МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТ:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема «Поисковик авиабилетов»

Исполнитель

студент (ка) 2 курса группы 4 Козека Елизавета Максимовна

(Ф.И.О.)

Руководитель работы преп.-ст. Север А.С.

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

Курсовой проект защищен с оценкой

Председатель Смелов В.В.

(подпись)

Минск 2024

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc135313107)

[1 Аналитический обзор прототипов и литературных источников 5](#_Toc135313108)

[1.1 Анализ прототипов 5](#_Toc135313109)

[1.2 Аналитический обзор источников 10](#_Toc135313110)

[1.3 Требования к проекту 11](#_Toc135313110)

[2 Анализ требований к программному средству 12](#_Toc135313111)

[2.1 Описание средств разработки 12](#_Toc135313112)

[2.1.1 Microsoft Visual Studio 2022 12](#_Toc135313113)

[2.1.2 Программная платформа .NET 8.0 12](#_Toc135313114)

[2.1.3 Язык программирования C# 13](#_Toc135313115)

[2.1.4 Технология WPF 13](#_Toc135313116)

[2.1.5 Расширяемый язык разметки XAML 13](#_Toc135313117)

[2.1.6 Технология ADO.NET 13](#_Toc135313115)

[2.1.7 Microsoft SQL Server 13](#_Toc135313116)

[2.1.8 Паттерн MVVM 14](#_Toc135313116)

[2.2 Архитектура системы 14](#_Toc135313118)

[2.3 Диаграмма вариантов использования 16](#_Toc135313119)

[3 Проектирование программного средства 17](#_Toc135313120)

[3.1 Общая структура приложения 17](#_Toc135313121)

[3.2 Взаимоотношение между классами 19](#_Toc135313122)

[3.3 Проектирование логической структуры базы данных 20](#_Toc135313123)

[3.4 Проектирование архитектуры приложения 23](#_Toc135313124)

[3.5 Проектирование архитектуры окон 24](#_Toc135313124)

[3.6 Схема работы приложения 27](#_Toc135313124)

[4 Реализация программного средства 28](#_Toc135313125)

[4.1 Диаграмма классов 28](#_Toc135313126)

[4.2 Реализация паттерна MVVM 28](#_Toc135313127)

[4.3 Реализация взаимодействия с базой данных 31](#_Toc135313129)

[4.4 Реализация методов регистрации и авторизации пользователей 32](#_Toc135313130)

[5 Тестирование, проверка работоспособности и анализ результатов 33](#_Toc135313134)

[5.1 Тестировании регистрации и авторизации 33](#_Toc135313135)

[5.2 Тестирование формы добавления аэропорта 36](#_Toc135313136)

[5.3 Тестирование формы добавления маршрута 38](#_Toc135313136)

[5.4 Тестирование формы добавления авиарейса 40](#_Toc135313136)

[5.5 Тестирование форм поиска и бронирования авиарейса 43](#_Toc135313136)

[6 Руководство по использованию 45](#_Toc135313137)

[ЗАКЛЮЧЕНИЕ 50](#_Toc135313138)

[Список использованных источников 51](#_Toc135313139)

[ПРИЛОЖЕНИЕ А 52](#_Toc135313140)

[ПРИЛОЖЕНИЕ Б 53](#_Toc135313141)

[ПРИЛОЖЕНИЕ В 55](#_Toc135313142)

[ПРИЛОЖЕНИЕ Г 56](#_Toc135313143)

[ПРИЛОЖЕНИЕ Д 57](#_Toc135313144)

[ПРИЛОЖЕНИЕ Е 58](#_Toc135313145)

[ПРИЛОЖЕНИЕ Ж 61](#_Toc135313146)

**ВВЕДЕНИЕ**

Раньше для того, чтобы заказать авиарейс приходилось проводить много времени в очередях и в аэропорту в целом. Также многие не могли попасть на нужный рейс по причине того, что просто опоздали.

Поэтому для того, чтобы избавить заказчиков от ненужных хлопот и сохранить их время разрабатываются приложения, которые ускоряют процесс заказа рейса, а также делают его удобным.

Данные приложения позволяют пользователям с легкостью заказать билет в любую точку мира и при этом они потратят гораздо меньше времени, чем если бы они производили заказ в аэропорту.

Целью данного курсового проекта является разработка ПО, осуществляющего поиск и бронирование авиарейсов, регистрацию и аутентификацию пользователей. Также необходимо реализовать сохранение пользовательских рейсов, возможность добавления рейсов и маршрутов администратором.

Для успешной реализации курсового проекта необходимо:

* провести анализ соответствующей литературы;
* ознакомиться с прототипами программных средств выбранной тематики;
* определить функциональные требования;
* продумать структуру базы данных;
* продумать структуру проекта;
* реализовать программное средство;
* протестировать программное средство;
* написать руководство пользователя.

Главная задача данного курсового проектирования — это разработка программного средства, которое реализует все вышеперечисленные функции и решает поставленные задачи. Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП, база данных MS SQL Server, и технология Windows Presentation Foundation (WPF).

Содержание данной пояснительной записки отражает все этапы выполнения курсового проекта.

# 1 Аналитический обзор литературы и формирование требований

## **1.1 Анализ прототипов**

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов и литературных источников.

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

Были рассмотрены 3 аналога:

В качестве первого аналога был выбран метапоисковый сервис «Aviasales». Данное программное средство не продаёт билеты. Найдя нужный билет, пользователь переходит на страницу авиакассы или авиакомпании, чтобы совершить оплату. Оплата происходит на странице авиакомпании или авиакассы.

Плюсы:

* обширный выбор авиакомпаний и онлайн-агенств;
* удобный интерфейс и навигация;
* гибкость поиска (различные фильтры и опции для уточнения поиска);
* информирование пользователей о скидках и специальных предложениях.

Минусы:

* ограниченные опции бронирования (перенаправление на сайт авиакомпа-нии или агентства при бронировании билета);
* не предлагаются все доступные авиакомпании;
* зависимость от сторонних поставщиков.

Чтобы найти билеты, нужно заполнить поля с направлениями и датой. Далее нужно дождаться конца поиска, чтобы увидеть все доступные варианты, в том числе самые быстрые и дешёвые.

Интерфейс для поиска билета представлен на рис. 1.1.

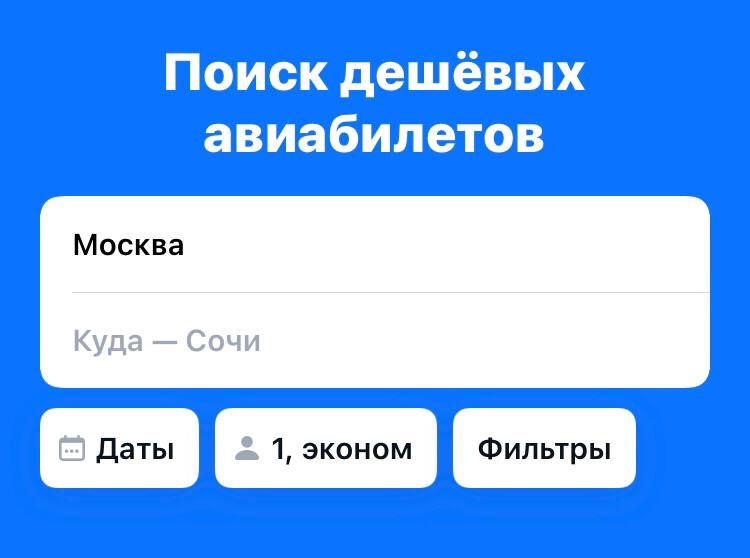


Рисунок 1.1 — Поиск билета в приложении «Aviasales»

Результаты поиска — это варианты перелётов от агентств и авиакомпаний с ценами, датой и временем вылета и прилёта, аэропортами, количеством пересадок и продолжительностью полёта, наличием или отсутствием ручной клади/багажа.

Интерфейс для отображения результатов поиска представлен на рис. 1.2.

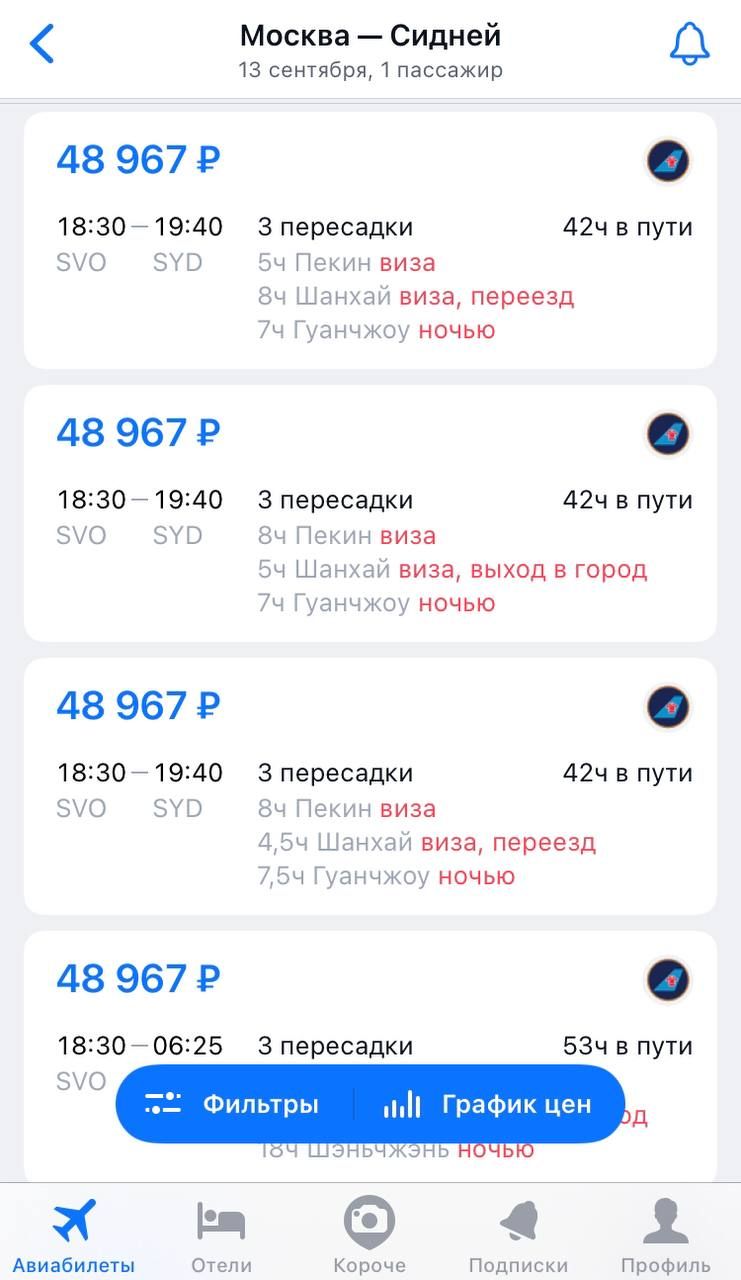


Рисунок 1.2 — Результат поиска билета в приложении «Aviasales»

Однако программное средство «Aviasales» не предлагает функцию хранения истории купленных билетов, так как приложение лишь подбирает подходящие пользователю варианты.

В качестве второго аналога было выбрано российское туристическое агентство «Anywayanyday», работающее в режиме онлайн. Все услуги предоставляются через веб-сайт и мобильные приложения. «Anywayanyday» было создано в 2008 году на месте закрывшегося интернет-портала по продаже авиабилетов «Avantix», потому и сохранило данный функционал.

Плюсы:

* широкий выбор авиакомпаний;
* удобный поиск и бронирование;
* оформление и оплата заказа занимают относительно небольшое количество времени;
* цена на авиабилеты отображается окончательная и не меняется в процессе бронирования;
* при оформлении заказа можно получить бонусные баллы, которые могут быть использованы при следующих заказах.

Минусы:

* ограниченный выбор региональных авиалиний;
* отсутствие гибкого поиска по датам;
* возможно небольшое расхождение в ценах по сравнению с другими платформами или непосредственно с сайтами компаний.

В главном меню приложения пользователь может осуществить поиск необходимого ему рейса.

Форма для поиска авиабилетов представлена на рис. 1.3.

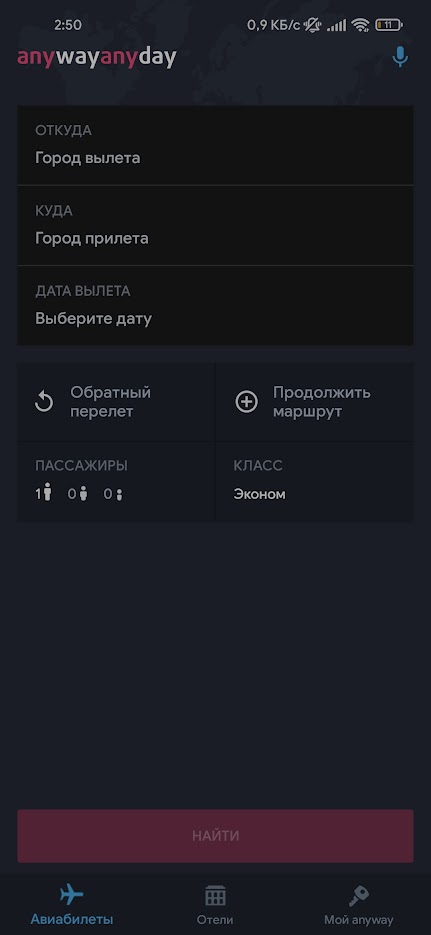


Рисунок 1.3 — Поиск авиабилета в приложении «Anywayanyday»

Результаты поиска авиарейса по введенным пользователем данным представлены на рис. 1.4.

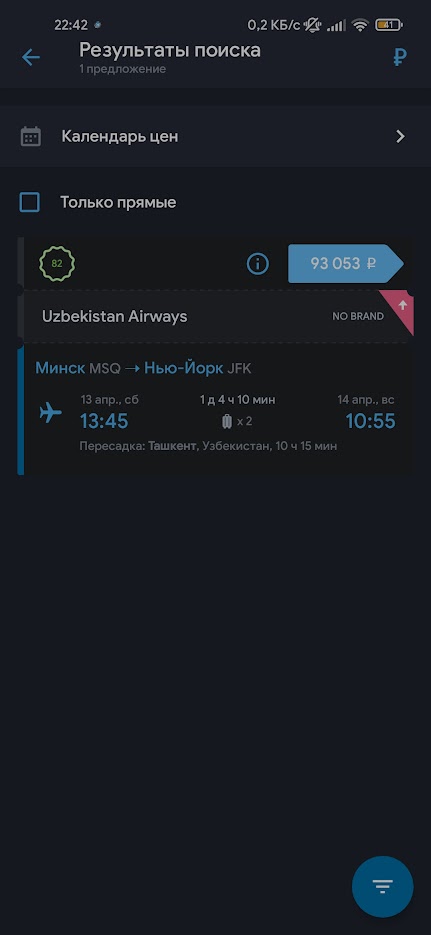


Рисунок 1.4 — Результат поиска билета в приложении «Anywayanyday»

Рассматриваемое программное средство также предлагает возможность сохранения истории заказов пользователя, а также возможность просмотра календаря цен и возможность показа только билетов для авиарейсов без пересадок. Пример представлен на рис. 1.5.



Рисунок 1.5 — Хранение истории заказов в приложении «Anywayanyday»

«Anywayanyday» также имеет систему регистрации и авторизации пользователей для синхронизации данных между сайтом и мобильным приложением. Страница регистрации и авторизации представлена на рис. 1.6.

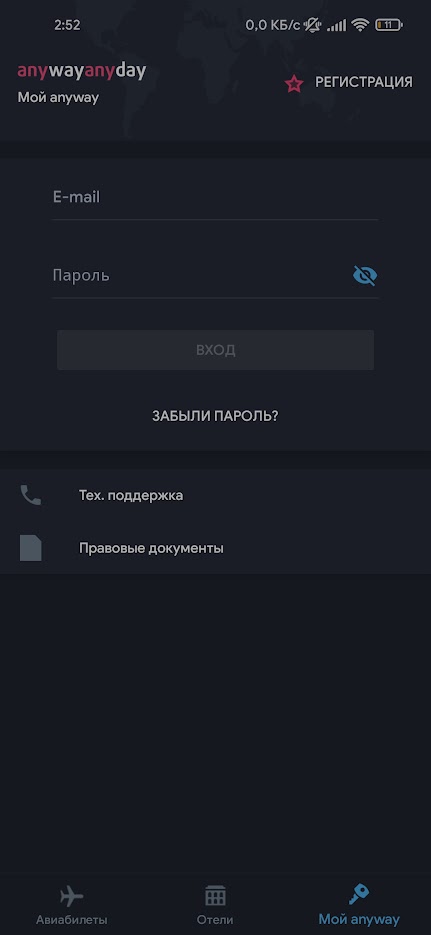


Рисунок 1.6 — Система регистрации и авторизации в приложении «Anywayanyday»

В качестве третьего аналога был выбран сайт национальной белорусской авиакомпании «Belavia». Основной акцент в своей работе авиакомпания делает на предоставлении возможности клиентам летать как прямыми рейсами, так и без промедления продолжить путешествие в любую точку мира, стараясь учесть при этом все пожелания пассажира.

Плюсы:

* легкий удобный поиск, выбор и бронирование желаемого рейса;
* подробная информация о рейсах;
* поддерживает систему управления бронированиями;
* имеется программа поощрения пассажиров.

Минусы:

* предлагает рейсы только на своих собственных маршрутах;
* отсутствие мобильного приложения;
* ограниченные способы оплаты.

На главной странице сайта пользователь может осуществить поиск интересующего его авиарейса. Форма для поиска билета представлена на рис. 1.7.

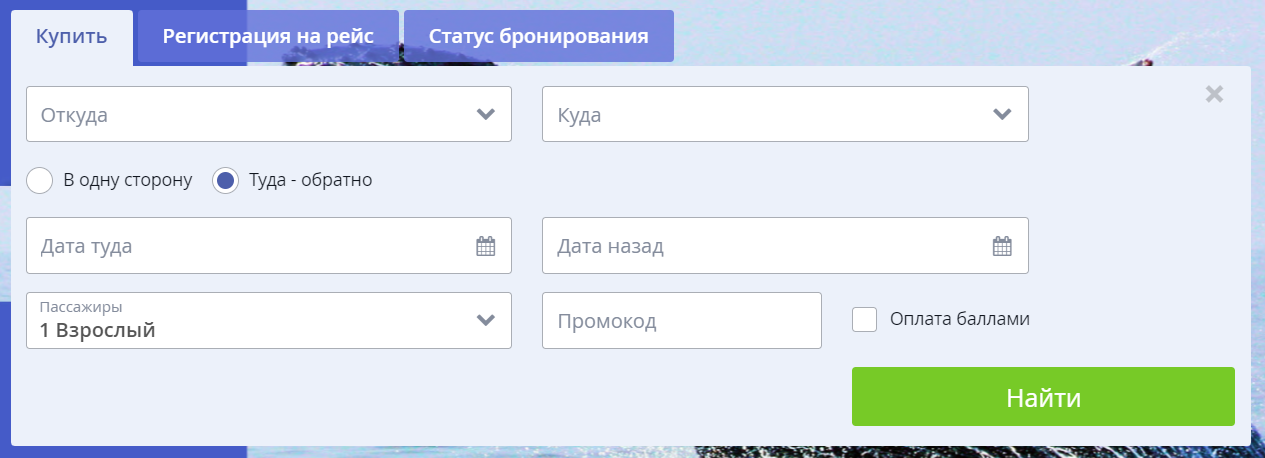


Рисунок 1.7 — Поиск авиарейса на сайте «Belavia»

После заполнения формы сайт направляет пользователя на страницу с результатами поиска. Она изображена на рис. 1.8.

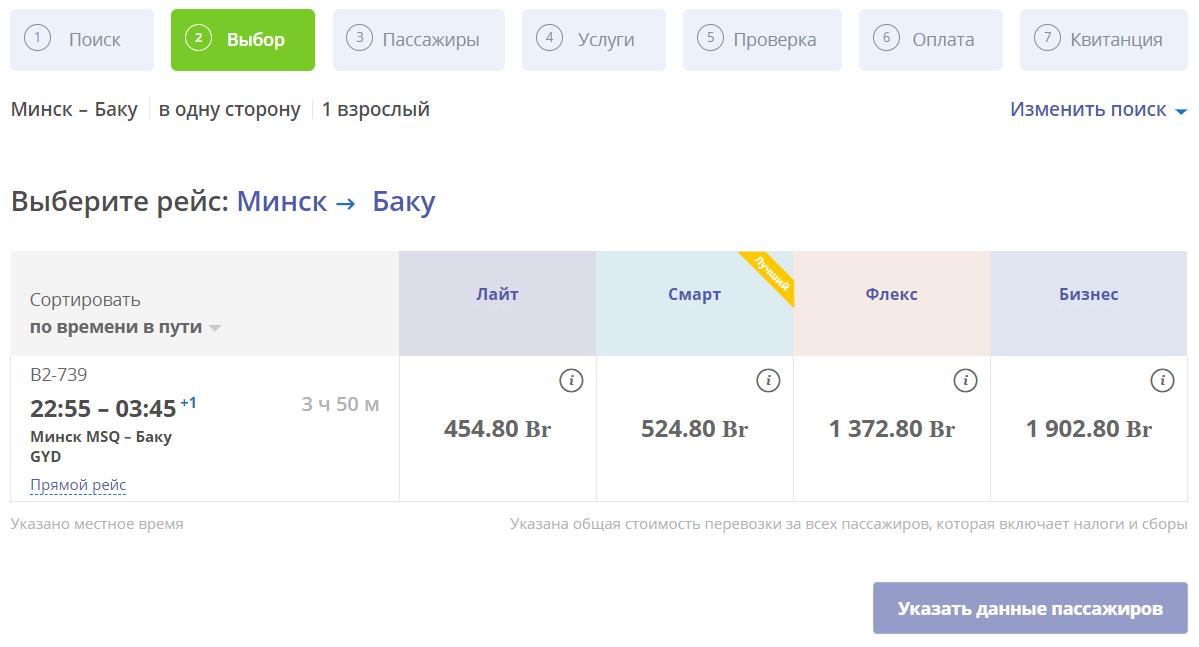


Рисунок 1.8 — Результат поиска билета на сайте «Belavia»

Как видно на рисунке, результаты поиска можно сортировать по различным критериям.

Рассматриваемый сайт также поддерживает систему регистрации и авторизации пользователей. Форма для регистрации/авторизации представлена на рис. 1.9.

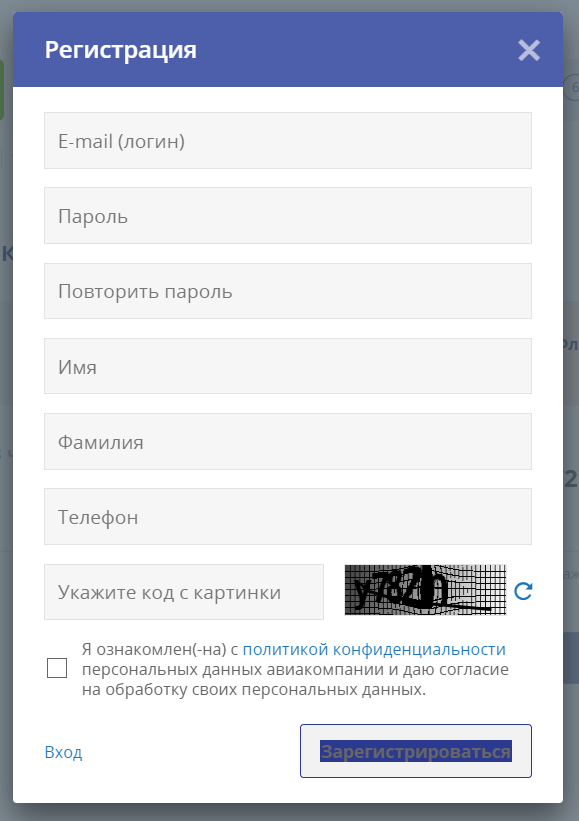


Рисунок 1.9 — Система регистрации и авторизации на сайте «Belavia»

Анализируя аналоги, можно сделать вывод, что все три приложения обладают современным интерфейсом и стараются обеспечить удобство и легкость в использовании для клиентов. Однако, каждое приложение может иметь свои особенности и преимущества, которые следует учитывать.

1.2 Аналитический обзор источников

В ходе подготовки была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

При разработке окон для регистрации и авторизации был использован подход, описанный в статье «WPF – система авторизации и регистрации». В статье были рассмотрены алгоритм работы системы авторизации и регистрации и пример создания окон регистрации и авторизации.

Принцип работы с SQL были получены из статьи «Подключение к базе данных». В статье было рассмотрено подключение необходимых библиотек, работа с SQL.

Дополнительная информации о принципах работы с WPF была получена из интернет-источника «Metanit», содержащего практические советы по работе с технологией.

## **1.3 Требования к проекту**

После рассмотрения представленных выше аналогов было решено использовать следующий функционал: возможность авторизации и регистрации пользователей, поиск и бронирование билетов пользователями, просмотр истории заказов, а также возможность добавления администратором новых маршрутов и рейсов.

Обзор вышеперечисленных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте.

Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

* управление администратором базой данных: добавление новых самолётов, аэропортов, компаний, маршрутов и билетов, а также просматривать и удалять билеты и заказы пользователей;
* возможность создать новый аккаунт пользователя или войти в уже существующий;
* возможность пользователей просматривать историю заказов;
* возможность пользователей осуществлять поиск авиабилетов;
* возможность пользователей осуществлять бронирование авиабилетов.

Таким образом, в ходе работы над этим разделом были сформулированы основные функциональные требования для проектирования программного средства.

# 2 Анализ требований к программному средству и разработка функциональных требований

Анализ требований — это процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения.

Основная цель анализа требований в проектах заключается в получении максимально полной информации о клиенте, его задачах и особенностях проекта. Анализ требований помогает определить границы проекта, выявить возможные риски и ключевые требования, как методологические, так и технологические. На этом этапе формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые будут использоваться для оценки результатов внедрения. Определение и описание требований является важным этапом, так как эти требования оказывают влияние на все последующие этапы проекта и определяют его успех.

# 2.1 Описание средств разработки

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio 2022;
* программная платформа .NET Framework 8.0;
* язык программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* технология ADO.NET;
* СУБД Microsoft SQL Server;
* паттерн MVVM.

# 2.1.1 Microsoft Visual Studio 2022

Microsoft Visual Studio 2022 — это интегрированная среда разработки (IDE) от Microsoft, которая обеспечивает разработчиков инструментами и возможностями для создания приложений с использованием Windows Presentation Foundation (WPF).

# 2.1.2 Программная платформа .NET 8.0

Платформа .NET — это программный фреймворк, разработанный компанией Microsoft, который в основном работает на операционной системе Microsoft Windows. Он предоставляет среду выполнения для запуска приложений и набор библиотек, которые используются для создания программных приложений. Фреймворк .NET состоит из нескольких компонентов, включая общую языковую среду (Common Language Runtime — CLR), библиотеку классов .NET Framework и различные инструменты разработки.

# 2.1.3 Язык программирования C#

В качестве языка программирования используется C# — основной язык разработки в .NET. Язык объектно-ориентированный, имеет строгую статическую типизацию, поддерживает перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения. Используется как основной язык в технологии WPF.

# 2.1.4 Технология WPF

Для предоставления пользовательского интерфейса и разграничения дизайна и бизнес-логики используется технология Microsoft WPF — аналог WinForms, система для построения клиентских приложений Windows с возможностями взаимодействия с пользователем и графическая подсистема в составе .NET, использующая язык разметки XAML.

# 2.1.5 Расширяемый язык разметки XAML

WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. XAML представляет собой язык декларативного описания интерфейса, основанный на XML.

# 2.1.6 Технология ADO.NET

ADO.NET (ActiveX Data Objects .NET) — это набор технологий, предоставляющих программистам доступ к данным из различных источников данных в среде .NET. ADO.NET является частью .NET Framework и обеспечивает эффективное взаимодействие с базами данных и другими источниками данных.

ADO.NET поддерживает различные источники данных, включая реляционные базы данных, XML-файлы и объекты в памяти. Он также предоставляет возможности для выполнения транзакций, обработки ошибок, кэширования данных и других задач, связанных с доступом к данным.

# 2.1.7 Microsoft SQL Server

Для организации баз данных MS SQL Server использует реляционную модель, которая предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта. Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос 10 на языке SQL должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения. Основной используемый язык запросов — Transact-SQL — реализован на структурированном языке запросов (SQL) с расширениями.

# 2.1.8 Паттерн MVVM

MVVM (Model-View-ViewModel) - это паттерн архитектуры, используемый в разработке программного обеспечения для разделения пользовательского интерфейса (View) от бизнес-логики (Model) и связывания их через промежуточный слой ViewModel.

В MVVM модель представляет данные и бизнес-логику приложения, которая может включать в себя операции чтения/записи данных из и в источники данных, такие как база данных или веб-сервисы.

Представление (View) отвечает за отображение данных и взаимодействие с пользователем. Оно не содержит бизнес-логику и должно быть максимально независимым от модели.

ViewModel представляет промежуточный слой между View и Model. Он содержит логику, необходимую для обработки пользовательских действий, обновления данных в модели и уведомления View об изменениях. ViewModel предоставляет свойства и команды, которые привязываются к элементам пользовательского интерфейса во View, позволяя им отображать данные и реагировать на действия пользователя.

MVVM позволяет достичь разделения ответственности между компонентами приложения, облегчает тестирование и повышает переиспользуемость кода. Он также способствует лучшей поддержке параллельной разработки пользовательского интерфейса и бизнес-логики, так как разработчики могут работать независимо над своими частями приложения.

**2.2 Архитектура системы**

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

* Клиентская часть. В этом модуле осуществляется взаимодействие пользователя с приложением с помощью графического интерфейса. В клиентской части пользователь сможет получить полную информацию об объектах приложения, в зависимости от присвоенной ему роли (пользователь или администратор).
* База данных. В базе данных описываемого приложения хранится информация о рейсах, аэропортах, самолётах, авиакомпаниях, билетах и других связанных с авиаперевозками сущностях.

Различают три уровня требований к проекту:

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

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

* простота и лёгкость интерфейса;
* использование принципов ООП;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД).

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

Следующими требованиями являются требования пользователей. Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы.

Программное средство должно предоставлять следующие функциональные возможности:

Для пользователя:

* регистрация;
* авторизация;
* просмотр доступных рейсов;
* поиск авиабилетов;
* фильтрация авиабилетов;
* просмотр истории заказов;
* бронирование места на рейс;
* возможность сменить пароль.

Для администратора:

* авторизация;
* просмотр авиарейсов;
* просмотр заказов пользователей;
* удаление заказов пользователей;
* добавление нового аэропорта;
* удаление аэропорта;
* добавление нового самолёта;
* удаление самолёта;
* добавление новой авиакомпании;
* удаление авиакомпании;
* добавление нового маршрута;
* удаление маршрута;
* добавление нового авиарейса (билета);
* удаление авиарейса (билета).

После проведения анализа были выявлены следующие функциональные требования:

* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить авторизацию и входить в систему под соответствующим пользовательским именем.

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

2.3 Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram) является графическим средством моделирования, которое позволяет представить функциональность системы или программного продукта с точки зрения ее пользователей. Она помогает идентифицировать и описать различные сценарии использования системы, включая взаимодействие между актерами (пользователями) и самой системой.

Диаграмма вариантов использования состоит из актеров, вариантов использования (Use Case) и связей между ними. Актеры представляют различных пользователей или внешние системы, которые взаимодействуют с программой. Варианты использования представляют собой функциональные возможности системы, которые могут быть выполнены пользователями или другими актерами.

Преимущества использования диаграммы вариантов использования включают:

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

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

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

UML-диаграмма вариантов использования для программного средства «BelAirWays» представлена в приложении А.

3 Проектирование программного средства

3.1 Общая структура приложения

Общая структура проекта Windows Presentation Foundation (WPF) – это организация файлов и папок в рамках проекта WPF. Структура разрабатываемого проекта «BelAirWays» представлена на рисунке 3.1.

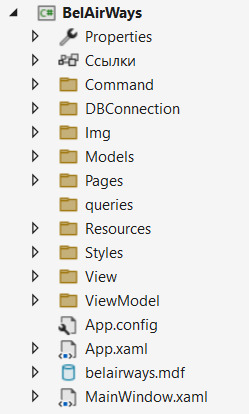


Рисунок 3.1 — Структура проекта

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

Таблица 3.1 — Описание структуры папок и файлов проекта

|  |  |
| --- | --- |
| Имя файла/папки | Содержание |
| App.config | Файл с параметрами проекта |
| App.xaml | Определение глобальных ресурсов |
| Папка Command | Директория, содержащая класс RelayCommand, реализующий работу с командами, который содержит все необходимые методы для работы с командами |
| Папка DBConnection | Директория, содержащая Singleton-класс, использующийся для создания и открытия подключения к базе данных. |
| Папка Img | Директория, содержащая графические материалы (картинки, иконки, курсоры) |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Папка Models | Директория, в которой располагаются файлы, отвечающие за логику приложения и всех страниц и окон, каждой странице соответствует свой Model класс. |
| Папка Pages | Директория, в которой располагаются классы страниц приложения, для более удобной навигации и отображения. |
| Папка Queries | Директория, содержащая файлы с SQL-запросами |
| Папка Resources | Директория, содержащая ресурсы приложения |
| Папка Styles | Директория, содержащая словари ресурсов стилей |
| Папка View | Директория, содержащая классы окон приложения, а также графическую информацию, использующуюся в окнах |
| Папка ViewModel | Директория, содержащая команды для связи Model и View для реализации MVVM |
| belairways.mdf | Файл базы данных |
| MainWindow.xaml | Главное окно приложения |

Таким образом, сформированная таблица помогает понять общую структура проекта проектируемого программного средства.

Структура папки Models (сущностей) представлена на рисунке 3.2.

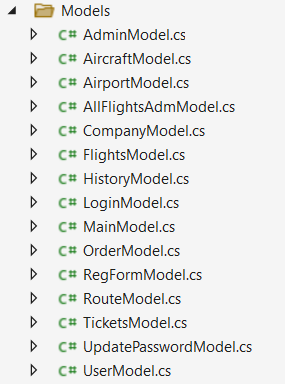


Рисунок 3.2 — Содержание папки Models

Более подробное описание структуры и файлов данной папки представлено в таблице 3.2.

Таблица 3.2 — Описание структуры папки Models

|  |  |
| --- | --- |
| Имя файла | Содержание |
| AdminModel.cs | Класс для определения и хранения учётной записи администратора |
| AircraftModel.cs | Класс для определения и хранения информации о самолётах |
| AirportModel.cs | Класс для определения и хранения информации об аэропортах |
| AllFlightsAdmModel.cs | Класс для определения и хранения информации о билетах на стороне администратора |
| CompanyModel.cs | Класс для определения и хранения информации о компаниях |
| FlightsModel.cs | Класс для определения и хранения информации об авиарейсах |
| HistoryModel.cs | Класс для определения и хранения истории заказов |
| LoginModel.cs | Класс для определения и хранения функции входа в учётную запись |
| MainModel.cs | Класс для определения и хранения функций стартового окна |
| OrderModel.cs | Класс для определения и хранения информации о заказах |
| RegFormModel.cs | Класс для определения и хранения функции регистрации новой учётной записи |
| RouteModel.cs | Класс для определения и хранения информации о маршрутах |
| TicketsModel.cs | Класс для определения и хранения информации о билетах на стороне пользователя |
| UpdatePasswordModel.cs | Класс для определения и хранения функции изменения пароля |
| UserModel.cs | Класс для определения и хранения учётной записи пользователя |

В целом, описание структуры проекта позволяет лучше понимать, как устроено программное средство и какие компоненты в нем присутствуют.

**3.2 Взаимоотношение между классами**

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

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена в приложении Б.

**3.3 Проектирование логической структуры базы данных**

Для создания базы данных описываемого приложения использовалась система управления реляционными базами данных Microsoft SQL Server 2022.

База данных — это совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимо от прикладных программ.

Система управления базами данных — совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами.

База данных данного курсового проекта состоит из 8 таблиц. Схема базы данных BelAirWays изображена на рисунке 3.3.

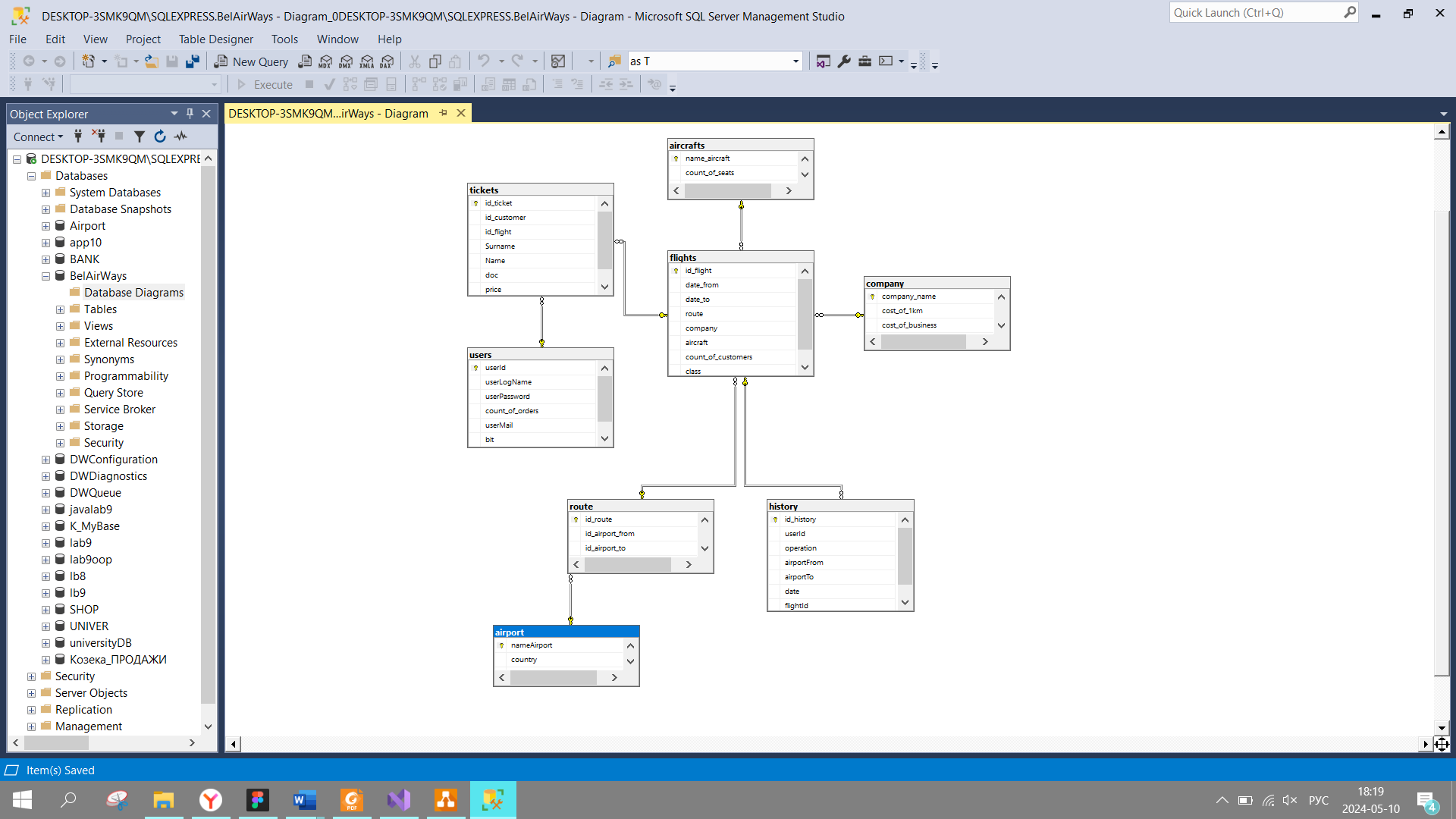


Рисунок 3.3 — Схема базы данных приложения

Остановимся более подробно на некоторых основных таблицах и рассмотрим, какие поля они содержат.

В таблице 3.3 проиллюстрирована структура таблицы «airports», которая содержит информацию об аэропортах.

Таблица 3.3 — Таблица «airports»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| nameAirport | nvarchar(50) | Содержит название аэропорта |
| country | nvarchar(50) | Содержит название страны, в которой находится аэропорт |
| town | nvarchar(50) | Содержит название города, в котором находится аэропорт |

В таблице 3.4 проиллюстрирована структура таблицы «aircrafts», которая содержит информацию о самолётах. В данной таблице есть 3 поля: nameAircraft —содержит название самолёта; countOfSeats — содержит количество мест в самолёте, в которой находится аэропорт; speed — содержит скорость самолёта.

Таблица 3.4 — Таблица «aircrafts»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| nameAircraft | nvarchar(50) | Содержит название самолёта |
| countOfSeats | int | Содержит количество мест в самолёте, в которой находится аэропорт |
| speed | int | Содержит скорость самолёта |

В таблице 3.5 проиллюстрирована структура таблицы «companies», которая содержит информацию о компаниях, предоставляющих авиарейсы.

Таблица 3.5 — Таблица «companies»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| nameCompany | nvarchar(50) | Содержит название компании |
| costOfKm | real | Содержит цену 1 километра полёта |
| costOfBusiness | real | Содержит цену на самолёт бизнес-класса |
| costOfEconom | real | Содержит цену на самолёт эконом-класса |

На рисунке 3.7 проиллюстрирована структура таблицы «tickets», которая содержит информацию о заказах.

Таблица 3.5 — Таблица «companies»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| idTicket | int | Содержит идентификатор заказа в базе данных |
| idCustomer | int | Содержит идентификатор пользователя, заказавшего билет |
| idFlight | int | Содержит идентификатор авиарейса в базе данных |
| surname | nvarchar(200) | Содержит фамилию пользователя |

Продолжение таблицы 3.5

|  |  |  |
| --- | --- | --- |
| name | nvarchar(200) | Содержит имя пользователя |
| doc | nvarchar(100) | Содержит номер паспорта пользователя |
| price | int | Содержит цену на билет |

В таблице 3.6 проиллюстрирована структура таблицы «flights», которая содержит информацию об авиарейсах.

Таблица 3.6 — Таблица «flights»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| idFlight | int | Содержит идентификатор авиарейса в базе данных |
| dateFrom | date | Содержит дату отправления |
| dateTo | date | Содержит дату прибытия |
| idRoute | int | Содержит идентификатор маршрута в базе данных |
| company | nvarchar(50) | Содержит название компании, предоставляющей рейс |
| aircraft | nvarchar(50) | Содержит название самолёта |
| countOfCustomers | int | Содержит количество пользователей, оформивших заказ на авиарейс |
| class | nvarchar(50) | Содержит класс самолёта |

В таблице 3.7 проиллюстрирована структура таблицы «routes», которая содержит информацию о маршрутах.

Таблица 3.7 — Таблица «routes»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| idRoute | int | Содержит идентификатор маршрута в базе данных |
| idAirportFrom | nvarchar(50) | Содержит идентификатор аэропорта, из которого отправляется самолёт |
| idAirportTo | nvarchar(50) | Содержит идентификатор аэропорта, в который прибывает самолёт |
| lengthOfRoute | int | Содержит длину маршрута |

В таблице 3.8 проиллюстрирована структура таблицы «users», которая содержит информацию о зарегистрированных пользователях.

Таблица 3.8 — Таблица «users»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| idUser | int | Содержит идентификатор пользователя в базе данных |
| userLogName | nvarchar(200) | Содержит логин пользователя |
| userPassword | nvarchar(500) | Содержит пароль пользователя |

Продолжение таблицы 3.8

|  |  |  |
| --- | --- | --- |
| userMail | int | Содержит адрес электронной почты пользователя |
| countOfOrders | nvarchar(200) | Содержит количество заказов, сделанных пользователем |
| bit | int | Содержит числовое значение, которое определяет, является ли пользователь администратором или обычным пользователем |

В таблице 3.9 проиллюстрирована структура таблицы «history», которая содержит историю заказов.

Таблица 3.9 — Таблица «history»

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| idHistory | int | Содержит идентификатор записи в истории в базе данных |
| idUser | int | Содержит идентификатор пользователя в базе данных |
| townFrom | nvarchar(100) | Содержит город, из которого отправляется самолёт |
| townTo | nvarchar(100) | Содержит город, в который прибывает самолёт |
| date | date | Содержит дату отправления |
| idFlight | int | Содержит идентификатор авиарейса в базе данных |

В целом, структура базы данных «BelAirWays» предоставляет необходимую основу для хранения и управления информацией в приложении «BelAirWays».

3.4 Проектирование архитектуры приложения

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

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

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

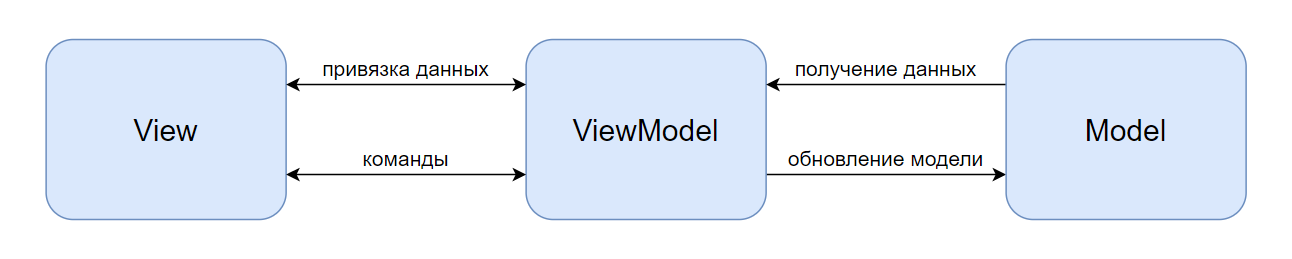


Рисунок 3.12 — Структура шаблона MVVM

Model (модель) описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную с этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления. Часто модель реализует интерфейс INotifyPropertyChanged, который позволяет системе автоматически обнаруживать изменения свойств модели и облегчает привязку к представлению, хотя сама модель не взаимодействует напрямую с представлением.

View (представление) определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление — это код XAML, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов. Представление не обрабатывает события, а в основном выполняет действия посредством команд.

ViewModel (модель представления) связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны. Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд.

3.5 Проектирование архитектуры окон

Приложение включает в себя 5 окон и 10 страниц.

При запуске приложения пользователь видит стартовое окно, в котором размещены 2 кнопки: «Вход» и «Регистрация». Пользователь может либо перейти к окну «Вход», ввести данные и получить доступ к стандартному функционалу приложения (при входе под ролью обычного пользователя) или же к расширенному функционалу (при входе под ролью администратора), либо перейти к окну «Регистрация», после которого он может ввести данные в форму входа и также получить доступ к приложению.

Внешний вид стартового окна приложения «BelAirWays» представлен на рисунке 3.13.

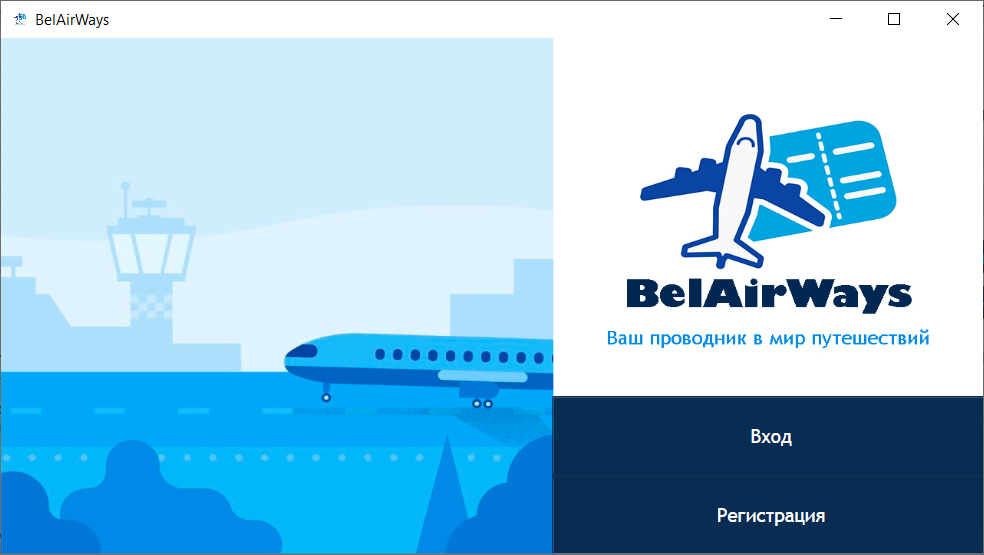
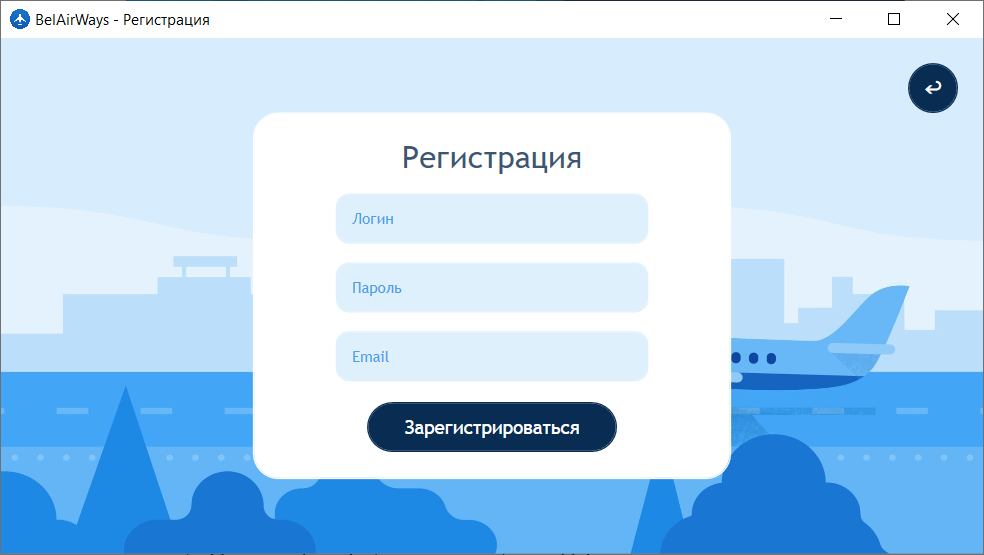
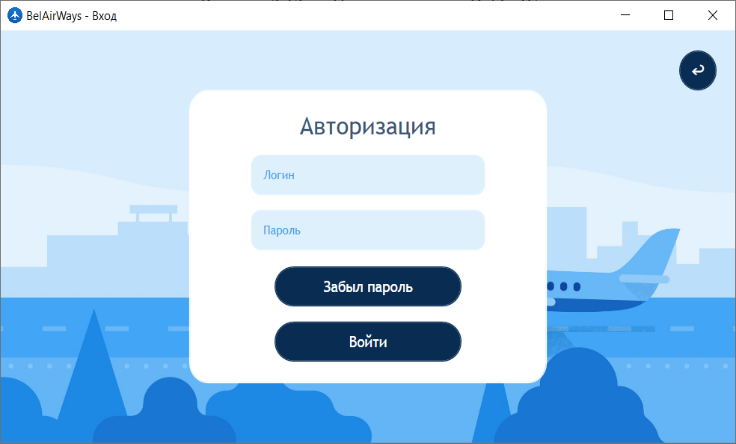


Рисунок 3.13 — Стартовое окно «BelAirWays»

Внешний вид окна авторизации представлен на рисунке 3.14.



а б

Рисунок 3.14 — Окна авторизации (а) и регистрации (б)

После входа в приложение, в зависимости от роли, пользователь переходит либо к окну администратора, либо к окну стандартного пользователя.

Боковое меню состоит из набора кнопок, которые относятся к определенному типу пользователя согласно его уровню доступа. Эти кнопки позволяют перейти на страницы. Далее рассмотрим варианты окон.

Окно стандартного пользователя дает доступ к стандартному функционалу приложения. Пункты меню окна изображены на рисунке 3.15.

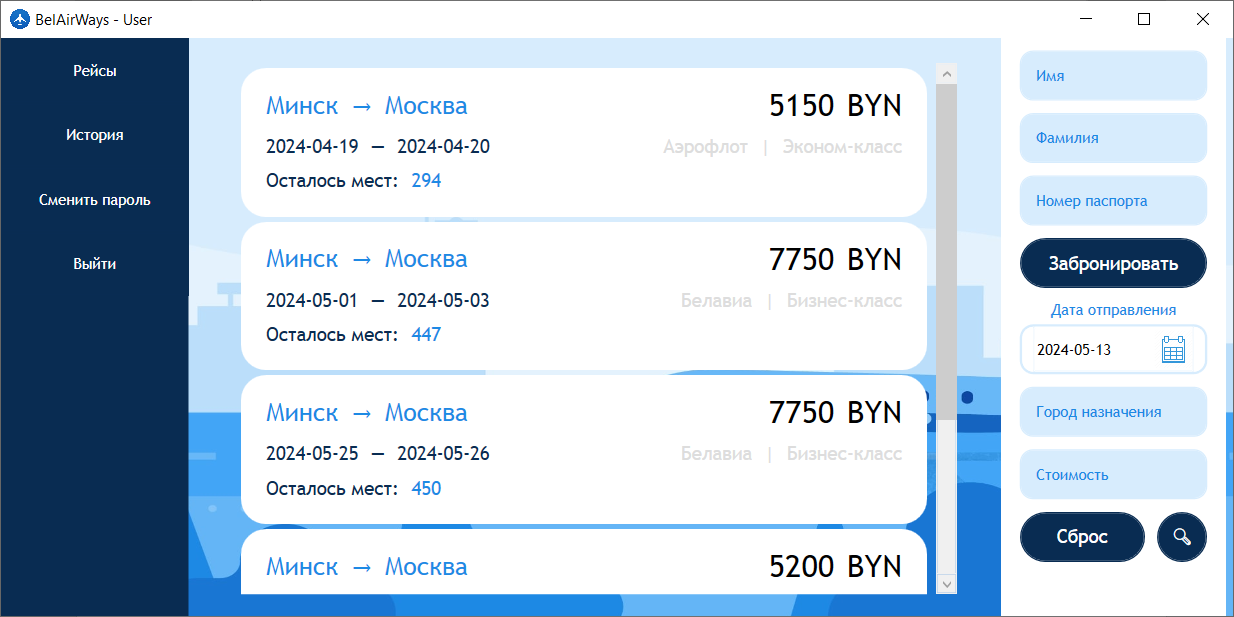


Рисунок 3.15 — Пункты меню для пользователя

При авторизации под логином и паролем администратора, можно попасть на страницу администрирования, пункты меню изображены на рисунке 3.16. Администратор имеет больше технических возможностей, чем остальные пользователи. Об этом свидетельствуют пункты меню, которые позволяют перейти на страницы с возможностью просмотра и редактирования таблиц БД.

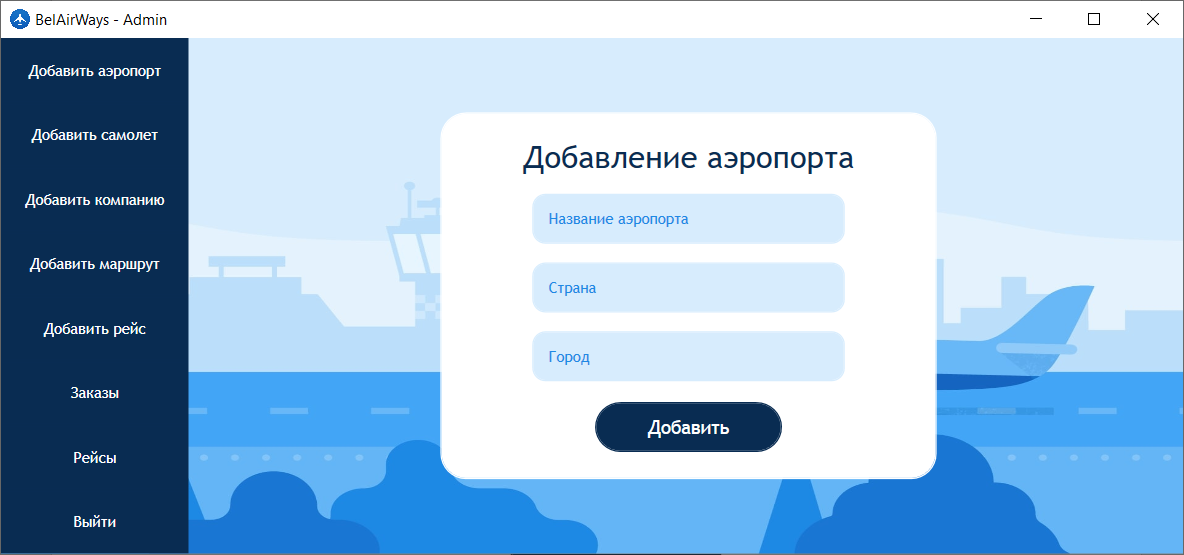


Рисунок 3.15 — Пункты меню для администратора

Таким образом, мы можем заметить, что для разных ролей пользователей приложение предоставляет разный функционал и, соответственно, разные окна и страницы.

3.6 Схема работы приложения

Для описания маршрутов работы программного продукта была создана диаграмма деятельности, которая показывает пути перехода между страницами приложения, представлена на рисунке 3.16.

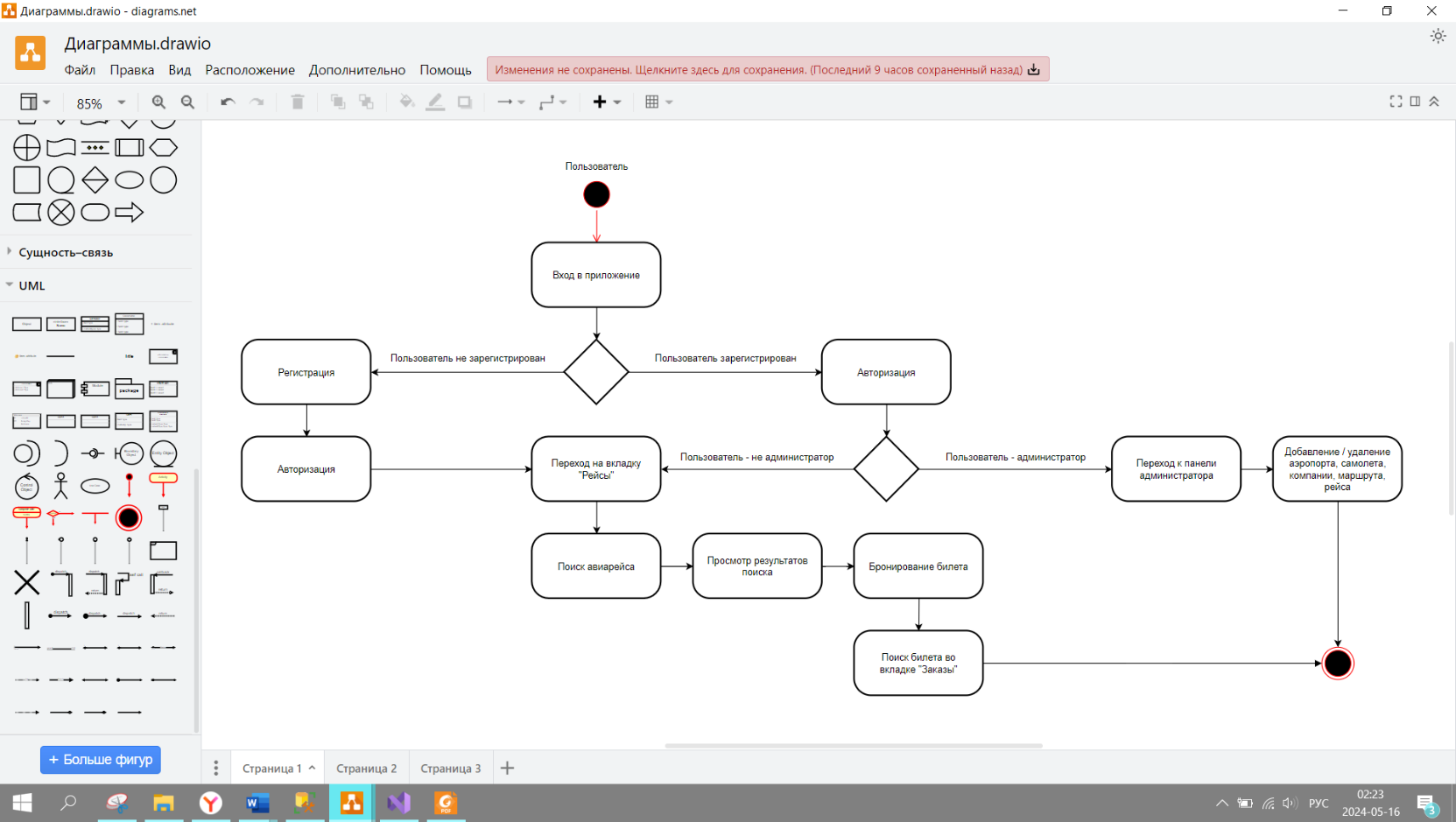


Рисунок 3.16 — Диаграмма деятельности

Для наглядного отображения взаимодействия между объектами системы в различные моменты времени в рамках одного сценария использования применяется диаграмма последовательностей в UML. Эта диаграмма позволяет иллюстрировать, как объекты системы взаимодействуют друг с другом для выполнения определенной функции, а также отображает порядок их взаимодействия в рамках конкретного случая использования.

Для отображения времени на диаграмме используется линия жизни объекта, которая представляет период его существования в системе и изображается вертикальной штриховой линией, направленной вниз. Объекты представлены прямоугольниками, а сообщения, которыми они обмениваются, обозначаются линиями с стрелками.

Диаграмма последовательностей позволяет наглядно представить взаимодействие между объектами и порядок выполнения действий во времени. Она является полезным инструментом для анализа и проектирования системы, а также может использоваться для коммуникации между разработчиками и заинтересованными сторонами.

Пример диаграммы последовательности представлен в приложении В.

4 Реализация программного средства

Следующим этапом разработки приложения является непосредственная реализация программного решения в соответствии с уже сформированными требованиями и шаблонами.

4.1 Диаграмма классов

В рамках разработки программного решения для организации задач и управления проектами, диаграмма классов является одним из ключевых инструментов моделирования и проектирования системы. Диаграмма классов представляет структуру системы, ее классы, атрибуты и отношения между классами.

В процессе проектирования программы для организации задач и управления проектами, диаграмма классов помогает определить основные классы, их свойства и методы. Она позволяет увидеть структуру данных, которые будут использоваться в программе, а также взаимосвязи между классами. Диаграмма классов также помогает лучше понять концептуальную модель системы и обеспечивает основу для дальнейшей разработки.

Диаграмма классов представлена в приложении Б.

4.2 Реализация паттерна MVVM

Как отмечалось в предыдущих разделах, при разработке приложения «BelAirWays» использовался архитектурный паттерн MVVM. Для его реализации необходимо использовать следующие классы:

* ViewModel (модель представления);
* View (визуальный интерфейс);
* Model (модель данных).

На примере страницы AddAirport рассмотрим пример использования вышеупомянтой модели. В листинге 4.1 изображён фрагмент кода XAML, демонстрирующий использование связывания для команды. В приложении Г можно ознакомиться с примером реализации XAML-кода для всей страницы.

<Button Style="{DynamicResource AddAirport}"

Command="{Binding AddAirport}" Panel.ZIndex="3"/>

Листинг 4.1 — XAML-код при связывании команд

Связывание команды AddAirport происходит при помощи Binding, которое передаётся в свойство Command. Вызов данной команды происходит при нажатии на кнопку добавления аэропорта.

В листинге 4.2 приводится пример модели для страницы AddAirport. В приложении Д реализация модели представления для всей страницы. Все классы, реализующие Model и ViewModel реализуют пользовательский интерфейс INotifyPropertyChanged, который позволяет использовать привязку данных и динамически обновлять данные в визуальном интерфейсе в случае их изменения.

namespace BelAirWays.Models

{

class AirportModel : INotifyPropertyChanged

{

private string \_nameAirport;

public string nameAirport { get { …; } set { …; } }

private string \_country;

public string country { get { …; } set { …; } }

private string \_town;

public string town { get { …; } set { …; } }

public void Add()

{

bool flag = true;

SqlCommand sqlCommand = new SqlCommand();

sqlCommand.CommandText = "select \* from airport";

sqlCommand.Connection =

DBConnection.DBConnection.SqlConnection;

SqlDataReader reader = sqlCommand.ExecuteReader();

foreach (var i in reader) { …; }

if (flag)

{

if (nameAirport != null && country != null && town != null )

{

if (nameAirport != "" && country != "" && town != "") { …; }

else

{

MessageBox.Show("Заполнены не все поля");

reader.Close();

}

}

else

{

MessageBox.Show("Заполнены не все поля");

reader.Close();

}

}

reader.Close();

}

public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged(string propertyName) { …; }

}

}

Листинг 4.2 — Модель для страницы AddAirport

Можем обратить внимание на реализацию метода OnPropertyChanged, который, как правило, реализовыван внутри свойств, что позволяет при передачи туда информации, уведомлять об этом систему, что позволит обновлять данные.

Реализация классом интерфейса предполагает генерацию события PropertyChanged каждый раз, когда значение свойства объекта изменяется. Такое поведение позволяет привязкам данных отслеживать состояние объекта и обновлять данные пользовательского интерфейса при изменении значения связанного свойства.

Пример реализации данной конструкции приведён в листинге 4.3.

namespace BelAirWays.ViewModel

{

class AirportVM : INotifyPropertyChanged

{

private AirportModel models;

public AirportModel Models

{

get { return models; }

set

{

models = value;

OnPropertyChanged("Models");

}

}

private RelayCommand addAirport;

public RelayCommand AddAirport

{

get

{

return addAirport ??

(addAirport = new RelayCommand(obj =>

{

models.Add();

}));

}

}

public AirportVM()

{

models = new AirportModel();

}

public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new

PropertyChangedEventArgs(propertyName));

}

}

}

Листинг 4.3 — Модель представления для страницы AddAirport

Основной командой, реализованной в проекте, является команда действия RelayCommand, конструктор которой принимает два параметра: первый – это делегат, который вызывается при выполнении команды (например, в ответ на нажатие кнопки). Второй параметр – это делегат, который возвращает логическое значение, указывающее, может ли команда выполниться; если параметр не передан, то команду можно выполнить всегда.

Реализация RelayCommand приведена в листинге 4.4.

namespace BelAirWays.Command

{

public class RelayCommand : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public RelayCommand(Action<object> execute, Func<object,

bool> canExecute = null)

{

this.execute = execute;

this.canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return this.canExecute == null ||

this.canExecute(parameter);

}

public void Execute(object parameter)

{

this.execute(parameter);

}

}

}

Листинг 4.4 — Реализация RelayCommand

Если же делегат возвращает значение «false», то он отключает кнопку.

4.3 Реализация взаимодействия с базой данных

Для обеспечения работоспособности программного средства была разработана база данных «BelAirWays», что являлось обязательным условием курсового проекта. База данных была тщательно сконструирована до начала написания самого приложения. Для ее создания использовалась реляционная система управления базами данных (РСУБД) MS SQL. Её структура была рассмотрена в разделе 3.3.

Для доступа к данным был разработан класс DBConnection, код которого представлен в листинге 4.5.

namespace BelAirWays.DBConnection

{

class DBConnection

{

private static DBConnection instance;

public static SqlConnection SqlConnection

{

get; private set;

}

private DBConnection()

{

SqlConnection = new SqlConnection(@"Data

Source=.\SQLEXPRESS;Initial

Catalog=BelAirWays;Integrated Security=True");

SqlConnection.Open();

}

public static DBConnection GetInstance()

{

if (instance == null)

instance = new DBConnection();

return instance;

}

public static void Close()

{

if (SqlConnection.State ==

System.Data.ConnectionState.Open)

SqlConnection.Close();

}

}

}

Листинг 4.5 — Реализация DBConnection

Как упоминалось ранее, для взаимодействия с базой данных была использована технология ADO.NET, так как она предоставляет самый прямой способ доступа к данным в .NET Framework.

4.4 Реализация методов регистрации и авторизации пользователей

При запуске приложения пользователь первым делом видит стартовое окно, в котором располагаются 2 кнопки: «Вход» и «Регистрация». Если у пользователя есть аккаунт, он нажимает на кнопку «Вход» и, в случае ввода корректных данных, он переходит к функционалу приложения, который отличается в зависимости от роли пользователя. Листинг модели представления для авторизации приводится в приложении Е.

При отсутствии аккаунта пользователь может зарегистрироваться. В приложении Ж приводится реализация модели представления для регистрации.

5. Тестирование, проверка работоспособности и анализ полученных результатов

Тестирование приложения — это проведение проверки и оценки соответствия между реальным и ожидаемым поведением программы.

Цели тестирования:

* убедиться, что ПО отвечает заявленным требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или несоответствующим требованиям.

Задачи тестирования:

* предотвратить как можно больше дефектов;
* проверить, что известные дефекты устранены;
* проверить, что при устранении известных дефектов, не было внесены новые дефекты.

Для обеспечения корректности работы программы обрабатываются различные ошибки, возникающие в процессе работы. Данное программное средство использует подключение к базе данных, следовательно, неправильно введенные данные или же их отсутствие может повлечь за собой неработоспособность приложения.

5.1 Тестирование регистрации и авторизации

Начнём тестирование с авторизации. Попробуем первым делом ввести неверные данные. В результате этих действий будет выведено соответствующее сообщение, которое демонстрируется на рисунке 5.1.

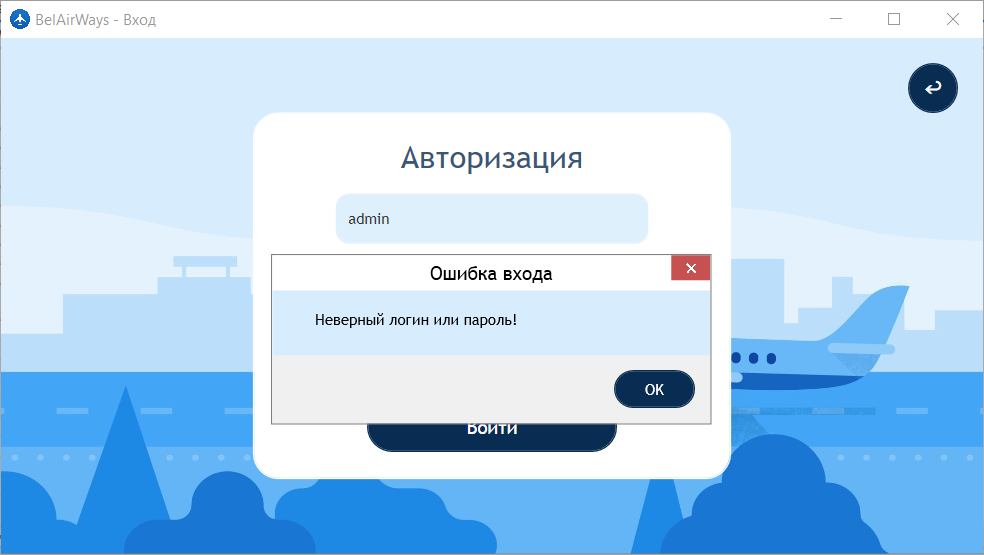


Рисунок 5.1 — Ошибка авторизации

Та же ошибка появится, если оставить все поля незаполненными.

Допустим, пользователь забыл свой пароль от учётной записи. Чтобы восстановить доступ к ней, он может ввести свой логин и нажать на кнопку «Забыл пароль», после чего ему придёт сообщение с новым паролем. Однако, если логин указан неверно, пользователь получит сообщение об ошибке, представленное на рисунке 5.2.

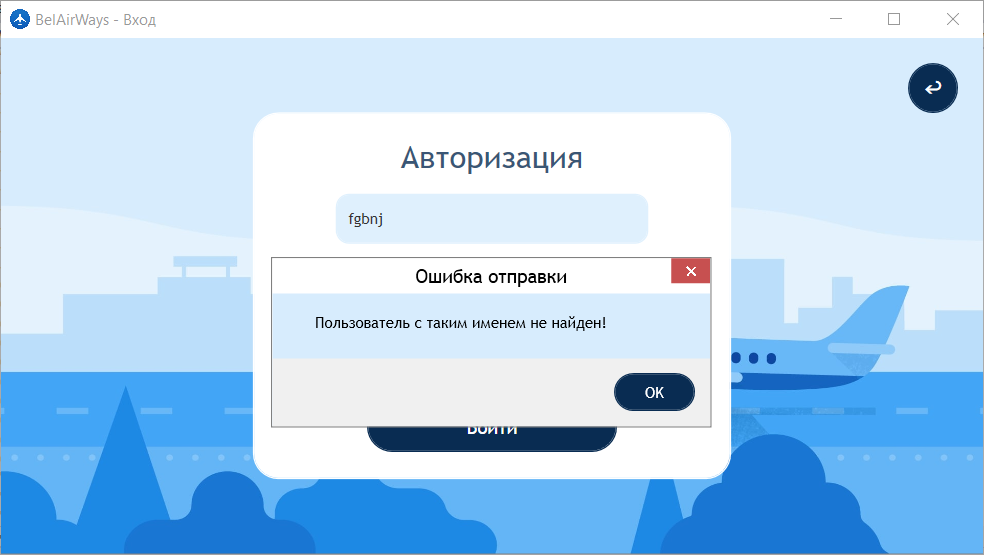


Рисунок 5.2 — Ошибка отправки сообщения (неверный логин)

Если же пользователь забыл ввести логин перед нажатием на кнопку «Забыл пароль», появится соответствующее сообщение, показанное на рисунке 5.3.

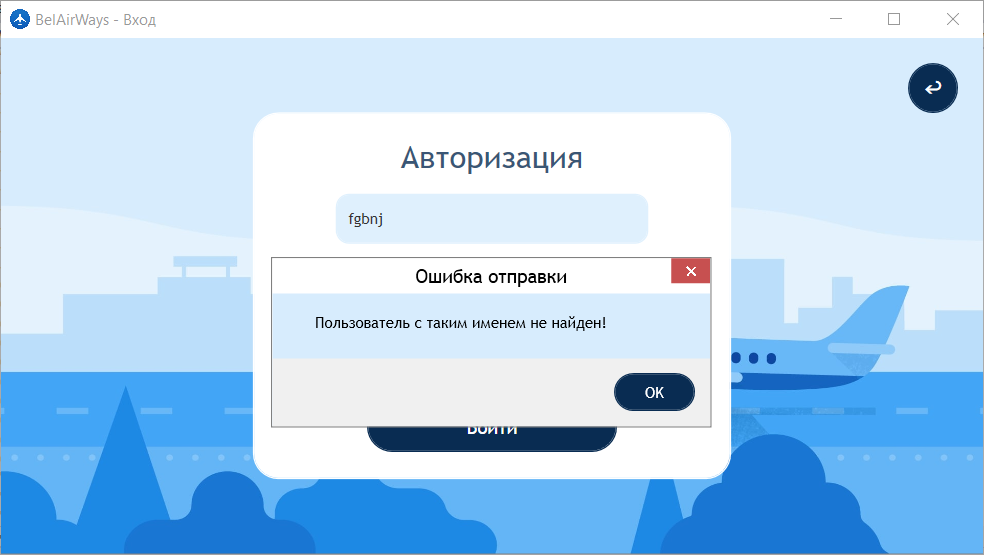


Рисунок 5.3 — Ошибка отправки сообщения (не введён логин)

Теперь начнём тестирование регистрации нового пользователя. Попробуем зарегистрироваться под логином пользователя, который уже существует в базе данных приложения. На рисунке 5.4 демонстрируется окно ошибки, в котором выводится соответствующее сообщение.

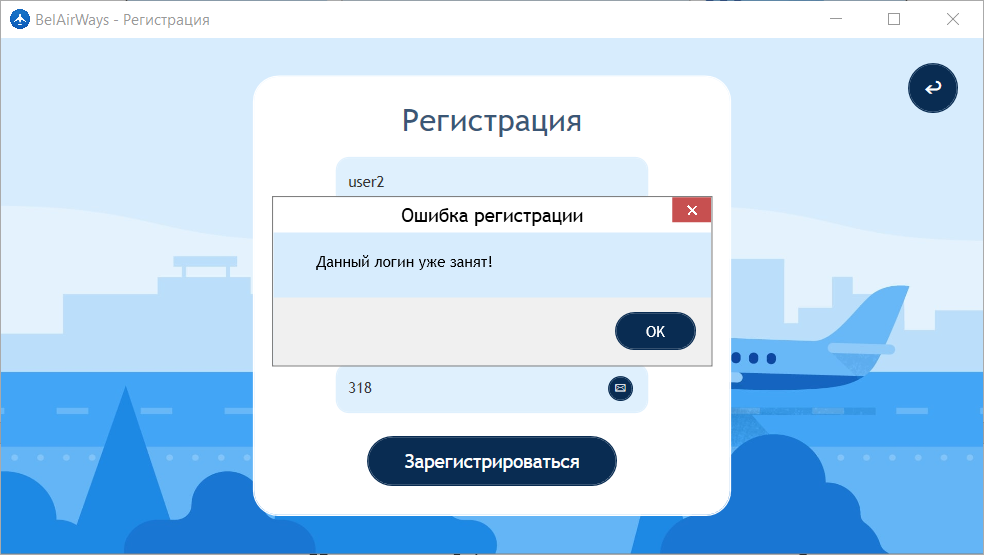


Рисунок 5.4 — Ошибка регистрации (логин занят другим пользователем)

Также пользователь не может использовать для регистрации почту, которая уже используется другим пользователем. Соответствующая ошибка представлена на рисунке 5.5.

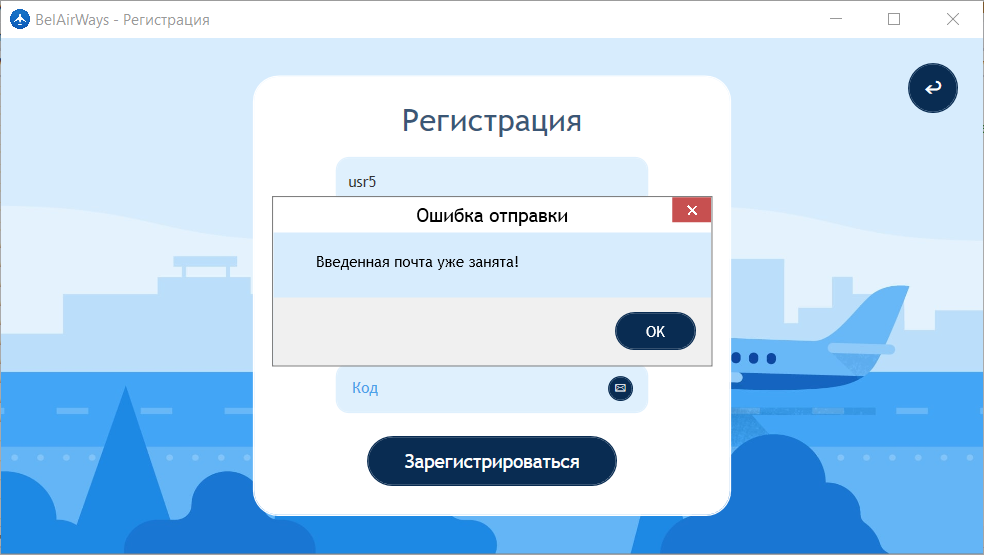


Рисунок 5.5 — Ошибка регистрации (почта занята другим пользователем)

Для регистрации необходимо ввести код подтверждения, отправленный на почту, которую указал пользователь. Если пользователь введёт неправильный код, он получит уведомление о том, что код введён неправильно. Данное сообщение об ошибке можно увидеть на рисунке 5.6.

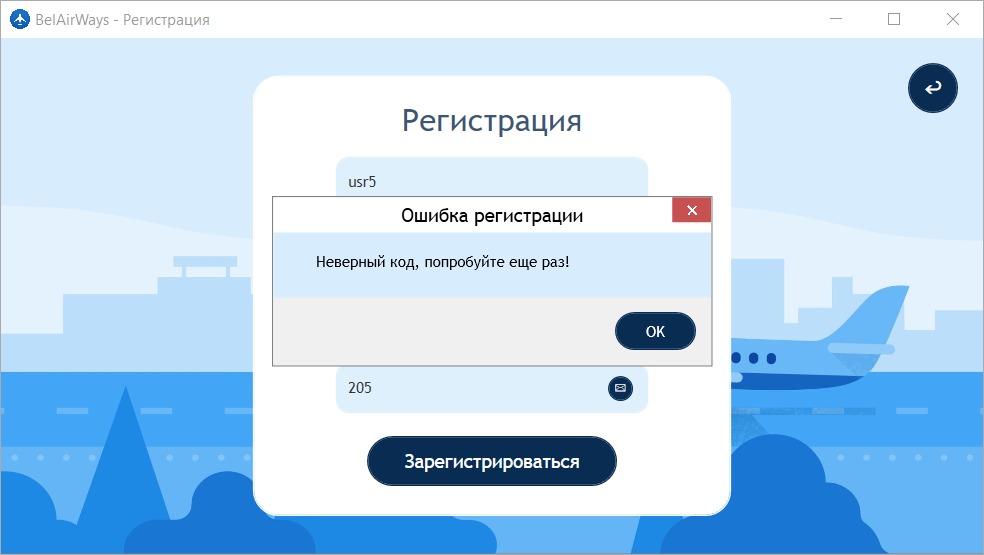


Рисунок 5.6 — Ошибка регистрации (неверный код)

Таким образом, можно сделать вывод, что окна для авторизации и регистрации обрабатывают все основные случаи возникновения ошибок.

5.2 Тестирование формы добавления аэропорта

Допустим, администратор хочет добавить новый аэропорт. Первым делом, попробуем добавить аэропорт, который уже существует. Результат тестирования представлен на рисунке 5.7.

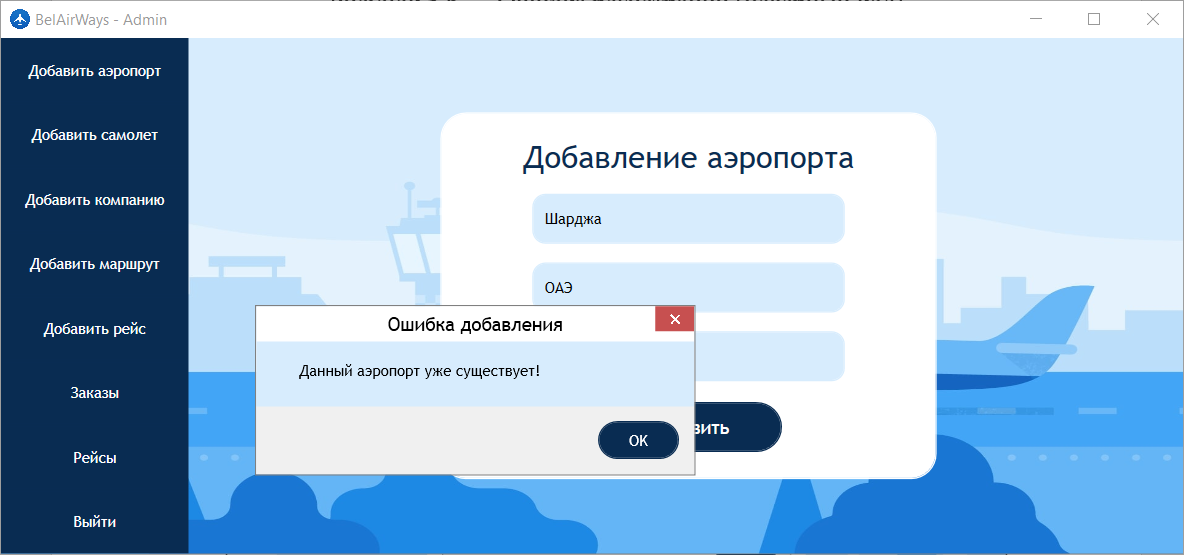


Рисунок 5.7 — Ошибка добавления (аэропорт уже существует)

Все поля формы добавления нового аэропорта являются обязательными для заполнения. Теперь попробуем добавить новый аэропорт, заполнив при этом не все поля формы. В результате тестирования мы получим сообщение, представленное на рисунке 5.8.

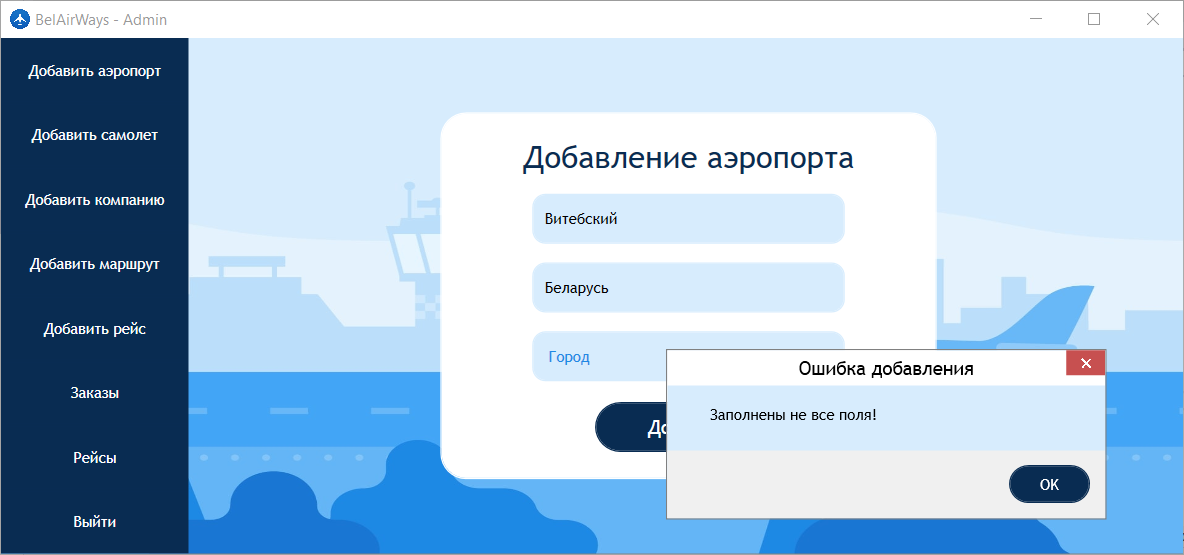


Рисунок 5.8 — Ошибка добавления (заполнены не все поля)

Также для всех полей формы была добавлена валидация, реализованная с помощью регулярных выражений. Например, если текст, введённый в поле «Название аэропорта», не соответствует шаблону, пользователь увидит сообщение об ошибке, представленное на рисунке 5.9.

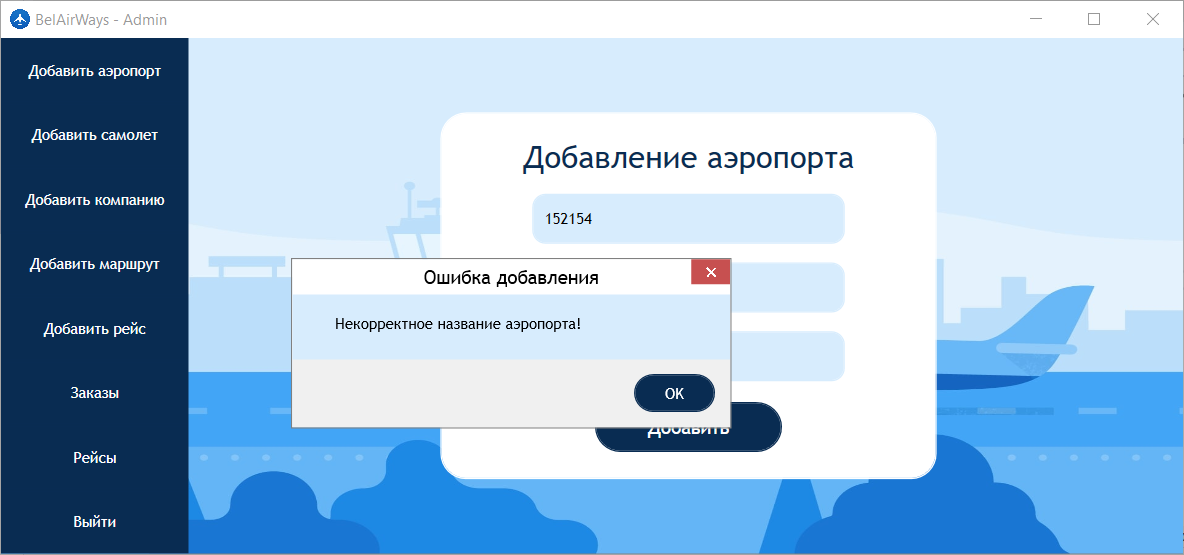


Рисунок 5.9 — Ошибка добавления (некорректное название аэропорта)

Аналогичная валидация с использованием регулярных выражений была применена к формам добавления нового самолёта и нового аэропорта.

5.3 Тестирование формы добавления маршрута

Теперь проведём тестирование добавления в базу данных нового маршрута. Первым делом, попробуем добавить маршрут, который уже существует. Результат тестирования представлен на рисунке 5.10.

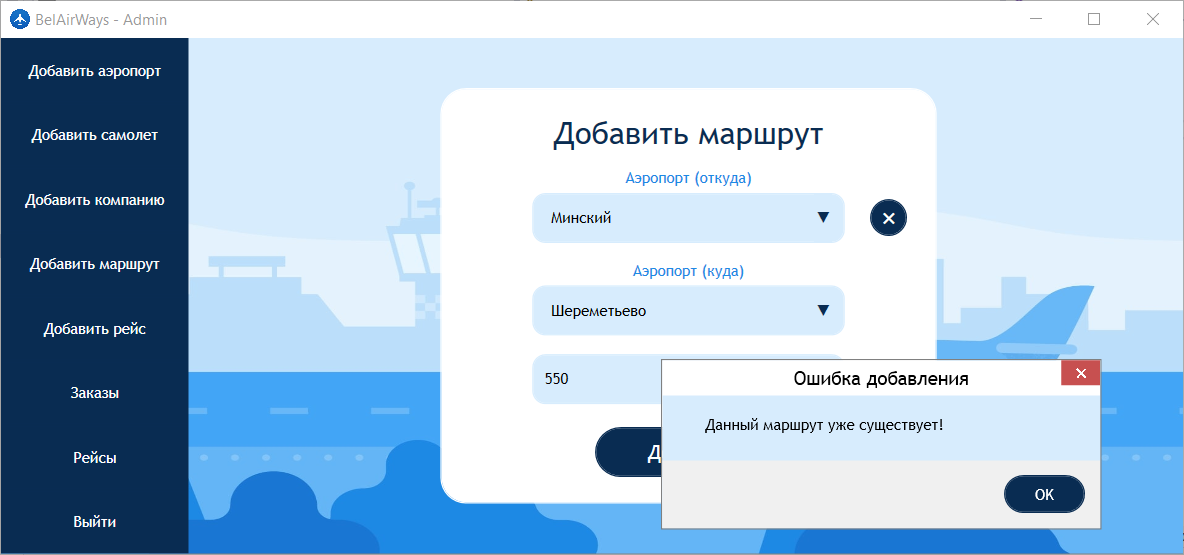


Рисунок 5.10 — Ошибка добавления (маршрут уже существует)

Попробуем добавить маршрут, пункты отправления и назначения которого совпадают. В результате тестирования мы получим сообщение, представленное на рисунке 5.11.

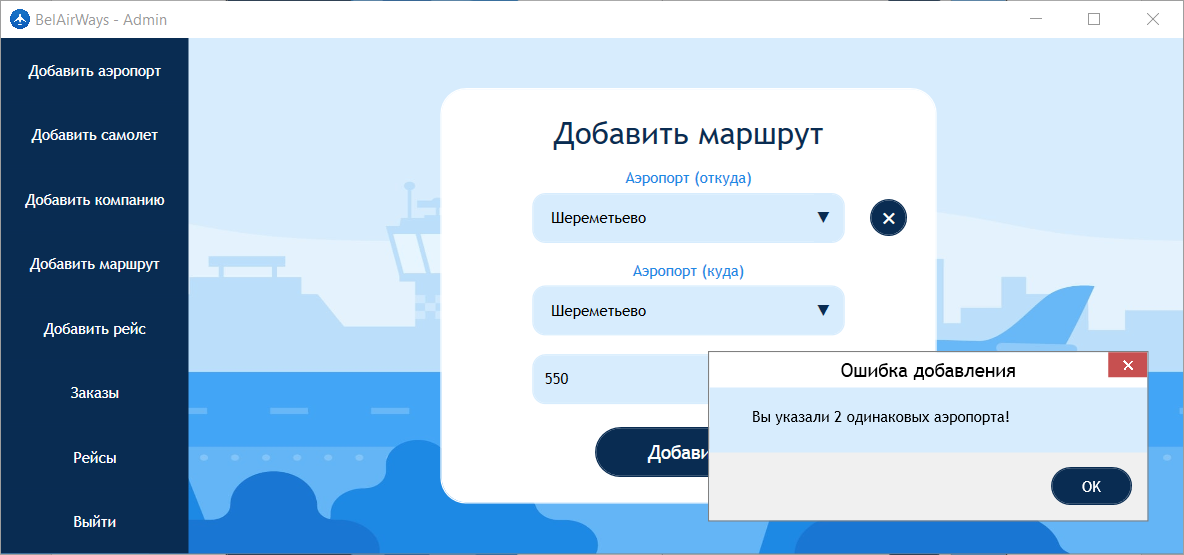


Рисунок 5.11 — Ошибка добавления (выбрано 2 одинаковых аэропорта)

Все поля формы добавления нового маршрута являются обязательными для заполнения. Теперь попробуем добавить новый маршрут, заполнив при этом не все поля формы. Например, не выберем значение в поле «Аэропорт (откуда)». В результате тестирования мы получим сообщение об ошибке, представленное на рисунке 5.12.

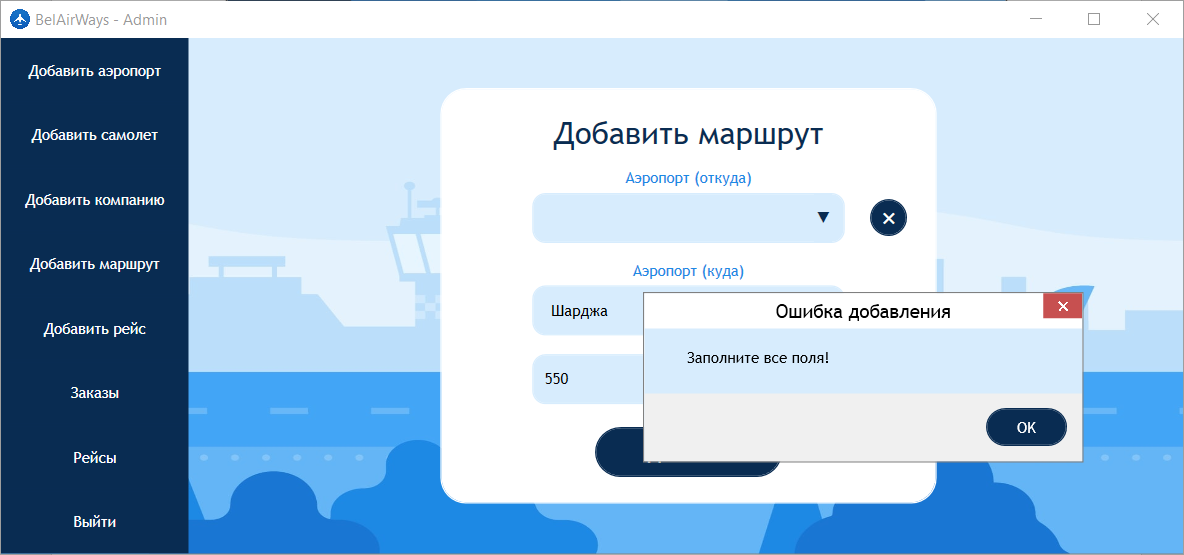


Рисунок 5.12 — Ошибка добавления (заполнены не все поля)

Также администратор не может добавить маршрут, длина которого меньше 100 километров. Если мы попытаемся добавить маршрут, длина которого короче заданного значения, то получим сообщение об ошибке добавления, представленное на рисунке 5.13.



Рисунок 5.13 — Ошибка добавления (слишком короткий маршрут)

Также для поля «Длина маршрута» была добавлена валидация, реализованная с помощью регулярных выражений. Значение, введённое в данное поле, должно соответствовать данным правилам: оно не должно начинаться с нуля или содержать какие-либо символы, не являющиеся числовыми. Если при тестировании мы нарушим хотя бы одно из этих правил, то получим сообщение об ошибке, представленное на рисунке 5.14.

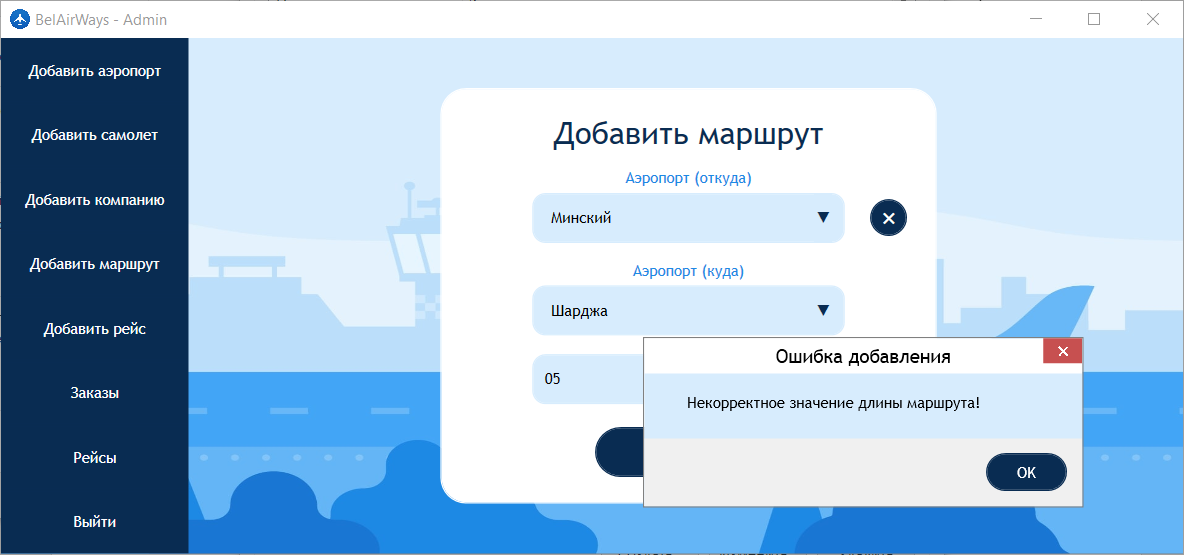


Рисунок 5.14 — Ошибка добавления (некорректное значение длины маршрута)

Также на этой странице мы можем произвести удаление аэропорта. Однако, если перед нажатием на кнопку удаления мы не выберем аэропорт, то увидим сообщение об ошибке, показанное на рисунке 5.15.

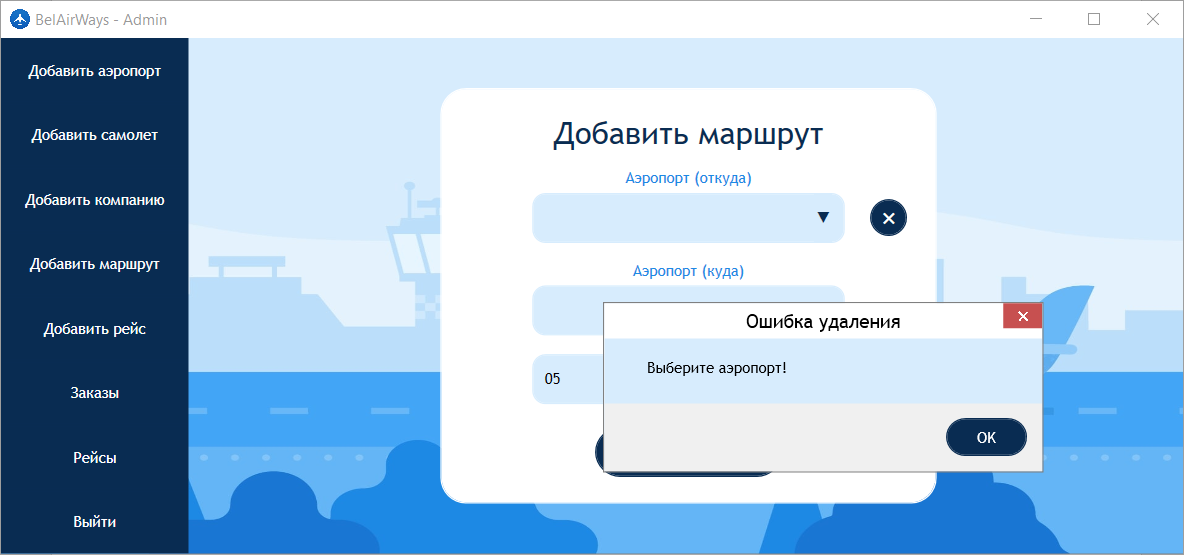


Рисунок 5.15 — Ошибка удаления (не выбран аэропорт)

Аналогичная ошибка появится, если на странице добавления авиарейса попытаться удалить компанию, маршрут или самолёт, не выбрав значение.

5.4 Тестирование формы добавления авиарейса

Теперь проведём тестирование формы добавления в базу данных нового авиарейса. Для начала попробуем добавить авиарейс, который уже существует. Для этого введём в форму данные для авиарейса, который уже был добавлен ранее. Результат тестирования в виде сообщения об ошибке представлен на рисунке 5.16.

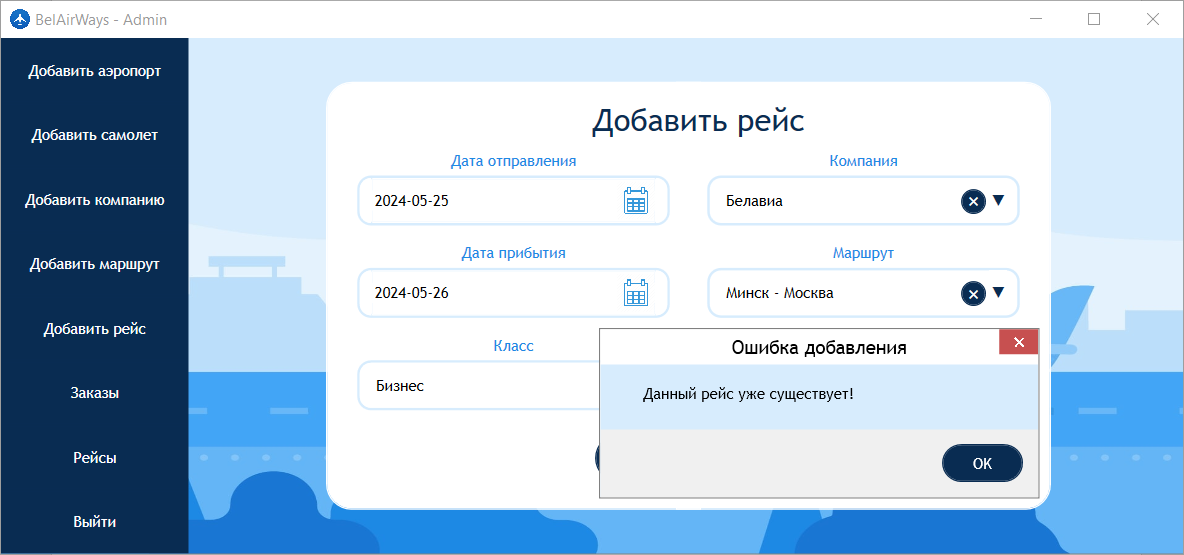


Рисунок 5.16 — Ошибка добавления (рейс уже существует)

Администратор не может добавить авиарейс, дата отправления которого уже прошла, если дата отправления является сегодняшней датой или если дата отправления больше, чем дата прибытия. Если мы попробуем добавить рейс с неправильными датами, то получим сообщение, показанное на рисунке 5.17.

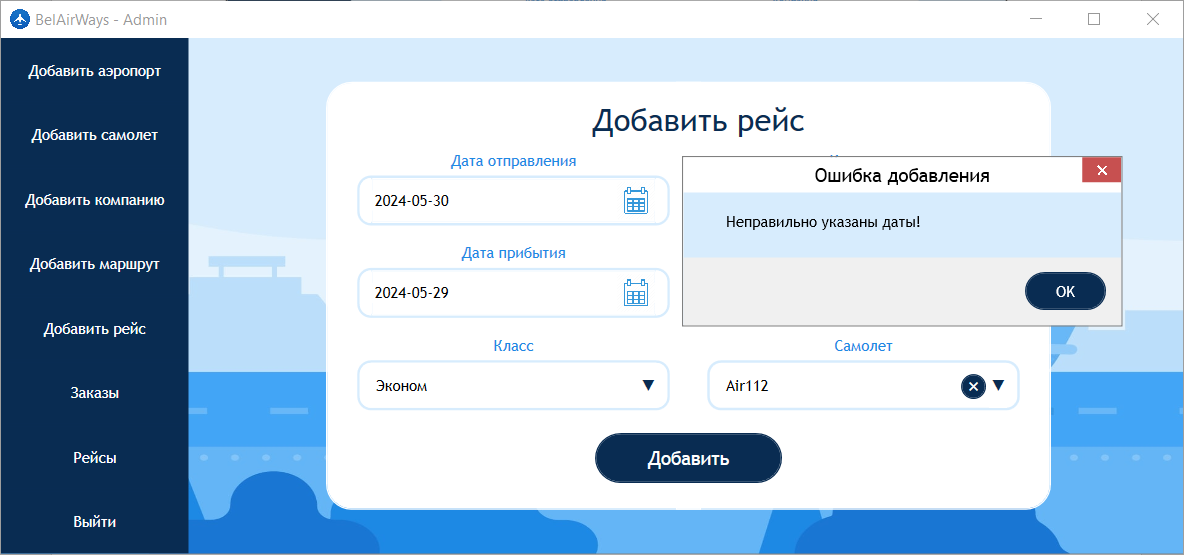


Рисунок 5.17 — Ошибка добавления (некорректные даты)

Все поля формы добавления нового авиарейса в базу данных приложения являются обязательными для заполнения. Теперь попробуем добавить новый авиарейс, заполнив при этом не все поля формы. Например, не выберем значения в полях «Компания», «Маршрут» и «Самолёт». В результате тестирования заполнения формы мы получим соответствующее сообщение об ошибке, представленное на рисунке 5.18.

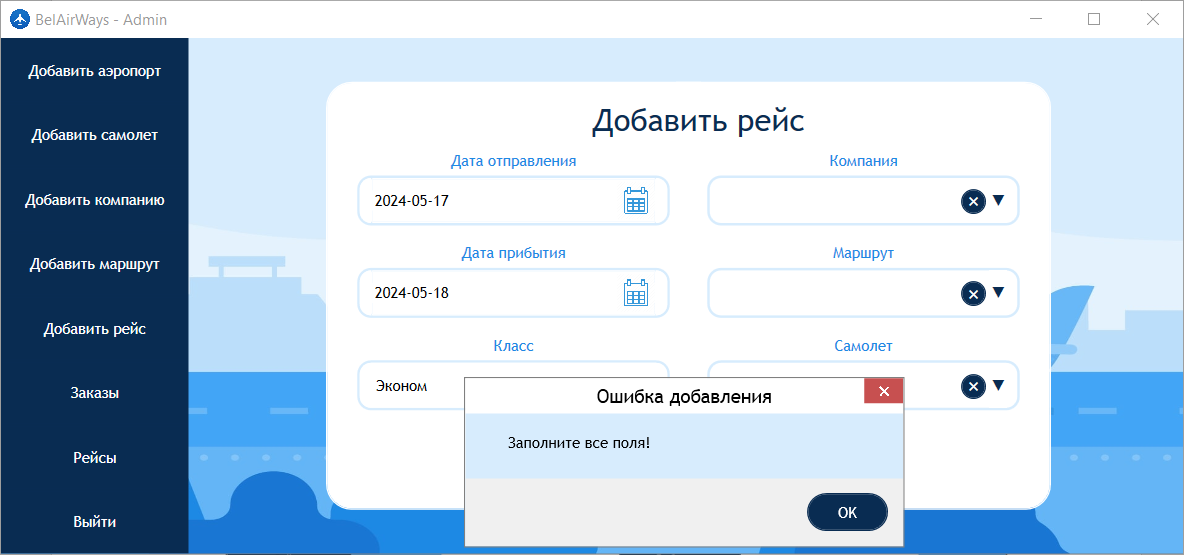


Рисунок 5.18 — Ошибка добавления (есть незаполненные поля)

Также на этой странице мы можем произвести удаление компании, маршрута или самолёта. Однако, например, если перед нажатием на кнопку удаления компании мы не выберем значение из списка, то увидим сообщение об ошибке, показанное на рисунке 5.19.

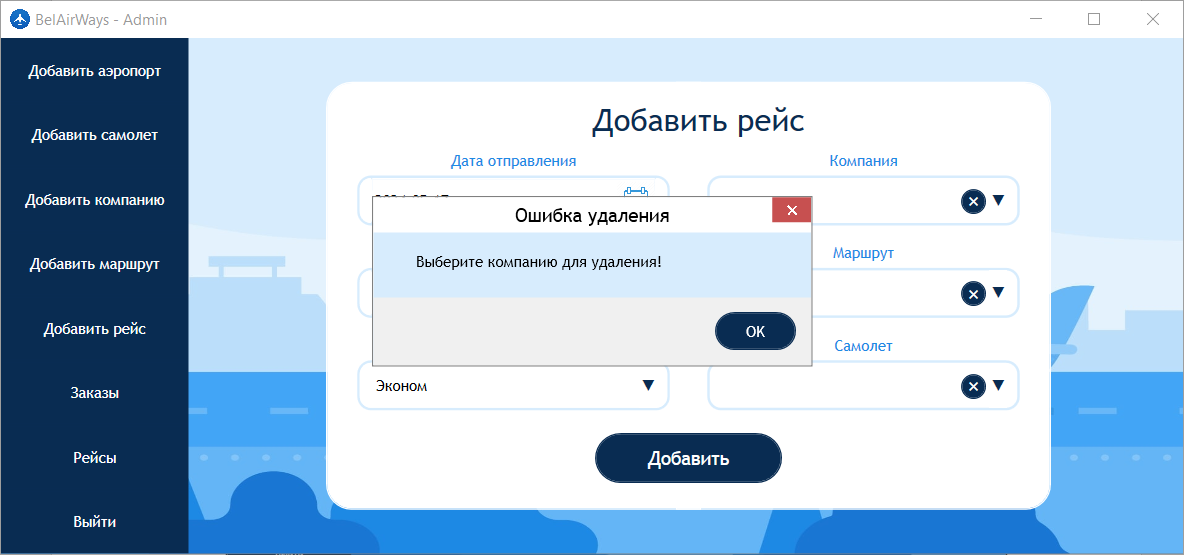


Рисунок 5.19 — Ошибка удаления (не выбрана компания)

Аналогичная ошибка появится, если при удалении маршрута или самолёта не выбрать значение из списка, или если на страницах с рейсами и заказами попытаться удалить рейс, не выбрав его.

5.5 Тестирование форм поиска и бронирования авиарейса

Проведём тестирование формы поиска авиарейса на стороне пользователя. Чтобы найти билет на интересующий рейс, нужно ввести дату отправления, а также можно выполнить поиск по городу назначения или по цене. Если пользователь введёт некорректное название города или некорректную цену, он получит соответствующее сообщение, представленное на рисунке 5.20.

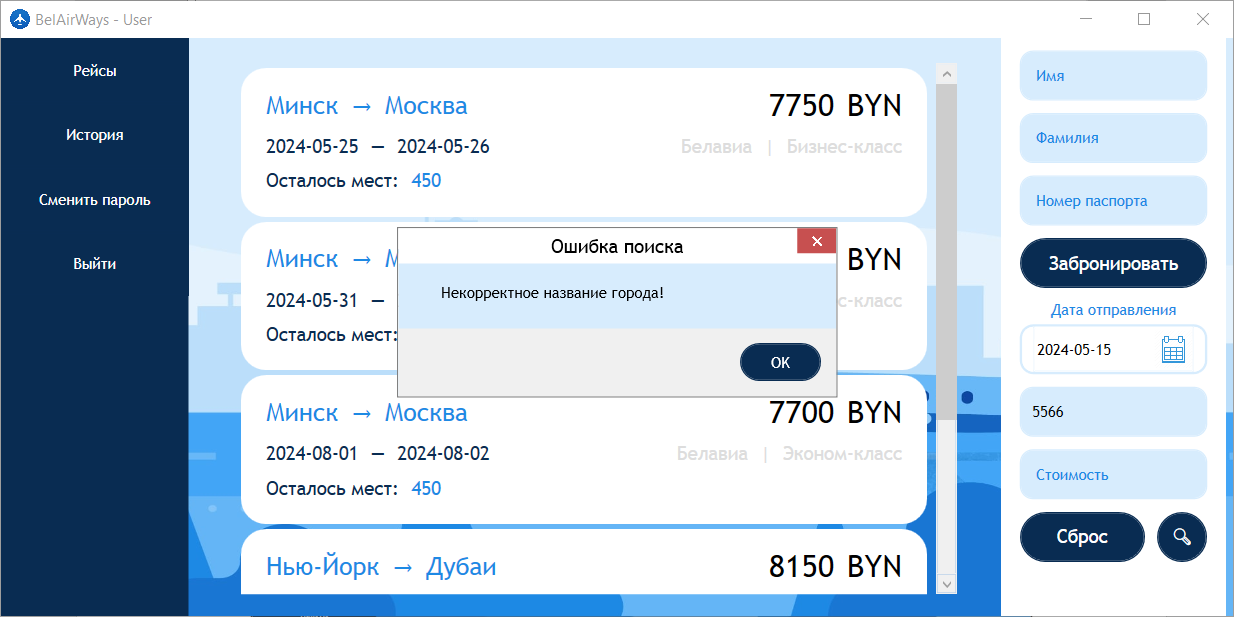


Рисунок 5.20 — Ошибка поиска (некорректное название города)

Данная валидация была осуществлена с помощью регулярных выражений и применена также к полю «Стоимость» формы бронирования, а также ко всем полям в форме бронирования рейса, которая будет рассмотрена ниже.

Протестируем бронирование рейса. Чтобы забронировать билет на рейс, нужно ввести свои имя, фамилию и номер паспорта. Если пропустить хотя бы одно поле, появится сообщение об ошибке, показанное на рисунке 5.21.

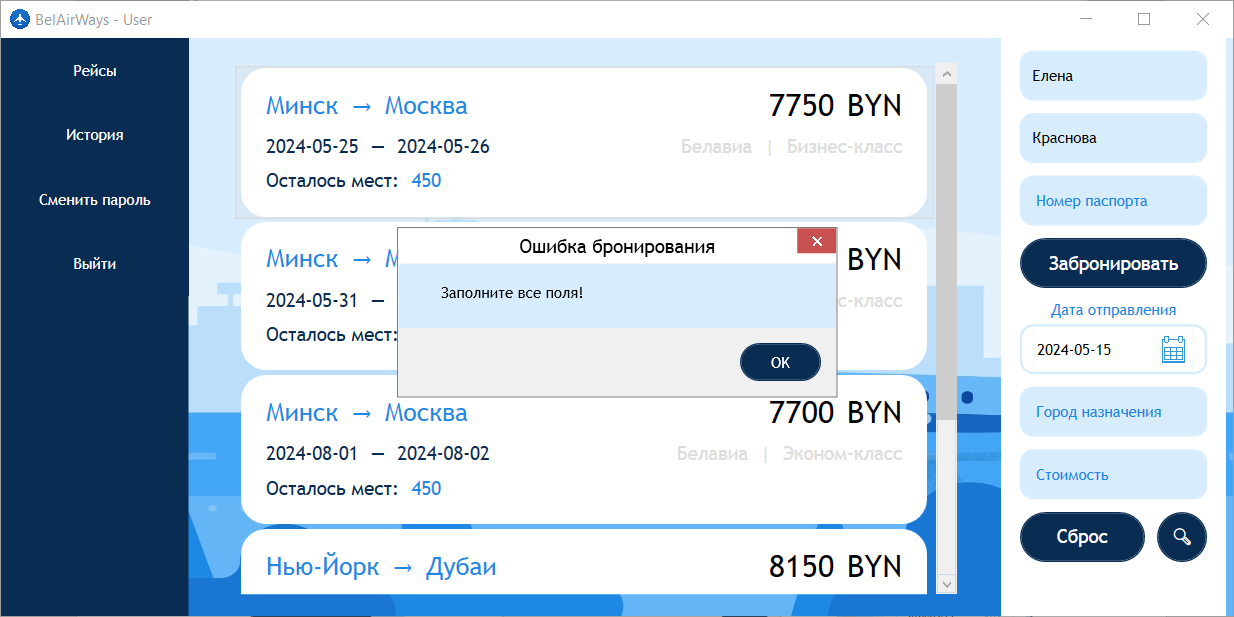


Рисунок 5.21 — Ошибка бронирования (есть незаполненные поля)

Если попробовать забронировать билет на авиарейс, места на который закончились, то появится сообщение об ошибке бронирования рейса с соответствующим содержанием. Всплывающее окно с данной ошибкой представлено на рисунке 5.22.

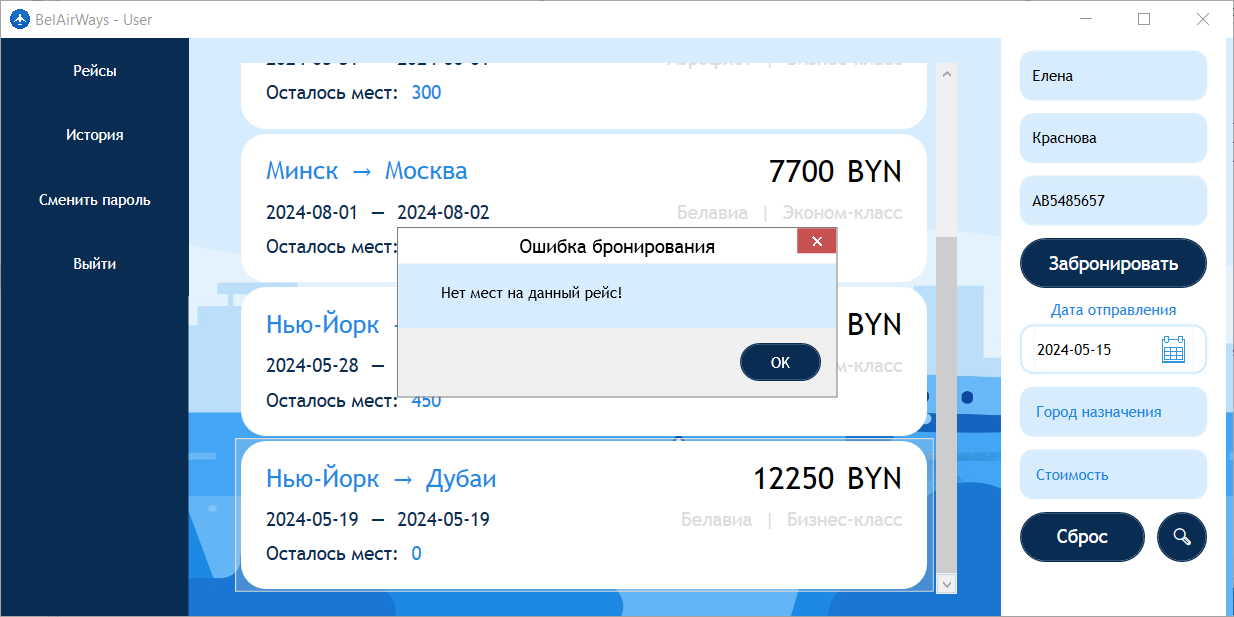


Рисунок 5.22 — Ошибка бронирования (нет мест на рейс)

Если пользователь уже забронировал место на рейс, то купить ещё одно место на то же имя он не сможет. В этом случае появится ошибка, представленная на рисунке 5.23.

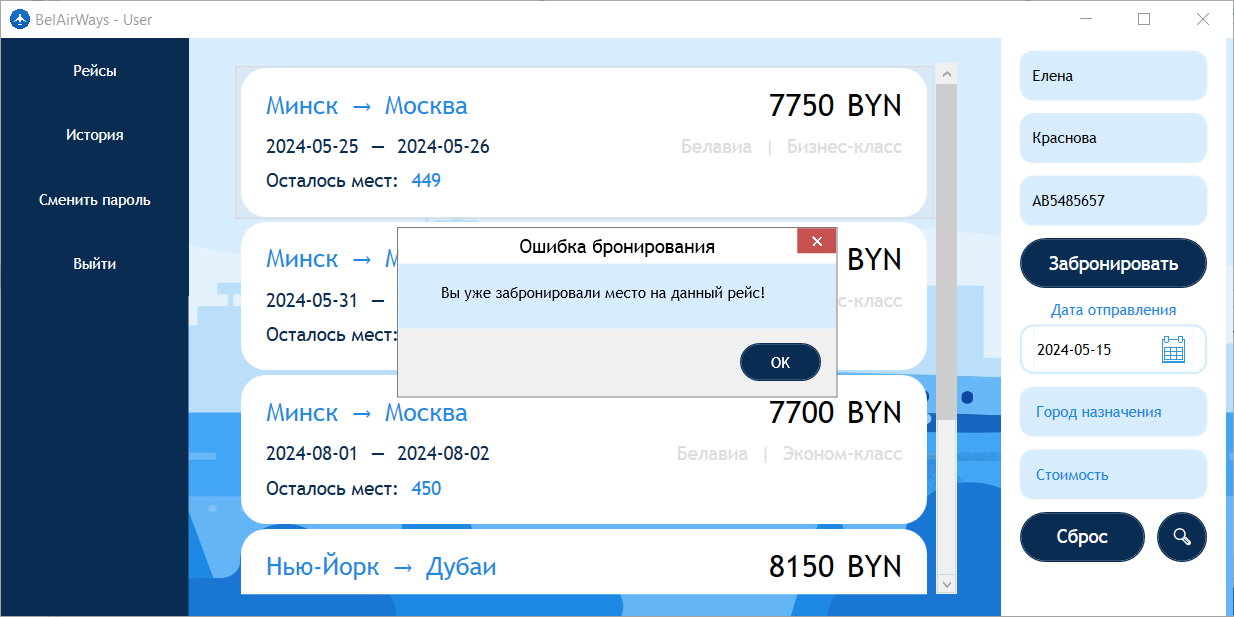


Рисунок 5.23 — Ошибка бронирования (место уже забронировано)

Также для этой формы была разработана валидация, аналогичная валидации для формы поиска авиарейса.

6 Руководство по установке и использованию

При запуске приложения мы видим стартовое окно (рисунок 6.1), которое предлагает нам либо авторизоваться, либо зарегистрироваться.

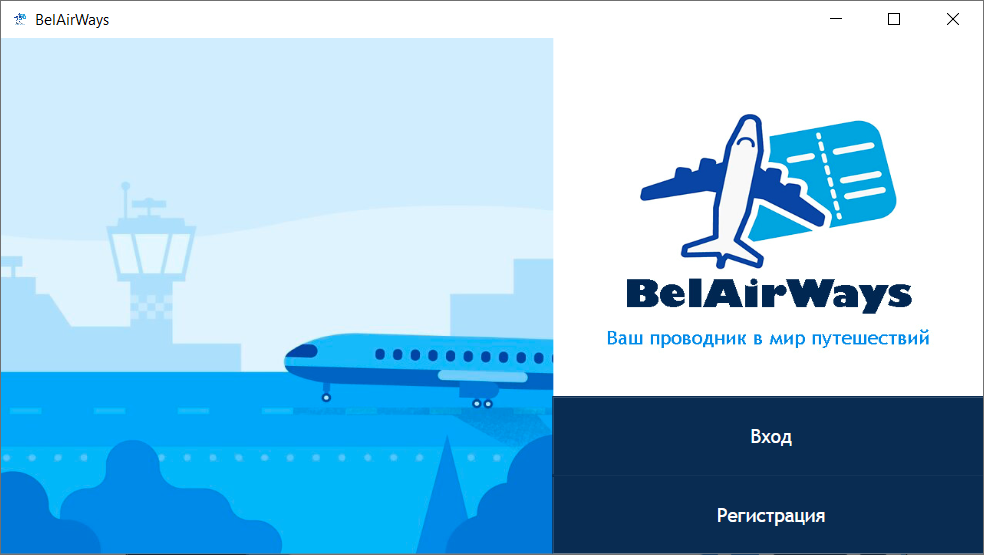


Рисунок 6.1 — Стартовое окно

Если у нас есть учётная запись, нужно нажать на кнопку «Вход», после чего появится окно авторизации, которое потребует ввода логина и пароля. Данное окно представлено на рисунке 6.2.

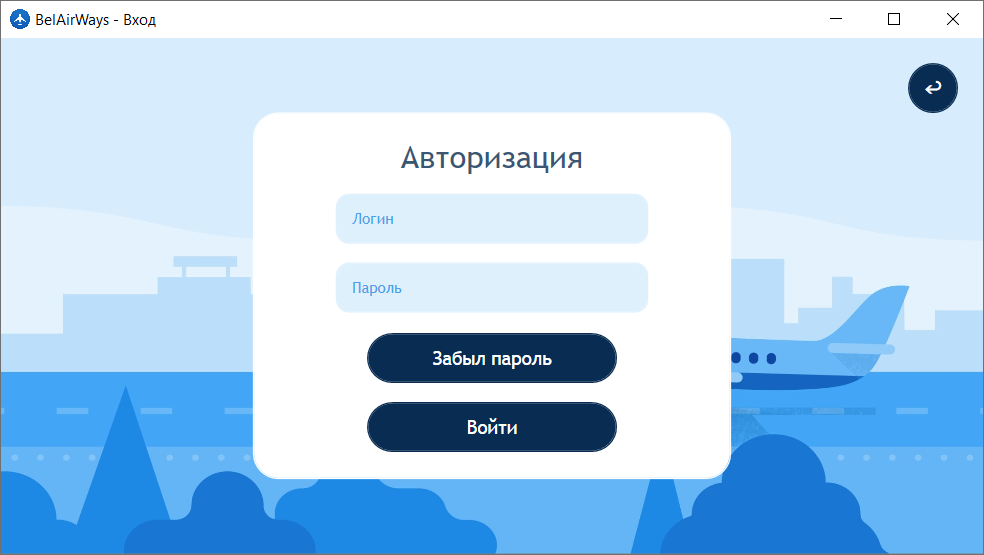


Рисунок 6.2 — Окно авторизации

Если пользователь забыл пароль, он может восстановить его, нажав на кнопку «Забыл пароль», после чего он увидит сообщение с новым паролем, который поможет пользователю восстановить доступ к учётной записи. Вид этого окна показан на рисунке 6.3.

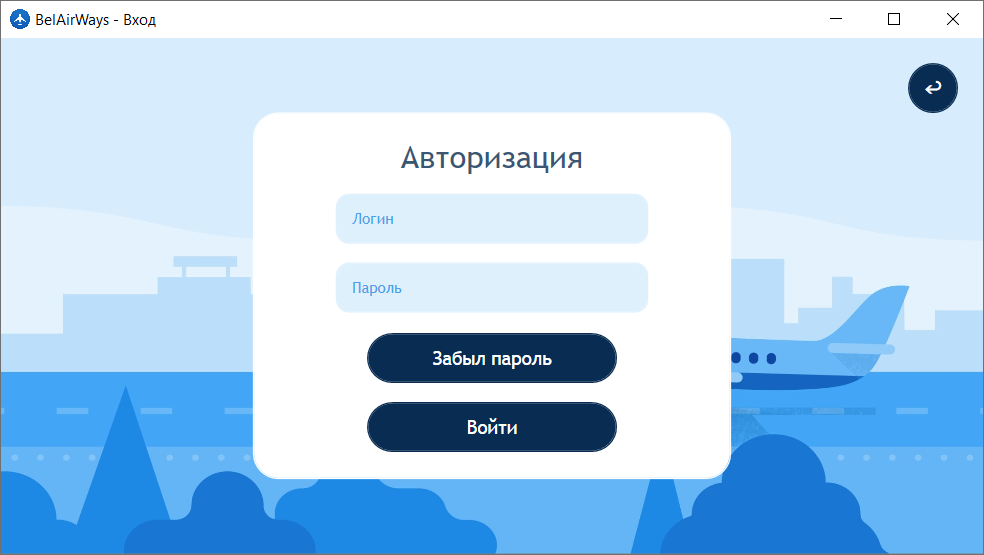


Рисунок 6.3 — Окно авторизации

В случае, если у нас нету своей учётной записи, её можно создать, нажав на кнопку «Регистрация» в стартовом окне. Мы будем направлены в окно регистрации, где нужно будет обязательно заполнить все поля. Форма регистрации представлена на рисунке 6.4.

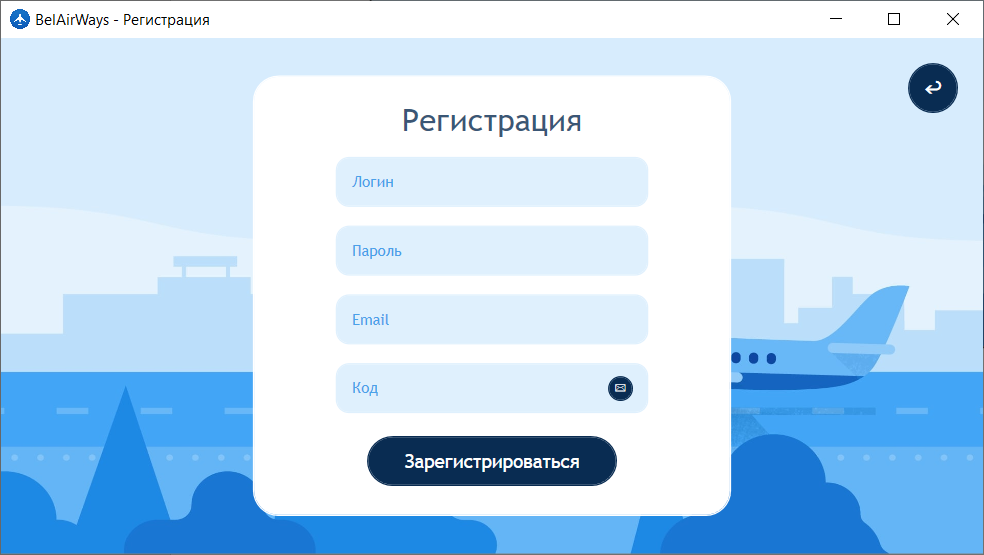


Рисунок 6.4 — Окно регистрации

Для регистрации нужно ввести код, который будет отправлен нам в сообщении после нажатия на кнопку отправления. Код регистрации представляет из себя трёхзначное число. Сообщение, содержащее код для регистрации, предоставлено на рисунке 6.5.

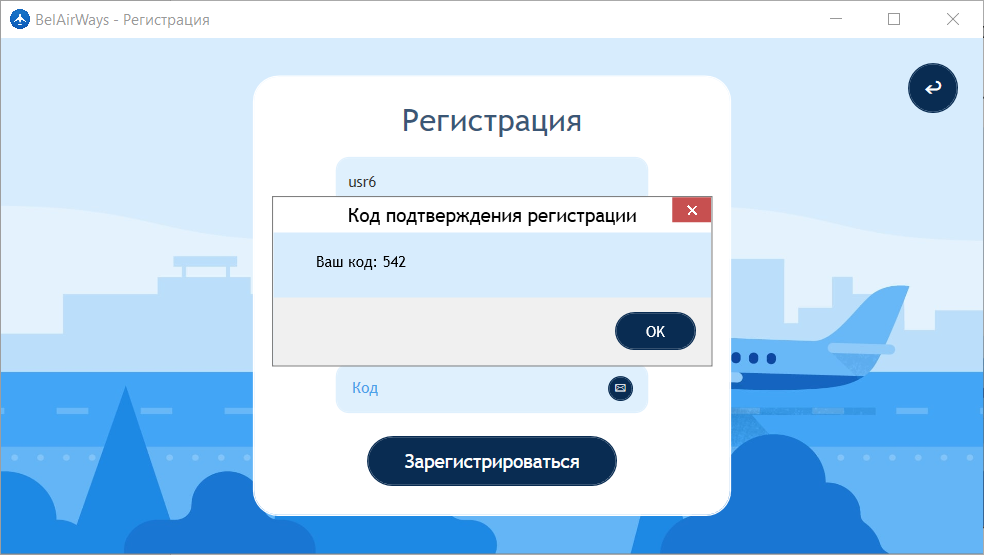


Рисунок 6.5 — Сообщение с кодом

После успешной регистрации нам нужно будет вернутся в окно авторизации и ввести данные, указанные при регистрации.

Если мы войдём в приложение под аккаунтом обычного пользователя, то увидим страницу со всеми доступными авиарейсами. Данная страница представлена на рисунке 6.6.

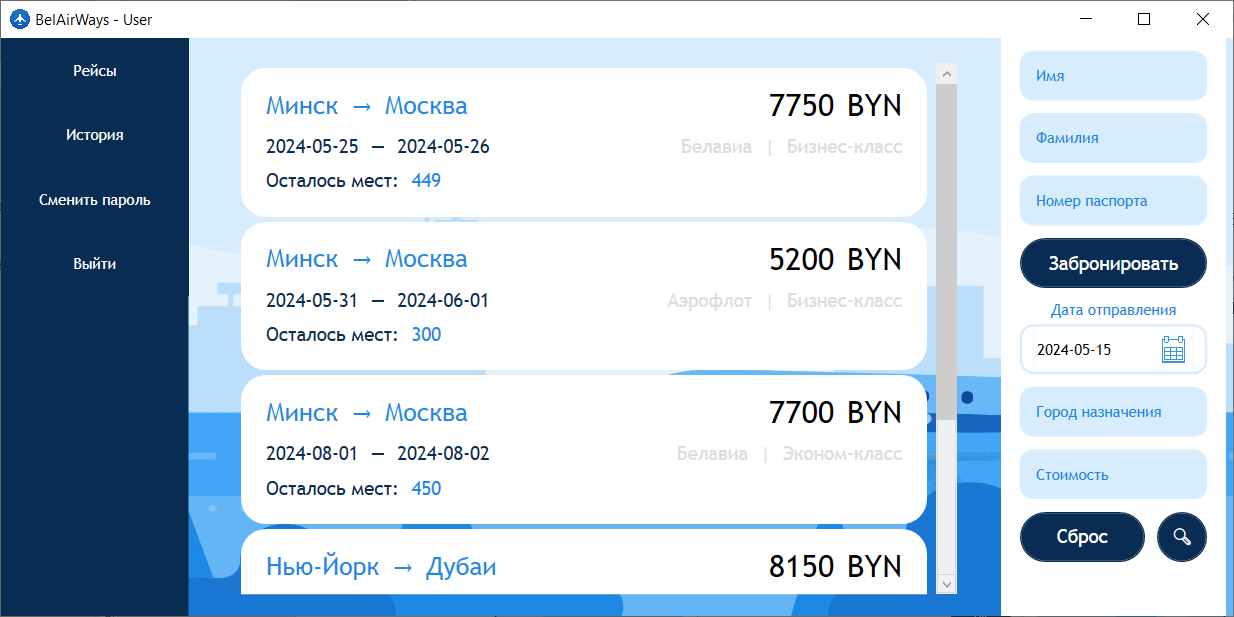


Рисунок 6.6 — Страница «Рейсы»

На этой странице мы можем произвести поиск билета на интересующий авиарейс. Для этого нужно указать дату отправления рейса, а также можно указать город назначения и стоимость полёта, после чего нажать на кнопку поиска. При необходимости все фильтры можно сбросить с помощью соответствующей кнопки. Пример применения фильтрации авиабилетов представлен на рисунке 6.7.

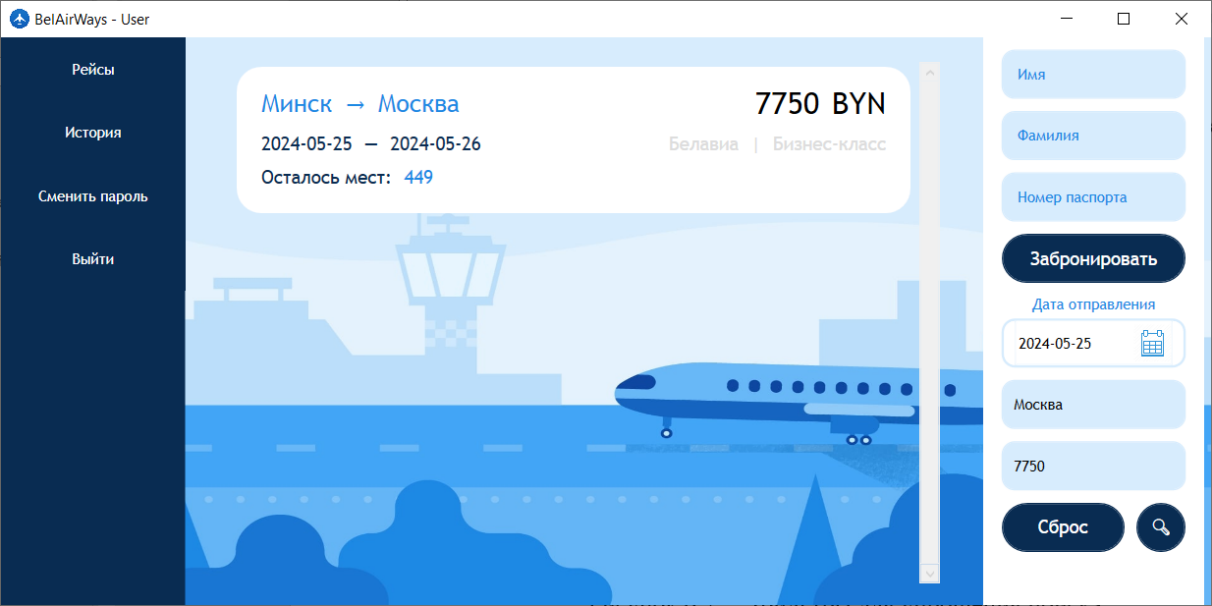


Рисунок 6.7 — Фильтры для упрощения поиска

На этой же странице находится форма для бронирования билета на авиарейс. В ней нужно указать свои имя и фамилию, а также номер паспорта. При успешном бронировании билета мы увидим сообщение, показанное на рисунке 6.8.

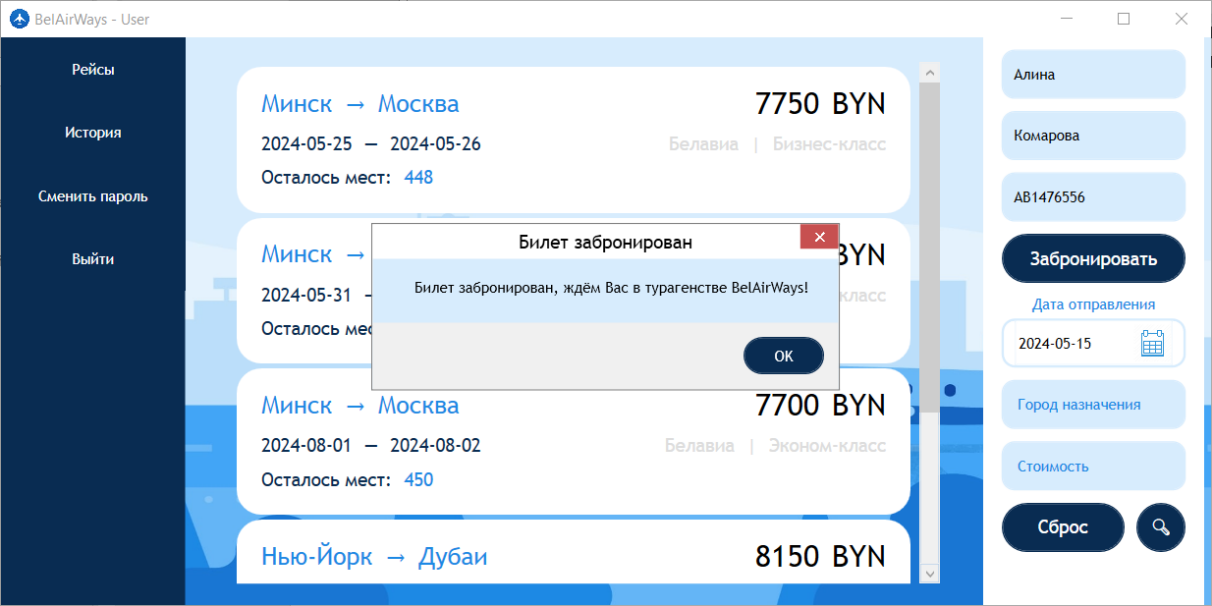


Рисунок 6.8 — Успешное бронирование билета

После того, как мы забронировали билет, мы можем увидеть его в списке заказов на странице «Заказы». Для этого нужно перейти на данную страницу и нажать на кнопку «Обновить», после чего в конце списка отобразиться наш новый заказ. Вид страницы «Заказы» представлен на рисунке 6.9.

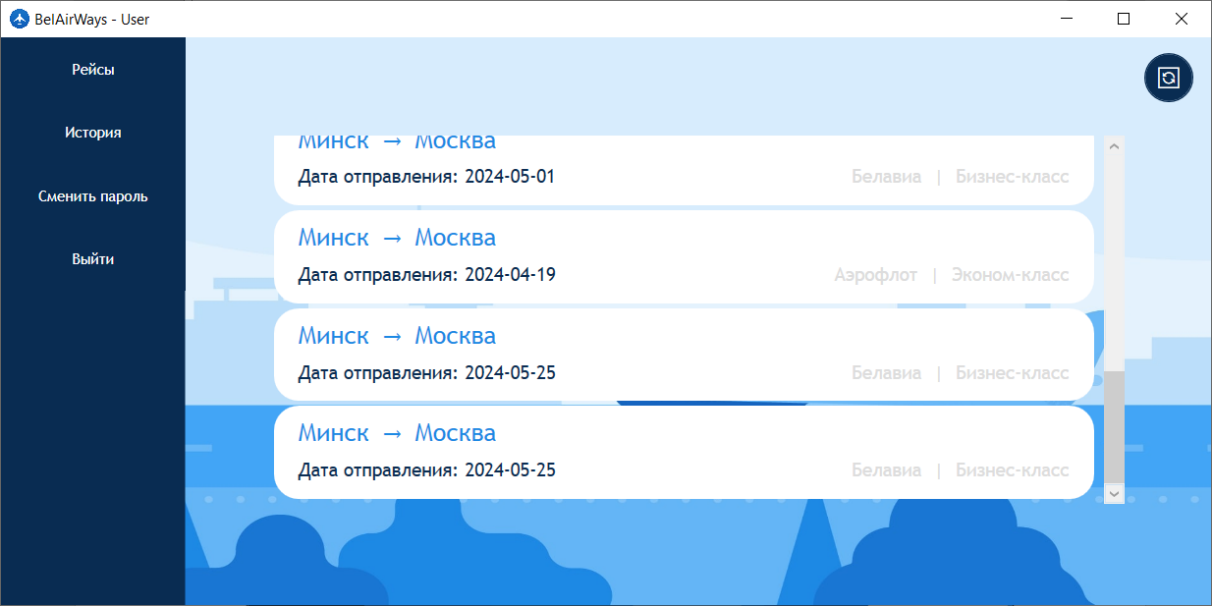


Рисунок 6.9 — Новый заказ в списке

Если пользователь восстанавливал пароль и новый пароль показался ему слишком сложным для запоминания, он может изменить его на соответствующей странице, вид которой представлен на рисунке 6.10.

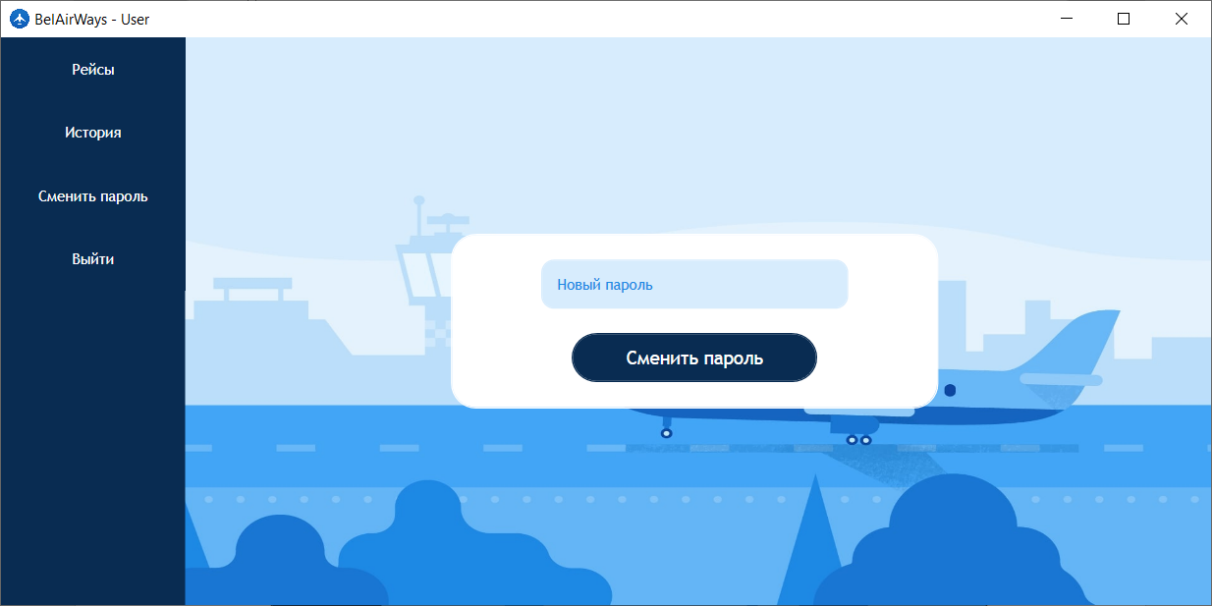


Рисунок 6.10 — Страница «Сменить пароль»

Можно было заметить, что при разработке пользовательского интерфейса, особый акцент был направлен на проработку интуитивно понятного интерфейса для любой аудитории в любом возрасте.

# ЗАКЛЮЧЕНИЕ

Данное курсовое проектирование было выполнено с целью разработки программного средства, которое позволяет пользователям искать и бронировать билеты на авиарейсы, а также администратору управлять базой данных и обеспечивать техническую поддержку.

В процессе выполнения проекта были достигнуты следующие результаты:

* разработано интуитивно понятное пользовательское приложение с удобным интерфейсом для просмотра и выбора эскизов татуировок;
* реализована функциональность записи пользователей на сеансы татуировок и обработка данных в базе данных;
* разработана административная часть для управления расписанием мастеров и предоставления технической поддержки клиентам;
* произведено тестирование программного средства для проверки его функциональности и корректности работы.

Для реализации приложения был выбран язык программирования C# и технология Windows Presentation Foundation (WPF), обеспечивающая разработку удобного и современного пользовательского интерфейса. В качестве базы данных была использована MS SQL Server.

В процессе разработки программного средства «BelAirWays» был применен паттерн проектирования MVVM (Model-View-ViewModel). MVVM является одним из наиболее популярных паттернов для разработки пользовательских интерфейсов в технологии WPF.

В процессе разработки программного средства «BelAirWays» ViewModel была активно использована для связи пользовательского интерфейса с бизнес-логикой и данными. Благодаря MVVM, разработка стала более структурированной и управляемой, а код стал более поддерживаемым и расширяемым.

Таким образом, применение паттерна MVVM в программном средстве «BelAirWays» обеспечило эффективное управление данными и логикой приложения, улучшенную архитектуру и обеспечило высокую отзывчивость и удобство использования пользовательского интерфейса.

В результате выполнения данного курсового проекта было создано функциональное и удобное программное средство «BelAirWays». Программа имеет потенциал для дальнейшего развития и расширения функциональности в соответствии с потребностями пользователей.

# Список использованных источников

1. Microsoft Visual Studio [Электронный ресурс] – https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio – Дата доступа 13.03.2024
2. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. Режим доступа: https://metanit.com/sharp/tutorial/ – Дата доступа: 13.03.2024
3. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
4. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 15.03.2024
5. Руководство по XAML // [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/xaml/index.htm– Дата доступа: 15.04.2024
6. Блинова, Е.А. Курс лекций по Базам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

# ПРИЛОЖЕНИЕ А

Диаграмма вариантов использования



# ПРИЛОЖЕНИЕ Б

Диаграмма классов для окон (Windows)

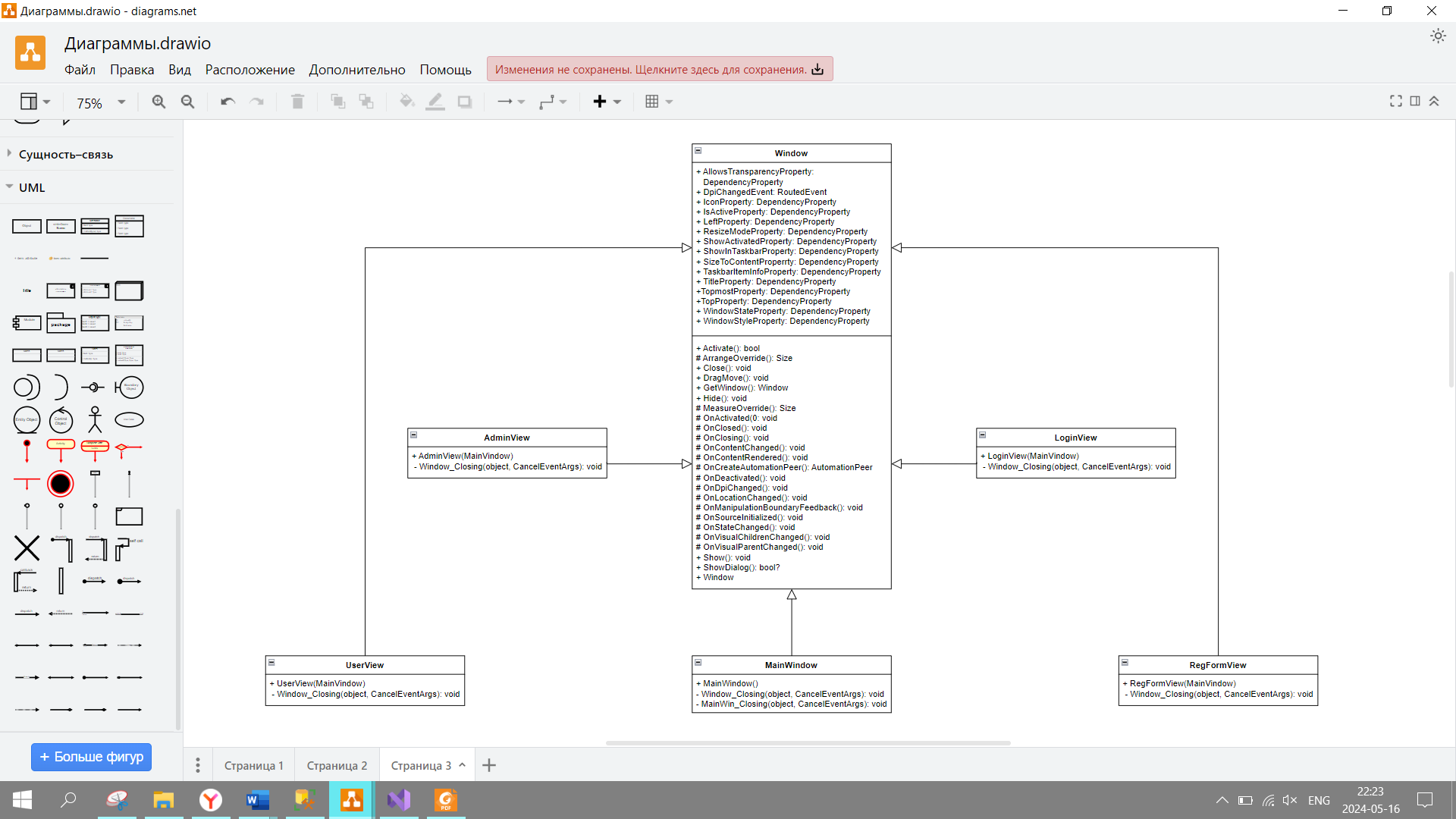


Диаграмма классов для страниц (Pages)

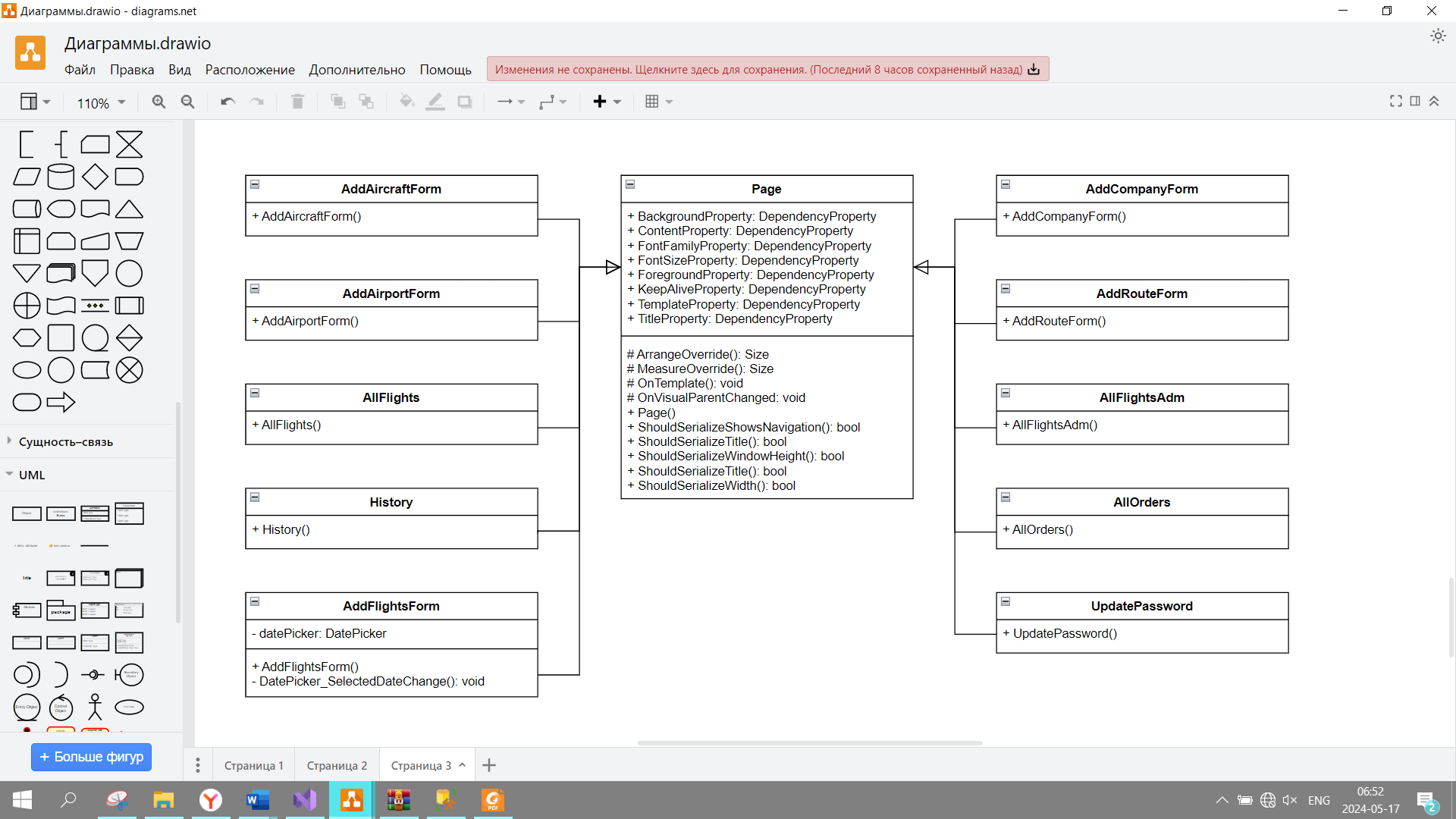
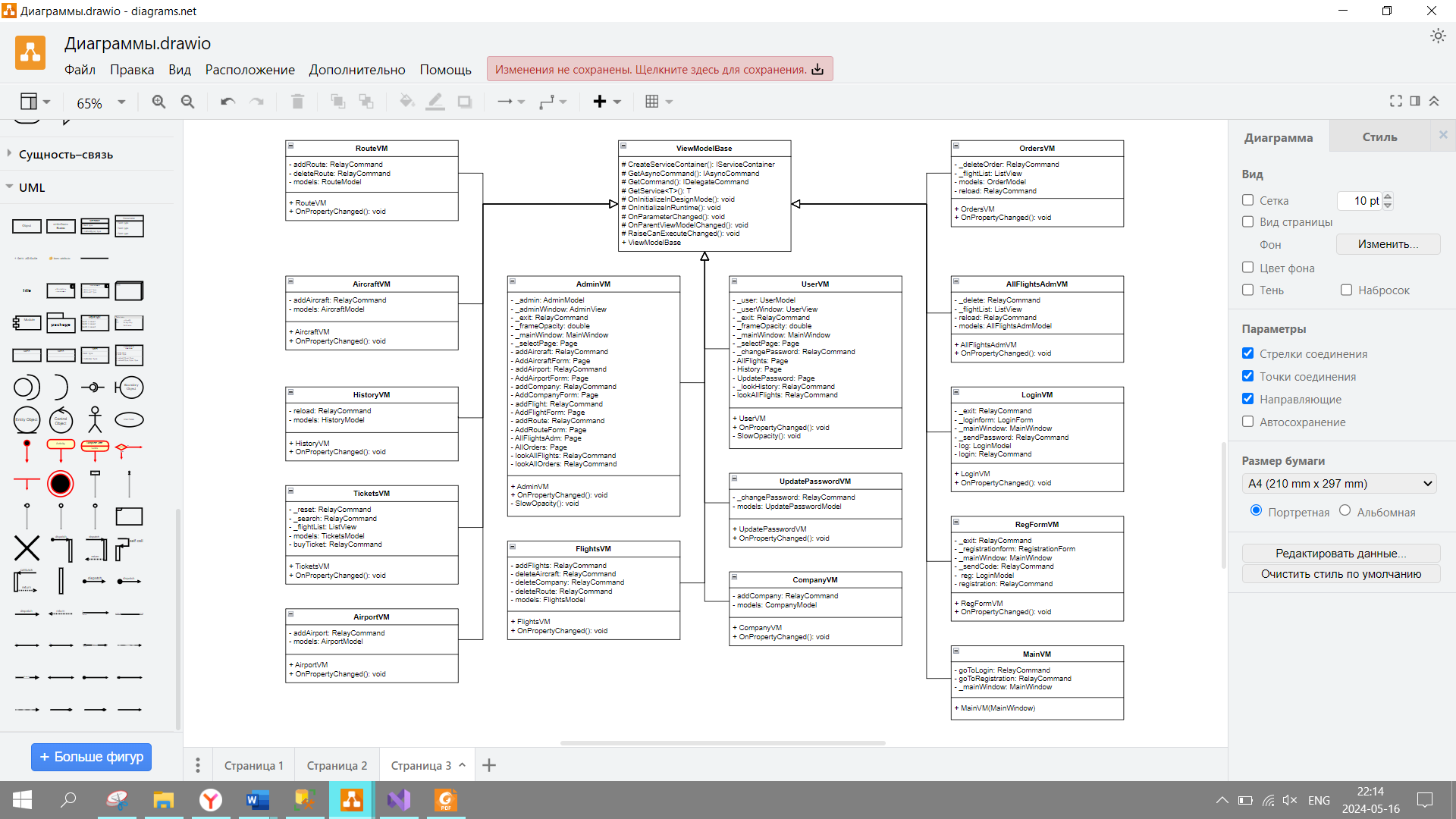
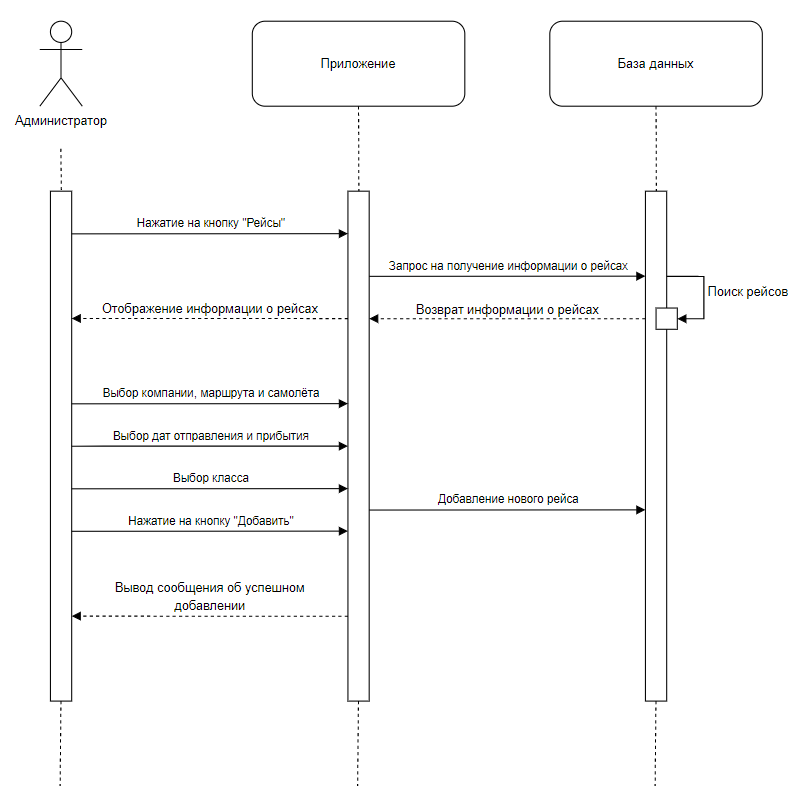


Диаграмма ViewModel



# ПРИЛОЖЕНИЕ В

Диаграмма последовательности



# ПРИЛОЖЕНИЕ Г

Листинг файла «AddAirport.xaml»

<Page x:Class="BelAirWays.Pages.AddAirportForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-

compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:BelAirWays.Pages"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800"

Title="AddAiraportForm">

<Grid Margin="201.5 59.5">

<Border CornerRadius="20" BorderBrush="White"

BorderThickness="1" Panel.ZIndex="2"

Background="White">

<StackPanel Style="{DynamicResource Panel}"

DataContext="{Binding Models}" Panel.ZIndex="1"

Background="{x:Null}">

<TextBlock Style="{DynamicResource AddAirportWord}"

Width="300">Добавление аэропорта</TextBlock>

<TextBox Style="{DynamicResource NameAirport}"

Text="{Binding nameAirport,

UpdateSourceTrigger=PropertyChanged}"/>

<TextBox Style="{DynamicResource Country}"

Text="{Binding country,

UpdateSourceTrigger=PropertyChanged}" />

<TextBox Style="{DynamicResource Town2}"

Text="{Binding town,

UpdateSourceTrigger=PropertyChanged}"/>

</StackPanel>

</Border>

<Button Style="{DynamicResource AddAirport}"

Command="{Binding AddAirport}" Panel.ZIndex="3"/>

</Grid>

</Page>

# ПРИЛОЖЕНИЕ Д

Листинг файла «AirportVM.cs»

namespace BelAirWays.ViewModel

{

class AirportVM : INotifyPropertyChanged

{

private AirportModel models;

public AirportModel Models

{

get { return models; }

set

{

models = value;

OnPropertyChanged("Models");

}

}

private RelayCommand addAirport;

public RelayCommand AddAirport

{

get

{

return addAirport ?? (addAirport = new

RelayCommand(obj =>

{

models.Add();

}));

}

}

public AirportVM()

{

models = new AirportModel();

}

public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new

PropertyChangedEventArgs(propertyName));

}

}

}

</Page>

# ПРИЛОЖЕНИЕ Е

Листинг файла «LoginVM.cs»

namespace BelAirWays.ViewModel

{

class LoginVM : INotifyPropertyChanged

{

private MainWindow \_mainWindow;

private LoginForm \_loginform;

private LoginModel log;

public LoginModel Log

{

get { return log; }

set

{

log = value;

OnPropertyChanged("Log");

}

}

private RelayCommand \_exit;

public RelayCommand Exit

{

get

{

return \_exit ?? (\_exit = new RelayCommand (obj =>

{

log.Exit(\_mainWindow, \_loginform);

}));

}

}

private RelayCommand \_sendPassword;

public RelayCommand SendPassword

{

get

{

return \_sendPassword ?? (\_sendPassword = new

RelayCommand (obj =>

{

log.SendNewPassword();

}));

}

}

private RelayCommand login;

public RelayCommand Login

{

get

{

return login ?? (login = new RelayCommand (obj =>

{

var passwordBox = obj as PasswordBox;

string passwordEnter =

log.GetHashString(passwordBox.Password);

if (log.Check(passwordEnter) == 0)

{

UserView userView = new

UserView(\_mainWindow);

\_loginform.Hide();

userView.ShowDialog();

}

else if (log.Check(passwordEnter) == 1)

{

AdminView adminView = new

AdminView(\_mainWindow);

\_loginform.Hide();

adminView.ShowDialog();

}

else

{

Xceed.Wpf.Toolkit.MessageBox mbox = new

Xceed.Wpf.Toolkit.MessageBox();

FontFamily fontFam = new

FontFamily("Trebuchet MS");

string hexColor1 = "#d7ecfd";

Color color1 =

(Color)ColorConverter.ConvertFromString(hexColor1);

SolidColorBrush backgroundBrush = new

SolidColorBrush(color1);

mbox.OkButtonStyle = (Style)Application.Current.Resources["OkButtonStyle"];

mbox.Background = backgroundBrush;

mbox.Caption = "Ошибка входа";

mbox.Text = "Неверный логин или пароль!";

mbox.FontFamily = fontFam;

mbox.ShowDialog();

}

}));

}

}

public LoginVM( LoginForm loginForm, MainWindow mainWindow)

{

\_loginform = loginForm;

\_mainWindow = mainWindow;

log = new LoginModel();

}

public LoginVM() { }

public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new

PropertyChangedEventArgs(propertyName));

}

}

}

# ПРИЛОЖЕНИЕ Ж

Листинг файла «RegFormVM.cs»

namespace BelAirWays.ViewModel

{

class RegFormVM : INotifyPropertyChanged

{

private RegistrationForm \_registration\_form;

private MainWindow \_main\_window;

private RegFormModel reg;

public RegFormModel Reg

{

get { return reg; }

set

{

reg = value;

OnPropertyChanged("Reg");

}

}

private RelayCommand \_registration;

public RelayCommand Registration

{

get

{

return \_registration ??

(\_registration = new RelayCommand(obj =>

{

var passwordBox = obj as PasswordBox;

string passwordEnter =

reg.GetHashString(passwordBox.Password);

reg.Add(passwordEnter,

\_registration\_form,\_main\_window);

}));

}

}

private RelayCommand \_sendCode;

public RelayCommand SendCode

{

get

{

return \_sendCode ??

(\_sendCode = new RelayCommand(obj =>

{

reg.SendCode();

}));

}

}

private RelayCommand \_exit;

public RelayCommand Exit

{

get

{

return \_exit ??

(\_exit = new RelayCommand(obj =>

{

reg.Exit( \_main\_window, \_registration\_form);

}));

}

}

public RegFormVM(RegistrationForm regform, MainWindow

mainWindow)

{

\_registration\_form = regform;

\_main\_window = mainWindow;

reg = new RegFormModel();

}

public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new

PropertyChangedEventArgs(propertyName));

}

}

}