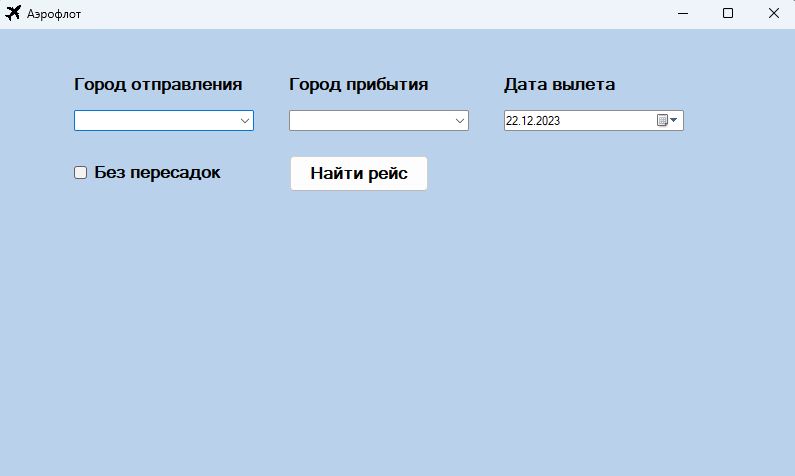


Рис. 1 — Внешний вид приложения.

Приложение устанавливает соединение с удалённой базой данных, и ищет подходящие запросам пользователя рейсы. В случае, если рейсы, соответствующие пользовательским запросам, существуют – необходимо выводить на экран номер рейса, города и время отправления и прибытия, пункт пересадки, количество билетов и цену за 1 билет. Если же подходящих рейсов нет, на экран выводится сообщение о том, что на выбранную данную по выбранному маршруту рейсов нет, или все билеты распроданы.

Также приложение должно осуществлять проверку всех неудачных сценариев. Примерами таких сценариев может послужить: выбор пользователем даты вылета раньше сегодняшней даты, вывод на экран рейсов, которые уже осуществили вылет, выбор пользователем в качестве города отправления и прибытия одинакового города и т.п. В случае столкновения с такими сценариями программа должна выводить сообщения о соответствующей ошибке и продолжать выполнение, не блокируя пользователю взаимодействие с приложением.

# Проектный план

Проектный план состоит из функциональных требований к разработке приложения, оценки сложности реализации и статуса их выполнения. (Рис. 2).

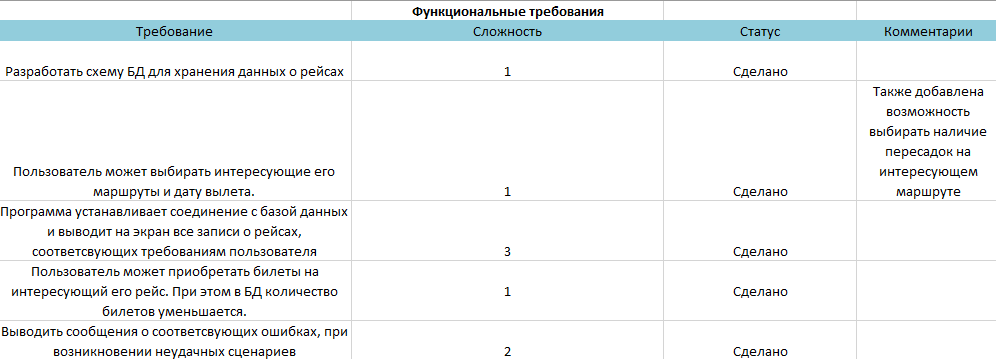


Рис. 2 — Функциональные требования.

Далее был разработан проектный план работы и назначены сроки выполнения работы. (Рис. 3).



Рис. 3 — Проектный план

# Руководство программиста

Данная программа разработана на технологии C# WinForms, входящая в .NetFramework.

У нас имеется главная форма FormSearch, содержащая главное меню, с помощью которого вызываются другие формы.

FormSearch() — конструктор класса. Вызывает метод, прорисовывающий объекты в форме. (Рис. 4).

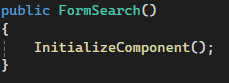
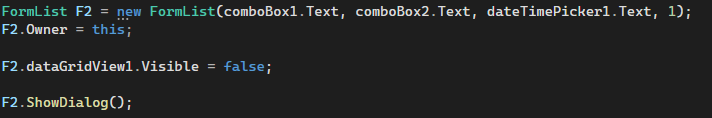


Рис. 4 — Конструктор класса.

На главной форме расположены компоненты comboBox предназначенные для выбора маршрута пользователем, компонент dateTimePicker, с помощью которого пользователь выбирает дату вылета, checkbox, позволяющий пользователю определить наличие пересадок на выбранном маршруте и компоненты label, служащие источником необходимой для взаимодействия с программой информации для пользователя. Кроме того, на форме присутствует компонент button, при нажатии на который программа проверяет предоставленные пользователем запросы, и, если все запросы верны, вызывает вторую форму FormList. В ином случае, программа выводит на экран сообщения, указывающие пользователю на ошибку во входных данных. Процедура button3\_Click() представлена в приложении А.

По нажатии на компонент button в форме FormSearch программа вызывает форму FormList, при этом передавая запросы пользователя в эту форму. (Рис. 5)

  
Рис. 5 — Передача параметров из формы FormSearch в форму FormList.

После инициализации элементов формы FormList программа устанавливает соединение с базой данных (рис. 6) и выводит на экран подходящие под запрос пользователя рейсы. Процедура LoadData() представлена в приложении Б.



Рис. 6 — Установка соединения с базой данных.

Считывание данных из базы данных осуществляется построчно с помощью классов MySqlDataReader, MySqlCommand и метода ExecuteReader(). (Рис. 7)



Рис. 7 — Считывание данных из базы.

Заполнение таблицы dataGridView подходящими рейсами осуществляется построчно с помощью ранее описанных классов MySqlDataReader, MySqlCommand и метода ExecuteReader(), а так же метода Add(). Подробнее с этим процессом можно ознакомиться в приложении Б.

После заполнения таблицы dataGridView программа вызывает окно формы FormList. Если есть подходящие пользователю рейсы, программа выводит их список, а также даёт пользователю возможность купить билеты на интересующий его рейс. Если же таких рейсов нет, программа выводит сообщение о том, что подходящих пользовательским запросам рейсов нет, или все билеты распроданы. (Рис. 8 и 9)

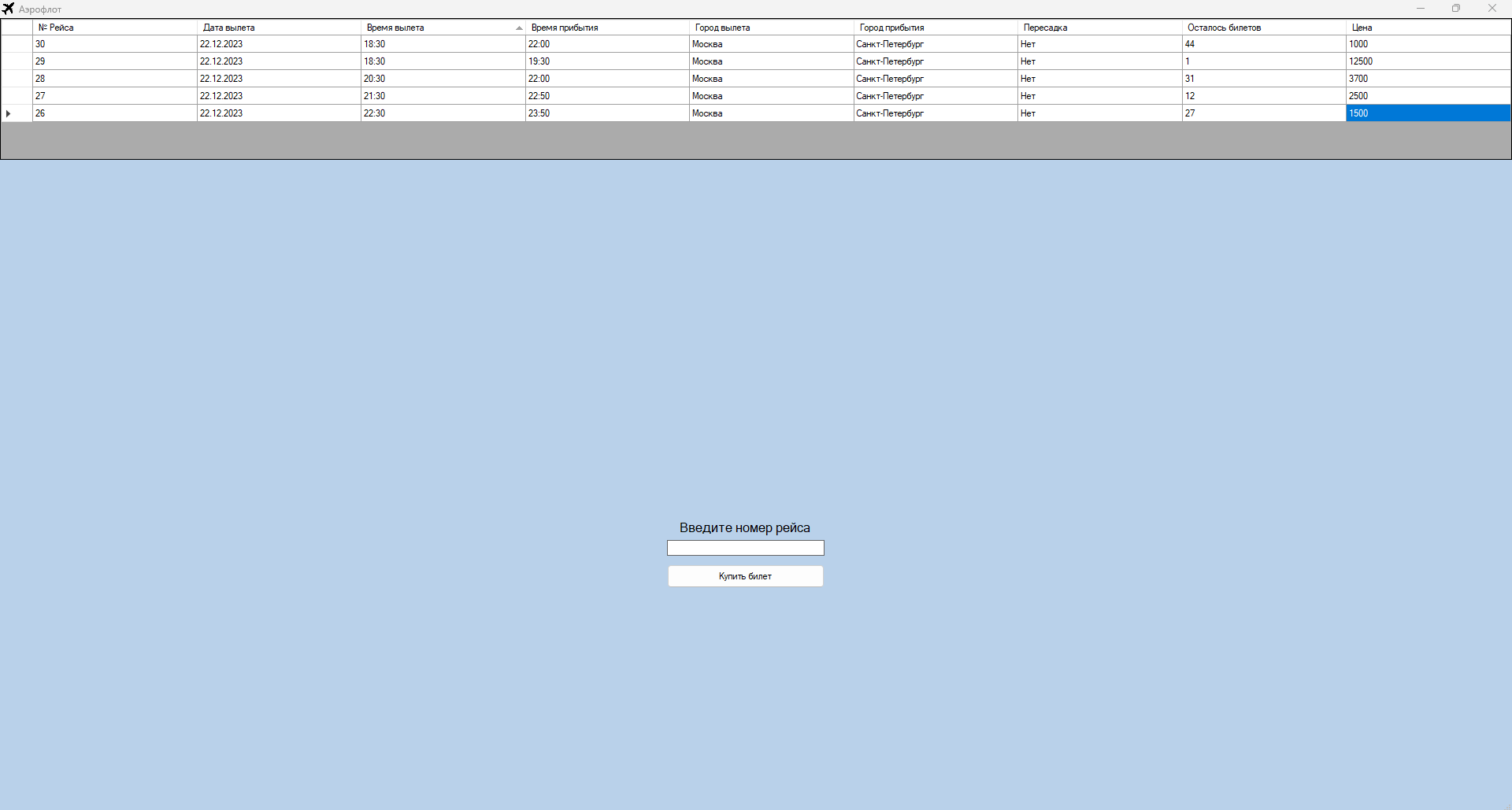


Рис. 8 — Внешний вид приложения при наличии рейсов.

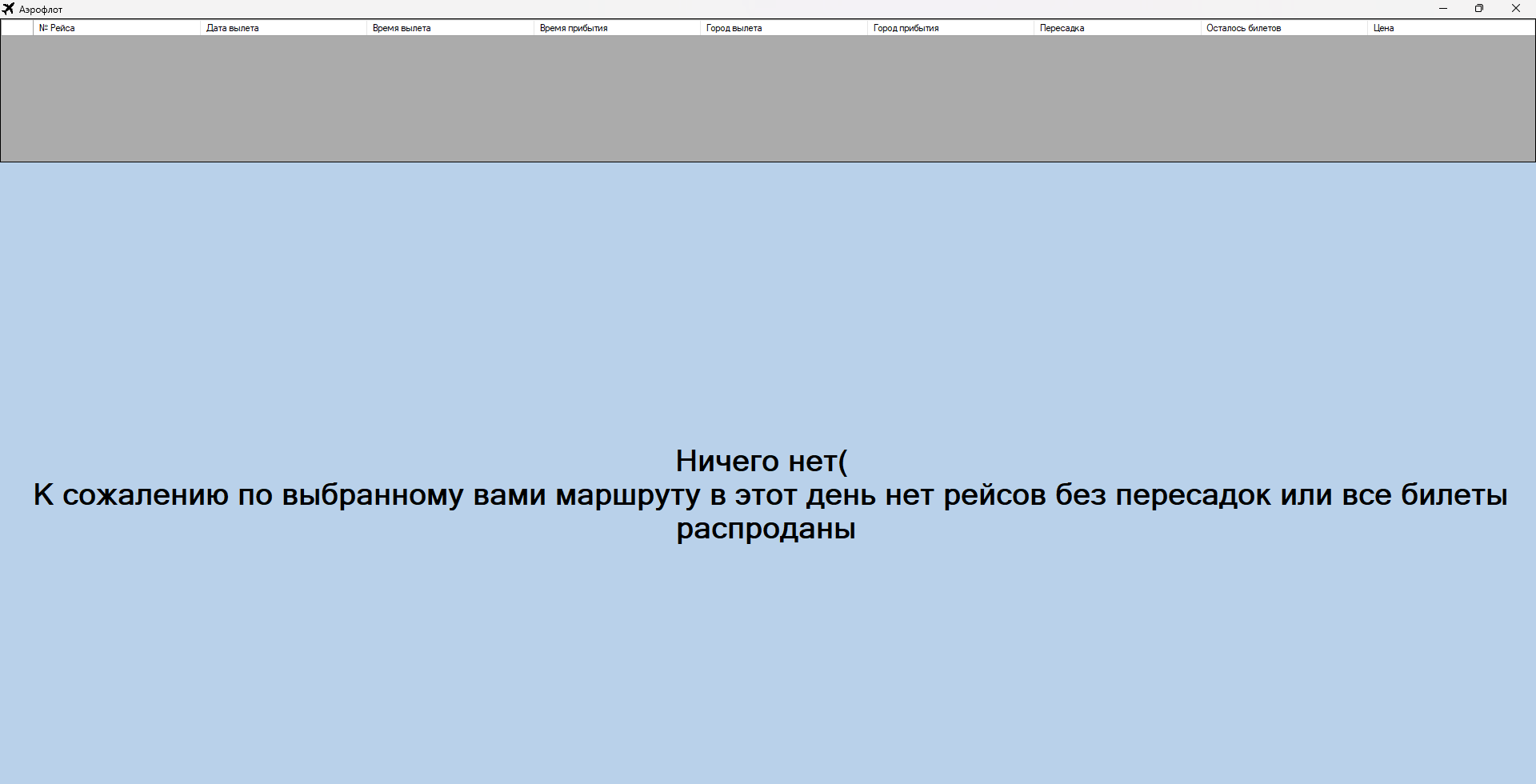


Рис. 9 — Внешний вид приложения при отсутствии подходящих рейсов.

Выбор желаемого пользователем рейса осуществляется при помощи компонента textbox для ввода номера рейса и компонента button, при нажатии на который программа проверит, есть ли такой рейс в таблице и вызовет третью форму FormBuy, при этом передавая ей в качестве аргумента номер рейса, выбранного пользователем, если рейс есть, либо выведет сообщение о пользовательской ошибке, если таких рейсов нет. (Рис. 10)

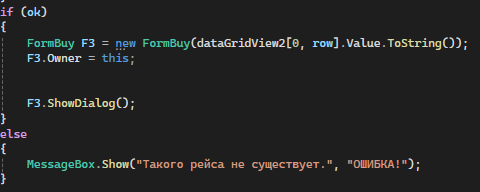


Рис. 10 — Работа процедуры button1\_Click() формы FormList.

В форме FormBuy программа вновь устанавливает соединение с базой данных и по переданному ей из формы FormList номеру рейса выведет на экран номер рейса, количество билетов доступных для покупки и цену за 1 билет. Подробнее в приложении В.

После этого программа запрашивает у пользователя персональные данные необходимые для оформления билетов (серия, номер паспорта, дата рождения) и данные необходимые для покупки (количество билетов, номер банковской карты, cvc код карты, и срок её действия). Далее, по нажатии на кнопку “оформить билеты” программа проверяет правильность введённых данных и, если всё введено верно, осуществляет покупку. При этом пользователь получает сообщение об успешном завершении сделки, а количество оставшихся билетов по данному рейсу уменьшается на число купленных пользователем. (Рис. 11)

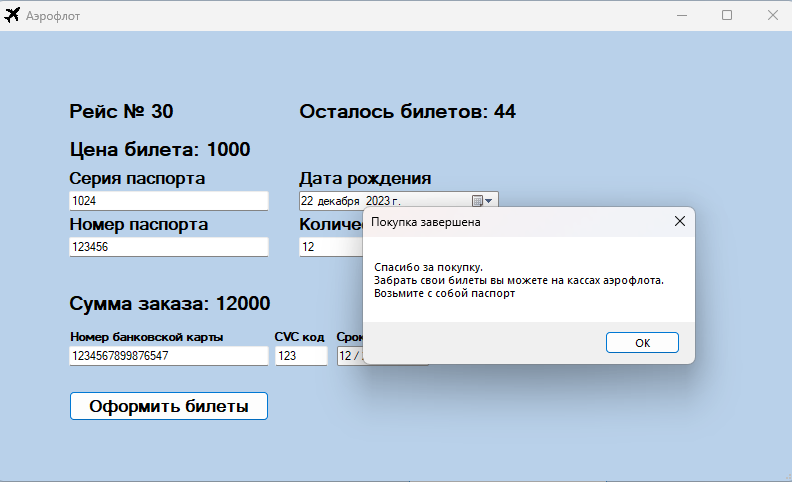


Рис. 11 — Покупка билета.

# Руководство пользователя

Данное руководство служит пользователю инструкцией по использованию приложения.

При запуске программы появляется главное окно программы. (Рис. 12)

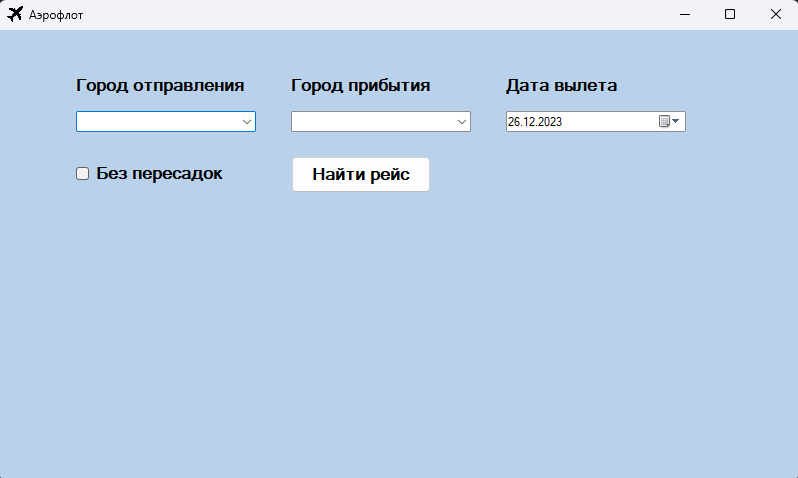


Рис. 12 — Главное окно программы.

При запуске значение даты вылета автоматически устанавливается на нынешнюю дату. В этом окне пользователь может выбрать город отправления, город прибытия, дату вылета и наличие пересадок на рейсе. Интерфейс интуитивно понятен и удобен. Если пользователь не переводит checkBox “Без пересадок” в состояние checked, программа ищет рейсы как с пересадками, так и без них.

В случае ввода неправильных данных пользователем программа выводит сообщения об ошибках, которые позволяют понять, в чём его ошибка. (Рис. 13, 14, 15)

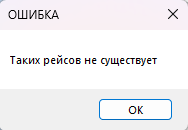


Рис. 13 — Ошибка выбора одинаковых пунктов вылета и прилёта.

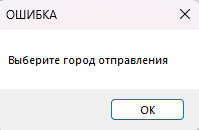


Рис. 14 — Сообщение об ошибке, если пользователь не выбрал город отправления.

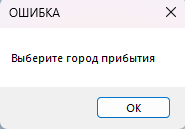


Рис. 15 — Сообщение об ошибке, если пользователь не выбрал город прибытия.

Для осуществления поиска интересующих рейсов пользователь должен нажать кнопку “Найти рейс”. После чего, если пользователь сделал всё верно, появится форма списка рейсов. Эта форма может быть представлена в 3 вариантах:  
таблица рейсов с пересадками и без них и поле покупки билета (рис. 16), таблица рейсов без пересадок и поле покупки билета (рис. 17), и форма в которой будет сказано, что рейсов по интересующему пользователя маршруту нет, или все билеты распроданы (рис. 18).

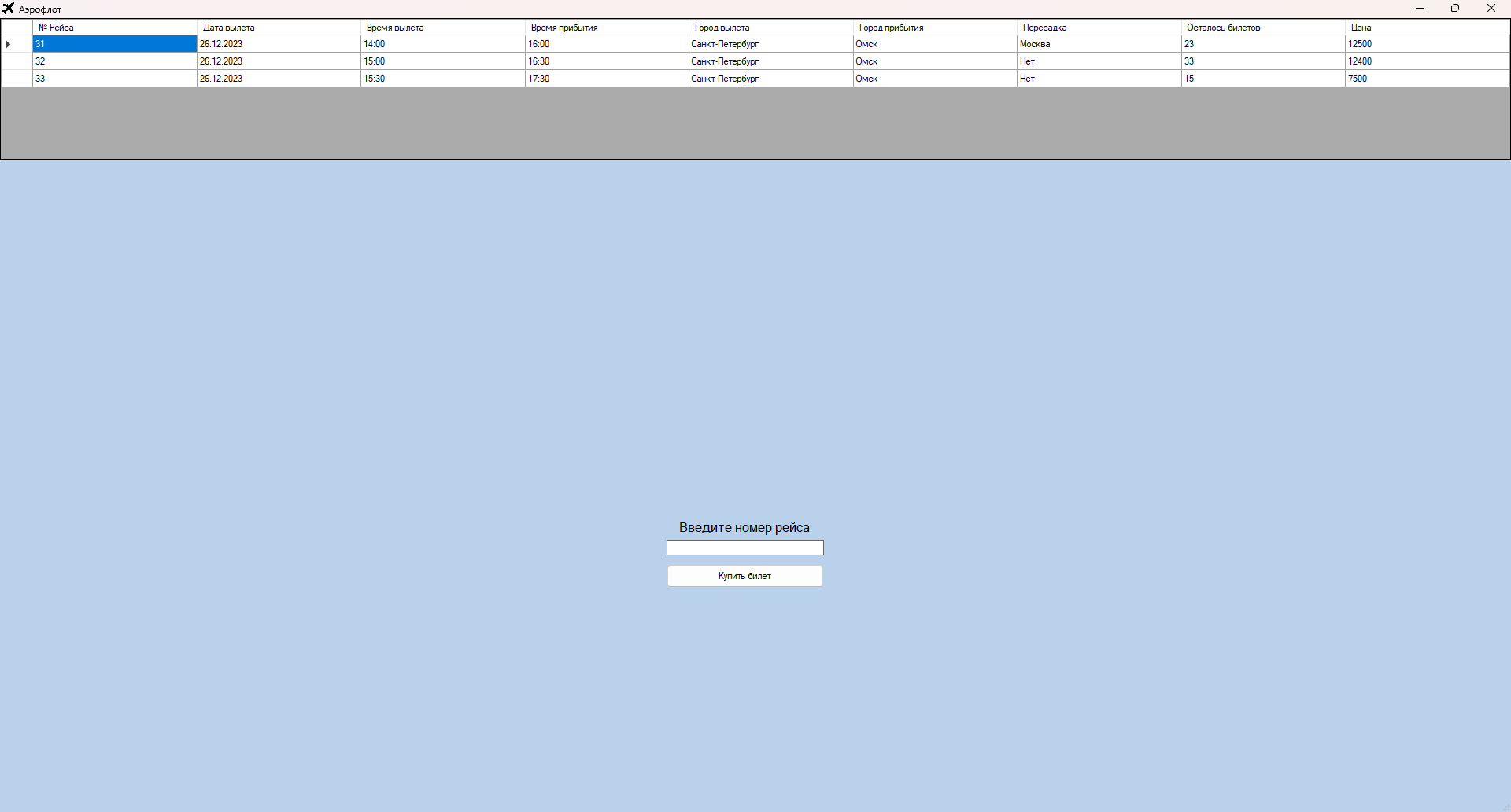


Рис. 17 — Первый вариант формы “FormList”. С пересадками и без.

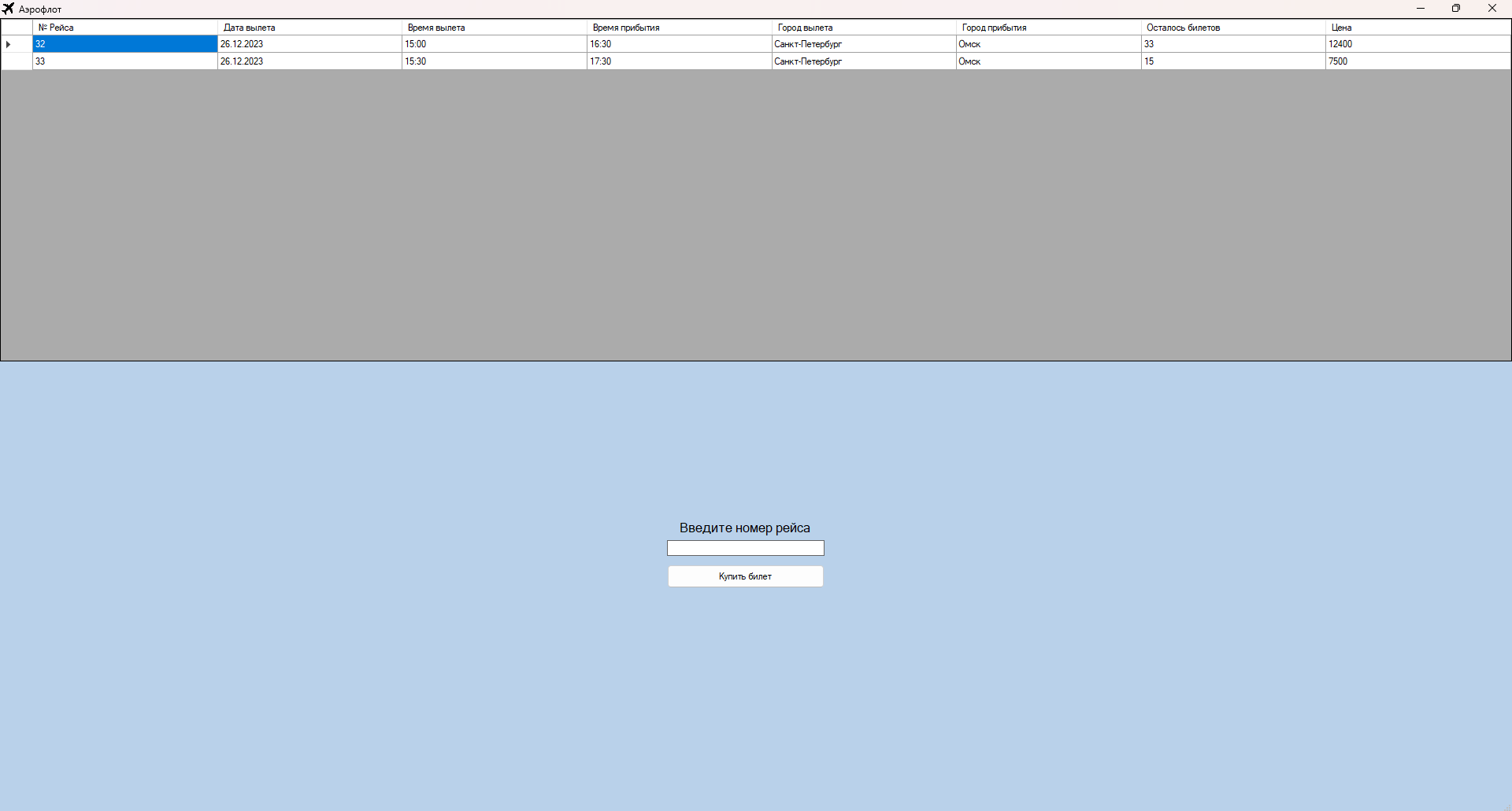


Рис. 18 — Второй вариант формы “FormList”. Без пересадок.

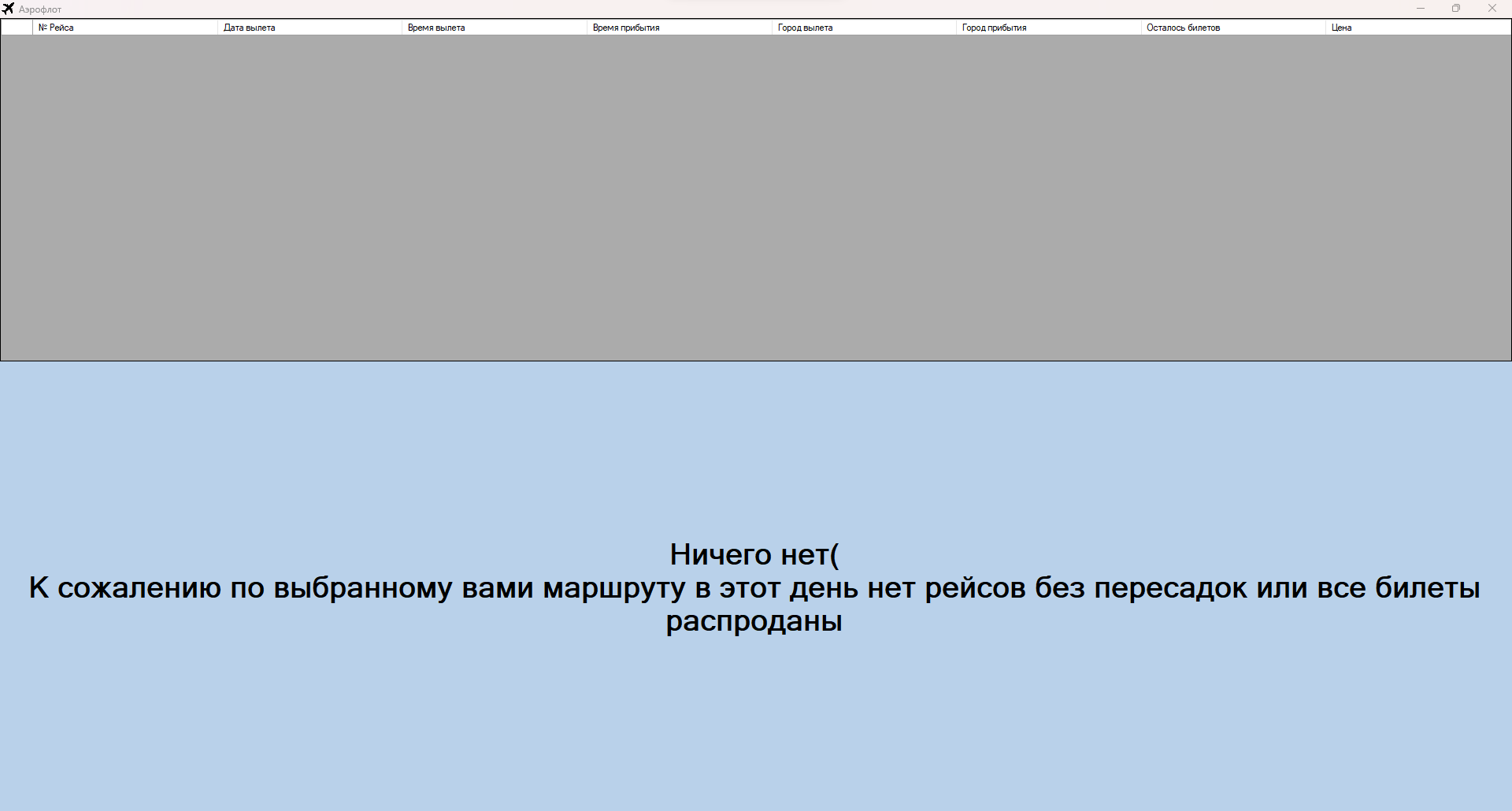


Рис. 19 — Третий вариант формы “FormList”. Рейсов нет.

Для рейса нужно ввести его номер в соответствующее поле ввода. В случае неправильного ввода программа выведет сообщения об ошибках, которые позволят пользователю понять, что он сделал не так. (Рис. 20, 21)

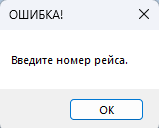


Рис. 20 — Сообщение об ошибке “Пустое поле ввода”.

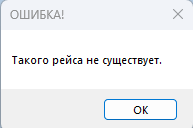


Рис. 21 — Сообщение об ошибке “Неправильный номер рейса”.

Если пользователь выбрал верный рейс, программа вызовет форму “FormBuy”, на которой пользователь сможет купить необходимое ему количество билетов на интересующий его рейс. (Рис. 22)

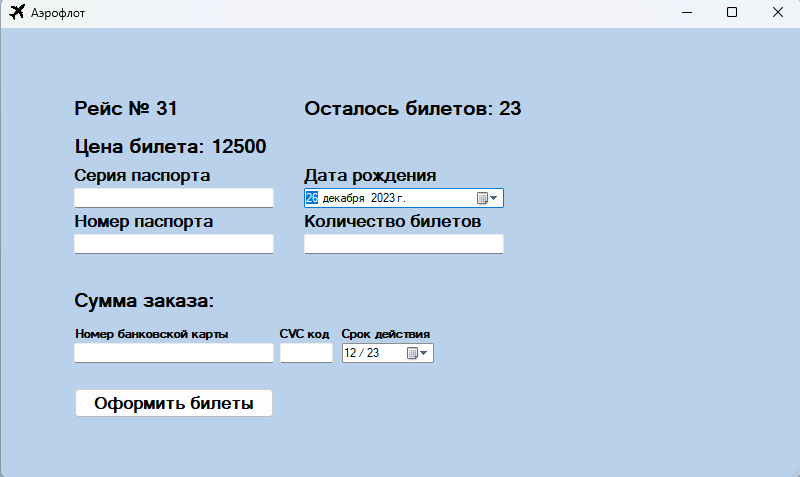


Рис. 22 — Форма покупки билетов “FormBuy”.

Заполнив все необходимые данные и нажав кнопку “Оформить билеты”, пользователь получит сообщение об успешном завершении процесса покупки. (Рис. 23)

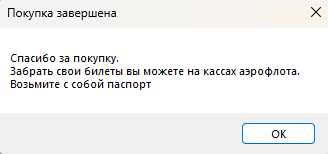


Рис. 23 Сообщение об успешной покупке билетов.

## **ЗАКЛЮЧЕНИЕ**

В результате курсовой работы мы улучшили свои навыки владения языками семейства C, в частности языком C#. Научились работать с .Net framework в рамках языка C#. Улучшили навыки работы с windows forms. Был разработан интерфейс приложения кассы аэрофлота, установлено соединение с базой данных, а также установлено взаимодействие между классами программы, что отражает принципы объектно-ориентированного программирования, проведено тестирование конечного продукта, что позволило предупредить и исправить все возможные ошибки.

Были решены следующие задачи, поставленные для достижения цели:

1. Реализована база данных;
2. Сделан внешний вид приложения;
3. Установлено взаимодействие между формами программы;
4. Проведено тестирование и исправление ошибок;

В дальнейшем наше приложение можно будет доработать, добавить выбор места в самолёте, улучшить дизайн, сделать страницу администратора, через которую он сможет добавлять новые рейсы в базу данных, добавить поле “Багаж”.

Далее этот опыт может помочь нам в разработке реального продукта, в котором мы уже сможем установить соединение с платёжной системой и проверять документы пользователя на правильность.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Евсеева О.Н. Основы языка C# 2005: учеб. / О.Н. Евсеева, А.Б. Шамшев. – Ульяновск :УлГТУ, 2008. – 132 с.
2. Лафоре Р. Объектно-ориентированное программирование в С++ / Р. Лафоре. – 4-е изд. –СПб.: Питер, 2004 – 922 с.
3. Пахомов Б.И. С/С++ и MSVisualC++ 2010 для начинающих / Б.И. Пахомов. – СПб.: БХВ-Петербург, 2011. – 736 с.: ил. + дистрибутив (на DVD).
4. Троелсен Э. Язык программирования C# 6.0 и платформа .NET 4.6. / Э. Троелсен, Ф. Джеликс ; пер. с англ. Ю.Н. Артеменко. – 7-е изд. – Москва [и др.]: Вильямс, 2017. - 1438 с.
5. Кручинин В.И. C# Лабораторный практикум по основам программирования.
6. Репозиторий проекта курсовой работы на хостинге для хранения проектов GitHub. https://github.com/StolzAn/Programming-technologies/

## **ПРИЛОЖЕНИЕ A**

ОПИСАНИЕ РАБОТЫ КНОПКИ “НАЙТИ РЕЙС”

public void button3\_Click(object sender, EventArgs e)

{

if(comboBox1.Text.Length == 0)

{

MessageBox.Show("Выберите город отправления", "ОШИБКА");

}

else if(comboBox2.Text.Length == 0)

{

MessageBox.Show("Выберите город прибытия", "ОШИБКА");

}

else if (comboBox2.Text == comboBox1.Text)

{

MessageBox.Show("Таких рейсов не существует", "ОШИБКА");

}

else

{

if (checkBox1.Checked)

{

FormList F2 = new FormList(comboBox1.Text, comboBox2.Text, dateTimePicker1.Text, 1);

F2.Owner = this;

F2.dataGridView1.Visible = false;

F2.ShowDialog();

}

else

{

FormList F2 = new FormList(comboBox1.Text, comboBox2.Text, dateTimePicker1.Text, 0);

F2.Owner = this;

F2.dataGridView1.Visible = true;

F2.dataGridView2.Visible = false;

F2.ShowDialog();

}

}

}

## **ПРИЛОЖЕНИЕ Б**

ОПИСАНИЕ ФУНКЦИИ LoadData() формы FormList

private void LoadData(string departure, string arrival, string date, int Is\_checked)

{

MySqlConnection connection = new MySqlConnection("server=localhost;port=3306;username=root;password=root;database=flightdb"); //Прописываем, с какой базой работаем

connection.Open(); //Подключаемся

string query = "SELECT \* FROM flight ORDER BY Departure\_time"; // SQL комманда

MySqlCommand command = new MySqlCommand(query, connection); // Активируем команду

MySqlDataReader reader = command.ExecuteReader(); // Комманда вывод консоли SQL

List<string[]> data = new List<string[]>(); //Создаём массив данных

if (Is\_checked == 0) // Пользователь не нажал "Без пересадок"

{

dataGridView1.Dock = DockStyle.Top; //Таблица с рейсами

int count = 0;

while (reader.Read()) //Начинаем считывание из консоли SQL

{

data.Add(new string[9]); // Определяем добавляем в массив 9 колонок

if ((DateTime.Parse(reader[1].ToString()).ToShortDateString() == date) //Если совпадает дата вылета и дата выбранная пользователем

&& (DateTime.Parse(DateTime.Parse(reader[2].ToString()).ToShortTimeString()) > DateTime.Parse(DateTime.Now.ToShortTimeString())) //Если время вылета больше времени сейчас(самолёт ещё не улетел)

&& (reader[4].ToString() == departure) && (reader[5].ToString() == arrival) //совпадают города вылета и прибытия

&& reader[8].ToString() != "0") //Количество билетов != 0

{

count++;

if (reader[6].ToString() == "0")

{

data[data.Count - 1][0] = reader[0].ToString(); //Начинаем заполнять. data[data.Count - 1] - data.Count - количество строк. С каждым проходом while data.Count увеличивается на 1

data[data.Count - 1][1] = reader[1].ToString();

data[data.Count - 1][1] = DateTime.Parse(data[data.Count - 1][1]).ToShortDateString();

data[data.Count - 1][2] = reader[2].ToString();

data[data.Count - 1][2] = DateTime.Parse(data[data.Count - 1][2]).ToShortTimeString();

data[data.Count - 1][3] = reader[3].ToString();

if(DateTime.Today > DateTime.Parse(data[data.Count - 1][3])) //Если дата прибытия не совпадает с датой вылета, выводить и дату и время

{

data[data.Count - 1][3] = DateTime.Parse(data[data.Count - 1][3]).ToShortDateString() + " " + DateTime.Parse(data[data.Count - 1][3]).ToShortTimeString();

}

else //Иначе - только время

{

data[data.Count - 1][3] = DateTime.Parse(data[data.Count - 1][3]).ToShortTimeString();

}

data[data.Count - 1][4] = reader[4].ToString();

data[data.Count - 1][5] = reader[5].ToString();

data[data.Count - 1][6] = "Нет";

data[data.Count - 1][7] = reader[8].ToString();

data[data.Count - 1][8] = reader[9].ToString();

}

else

{

data[data.Count - 1][0] = reader[0].ToString();

data[data.Count - 1][1] = reader[1].ToString();

data[data.Count - 1][1] = DateTime.Parse(data[data.Count - 1][1]).ToShortDateString();

data[data.Count - 1][2] = reader[2].ToString();

data[data.Count - 1][2] = DateTime.Parse(data[data.Count - 1][2]).ToShortTimeString();

data[data.Count - 1][3] = reader[3].ToString();

if (DateTime.Today > DateTime.Parse(data[data.Count - 1][3])) //Если дата прибытия не совпадает с датой вылета, выводить и дату и время

{

data[data.Count - 1][3] = DateTime.Parse(data[data.Count - 1][3]).ToShortDateString() + DateTime.Parse(data[data.Count - 1][3]).ToShortTimeString();

}

else //Иначе - только время

{

data[data.Count - 1][3] = DateTime.Parse(data[data.Count - 1][3]).ToShortTimeString();

}

data[data.Count - 1][4] = reader[4].ToString();

data[data.Count - 1][5] = reader[5].ToString();

data[data.Count - 1][6] = reader[7].ToString();

data[data.Count - 1][7] = reader[8].ToString();

data[data.Count - 1][8] = reader[9].ToString();

}

}

}

if (count == 0)

{

label1.Text = "Ничего нет( \n К сожалению по выбранному вами маршруту в этот день нет рейсов без пересадок или все билеты распроданы";

label1.Visible = true;

label1.Dock = DockStyle.Fill;

flowLayoutPanel1.Visible = false;

}

else

{

foreach (string[] s in data)

{

if (s[1]!=null) //Исключаем добавление пустых строк. Пустые строки появляются, когда мы делаем data.Add(new string[9]);

{

dataGridView1.Rows.Add(s); //В dataGridView1 добавляем всё из строки s массива data.

}

}

}

}

else

{

dataGridView1.Dock = DockStyle.Top;

int count = 0;

while (reader.Read())

{

data.Add(new string[8]);

if ((reader[6].ToString() == "0") //Рейс без пересадок

&&(DateTime.Parse(reader[1].ToString()).ToShortDateString() == date) //Дата вылета совпадает с датой выбранной пользователем

&& (DateTime.Parse(DateTime.Parse(reader[2].ToString()).ToShortTimeString()) > DateTime.Parse(DateTime.Now.ToShortTimeString())) //Если время вылета больше времени сейчас(самолёт ещё не улетел)

&& (reader[4].ToString() == departure) && (reader[5].ToString() == arrival) //совпадают города вылета и прибытия

&& reader[8].ToString() != "0")//Количество билетов != 0

{

count++;

data[data.Count - 1][0] = reader[0].ToString();

data[data.Count - 1][1] = reader[1].ToString();

data[data.Count - 1][1] = DateTime.Parse(data[data.Count - 1][1]).ToShortDateString();

data[data.Count - 1][2] = reader[2].ToString();

data[data.Count - 1][2] = DateTime.Parse(data[data.Count - 1][2]).ToShortTimeString();

data[data.Count - 1][3] = reader[3].ToString();

if (DateTime.Today > DateTime.Parse(data[data.Count - 1][3])) //Если дата прибытия не совпадает с датой вылета, выводить и дату и время

{

data[data.Count - 1][3] = DateTime.Parse(data[data.Count - 1][3]).ToShortDateString() + DateTime.Parse(data[data.Count - 1][3]).ToShortTimeString();

}

else //Иначе - только время

{

data[data.Count - 1][3] = DateTime.Parse(data[data.Count - 1][3]).ToShortTimeString();

}

data[data.Count - 1][4] = reader[4].ToString();

data[data.Count - 1][5] = reader[5].ToString();

data[data.Count - 1][6] = reader[8].ToString();

data[data.Count - 1][7] = reader[9].ToString();

}

}

if (count == 0)

{

label1.Text = "Ничего нет(\nК сожалению по выбранному вами маршруту в этот день нет рейсов без пересадок или все билеты распроданы";

label1.Visible = true;

label1.Dock = DockStyle.Fill;

flowLayoutPanel1.Visible = false;

}

else

{

foreach (string[] s in data)

{

if (s[1] != null)

{

dataGridView2.Rows.Add(s);

}

}

}

}

reader.Close();

connection.Close();

}

## **ПРИЛОЖЕНИЕ В**

ОПИСАНИЕ ФУНКЦИИ LoadData() формы FormBuy

private void LoadData(string flight\_number\_value)

{

MySqlConnection connection = new MySqlConnection("server=localhost;port=3306;username=root;password=root;database=flightdb");

connection.Open();

string query = "SELECT Tickets, Price FROM flight WHERE Flight\_number = @fn";

MySqlCommand command = new MySqlCommand(query, connection);

command.Parameters.Add("@fn", MySqlDbType.VarChar).Value = flight\_number\_value;

MySqlDataReader reader = command.ExecuteReader();

string data = "";

string price = "";

if (reader.Read())

{

data = reader[0].ToString();

price = reader[1].ToString();

}

label7.Text = flight\_number\_value;

label8.Text = data;

label10.Text = price;

label11.Text = "Сумма заказа: ";

reader.Close();

connection.Close();

}