МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Профиль Проектирование, разработка и эксплуатация информационных систем

БАКАЛАВРСКАЯ РАБОТА

на тему

Автоматизированная информационная система продажи авиабилетов

Студент _____ Цапин Антон Андреевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Самуйлов С.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Попова Н.А.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2016

Содержание

Реферат

Пояснительная записка содержит 47 листов, 20 рисунков, 15 использованных источников, 17 таблиц и 3 приложения.

АВИАБИЛЕТЫ, РЕЙСЫ, МАРШРУТЫ, ПАССАЖИРЫ, ПОЛЬЗОВАТЕЛИ, КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ, БАЗА ДАННЫХ, ПРЕДМЕТНАЯ ОБЛАСТЬ, IDEFIX, MSSQL, UML, C#, ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ.

Объект исследования: программные средства для продаж авиабилетов.

Цель работы: разработать автоматизированную информационную систему продажи авиабилетов.

Технология разработки – Visual Studio 2015, MS SQL Server 2012, C#

Результаты работы: спроектирована и разработана автоматизированная информационная система продажи авиабилетов.

Введение

В настоящее время информационные системы занимают значимое место в жизни людей. Самые первые из них были созданы еще в 50-х годах прошлого столетия и осуществляли в основном арифметические расчеты, незначительно сокращая издержки производства и затраты времени. Развитие информационных систем не стояло на месте, продвигаясь в ногу со временем и деловыми потребностями человека. К банальным возможностям расчета зарплат добавились возможности анализировать информацию, упрощая процесс принятия решений для управленческого персонала. Также, с каждым годом степень автоматизации систем увеличивалась, позволяя все сильнее наращивать производственные показатели предприятий их использующих.

В современных условиях, человек вынужден работать с гигантскими объемами информации. В связи с этим разработка программных продуктов, служащих для автоматизированного учета, весьма актуальна. Системы обязаны представлять собой мощные средства, способные обрабатывать гигантские потоки данных высокой структурной сложности за минимум затраченного времени, обеспечивая дружественный диалог с пользователем

Целью данной выпускной квалификационной работы является создание автоматизированной информационной системы, осуществляющей продажи авиабилетов.

Разработка подобной системы весьма актуальна на данный момент. В современном мире самолеты являются не только самым быстрым видом транспорта, но и самым безопасным, в связи с этим авиаперелеты пользуются весьма высокой популярностью. Вследствие этого продаваемые на рейсы билеты востребованы и с высокой вероятностью найдут своего покупателя, при условии, что авиакомпания обеспечила клиенту полноценный доступ к нужной ему информации. Это и есть задача, решаемая современными автоматизированными информационными системами. Существует множество подобных разработок, позволяющих авиакомпаниям реализовывать авиабилеты, а пользователям приобретать их. Однако, зачастую, функциональность таких систем либо весьма

ограничена, либо предоставляет достаточное количество информации, жертвуя дружественностью к пользователю.

Для реализации поставленной задачи были выбраны следующие средства разработки: MS Visual Studio 2015, MS SQL Server 2012 и язык программирования C#.

1. Анализ предметной области и постановка задачи

1.1 Продажа авиабилетов

Целью данной выпускной квалификационной работы является создание программного средства, осуществляющего поиск и продажу авиабилетов.

Авиакомпания занимается авиаперевозками пассажиров. Также она устанавливает маршруты полетов. Рейсы осуществляются по установленным маршрутам согласно расписанию. На каждый рейс существует определенное количество билетов. Продажа билета пользователю осуществляется при отправке запроса на бронирование, при условии, что данный билет до сих пор есть в наличии. Приобретя билет, пользователь предоставляет информацию о себе и становится пассажиром. Совершеннолетние пассажиры обязаны иметь занесенные в БД паспортные данные. Несовершеннолетние обязаны иметь занесенные в БД данные из свидетельства о рождении. Администраторы системы могут ограничивать или расширять доступ пользователей и сотрудников к предоставляемой информации.

Система создаётся для обслуживания следующих групп пользователей:

- пользователи, приобретающие билеты на рейсы и осуществляющие их поиск;
- администраторы, осуществляющие контроль за пользователями и функциональностью системы;

Абсолютно каждая авиакомпания использует определенную систему дистрибуции. Наиболее развитые используют GDS (глобальные дистрибьюторские системы, которые формируются из основных международных компьютерных систем резервирования). В итоге сервисы продаж авиабилетов при поиске информации пользуются ресурсами глобальных дистрибьюторских систем. Однако доступ к GDS является не бесплатным, поэтому в роли дистрибутивной системы для разрабатываемого продукта будет выступать БД, созданная в MS SQL Server 2012[1].

В БД должна храниться информация:

- о маршрутах;

- о рейсах;
- о пользователях;
- о билетах;
- об авиакомпаниях;
- о сотрудниках;
- о пассажирах.

Разрабатываемая информационная система предназначена для продажи авиабилетов и упрощения доступа к нужной информации. Наличие данной разработки улучшает организационную работу авиаперевозчика за счёт отсутствия бумажной документации, поиск и систематизация которой занимали бы очень большое количество времени.

1.2 Сравнительный анализ существующих информационных систем продажи авиабилетов

На данный момент существует огромное количество информационных систем, занимающихся продажами авиабилетов. Они могут представлять собой как самостоятельные приложения, так и онлайн сервисы, предоставляя пользователю доступ к веб-службам поставщиков.

Были рассмотрены следующие средства, в настоящее время существующие на рынке:

Таблица 1 – Сравниваемые программные средства

Информационная система	Разработчик	Системные требования	Адрес в Интернете
Amadeus	Master Pricer	ОС:Microsoft Windows XP и выше Процессор: PIII 1.3 GHz ОЗУ: 512 Mo	www.amadeus.ru/
Galileo	Travelport	ОС:Windows, Windows XP, Windows Vista, Windows 7 Процессор: Pentium Pll - 266 ОЗУ: 128 MB RAM	http://Travelport.com/
Ryanair.com	Ryanair	Требует подключения к сети Интернет	http://Ryanair.com/

Продолжение таблицы 1

Nemo.travel	Mute Lab	ОС:Microsoft Windows XP и выше Процессор: PIII 1.3 GHz ОЗУ: 512 Мо	http://www.Nemo-travel.com/

Amadeus – самая популярная система продаж авиабилетов в мире. Является ведущим поставщиком программных средств в области систематизации и дистрибуции информации, а также в области электронных платежей. Разработанные решения широко используются в туристической индустрии, что позволяет сотрудничать с ведущими авиаперевозчиками и турагентствами. Годовой объем бронирований превышает полмиллиарда, а годовой объем обслуженных клиентов колеблется в районе 500 миллионов[2]. Однако такие показатели играют не только положительную роль для системы, потому как запрос на рейсы осуществляется на протяжении 40 секунд, что несопоставимо с поиском того же рейса через интернет-сервисы. Также минусом данной системы является интерфейс, недружелюбный для необученного пользователя. Отсутствие русского языка тоже является недостатком, однако это несильно влияет на ее популярность в пределах нашей страны, не мешая удерживать ведущую позицию.

Подключение к системе Amadeus может осуществляться несколькими способами[3]:

- подключение через выделенные линии связи (стационарное);
- телефонная версия Dial-Up;
- подключение через интернет;
- подключение с помощью мобильного телефона.

Однако абсолютно каждый способ требует установки и работы через дополнительное ПО системы Amadeus, что усложняет процесс подключения, делая его не только более продолжительным по времени, но и более затратным в финансовом плане.

Galileo – одна из самых востребованных систем продаж авиабилетов в мире. Обеспечивает пользователя возможностью бронировать не только авиаперевозки, но и сопутствующие туристические услуги. Также как и Amadeus

данная система предназначена для работы на терминалах с заранее обученными пользователями.

Поддерживает только два вида подключения:

- стационарное подключение;
- подключение через интернет.

Стационарное подключение осуществляется посредством линий SITA и стоит дороже подключения через интернет, но такая версия предусматривает бесплатное ПО, состоящее из шести программ, расширяющих графический интерфейс системы и предоставляющих дополнительные функции по бронированию. Подключение через интернет стоит дороже, однако поддерживает многопользовательский режим.

Система Galileo не поддерживает русский язык, как и Amadeus, но существует программа автоматизации деятельности туристических агенств Galileo Office, созданная специально для российского рынка услуг[4].

Ryanair.com – сайт авиакомпании Ryanair, которая является крупнейшим европейским бюджетным перевозчиком. В отличие от большинства других сайтов, осуществляющих продажу авиабилетов, не использует ни одну из GDS. Вся информация о рейсах, ценах и билетах предоставляется непосредственно самой авиакомпанией в виде БД, а диалог с потенциальным покупателем ведется при помощи сайта, оперирующего к данной БД. Ryanair.com позволяет осуществлять поиск авиабилетов по различным критериям, а также бронирование номеров в гостиницах и заказ автотранспорта. Также при авторизации пользователя становится доступен личный кабинет, позволяющий просматривать статистику по осуществленным перелетам и предлагаемые авиакомпанией рейсы, подобранные индивидуально.

Подобная структура системы имеет как плюсы, так и минусы – данная информационная система обладает высоким быстродействием и удобным интерфейсом, понятным рядовому пользователю, но, в силу отсутствия подключения к GDS, количество осуществляемых рейсов, а значит и продаваемых билетов ограничено интересами авиакомпании[5].

Nemo.travel – информационная система, позволяющая бронировать авиа и ЖД билеты, а также гостиничные номера. Представляет собой мощную платформу, настроенную н использование GDS и оснащенную множеством таких средств как:

- автоматизация процессов. Представляет собой набор плагинов, осуществляющих оптимизацию трудовых затрат на обработку заказов;
- микшер результатов. Аналитическое средство для фильтрации поисковых результатов конкретного пользователя с целью предоставления наиболее подходящего варианта;
- личный кабинет. Область, защищенная информационной системой и предоставляющая отчеты по действиям авторизированного пользователя, таким как бронирование билета или аннулирование заказа. Личные кабинеты разных групп пользователей различаются функциональными возможностями;
- мидл-офис. Область, в которой реализуются все функции и компоненты Nemo, такие как управление пользователями и их группами, настройки подключений, управление справочниками и т.п.;
- управление платежными методами. Nemo позволяет подключать различные способы оплаты, от расчета при помощи банковской карты в режиме онлайн до расчета через терминалы обслуживания;
- настройка локализации. Информационная система поддерживает не только русский, но и английский и украинский языки.

Архитектура Nemo представляет собой множество модулей, что позволяет потенциальным покупателям системы подключать только нужные функции, избегая лишних денежных затрат на неиспользуемый функционал. Несмотря на то, что разработчики рассматриваемой системы сотрудничают с самыми популярными GDS, оплачивать доступ к данным глобальным дистрибутивным системам необходимо отдельно, что вкупе с затратами на саму Nemo делает приобретение невыгодным для турагенств и совершенно невыгодным для рядового пользователя[6].

Результаты сравнения программных средств представлены в таблице 2.

Таблица 2 – Результаты сравнения программных средств

Название системы	Достоинства	Недостатки
Amadeus	Высокая стабильность, независимая от объема данных, многофункциональный интерфейс, высокая гибкость администрирования системы	Низкая скорость работы, сложный для рядового пользователя интерфейс, отсутствие русского языка
Galileo	Лучшие способы дистрибьюции и управления ресурсами в туристической индустрии, существует Windows версия системы	Недоступность для рядового пользователя, отсутствие русского языка
Ryanair.com	Простой и понятный интерфейс, бесплатное использование	Отсутствие подключения к GDS
Nemo.travel	Удобный интерфейс, обширный функционал, модульность, поддержка русского языка	Невыгодна для приобретения рядовым пользователем

По результатам проведенного анализа можно сказать, что на данный момент существует множество мощных информационных систем продаж авиабилетов, обладающих обширными функциональными средствами. Однако большая часть таких средств является недоступной для рядового пользователя либо в силу сложности интерфейса, либо в силу своей высокой цены.

1.3 Постановка задачи на разработку

Разрабатываемая система должна содержать в себе следующие подсистемы:

- подсистема администрирования, позволяющая осуществлять настройку системы и ее поддержку;

- клиентская подсистема, позволяющая просматривать справочную информацию и отправлять запросы на бронирование или возврат авиабилетов.

Для доступа к любой из данных подсистем пользователь должен пройти предварительную регистрацию или, если пользователь уже зарегистрирован, авторизацию. Разным группам пользователей доступны разные функциональные возможности и уровень доступа к информации.

В ходе данной выпускной квалификационной работы должна быть создана автоматизированная информационная система продажи авиабилетов, решающая следующие задачи:

- продажа авиабилетов на запланированные рейсы;
- поиск авиабилетов по запросу пользователя ;
- администрирование информационной системы;
- создание приложения, предоставляющего пользователям графический интерфейс для доступа к системе.

Время отклика информационной системы должно быть комфортным для пользователя и не превышать 3 секунд.

2. Анализ требований на разработку информационной системы продажи авиабилетов

2.1. Анализ функциональных возможностей

Функциональные возможности системы представлены в диаграмме вариантов использования (рисунок 1).

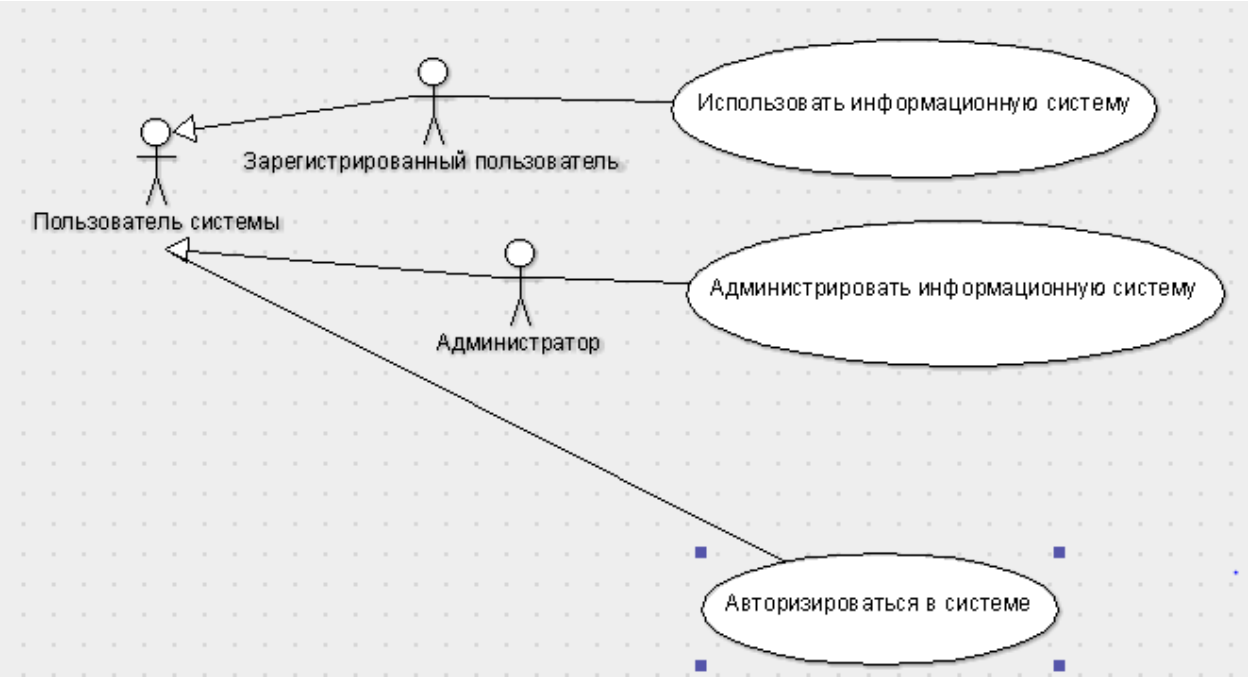

Рисунок 1 – Диаграмма вариантов использования

Диаграмма вариантов использования показывает отношения между актерами и прецедентами, описывая систему на концептуальном уровне. В данном случае существует три актера: пользователь системы, зарегистрированный пользователь и администратор, причем для предотвращения дублирования информации было использовано отношение обобщения. Также на диаграмме представлены три прецедента – администрировать информационную систему, использовать информационную систему и авторизоваться в системе. Они относятся к администратору, зарегистрированному пользователю и пользователю системы, соответственно. Далее будут представлены и описаны диаграммы декомпозиции прецедентов, представленных на рисунке 1.

На рисунке 2 представлена декомпозиция прецедента авторизации пользователя.

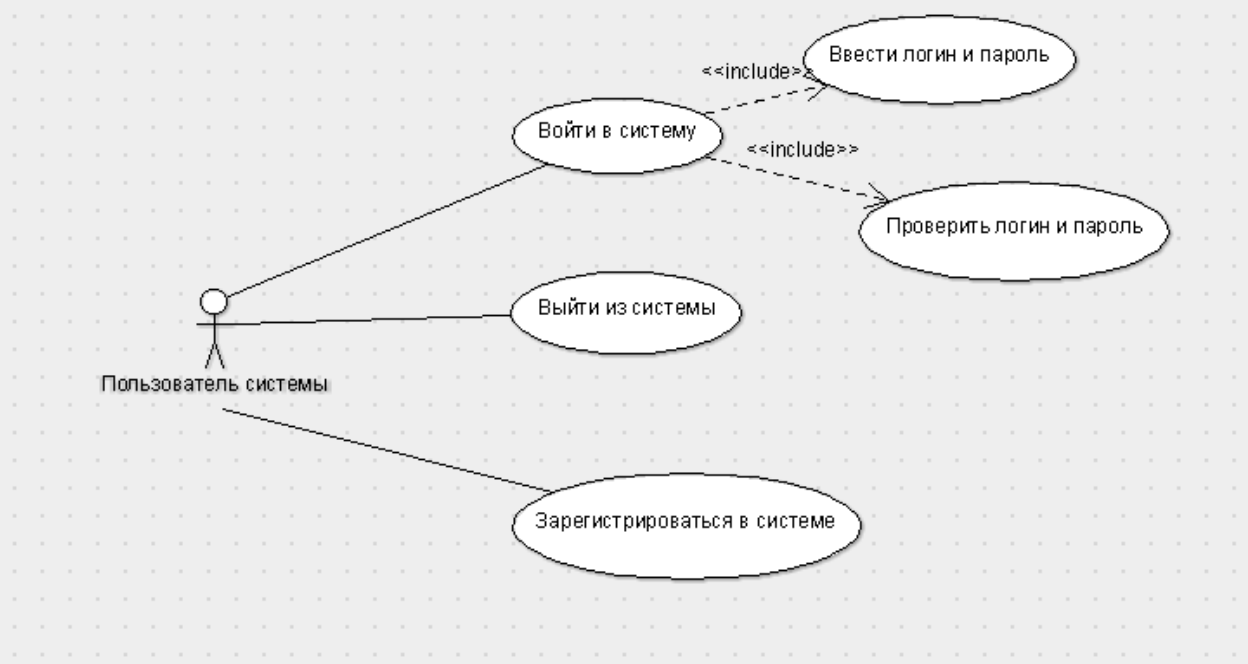

Рисунок 2 – Декомпозиция варианта использования «Авторизоваться в системе»

В таблице 3 описаны варианты использования, представленные на рисунке 2.

Таблица 3 – Варианты использования диаграммы «Авторизоваться в системе»

Варианты использования	Описание
Войти в систему	Вход пользователя в систему
Выйти из системы	Выход пользователя из системы
Зарегистрироваться в системе	Регистрация нового пользователя в системе
Ввести логин и пароль	Ввод логина и пароля пользователем
Проверить логин и пароль	Нажимая «вход в систему» пользователь инициирует проверку введенных данных

На рисунке 3 представлена декомпозиция прецедента «Администрирование информационной системы».

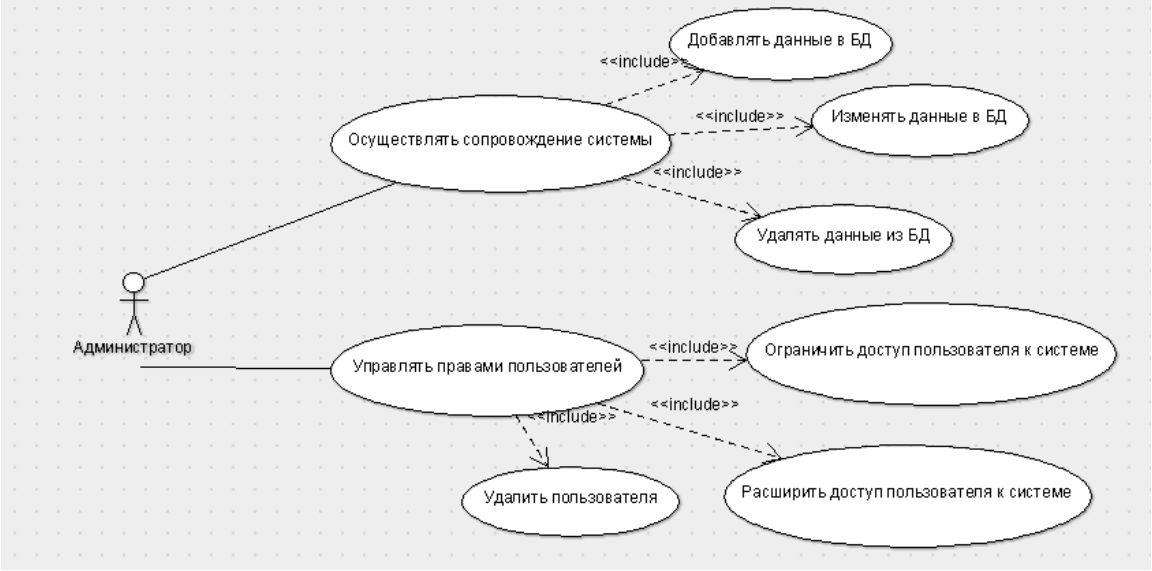

Рисунок 3 – Декомпозиция варианта использования «Администрировать информационную систему»

В таблице 4 описаны варианты использования, представленные на рисунке 3.

Таблица 4 – Варианты использования диаграммы «Администрировать информационную систему»

Варианты использования	Описание
Осуществлять сопровождение системы	Поддерживать работоспособность системы
Добавлять данные в БД	Добавлять новую информацию в БД системы
Изменять данные в БД	Изменять информацию в БД системы
Удалять данные из БД	Удалять информацию из БД системы
Управлять правами пользователей	Определять уровень доступа пользователя к системе
Ограничить доступ пользователя к системе	Ограничить права доступа пользователя системы
Расширить доступ пользователя к системе	Расширить права доступа пользователя системы
Удалить пользователя	Удалить пользователя и его данные из системы

На рисунке 4 представлена декомпозиция прецедента «Использовать информационную систему».

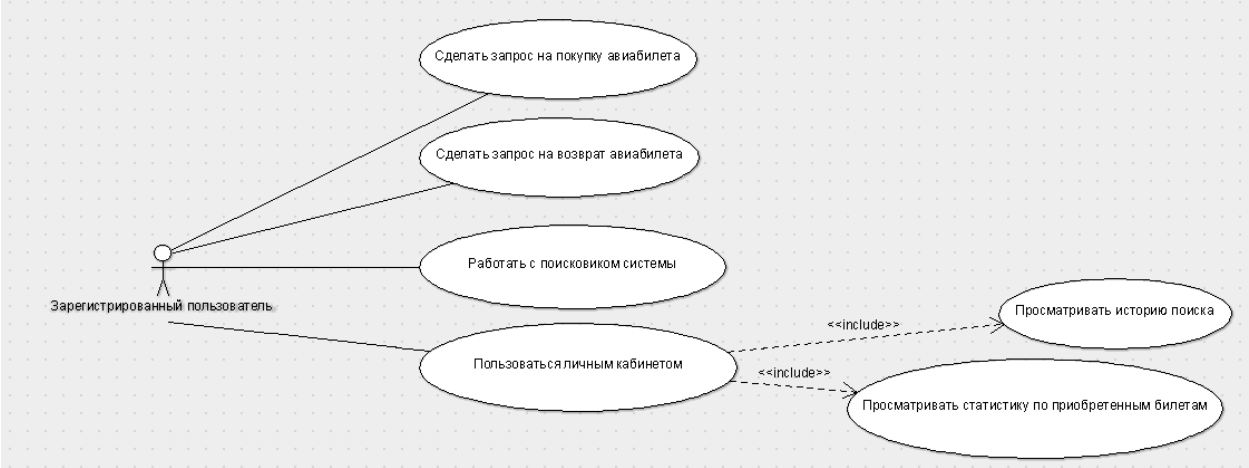

Рисунок 4 – Декомпозиция прецедента «Использовать информационную систему»

В таблице 5 описаны варианты использования, представленные на рисунке 4.

Таблица 5 – Варианты использования диаграммы «Использовать информационную систему»

Варианты использования	Описание
Сделать запрос на покупку билета	Отправить запрос к БД на бронирование авиабилета
Сделать запрос на возврат билета	Отправить запрос к БД на возврат авиабилета
Работать с поисковиком системы	Осуществлять поиск информации по рейсам
Пользоваться личным кабинетом	Использовать личный кабинет пользователя
Просматривать историю поиска	Осуществлять просмотр поисковых запросов, совершенных пользователем
Просматривать статистику по приобретенным билетам	Осуществлять просмотр статистики по приобретенным билетам

В ходе анализа функциональных возможностей были представлены диаграмма использования информационной системы продажи авиабилетов и ее декомпозиции, а также описаны основные прецеденты и актеры.

2.2. Выбор средств разработки

Для разработки информационной системы были выбраны язык программирования C#, MS Visual Studio 2015, язык программирования базы данных и запросов к ней SQL и система управления базами данных Microsoft SQL Server 2012.

Язык C# является объектно-ориентированным языком программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств, вроде Visual Studio.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++)[7].

Среда разработки Microsoft Visual Studio 2015 — это набор инструментов и средств, предназначенных для помощи разработчикам программ любого уровня квалификации в решении сложных задач и создания новаторских решений. Роль Visual Studio заключается в том, чтобы улучшить процесс разработки и упростить разработку высокоэффективных программ. В Visual Studio содержатся инструменты для всех этапов разработки программного обеспечения (разработка, тестирование, развертывание, интеграция и

управления). Visual Studio разрабатывается таким образом, чтобы обеспечить высокую надежность и совместимость. Visual Studio обладает удачным сочетанием безопасности, масштабируемости и взаимодействия[8].

SQL (англ. Structured Query Language — «язык структурированных запросов») – универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. SQL основывается на исчислении кортежей. Язык SQL ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц-отношений. Важнейшая особенность его структур – ориентация на конечный результат обработки данных, а не на процедуру этой обработки. Язык SQL сам определяет, где находятся данные, индексы и даже какие наиболее эффективные последовательности операций следует использовать для получения результата, а потому указывать эти детали в запросе к базе данных не требуется.

Основные особенности SQL:

- независимость от конкретной СУБД – несмотря на наличие диалектов и различий в синтаксисе, в большинстве своём тексты SQL-запросов, содержащие DDL и DML, могут быть достаточно легко перенесены из одной СУБД в другую. Существуют системы, разработчики которых изначально ориентировались на применение по меньшей мере нескольких СУБД (например: система электронного документооборота Documentum может работать как с Oracle, так и с Microsoft SQL Server и IBM DB2). Естественно, что при применении некоторых специфичных для реализации возможностей такой переносимости добиться уже очень трудно;
- наличие стандартов и набора тестов для выявления совместимости и соответствия конкретной реализации SQL общепринятому стандарту только способствует «стабилизации» языка. Правда, стоит обратить внимание, что сам по себе стандарт местами чересчур формализован и раздут в размерах (например, Core-часть стандарта SQL:2003 представляет собой более 1300 страниц текста);

- декларативность – с помощью SQL программист описывает только то, какие данные нужно извлечь или модифицировать. То, каким образом это сделать, решает СУБД непосредственно при обработке SQL-запроса. Однако не стоит думать, что это полностью универсальный принцип — программист описывает набор данных для выборки или модификации, однако ему при этом полезно представлять, как СУБД будет разбирать текст его запроса. Чем сложнее сконструирован запрос, тем больше он допускает вариантов написания, различных по скорости выполнения, но одинаковых по итоговому набору данных;
- реляционная основа языка – SQL является языком реляционных БД, поэтому он стал популярным тогда, когда получила широкое распространение реляционная модель представления данных. Табличная структура реляционной БД хорошо понятна, а потому язык SQL прост для изучения[9].

Microsoft SQL Server — система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями.

На базе Microsoft SQL Server 2012 могут быть построены решения для компаний малого, среднего и крупного бизнеса. SQL Server 2012 выпускается в двух основных редакциях Standard и Enterprise. На основе последней создана также редакция для разработчиков Developer Edition, лицензия на которую позволяет разрабатывать и тестировать системы и приложения.

Enterprise-версия системы SQL Server 2012 представляет собой комплексную платформу, которая позволяет работать даже с самыми требовательными корпоративными OLTP-системами и хранилищами данных. Она обладает значительной масштабируемостью, возможностью создавать громадные хранилища данных, продвинутыми средствами анализа и усиленной безопасностью, что позволяет использовать ее как основу для критически

важных бизнес-приложений. Эта редакция позволяет консолидировать серверы и выполнять большое число OLTP-операций и крупные отчеты.

Редакцию Microsoft SQL Server 2012 Enterprise характеризуют:

- высокий уровень доступности – непрерывность бизнес-процессов обеспечивается благодаря защите данных от дорогостоящих человеческих ошибок и максимальному уменьшению сроков аварийного восстановления;
- производильность и масштабируемость – инфраструктура на основе SQL Server 2012 Enterprise позволяет справиться с любыми пиковыми нагрузками;
- безопасность – вопросы конфиденциальности, а также соответствия нормативным требованиям решаются с помощью встроенных средств защиты от несанкционированного доступа;
- управляемость – автоматические диагностика, калибровка и настройка инфраструктуры позволяют управлять огромными объемами данных, значительно сократив издержки на управление и обслуживание;
- бизнес-аналитика. SQL Server 2012 Enterprise помогает легко собрать и проанализировать большие объемы данных из хранилищ или киосков.

Кроме редакций Enterprise и Standard, существуют специализированные редакции SQL Server 2012, одна из которых Express. Редакция Express — также доступная для бесплатной загрузки, эта редакция идеальна для обучения и создания настольных и небольших серверных приложений[10].

2.3. Архитектура информационной системы продажи авиабилетов

Архитектурой информационной системы называется концепция, определяющая структуру и взаимосвязь компонентов в данной системе. Компоненты информационной системы по выполняемым функциям можно разделить на три слоя: слой представления, слой бизнес-логики и слой доступа к данным. Разрабатываемой информационной системе продажи авиабилетов соответствует клиент-серверная архитектура (рисунок 5).

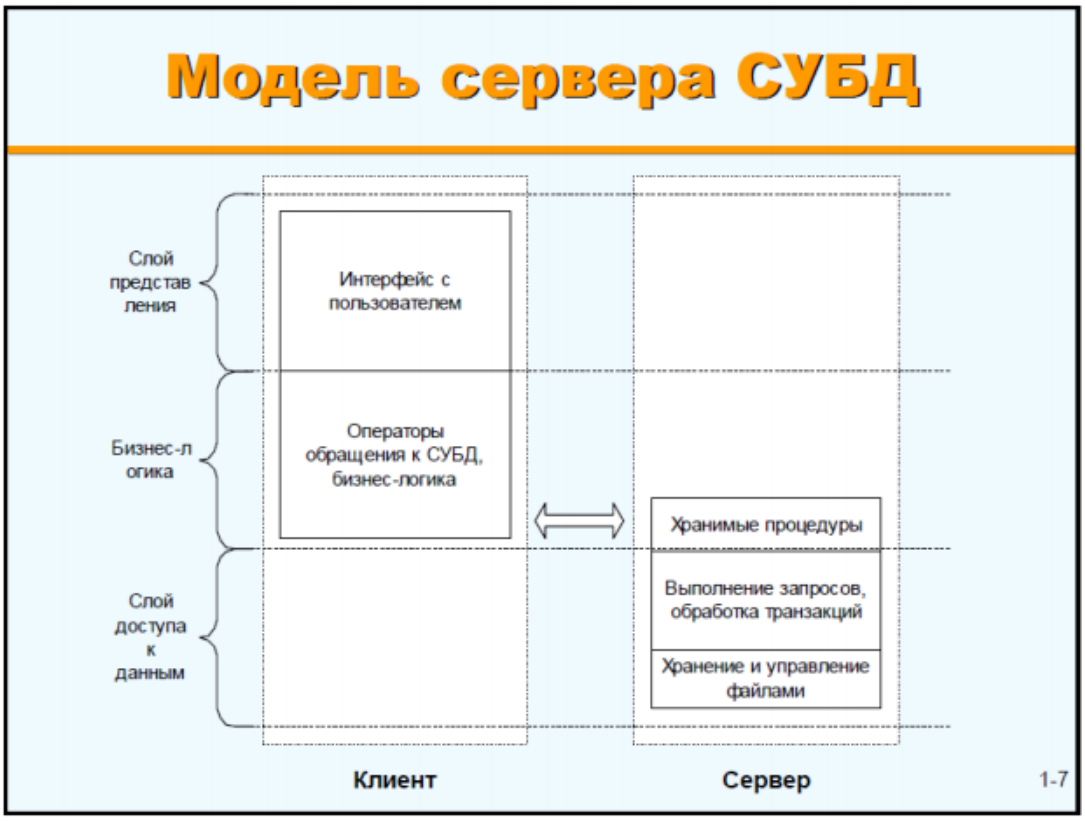

Рисунок 5 – Архитектура клиент-сервер

Такой подход обеспечивает многопользовательский режим доступа к информации, а также гарантирует целостность данных. Пользователь использует спроектированный графический интерфейс клиентской части системы, которая в свою очередь отправляет запросы серверному ПО. Также особенностью такой архитектуры является разделение функциональных возможностей приложения между клиентом и сервером[11].

2.4. Функционально-стоимостной анализ

Функционально стоимостной анализ – метод анализа объекта, позволяющий найти баланс между полезностью и себестоимостью функциональных возможностей данного объекта. Он используется для постоянного совершенствования издаваемой продукции и методов ее создания.

В рамках выполнения выпускной квалификационной работы функционально-стоимостной анализ предназначен для описания существующих бизнес-процессов и определения их стоимости.

На рисунке 6 представлена контекстная диаграмма бизнес-процесса

«Проектирование ВКР».

На рисунке 7 изображена декомпозиция бизнес–процесса «Выпускная квалификационная работа». Диаграммы бизнес-процессов выполнены в нотации IDEF0 в программной среде ERwin Process Modeler.

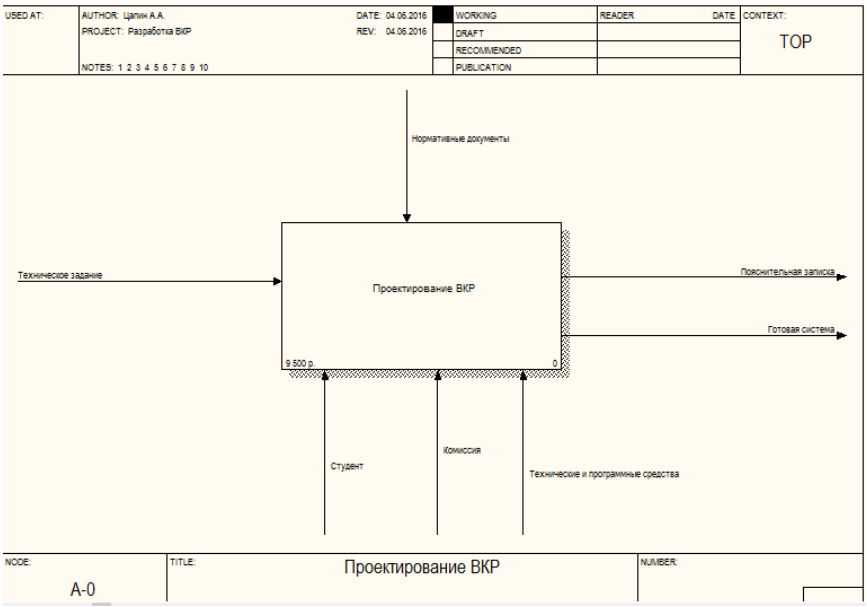

Рисунок 6 – Контекстная диаграмма «Выпускная квалификационная работа»

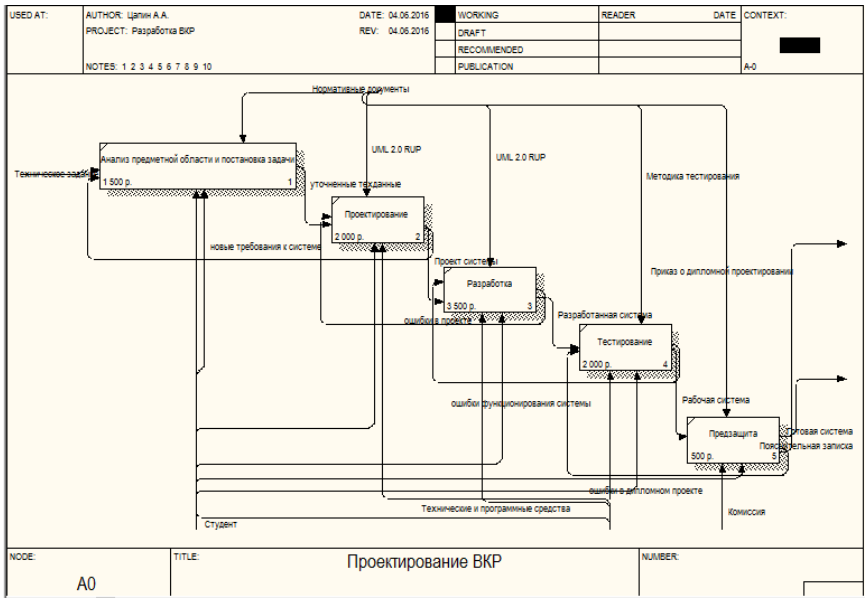

Рисунок 7 – Декомпозиция бизнес–процесса «Выпускная квалификационная работа»

При декомпозиции контекстной диаграммы процесс выполнения выпускной квалификационной работы разбит на 5 составляющих бизнес-процессов: «Анализ предметной области и постановка задачи», «Анализ требований на разработку», «Разработка приложения», «Тестирование» и

«Предзащита». Проведен анализ временных затрат для каждого этапа разработки и рассчитана стоимость финансовых затрат, учитывая тот факт, что разработкой занимается один программист. Результаты вычислений представлены в таблице 6.

Таблица 6 – Затраты на разработку

Этапы	Время	Тариф	Потраченные денежные средства
Анализ предметной области и постановка задачи	10ч	150руб/час	1500 руб
Анализ требований на разработку	10ч	200руб/час	2000 руб
Разработка приложения	20ч	175руб/час	3500 руб
Тестирование	10ч	200руб/час	2000 руб
Предзащита	2ч	250руб/час	500 руб
		Итого:	9500 руб

Для иллюстрации графика работ по данному проекту была использована диаграмма Ганта.

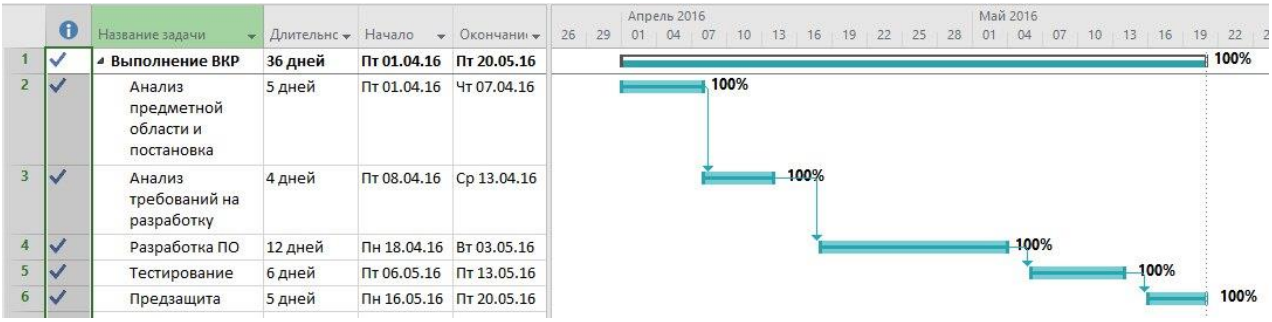

Рисунок 8 – Диаграмма Ганта для выполнения выпускной квалификационной работы

При проведении функционально стоимостного анализа были определены основные процессы выполнения выпускной квалификационной работы и стоимость их выполнения, равная 9500 рублей.

3. Разработка информационной системы продажи авиабилетов

3.1. Разработка базы данных информационной системы продажи авиабилетов

Процесс создания информационной модели начинается с определения концептуальных требований будущих пользователей БД. Концептуальная модель отображает предметную область в виде взаимосвязанных объектов без указания способов их физического хранения. Концептуальная модель представляет интегрированные концептуальные требования всех пользователей к базе данных данной предметной области. Возможно, что отраженные в концептуальной модели взаимосвязи между объектами окажутся впоследствии нереализуемыми средствами выбранной СУБД. Это потребует изменения концептуальной модели. Версия концептуальной модели, которая может быть реализована конкретной СУБД, называется логической моделью.

Логическая модель отражает логические связи между атрибутами объектов вне зависимости от их содержания и среды хранения и может быть реляционной, иерархической или сетевой. Таким образом, логическая модель отображает логические связи между информационными данными в данной концептуальной модели.

Различным пользователям в информационной модели соответствуют различные подмножества ее логической модели, которые называются внешними моделями пользователей. Таким образом, внешняя модель пользователя представляет собой отображение концептуальных требований этого пользователя в логической модели и соответствует тем представлениям, которые пользователь получает о предметной области на основе логической модели. Следовательно, насколько хорошо спроектирована внешняя модель, настолько полно и точно информационная модель отображает предметную область и настолько полно и точно работает автоматизированная система управления этой предметной областью.

Логическая модель отображается в физическую память, которая может быть построена на электронных, магнитных, оптических, биологических или других принципах.

Внутренняя модель предметной области определяет размещение данных, методы доступа и технику индексирования в данной логической модели и иначе называется физической моделью.

Физическая модель данных оперирует категориями, касающимися организации внешней памяти и структур хранения, используемых в данной операционной среде. В настоящий момент в качестве физических моделей используются различные методы размещения данных, основанные на файловых структурах: это организация файлов прямого и последовательного доступа, индексных файлов и инвертированных файлов, файлов, использующих различные методы хеширования, взаимосвязанных файлов. Кроме того, современные СУБД широко используют страничную организацию данных. Физические модели данных, основанные на страничной организации, являются наиболее перспективными[12].

Логическая и физическая модели базы данных информационной системы продажи авиабилетов представлены на рисунках 9 и 10, соответственно.

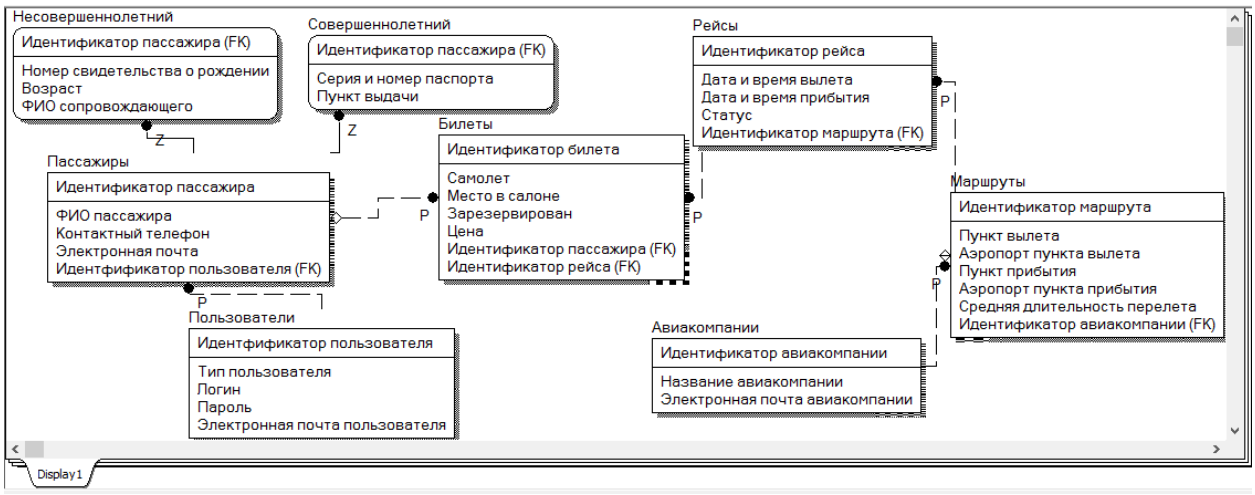

Рисунок 9 – Логическая модель базы данных продажи авиабилетов

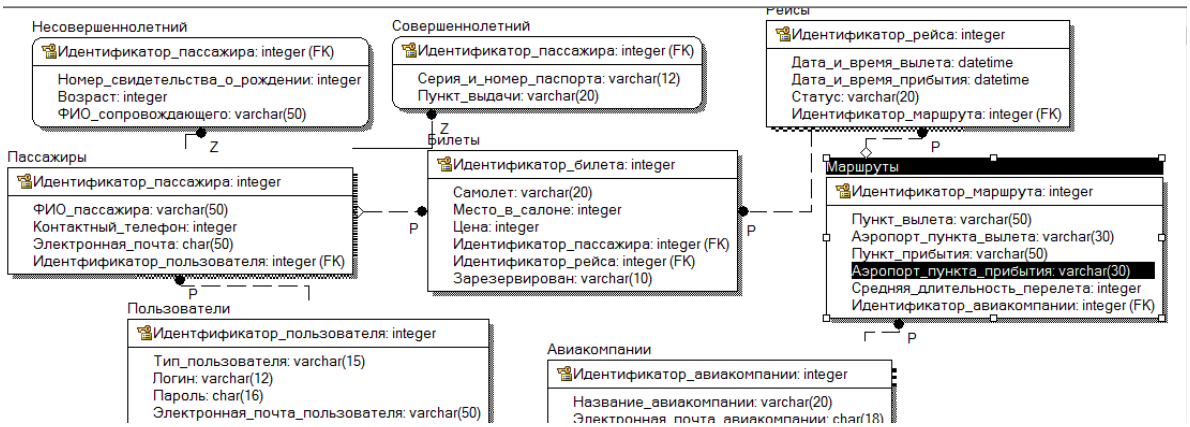

Рисунок 10 – Физическая модель базы данных продажи авиабилетов

Описание сущностей представлено в таблице 7.

Таблица 7 – Описание сущностей

Имя сущности	Описание сущности
Пассажиры	Сущность, хранящая данные о пассажирах
Совершеннолетний	Сущность, хранящая данные о совершеннолетних пассажирах
Несовершеннолетний	Сущность, хранящая данные о несовершеннолетних пассажирах
Пользователи	Сущность, хранящая данные о пользователях системы
Билеты	Сущность, хранящая данные о билетах
Рейсы	Сущность, хранящая данные о рейсах
Авиакомпании	Сущность, хранящая данные о авиакомпаниях
Маршруты	Сущность, хранящая данные о маршрутах

Полное описание атрибутов всех сущностей представлено в таблицах 8 – 15.

Таблица 8 – Сущность «Пассажиры»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор пассажира	Целый	Нет	1
ФИО Пассажира	Символьный	Нет	Иванов Иван Иванович
Контактный телефон	Целый	Нет	89765432108

Продолжение таблицы 8

Электронная почта	Символьный	Нет	Sheog.k@yandex,ru
Идентификатор пользователя	Целый	Нет	1

Таблица 9 – Сущность «Несовершеннолетний»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор пассажира	Целый	Нет	1
Номер свидетельства о рождении	Целый	Нет	12312354364346
Возраст	Целый	Нет	12
ФИО сопровождающего	Символьный	Да	Петров Петр Петрович

Таблица 10 – Сущность «Совершеннолетний»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор пассажира	Целый	Нет	1
Серия и номер паспорта	Символьный	Нет	5600 800111
Пункт выдачи	Символьный	Нет	ОУФМС России по пензенской области

Таблица 11 – Сущность «Пользователи»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор пользователя	Целый	Нет	1
Тип пользователя	Символьный	Нет	Администратор
Логин	Символьный	Нет	Login
Пароль	Символьный	Нет	Password
Электронная почта пользователя	Символьный	Нет	Sheog.k@yandex.ru

Таблица 12 – Сущность «Билеты»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор билета	Целый	Нет	1
Самолет	Символьный	Нет	Боинг 747
Место в салоне	Целый	Нет	1
Зарезервирован	Символьный	Нет	Да
Цена	Целый	Нет	10000
Идентификатор пассажира	Целый	Да	1
Идентификатор рейса	Целый	Нет	1

Таблица 13 – Сущность «Рейсы»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор рейса	Целый	Нет	1
Дата и время вылета	Дата-время	Нет	2016-05-08 12:17:20
Дата и время прибытия	Дата-время	Нет	2016-05-08 17:17:20
Статус	Символьный	Нет	Отменен
Идентификатор маршрута	Целый	Нет	1

Таблица 14 – Сущность «Маршруты»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор маршрута	Целый	Нет	1
Пункт вылета	Символьный	Нет	г. Москва
Аэропорт пункта вылета	Символьный	Нет	Домодедово
Пункт прибытия	Символьный	Нет	г. Симферополь
Аэропорт пункта прибытия	Символьный	Нет	Бельбек
Средняя длительность перелета	Целый	Да	210
Идентификатор авиакомпании	Целый	Нет	1

Таблица 15 – Сущность «Авиакомпании»

Содержательное описание	**Тип данных**	**Возможность значения Null**	**Пример**
Идентификатор авиакомпании	Целый	Нет	1
Название авиакомпании	Символьный	Нет	Россия
Электронная почта авиакомпании	Символьный	Нет	Sheog.k@yandex.ru

Скрипты создания и заполнения таблиц представлены в приложении А

3.2. Разработка интерфейса клиентской части информационной системы

На основе диаграмм вариантов использования были разработаны следующие экранные формы:

- форма авторизации для входа в систему;
- форма регистрации для занесения данных нового пользователя в базу данных;
- форма поиска информации по рейсам и билетам;
- форма для осуществления бронирования билета;
- форма администрирования информационной системы, доступная пользователям группы «Администратор»;
- форма «Личный кабинет», доступная пользователям группы «Обычный пользователь».

На рисунке 11 представлена форма авторизации.

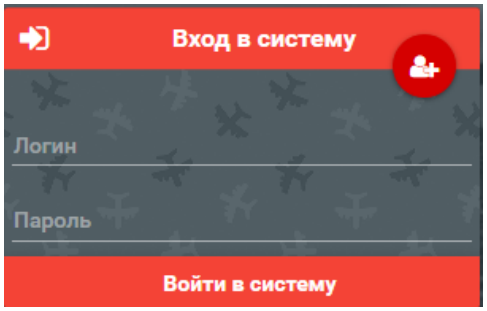

Рисунок 11 – Форма авторизации в систему

Форма регистрации нового пользователя системы представлена на рисунке 12.

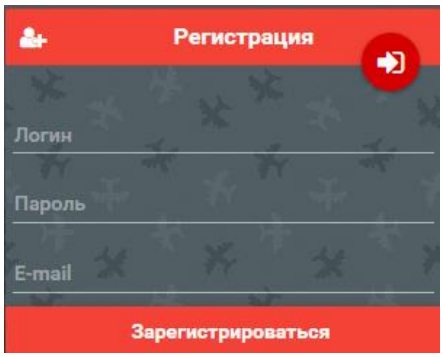

Рисунок 12 – Форма регистрации нового пользователя

Форма поиска информации по билетам представлена на рисунке 13.

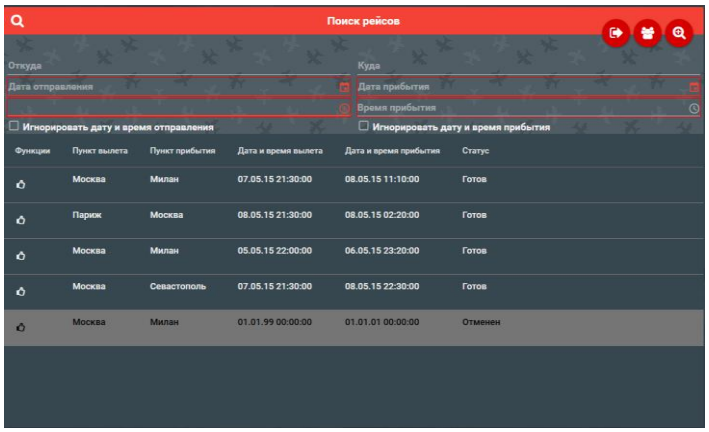

Рисунок 13 – Форма поиска информации в системе

Форма осуществления бронирования билета представлена на рисунке 14.

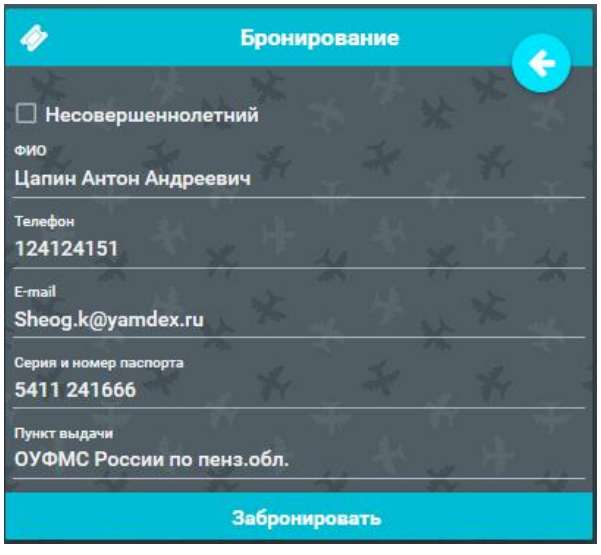

Рисунок 14 – Форма осуществления бронирования билета

На рисунке 15 представлена форма, предназначенная для администрирования информационной системы.

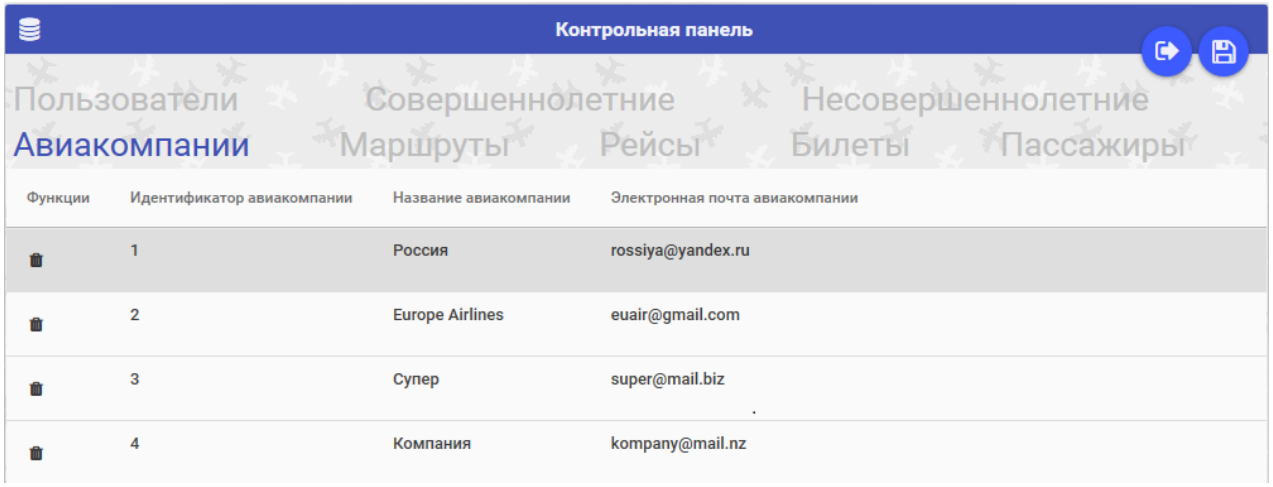

Рисунок 15 – Форма для администрирования системы

На рисунке 16 представлена форма, обеспечивающая пользователю доступ к личному кабинету.

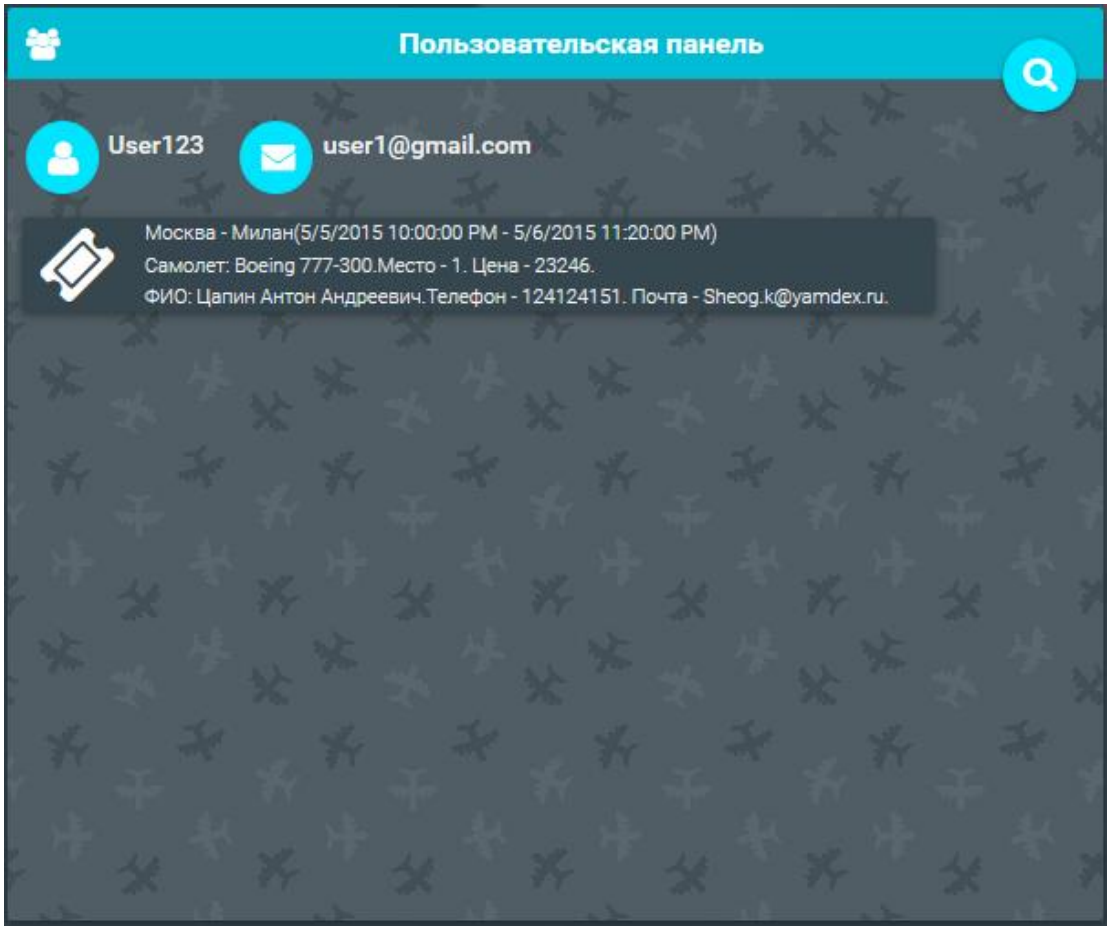

Рисунок 16 – Личный кабинет пользователя

Для демонстрации физической модели разработанного приложения была разработана диаграмма компонентов, представленная на рисунке 17. Она показывает разбиение программной системы на структурные единицы и зависимости между представленными компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п. Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом, иллюстрируются отношения клиент-источник между двумя компонентами. Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту[13].

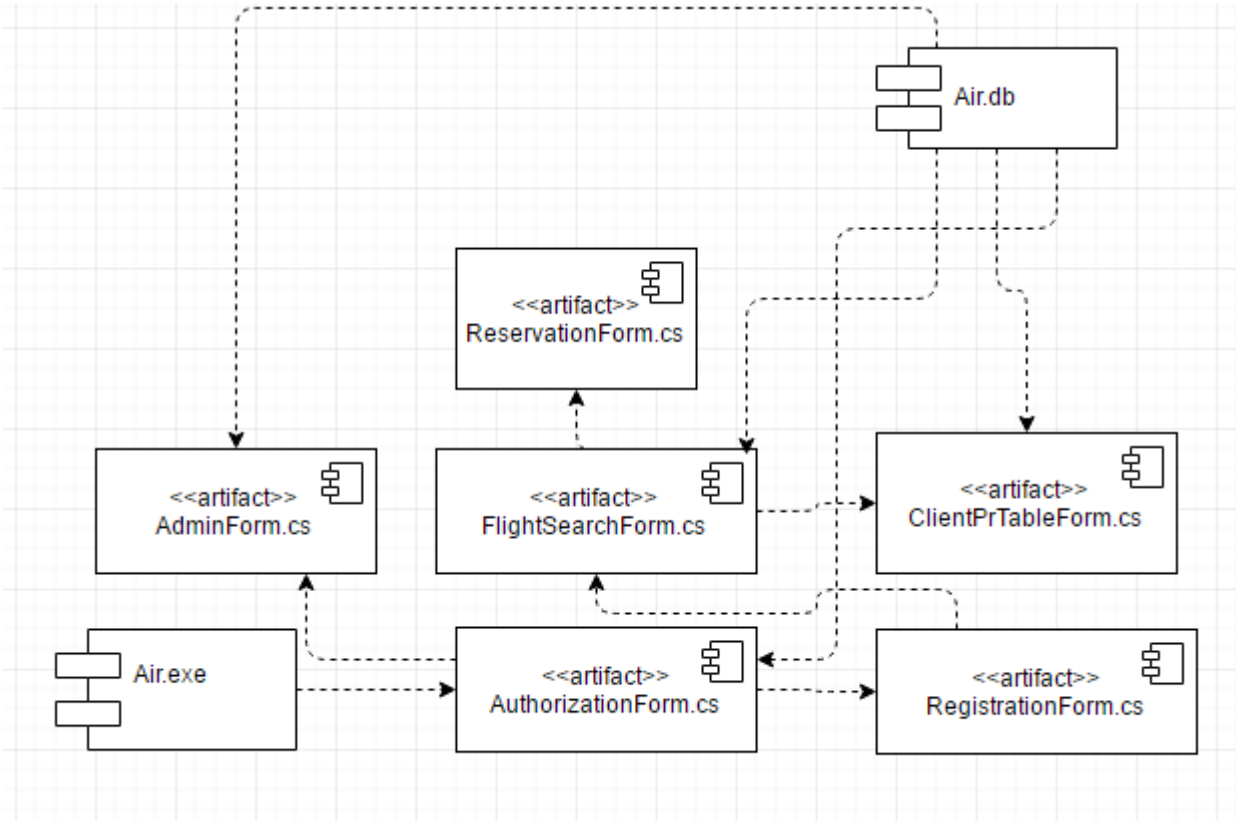

Рисунок 17 – Диаграмма компонентов

Диаграмма развертывания системы представлена на рисунке 18.

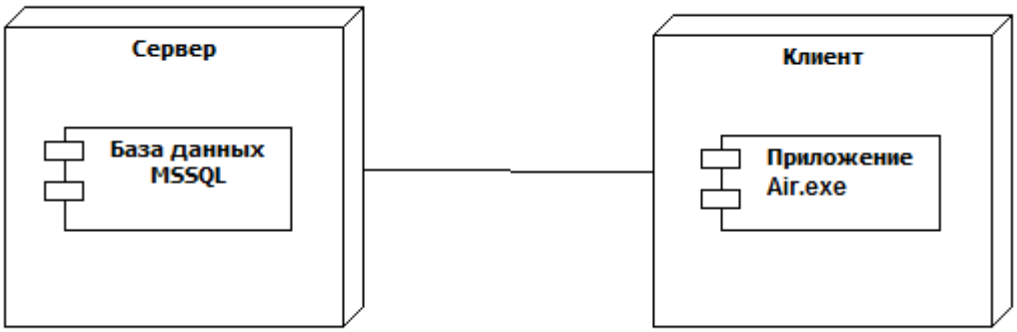

Рисунок 18 – Диаграмма развертывания

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения. Диаграмма показывает какие аппаратные компоненты («узлы») существуют (например, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, приложение, база данных), и как различные части этого комплекса соединяются друг с другом.

3.3. Разработка программных средств для пользовательского приложения информационной системы

Были разработаны классы программы, представленные на диаграмме классов (рисунок 19). Диаграммы классов являются центральным звеном методологии объектно-ориентированных анализа и проектирования. Диаграмма классов показывает классы и их отношения, тем самым представляя логический аспект проекта. Отдельная диаграмма классов представляет определенный ракурс структуры классов [14].

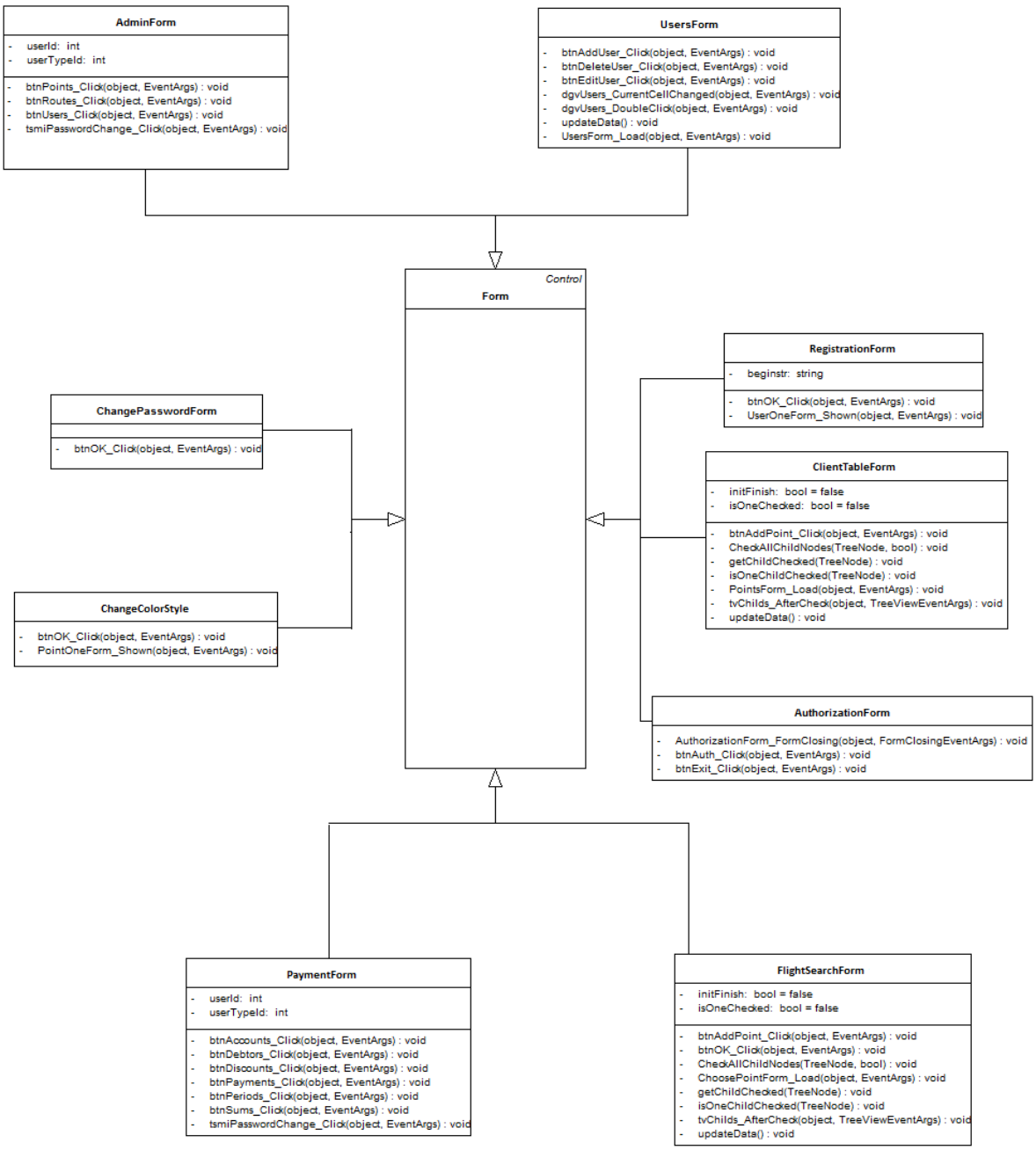

Рисунок 19 – Диаграмма классов

Основные функциональные назначения каждого из классов представлены в таблице 16.

Таблица 16 – Описание разрабатываемых классов

Имя класса	Функциональное назначение
AdminForm	Класс реализует главную форму администратора системы
UsersForm	Класс реализует форму пользователя системы
AuthorizationForm	Класс реализует форму авторизации в системе

Продолжение таблицы 16

ChangePasswordForm	Класс реализует форму для смены пароля
ClientTableForm	Класс реализует форму для отображения приобретенных пользователем билетов
ChangeColorStyle	Класс реализует форму для изменения цветовой гаммы системы
PaymentForm	Класс реализует форму для бронирования билета
FlightSearchForm	Класс реализует форму для поиска информации о билетах
RegistrationForm	Класс реализует форму для регистрации нового пользователя

Основной функцией разработанной информационной системы является реализация авиабилетов. Алгоритм работы данного прецедента показан при помощи диаграммы последовательности. Данный вид диаграмм используется, когда возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

На рисунке 20 изображена диаграмма последовательности.

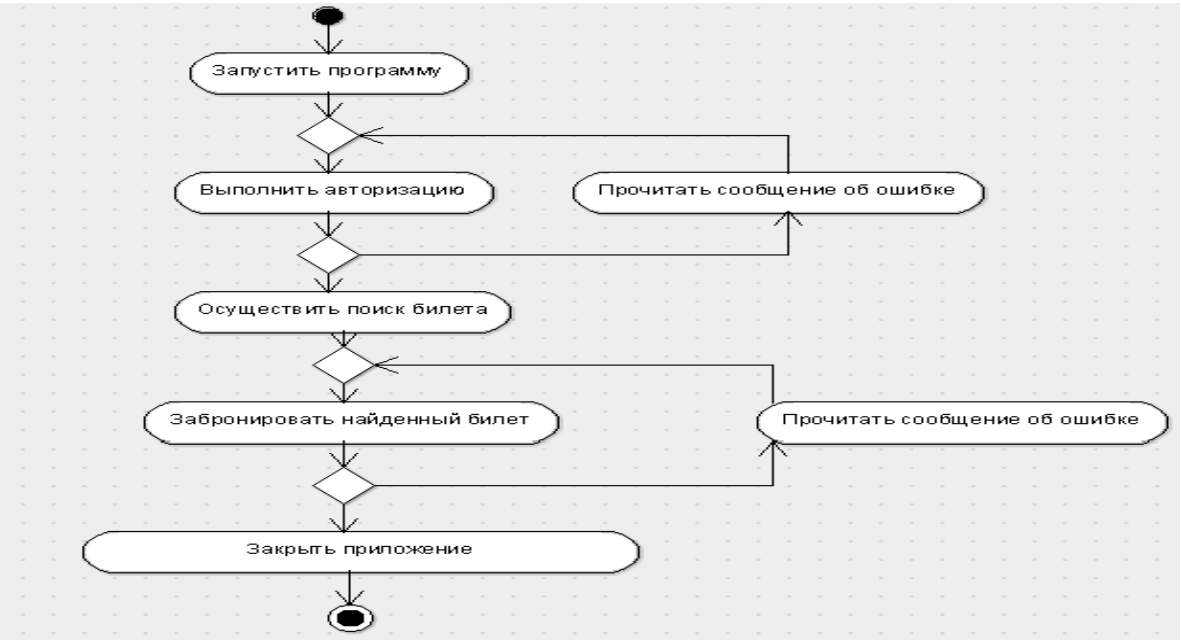

Рисунок 20 – Диаграмма деятельности для прецедента «Продажа авиабилета»

В данной главе была описана реализация серверного программного обеспечения на стороне сервера журнализации событий. Основные задачи, описанные в предметной области были выполнены.

4. Тестирование

4.1. Выбор режима тестирования

На данный момент существует несколько видов тестирования: функциональное, нефункциональное и тестирование, связанное изменениями.

При функциональном тестировании имитируется реальное использование системы с целью проверки работоспособности ее компонентов. Все тесты основываются на заявленных заранее функциях, описанных в функциональных спецификациях и прецедентах. Такое тестирование позволяет наглядным образом выявлять ошибки системы, однако у него также существуют недостатки – вероятность избыточного тестирования и возможность упущения логических ошибок.

В ходе нефункционального тестирования определяются характеристики программного обеспечения в различных измерениях. Существует несколько видов такого тестирования:

- нагрузочное тестирование;
- стрессовое тестирование;
- тестирование стабильности или надежности;
- объемное тестирование;
- тестирование установки;
- тестирование удобства пользования;
- тестирование на отказ и восстановление;
- конфигурационное тестирование.

Связанное с изменениями тестирование проводится после исправления ошибок в работе системы с целью проверки эффективности внесенных поправок[15].

Для проверки работоспособности разработанной системы было выбрано функциональное тестирование, потому что именно оно позволяет проверить реализацию заявленных возможностей наглядным образом.

4.2. Тестирование программных средств

Для функционального тестирования приложения были смоделированы различные ситуации, которые могли возникнуть во время работы приложения. Результаты тестирования приведены в таблице 17.

Таблица 17 – Результаты функционального тестирования

№ п/п	Название теста	Описание	Полученный результат
1	Авторизация зарегистрированного пользователя	При запуске приложения должна открыться форма авторизации, в которой необходимо ввести логин и пароль	После запуска приложения отображается форма авторизации. После ввода верного логина и пароля открывается форма поиска авиабилетов (рисунок Б.1). При неверном логине или пароле отображается сообщение об ошибке (Рисунок Б.2). Тест выполнен успешно.
2	Регистрация нового пользователя	Пользователь переходит из окна авторизации в окно регистрации, где указывает свои логин, пароль и email. После нажимает кнопку «зарегистрироваться» (рисунок Б.3).	После нажатия кнопки «зарегистрироваться» пользователь получает возможность войти в систему (рисунок Б.4). Тест выполнен успешно.
3	Поиск рейса	Пользователь осуществляет поиск авиабилетов по одному или нескольким критериям, предопределенным системой.	После заполнения одного или нескольких полей и нажатия кнопки «Поиск» рейсы отображаются в списке (рисунок Б.5). Тест выполнен успешно.
4	Бронирование билета	Пользователь, найдя интересующий его билет, нажимает на строку с билетом и попадает на форму бронирования	При нажатии кнопки «Забронировать» появляется всплывающее окно с информацией о билете (Рисунок Б.7). Тест выполнен успешно.

Продолжение таблицы 17

5	Просмотр личного кабинета пользователя	У каждого пользователя имеется личный кабинет, в котором фиксируются все совершенные покупки. Доступ к нему можно получить из формы поиска авиабилетов нажатием кнопки «Перейти в пользовательскую панель» (Рисунок Б.8).	После нажатия кнопки «Перейти в пользовательскую панель» пользователь попадает в личный кабинет (рисунок Б.9). Тест выполнен успешно.
6	Изменение цветового оформления приложения	В любой момент использования системы пользователь может перейти на панель настройки цветовой гаммы приложения, нажав кнопку «Изменить оформление» (Рисунок Б.10)..	После нажатия кнопки «Изменить оформление» пользователь попадает в меню настройки цветовой гаммы приложения, где может изменять дизайн приложения (Рисунок Б.11). Тест выполнен успешно.

Заключение

В данной выпускной квалификационной работе выполнены анализ требований, проектирование и реализация программных средств, предоставляющих пользователю возможность реализовать все функции программы. В результате выполнения работы было разработано приложение, автоматизирующее процесс продаж билетов клиентам. Приложение предоставляет пользователю следующие основные функции:

- авторизация пользователя;
- регистрация пользователя;
- поиск билетов для бронирования;
- бронирование билета;
- редактирование БД продаж авиабилетов, доступное пользователям группы «Администратор»;
- распределение прав доступа пользователей, доступное пользователям группы «Администратор»;
- личный кабинет пользователя, содержащий статистику по совершенным покупкам.

В ходе тестирования разработанной информационной системы было продемонстрировано, что система выполняет все поставленные задачи в соответствии со сформированными требованиями в разделе «Постановка задачи на разработку», демонстрируя корректную работу.

Список использованных источников

1. Электронный билет (воздушный транспорт) [Электронный ресурс] – Режим доступа:http://ru.wikipedia.org/wiki/Электронный_билет_(воздушный_транспорт) (Дата обращения: 05.06.2016).

2. Системы бронирования авиабилетов [Электронный ресурс] – Режим доступа: http://www.flyworld.ru/stati/sistemy-bronirovaniya-aviabiletov/ (Дата обращения: 05.06.2016).

3. Amadeus [Электронный ресурс] – Режим доступа: http://support.nemo.travel/ru/Amadeus (Дата обращения: 05.06.2016).

4. Travelport (Galileo) [Электронный ресурс] – Режим доступа: http://support.nemo.travel/ru/Travelport_(Galileo) (Дата обращения: 05.06.2016).

5. Авиабилеты. IT системы бронирования [Электронный ресурс] – Режим доступа: https://habrahabr.ru/company/buruki/blog/192384/ (Дата обращения: 05.06.2016).

6. Возможности Немо [Электронный ресурс] – Режим доступа: https://nemo.travel/bazovye-vozmozhnosti-nemo.html (Дата обращения: 05.06.2016).

7. Либерти, Дж. Создание .NET приложений Программирование на C#. – СПб.: Орейли, 2006.

8. Троелсен Э. C# и платформа .NET. Библиотека программиста. – СПб.: Питер, 2007.

9. Клайн К. SQL справочник. 2-е издание. – М.: «КУДИЦ-ОБРАЗ», 2006.

10. Р. Фрост, Д. Дей, К. Ван Слайк; пер. с англ. А.Ю. Кухаренко. Проектирование и разработка баз данных. Визуальный подход – М.: НТ Пресс, 2007.

11. Лекция 3. Архитектура ИС [Электронный ресурс] – Режим доступа: http://it-claim.ru/Education/Course/ISDevelopment/Lecture_3.pdf (Дата обращения: 06.06.2016).

12. ERwin Data Modeler [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/ERwin_Data_Modeler (Дата обращения: 06.06.2016).

13. Фаулер М. UML в кратком изложении. Применение стандартного языка объектного моделирования / Фаулер М., Скотт К. – М.: «Мир», 1999.

14. Крэг Л. Применение UML и шаблонов проектирования. – М.: Издательских дом «Вильямс», 2004.

15. Виды тестирования программного обеспечения [Электронный ресурс] – Режим доступа: http://www.protesting.ru/testing/testtypes.html (Дата обращения: 08.06.2016).

ТЕКСТ СКРИПТА СОЗДАНИЯ БАЗЫ ДАННЫХ

Приложение А

(обязательное)

```
CREATE TABLE Авиакомпании
(
	Идентификатор_авиакомпании integer identity(1,1) NOT NULL,
	Название_авиакомпании varchar(20) NOT NULL,
	Электронная_почта_авиакомпании char(18) NOT NULL,
	PRIMARY KEY (Идентификатор_авиакомпании ASC)
)
go


CREATE TABLE Маршруты
(
	Идентификатор_маршрута integer Identity(1,1) NOT NULL,
	Пункт_вылета varchar(50) NOT NULL,
	Аэропорт_пункта_вылета varchar(30) NOT NULL,
	Пункт_прибытия varchar(50) NOT NULL,
	Аэропорт_пункта_прибытия varchar(30) NOT NULL,
	Средняя_длительность_перелета integer NOT NULL,
	Идентификатор_авиакомпании integer NOT NULL,
	PRIMARY KEY (Идентификатор_маршрута ASC),
	FOREIGN KEY (Идентификатор_авиакомпании) REFERENCES
Авиакомпании(Идентификатор_авиакомпании)
)
go


CREATE TABLE Пользователи
(
	Идентификатор_пользователя integer Identity(1,1) NOT NULL,
	Тип_пользователя varchar(15) NOT NULL,
	Логин varchar(15) NOT NULL,
	Пароль varchar(15) NOT NULL,
	Электронная_почта_пользователя varchar(50) NOT NULL,
	PRIMARY KEY (Идентификатор_пользователя ASC)
)
go


CREATE TABLE Рейсы
(
	Идентификатор_рейса integer Identity(1,1) NOT NULL,
	Дата_и_время_вылета datetime NOT NULL,
	Дата_и_время_прибытия datetime NOT NULL,
	Статус varchar(20) NOT NULL,
	Идентификатор_маршрута integer NOT NULL,
	PRIMARY KEY (Идентификатор_рейса ASC),
	FOREIGN KEY (Идентификатор_маршрута) REFERENCES Маршруты(Идентификатор_маршрута)
)
go

CREATE TABLE Пассажиры
(
	Идентификатор_пассажира integer Identity(1,1) NOT NULL,
	ФИО_пассажира varchar(50) NOT NULL,
	Контактный_телефон integer NULL,
	Электронная_почта char(50) NOT NULL,
	Идентификатор_пользователя integer NOT NULL,
	PRIMARY KEY (Идентификатор_пассажира ASC),
```

```
	FOREIGN KEY (Идентификатор_пользователя) REFERENCES
Пользователи(Идентификатор_пользователя)
)
go

CREATE TABLE Билеты
(
	Идентификатор_билета integer NOT NULL,
	Самолет varchar(20) NOT NULL,
	Место_в_салоне integer NOT NULL,
	Цена integer NOT NULL,
	Идентификатор_пассажира integer,
	Идентификатор_рейса integer NOT NULL,
	Зарезервирован varchar(20) NOT NULL,
	PRIMARY KEY (Идентификатор_билета ASC),
	FOREIGN KEY (Идентификатор_пассажира) REFERENCES Пассажиры(Идентификатор_пассажира),
	FOREIGN KEY (Идентификатор_рейса) REFERENCES Рейсы(Идентификатор_рейса)
)
go

CREATE TABLE Совершеннолетний
(
	Серия_и_номер_паспорта varchar(12) NOT NULL,
	Пункт_выдачи varchar(20) NOT NULL,
	Идентификатор_пассажира integer NOT NULL,
	PRIMARY KEY (Идентификатор_пассажира ASC),
	FOREIGN KEY (Идентификатор_пассажира) REFERENCES Пассажиры(Идентификатор_пассажира)
)
go


CREATE TABLE Несовершеннолетний
(
	Номер_свидельства_о_рождении integer NULL,
	Возраст integer NOT NULL,
	ФИО_сопровождающего varchar(50) NULL,
	Идентификатор_пассажира integer	NOT NULL,
	PRIMARY KEY (Идентификатор_пассажира ASC),
	FOREIGN KEY (Идентификатор_пассажира) REFERENCES Пассажиры (Идентификатор_пассажира)
)
go



INSERT INTO Авиакомпании
VALUES ('Россия', 'rossiya@yandex.ru');
INSERT INTO Авиакомпании
VALUES ('Europe Airlines', 'euair@gmail.com');

INSERT INTO Маршруты
VALUES ('Москва', 'Домодедово','Севастополь','Крымский',120,1);
INSERT INTO Маршруты
VALUES ('Москва', 'Домодедово','Милан','Милан',220,1);
INSERT INTO Маршруты
VALUES ('Париж', 'Париж','Москва','Пулково',200,2);

INSERT INTO Рейсы
VALUES ('2015-05-07 21:30:00','2015-05-08 ','Готов',2);
INSERT INTO Рейсы
```

```
VALUES ('2015-05-08','2015-05-08 02:20:00','Готов',3);
INSERT INTO Рейсы
VALUES ('2015-05-05 22:00:00','2015-05-06 ','Готов',2);
INSERT INTO Рейсы
VALUES ('2015-05-07 21:30:00','2015-05-08 ','Готов',1);

INSERT INTO Билеты
VALUES ('Боинг-747',12,15000,1,1,'Зарезервирован');
INSERT INTO Билеты
VALUES ('Боинг-747',12,15000,2,1,'Зарезервирован');
INSERT INTO Билеты
VALUES ('Боинг-747',12,15000,3,2,'Зарезервирован');
INSERT INTO Билеты
VALUES ('Боинг-747',12,15000,4,2,'Зарезервирован');

INSERT INTO Пользователи
VALUES ('Пользователь','user1','user1','sheog.k@yandex.ru');
INSERT INTO Пользователи
VALUES ('Пользователь','user2','user2','sheog.k@yanx.ru');
INSERT INTO Пользователи
VALUES ('Администратор','admin','admin','sheog.k@y.ru');

INSERT INTO Пассажиры
VALUES ('Петров Петр Петрович',88888888888,'petrpetr@gmail.com');
INSERT INTO Пассажиры
VALUES ('Иванов Иван Иванович',89121542135,'ivanivan@gmail.com');
INSERT INTO Пассажиры
VALUES ('Алексеев Алексей Алексеевич',8352526777,'alex@gmail.com');
INSERT INTO Пассажиры
VALUES ('Васильев Михаил Александрович',8213124551,'brainpower@gmail.com');
```

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Приложение Б

(обязательное)

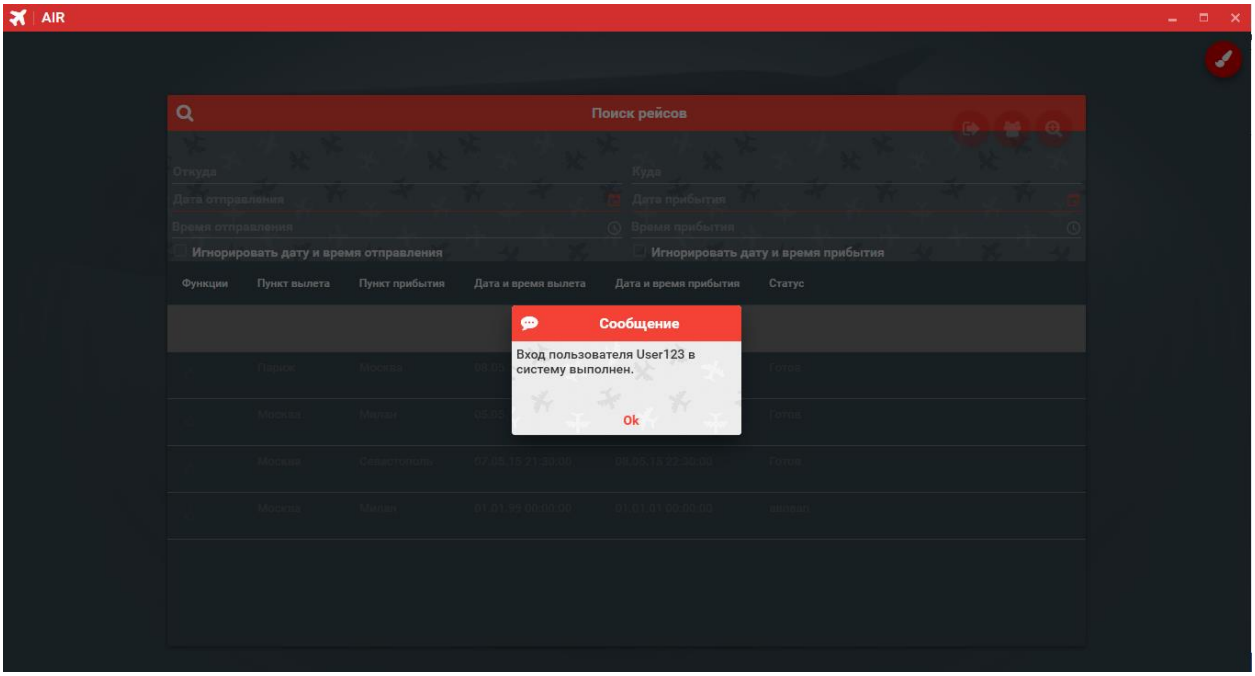

Рисунок Б.1 – Результат теста 1

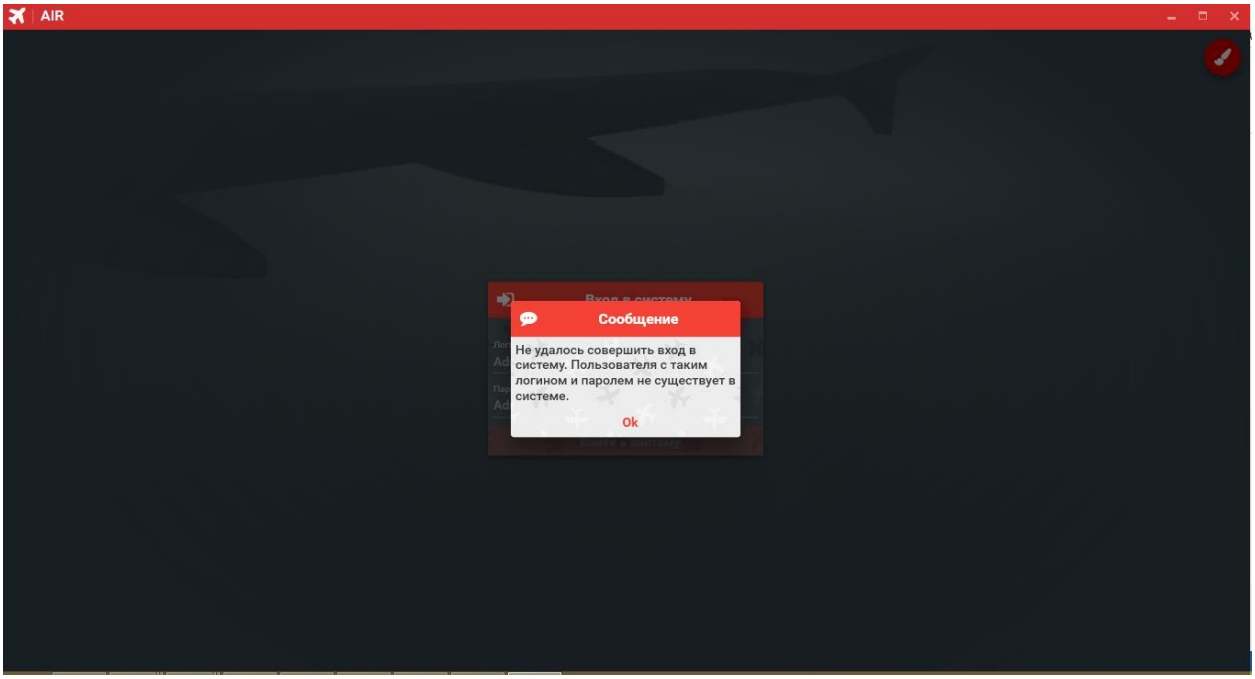

Рисунок Б.2 – Результат теста 1

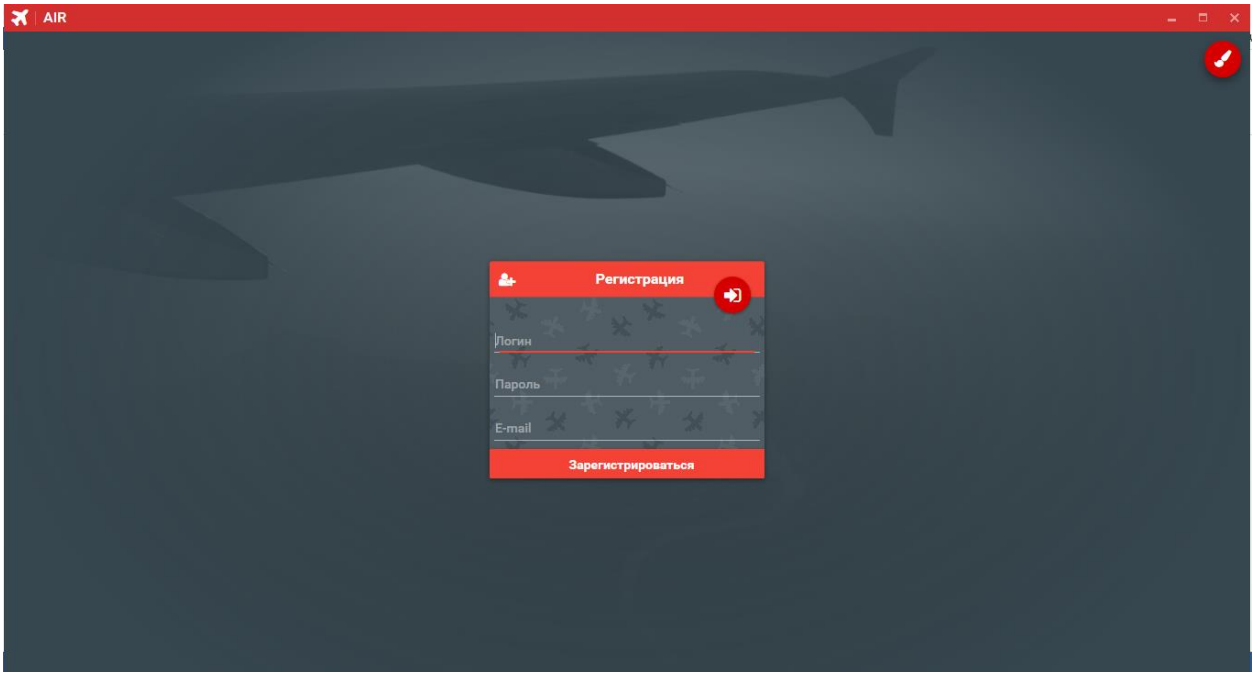

Рисунок Б3 – Описание теста 2

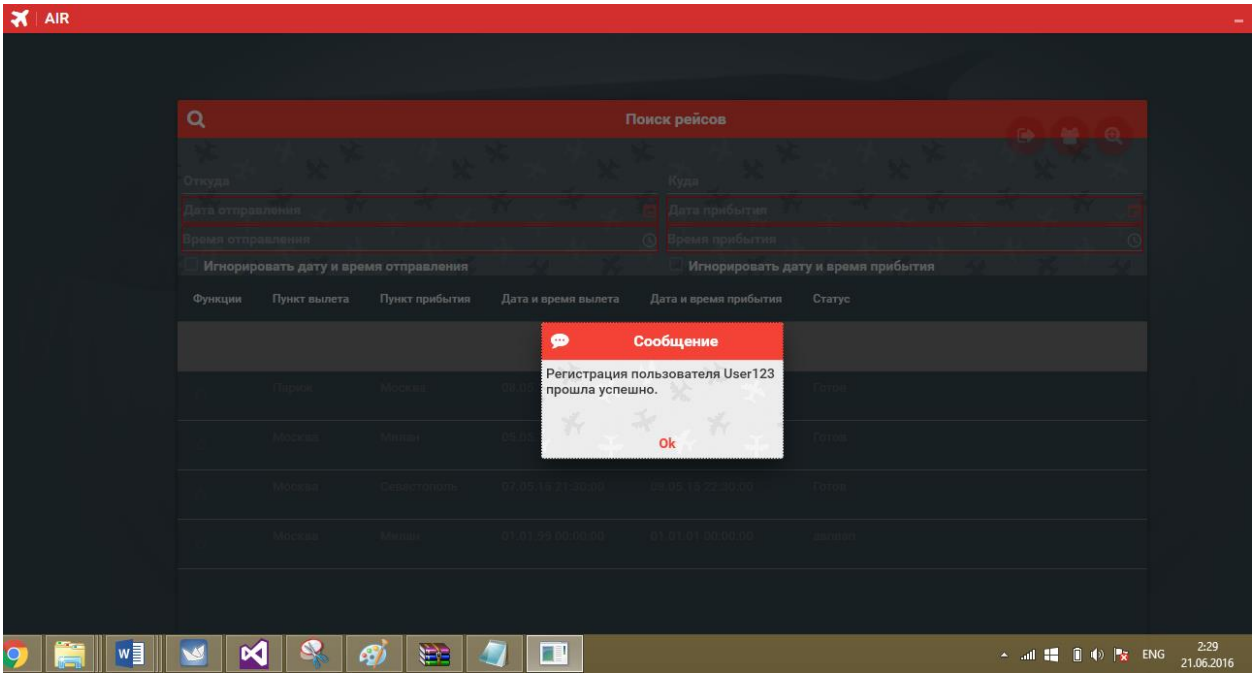

Рисунок Б.4 – Результат теста 2

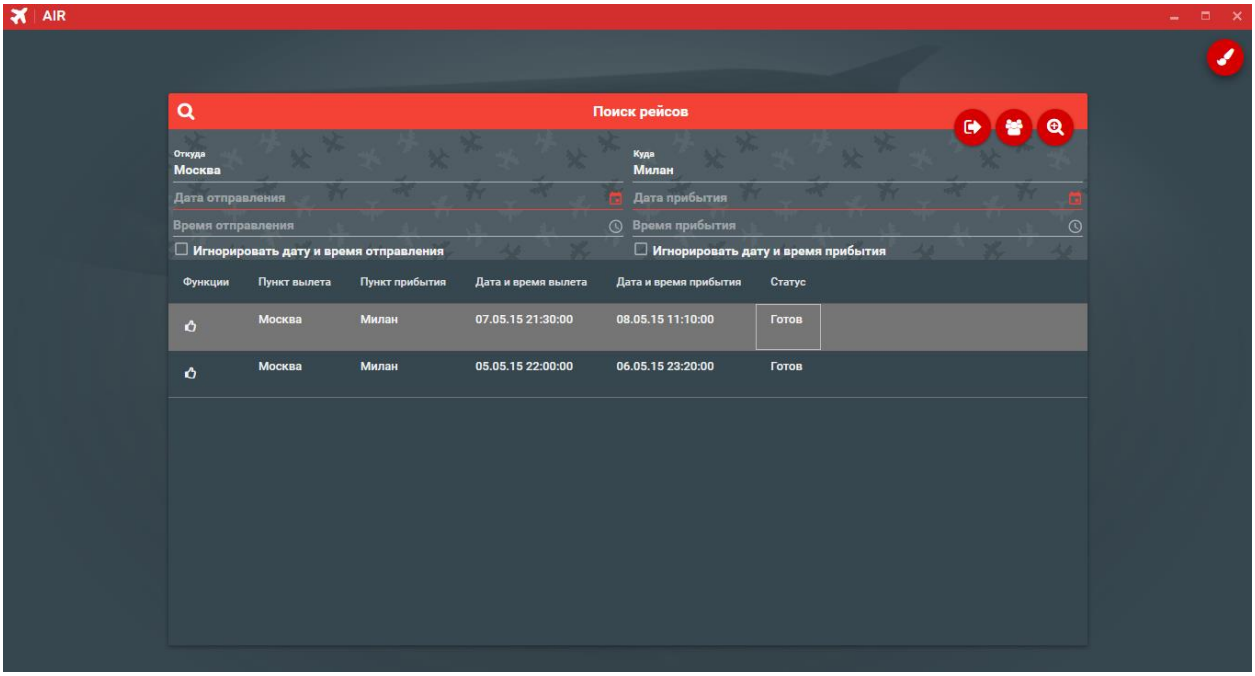

Рисунок Б.5 – Результат теста 3

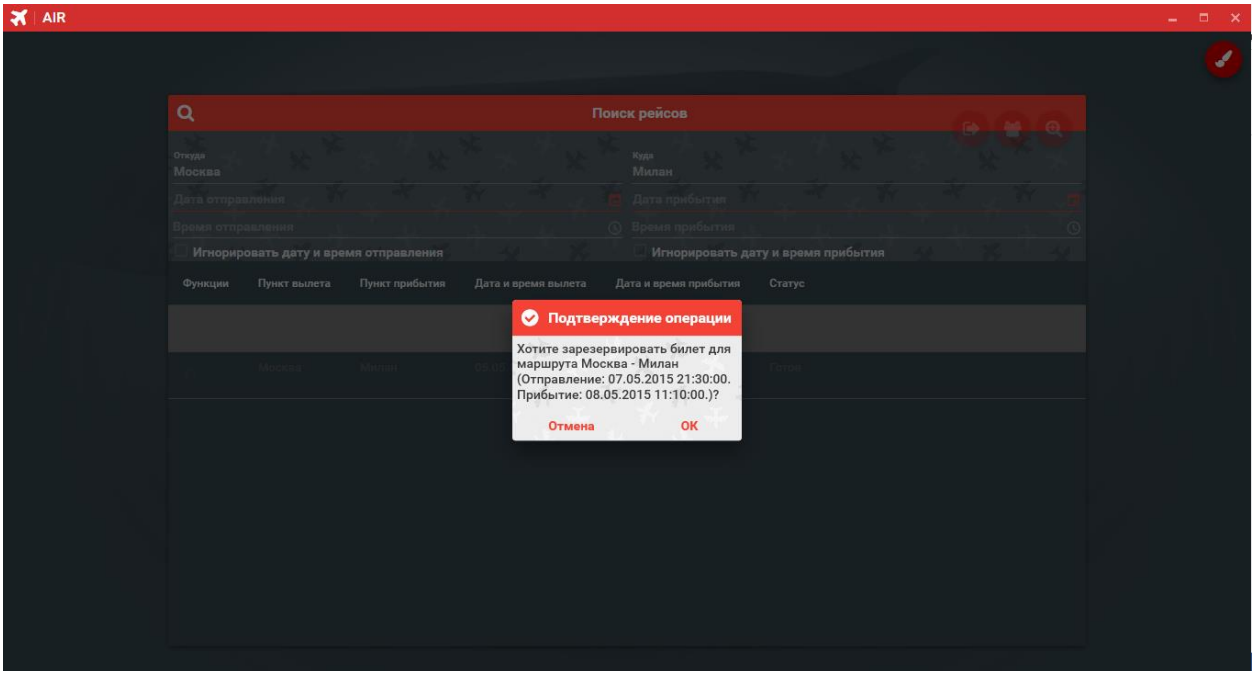

Рисунок Б.6 – Описание теста 4

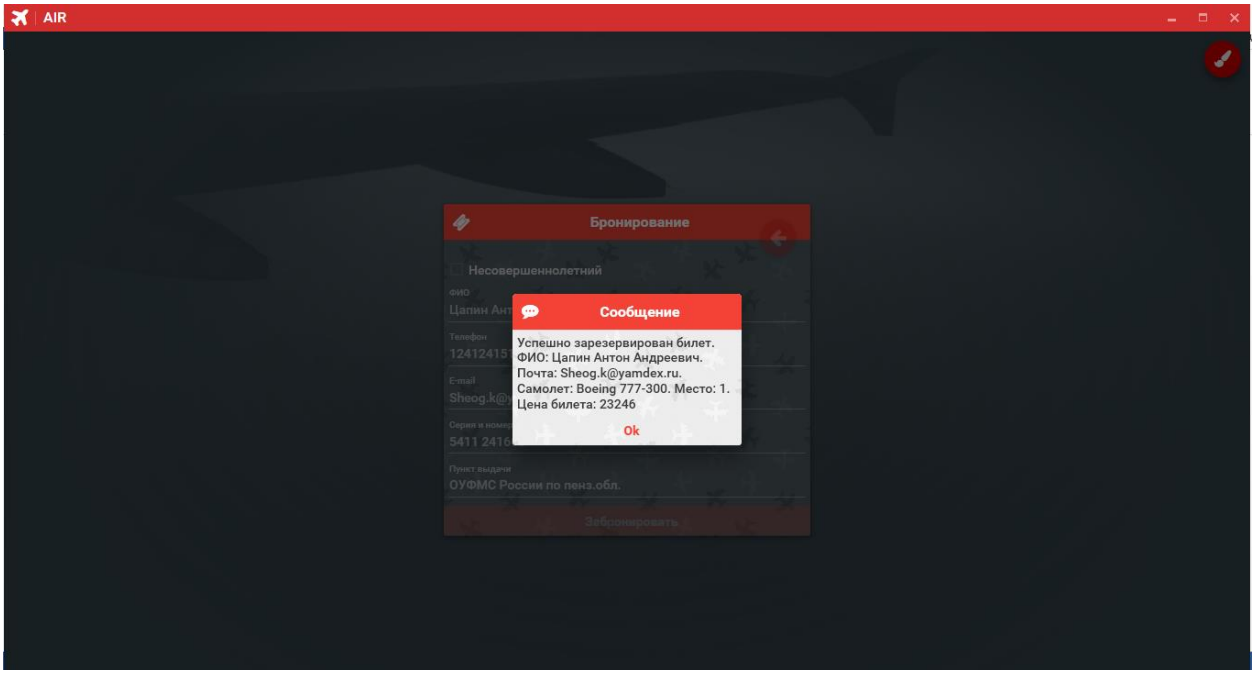

Рисунок Б.7 – Результат теста 4

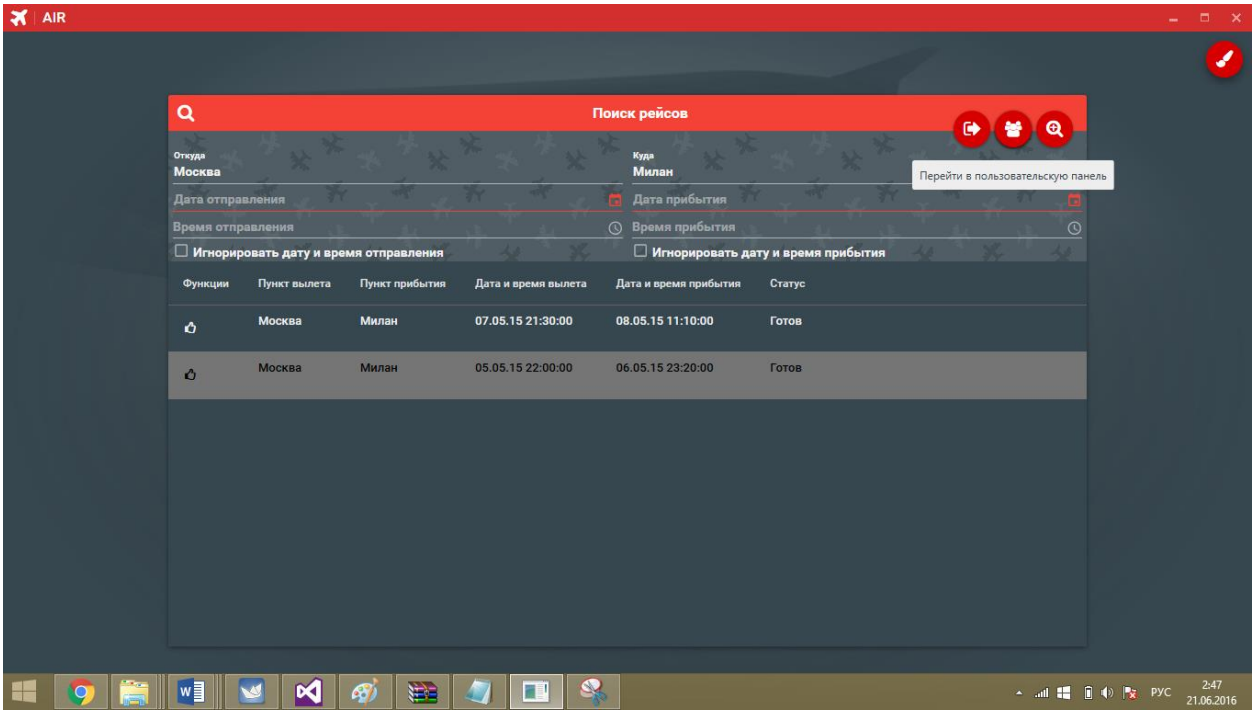

Рисунок Б.8 – Описание теста 5

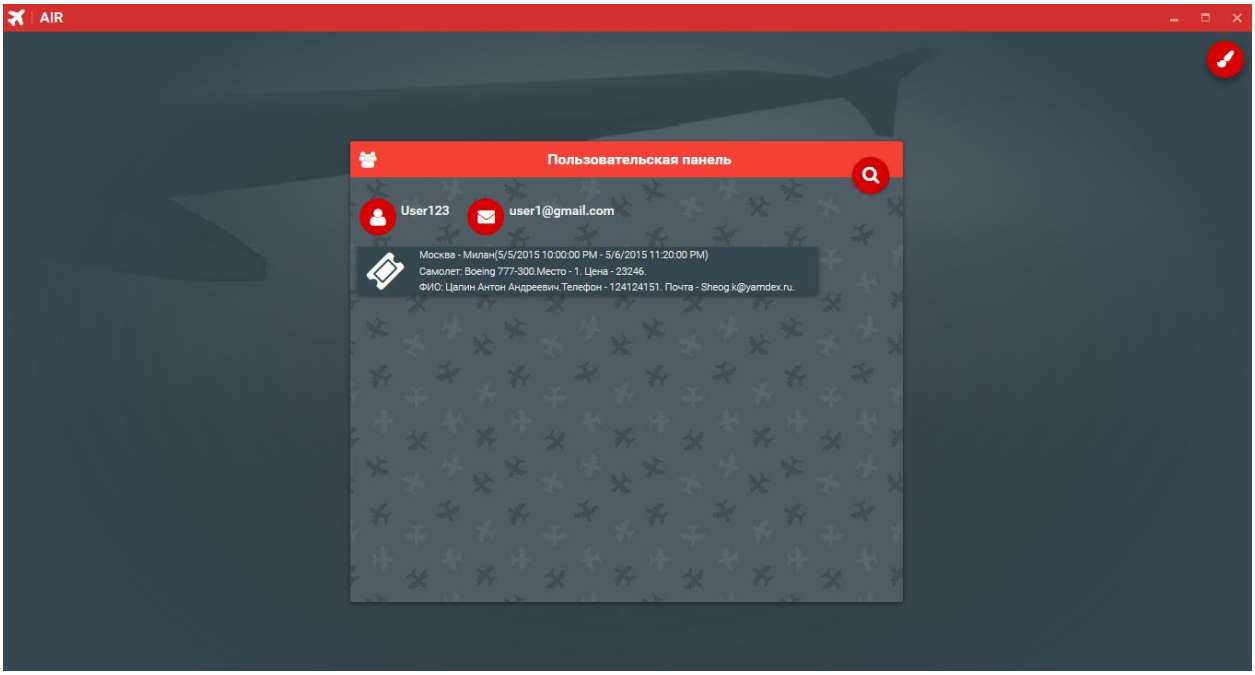

Рисунок Б.9 – Результат теста 5

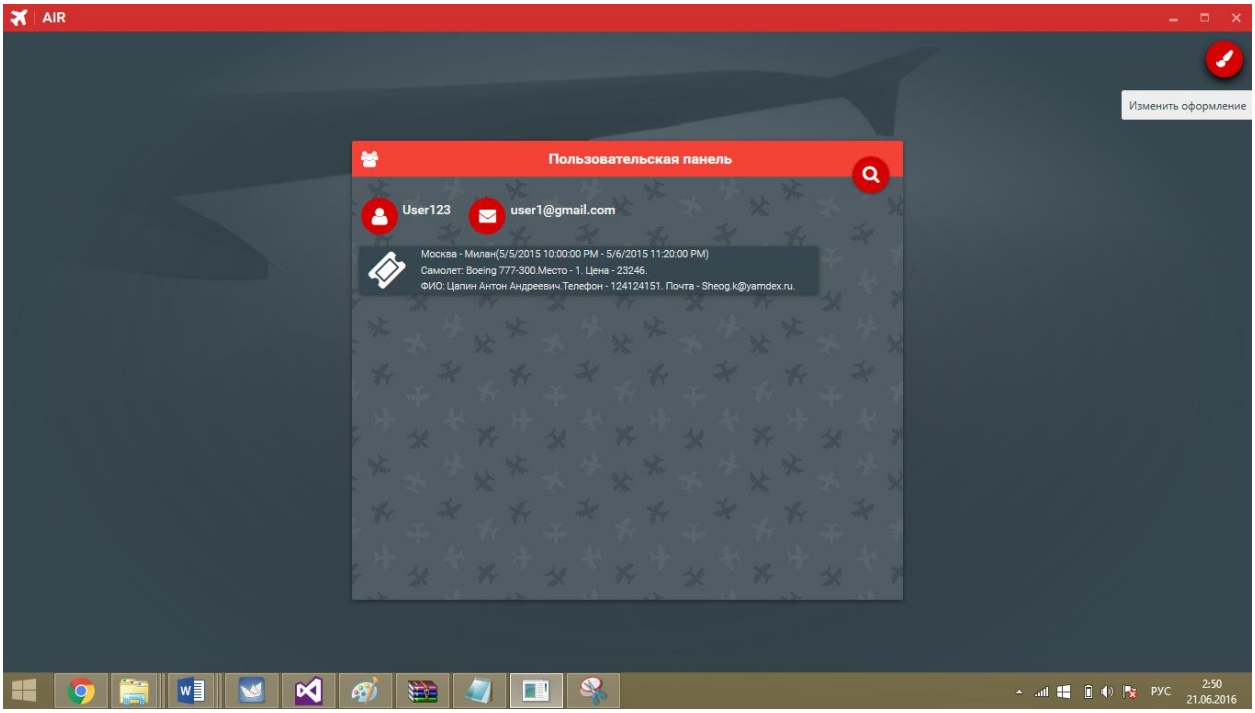

Рисунок Б.10 – Описание теста 6

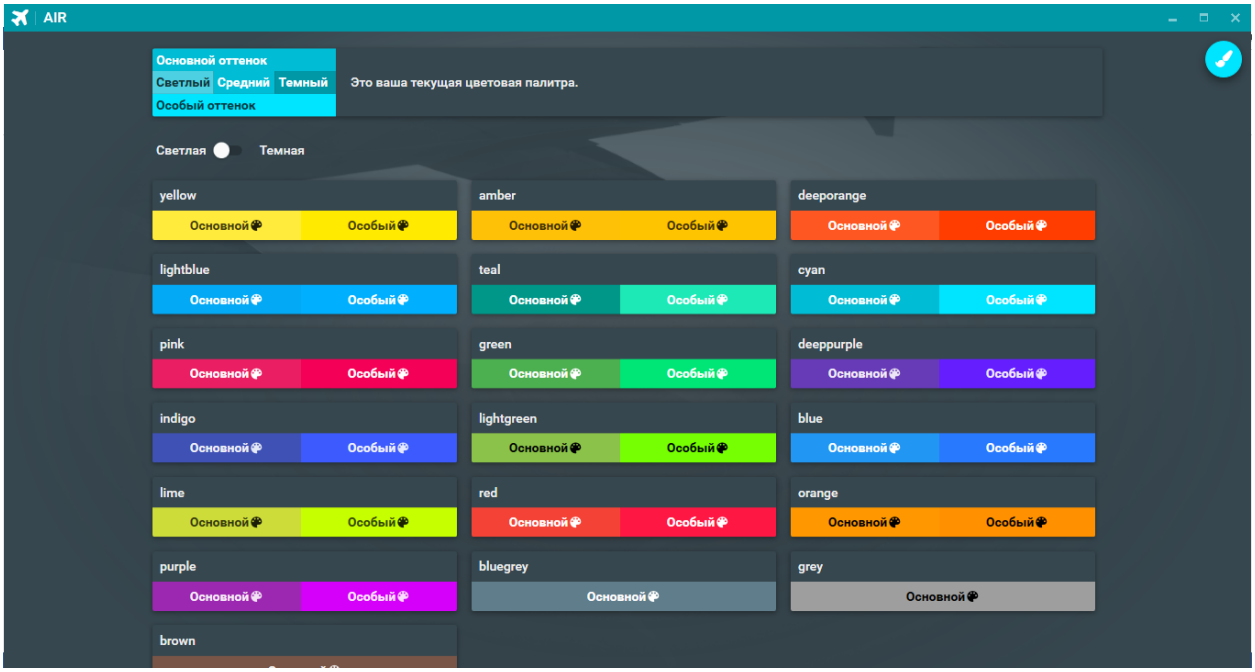

Рисунок Б.11 – Результат теста 6

ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ

Приложение В

(обязательное)

```
using Prism.Commands;
using System.Windows.Input;
using System.Data.Entity;
using System.Windows.Data;
using System;
using System.Collections.ObjectModel;
using System.Linq;
using Air.Views;
using System.Collections.Generic;
using System.Windows.Controls;
using System.Windows.Media.Imaging;
using System.IO;
using MaterialDesignThemes.Wpf;

namespace Air.ViewModels
{
    public class MainViewModel : PropertyChangedBase
    {
        private AirEntities1 _entities = new AirEntities1();

        private AuthorizationPanel _authorizationPanel = new
AuthorizationPanel();
        private RegistrationPanel _registrationPanel = new RegistrationPanel();
        private ReservationPanel _reservationPanel = new ReservationPanel();
        private MessageDialog _messageDialog = new MessageDialog();
        private MembersAreaPanel _membersAreaPanel = new MembersAreaPanel();
        private FlightsSearchPanel _flightsSearchPanel = new
FlightsSearchPanel();
        private ControlPanel _controlPanel = new ControlPanel();
        private ConfirmationDialog _confirmationDialog = new
ConfirmationDialog();
        private PaletteSelector _paletteSelector = new PaletteSelector();

        public BitmapSource BackgroundImage { get; set; }
        public UserControl CurrentUserControl { get; set; }
        private UserControl _previousUserControl;


        public CollectionViewSource AviaCompanyCollection { get; set; }
        public CollectionViewSource MarshrutCollection { get; set; }
        public CollectionViewSource TicketCollection { get; set; }
        public CollectionViewSource ReisCollection { get; set; }
        public CollectionViewSource PassagerCollection { get; set; }
        public CollectionViewSource UserCollection { get; set; }
        public CollectionViewSource OldCollection { get; set; }
        public CollectionViewSource YoungCollection { get; set; }

        public ObservableCollection<long> CompanyIds { get; set; }
        public ObservableCollection<long> MarshrutIds { get; set; }
        public ObservableCollection<long> TicketIds { get; set; }
        public ObservableCollection<long> ReisIds { get; set; }
        public ObservableCollection<long> PassagerIds { get; set; }
        public ObservableCollection<long> UserIds { get; set; }

        public List<Билеты> CurrentUserTickets { get; set; }
        public Пользователи CurrentUser { get; set; }
        public string FullName { get; set; }
        public int Phone { get; set; }
        public string EmailPeople { get; set; }
        public string PassportData { get; set; }
        public int Age { get; set; }
        public string FullNameAccompanying { get; set; }
        public int BirthCertificate { get; set; }
        public string PassportPointDelivery { get; set; }
```

```
        public bool IsYoung { get; set; }


        public string Login { get; set; }
        public string Password { get; set; }
        public string Email { get; set; }
        public string Message { get; set; }
        public object CurrentRow { get; set; }
        public Рейсы CurrentReis { get; set; }

        public string From { get; set; }
        public string Where { get; set; }
        public DateTime DepartureDate { get; set; }
        public DateTime ArrivalDate { get; set; }
        public DateTime DepartureTime { get; set; }
        public DateTime ArrivalTime { get; set; }
        public bool DepartureDateAndTimeIgnore { get; set; }
        public bool ArrivalDateAndTimeIgnore { get; set; }

        public MainViewModel()
        {
            List<string> imagesInDirectory = new
List<string>(Directory.GetFiles("Images"));
            Random random = new Random();

            string path = imagesInDirectory[random.Next(imagesInDirectory.Count
- 1)];

            BitmapImage bitmapImage = null;
            bitmapImage = new BitmapImage();
            bitmapImage.BeginInit();
            bitmapImage.UriSource = new Uri(path, UriKind.Relative);
            bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
            bitmapImage.CreateOptions = BitmapCreateOptions.DelayCreation;
            bitmapImage.DecodePixelWidth = 1200;
            bitmapImage.EndInit();
            bitmapImage.Freeze();

            FormatConvertedBitmap grayBitmap = new FormatConvertedBitmap();
            grayBitmap.BeginInit();
            grayBitmap.Source = bitmapImage;
            grayBitmap.DestinationFormat =
System.Windows.Media.PixelFormats.Gray8;
            grayBitmap.EndInit();
            grayBitmap.Freeze();

            BackgroundImage = grayBitmap;

            AviaCompanyCollection = new CollectionViewSource();
            MarshrutCollection = new CollectionViewSource();
            TicketCollection = new CollectionViewSource();
            PassagerCollection = new CollectionViewSource();
            ReisCollection = new CollectionViewSource();
            UserCollection = new CollectionViewSource();
            OldCollection = new CollectionViewSource();
            YoungCollection = new CollectionViewSource();

            CompanyIds = new ObservableCollection<long>();
            MarshrutIds = new ObservableCollection<long>();
            TicketIds = new ObservableCollection<long>();
            ReisIds = new ObservableCollection<long>();
            PassagerIds = new ObservableCollection<long>();
            UserIds = new ObservableCollection<long>();
```

```
            LoadDataFromDatabase();
            GetIdsTables();

            CurrentUserControl = _registrationPanel;
        }


        public ICommand MoveToUserPanelCommand { get { return new
DelegateCommand(() =>
        {
            CurrentUserTickets = _entities.Билеты.Local.Where(x =>
x?.Пассажиры?.Идентификатор_пользователя ==
CurrentUser?.Идентификатор_пользователя).ToList();
            CurrentUserControl = _membersAreaPanel;
        }); } }
        public ICommand MoveAutorizationPanelCommand { get { return new
DelegateCommand(() =>
        {
            CurrentUserControl = _authorizationPanel;
            Login = string.Empty;
            Password = string.Empty;
        }); } }
        public ICommand MoveRegistrationPanelCommand { get { return new
DelegateCommand(() =>
        {
            CurrentUserControl = _registrationPanel;
            Login = string.Empty;
            Password = string.Empty;
            Email = string.Empty;
        }); } }
        public ICommand MoveToPaletteSelectorCommand { get { return new
DelegateCommand(() =>
        {
            if (CurrentUserControl != _paletteSelector || _previousUserControl
== null)
            {
                _previousUserControl = CurrentUserControl;
                CurrentUserControl = _paletteSelector;
            }
            else
            {
                CurrentUserControl = _previousUserControl;
            }
        }); } }
        public ICommand MoveToReservationPanelCommand { get { return new
DelegateCommand(() => { CurrentUserControl = _reservationPanel; }); } }
        public ICommand MoveToSearchPanelCommand { get { return new
DelegateCommand(() => { CurrentUserControl = _flightsSearchPanel; }); } }
        public ICommand RegistrationCommand { get { return new
DelegateCommand(Registration); } }
        public ICommand AutorizationCommand { get { return new
DelegateCommand(Autorization); } }
        public ICommand ActiveFilterCommand { get { return new
DelegateCommand(ActiveFilter); } }
        public ICommand ReservationCommand { get { return new
DelegateCommand(Reserve); } }
        public ICommand OpenReservationPanelCommand { get { return new
DelegateCommand<object>(OpenReservationPanel); } }
        public ICommand LoadDataFromDatabaseCommand { get { return new
DelegateCommand(LoadDataFromDatabase); } }
        public ICommand SaveChangesDatabaseCommand { get { return new
DelegateCommand(SaveChangesDatabase); } }
```

```
        public ICommand RemoveRowFromDatabaseCommand { get { return new
DelegateCommand<object>(Delete); } }
        public CompositeCommand ConfirmationDialogCommand { get; set; }


        private async void OpenReservationPanel(object row)
        {
            ConfirmationDialogCommand = new CompositeCommand();

ConfirmationDialogCommand.RegisterCommand(DialogHost.CloseDialogCommand);

ConfirmationDialogCommand.RegisterCommand(MoveToReservationPanelCommand);

            CurrentReis = row as Рейсы;
            Message = string.Format("Хотите зарезервировать билет для маршрута
{0} - {1}(Отправление: {2}. Прибытие: {3}.)?  ",
                CurrentReis.Маршруты.Пункт_вылета,
CurrentReis.Маршруты.Пункт_прибытия, CurrentReis.Дата_и_время_вылета,
CurrentReis.Дата_и_время_прибытия);
            var view = new ConfirmationDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
        }

        private async void Reserve()
        {
            var error = false;
            Random rand = new Random();
            List<int> places = Enumerable.Range(1, 20).ToList(); ;

            try
            {
                if (!string.IsNullOrEmpty(FullName) &&
!string.IsNullOrEmpty(EmailPeople))
                {
                    Пассажиры newPassager = new Пассажиры() {
Идентификатор_пользователя = CurrentUser.Идентификатор_пользователя,
ФИО_пассажира = FullName, Электронная_почта = EmailPeople, Контактный_телефон =
Phone };

                    if (IsYoung)
                    {
                        if (Age >= 0 && BirthCertificate.ToString().Count() > 6
&& !string.IsNullOrEmpty(FullNameAccompanying))
                        {
                            _entities.Пассажиры.Add(newPassager);
                            _entities.SaveChanges();
                            var young = new Несовершеннолетний();
                            young.Идентификатор_пассажира =
newPassager.Идентификатор_пассажира;
                            young.Возраст = Age;
                            young.Номер_свидельства_о_рождении =
BirthCertificate;
                            young.ФИО_сопровождающего = FullNameAccompanying;
                            _entities.Несовершеннолетний.Add(young);
                            _entities.SaveChanges();
                        }
                        else
                        {
                            Message = string.Format(@"Произошла ошибка
резервирования билета.
                        Причины: неверное указан возраст; номер свительства о
рождении должен содержать больше 6 знаков; не заполнено ФИО сопровождающего.
");
                            error = true;
```

```
                    }
                }
                else
                {
                    if (!string.IsNullOrEmpty(PassportData) &&
!string.IsNullOrEmpty(PassportPointDelivery) && PassportData.Count() > 8)
                    {
                        _entities.Пассажиры.Add(newPassager);
                        _entities.SaveChanges();
                        var old = new Совершеннолетний();
                        old.Серия_и_номер_паспорта = PassportData;
                        old.Пункт_выдачи = PassportPointDelivery;
                        old.Идентификатор_пассажира =
newPassager.Идентификатор_пассажира;
                        _entities.Совершеннолетний.Add(old);
                        _entities.SaveChanges();
                    }
                    else
                    {
                        Message = string.Format(@"Произошла ошибка
резервирования билета.
                    Причины: не заполнены (Паспортные данные) или (Место
выдачи пасспорта); пассортные данные должны содержать больше 8 цифр. ");
                        error = true;
                    }

                }

                if (!error)
                {
                    var currentPlaces =
places.Except(_entities.Билеты.Local.Where(x => x.Рейсы.Идентификатор_рейса ==
CurrentReis.Идентификатор_рейса).Select(x => x.Место_в_салоне));
                    if (currentPlaces.Count() > 0) {

                        var ticket = new Билеты();
                        ticket.Зарезервирован = "Зарезервирован";
                        ticket.Идентификатор_пассажира =
newPassager.Идентификатор_пассажира;
                        ticket.Идентификатор_рейса =
CurrentReis.Идентификатор_рейса;
                        ticket.Самолет = "Boeing 777-300";
                        ticket.Цена = rand.Next(15000, 30000);
                        ticket.Место_в_салоне = currentPlaces.First();
                        _entities.Билеты.Add(ticket);
                        _entities.SaveChanges();
                        Message = string.Format(@"Успешно зарезервирован
билет. ФИО: {0}. Почта: {1}. Самолет: {2}. Место: {3}. Цена билета: {4} ",
                                newPassager.ФИО_пассажира,
newPassager.Электронная_почта, ticket.Самолет, ticket.Место_в_салоне,
ticket.Цена);
                    }
                    else
                    {
                        Message = string.Format(@"Произошла ошибка
резервирования билета.
                    Причины: все места заняты. ");
                    }
                }
            }
            else
                Message = string.Format(@"Произошла ошибка резервирования
билета.
```

```
                        Причины: не заполнены (ФИО пассажира) или (Почта
пассажира). ");
            }
            catch(Exception e)
            {
                Message = string.Format(@"Произошла непридвиденная ошибка
резервирования билета.
                        Исключение: {0}. ", e.Message);
            }

            var view = new MessageDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
            CurrentUserControl = _flightsSearchPanel;
        }

        private async void Delete(object row)
        {
            ConfirmationDialogCommand = new CompositeCommand();

ConfirmationDialogCommand.RegisterCommand(DialogHost.CloseDialogCommand);
            ConfirmationDialogCommand.RegisterCommand(new
DelegateCommand(DeleteRowFromDatabase));

            CurrentRow = row;
            Message = string.Format("Значение будет удалено из БД. ");
            var view = new ConfirmationDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
        }

        private async void DeleteRowFromDatabase()
        {
            if (CurrentRow == null) return;

            try
            {
                switch (CurrentRow.GetType().BaseType.Name)
                {
                    case nameof(Авиакомпании):

_entities.Авиакомпании.Remove((Авиакомпании)CurrentRow);
                        break;
                    case nameof(Билеты):
                        _entities.Билеты.Remove((Билеты)CurrentRow);
                        break;
                    case nameof(Маршруты):
                        _entities.Маршруты.Remove((Маршруты)CurrentRow);
                        break;
                    case nameof(Несовершеннолетний):

_entities.Несовершеннолетний.Remove((Несовершеннолетний)CurrentRow);
                        break;
                    case nameof(Пассажиры):
                        _entities.Пассажиры.Remove((Пассажиры)CurrentRow);
                        break;
                    case nameof(Пользователи):

_entities.Пользователи.Remove((Пользователи)CurrentRow);
                        break;
                    case nameof(Рейсы):
                        _entities.Рейсы.Remove((Рейсы)CurrentRow);
                        break;
                    case nameof(Совершеннолетний):

_entities.Совершеннолетний.Remove((Совершеннолетний)CurrentRow);
```

```
                    break;
                }

                Message = string.Format("Удаление произошло успешно. ");
                _entities.SaveChanges();
                GetIdsTables();
            }
            catch (Exception e)
            {
                Message = string.Format("Операция не выполнена. Исключение:
{0}. ", e.Message);
            }

            var view = new MessageDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
        }

        private void ActiveFilter()
        {
            ReisCollection.View.Filter = item =>
            {
                Рейсы reis = item as Рейсы;
                bool suitable = false;

                bool suitableDepartureDate = false,
                suitableArrivalDate = false,
                suitableDepartureTime = false,
                suitableArrivalTime = false,
                suitableFrom = false,
                suitableWhere = false;

                if (DepartureDateAndTimeIgnore) suitableDepartureDate = true;
                else if (reis.Дата_и_время_вылета.Day == DepartureDate.Day &&
reis.Дата_и_время_вылета.Month == DepartureDate.Month &&
reis.Дата_и_время_вылета.Year == DepartureDate.Year) suitableDepartureDate =
true;

                if (ArrivalDateAndTimeIgnore) suitableArrivalDate = true;
                else if (reis.Дата_и_время_прибытия.Day == ArrivalDate.Day &&
reis.Дата_и_время_прибытия.Month == ArrivalDate.Month &&
reis.Дата_и_время_прибытия.Year == ArrivalDate.Year) suitableArrivalDate =
true;

                if (DepartureDateAndTimeIgnore) suitableDepartureTime = true;
                else if (reis.Дата_и_время_вылета.Hour == DepartureTime.Hour &&
reis.Дата_и_время_вылета.Minute == DepartureTime.Minute) suitableDepartureTime
= true;

                if (ArrivalDateAndTimeIgnore) suitableArrivalTime = true;
                else if (reis.Дата_и_время_прибытия.Hour == ArrivalTime.Hour &&
reis.Дата_и_время_прибытия.Minute == ArrivalTime.Minute) suitableArrivalTime =
true;

                if (string.IsNullOrEmpty(From)) suitableFrom = true;
                else if (reis.Маршруты.Пункт_вылета == From) suitableFrom =
true;

                if (string.IsNullOrEmpty(Where)) suitableWhere = true;
                else if (reis.Маршруты.Пункт_прибытия == Where) suitableWhere =
true;

                suitable = suitableDepartureDate && suitableArrivalDate &&
suitableDepartureTime && suitableArrivalTime && suitableFrom &&suitableWhere;
```

```
                if (suitable)
                    return true;
                else
                    return false;
            };
        }

        private async void Autorization()
        {
            if
(((ObservableCollection<Пользователи>)UserCollection.Source).Any(x => x.Логин
== Login && x.Пароль == Password))
            {
                var user =
((ObservableCollection<Пользователи>)UserCollection.Source).First(x => x.Логин
== Login && x.Пароль == Password);
                CurrentUserControl = user.Тип_пользователя == "Пользователь" ?
(UserControl)_flightsSearchPanel : (UserControl)_controlPanel;
                Message = string.Format("Вход пользователя {0} в систему
выполнен. ", user.Логин);
                CurrentUser = user;
                ReisCollection.View.Filter = null;
            }
            else
            {
                Message = string.Format("Не удалось совершить вход в систему.
Пользователя с таким логином и паролем не существует в системе.");
            }
            var view = new MessageDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
        }

        private async void Registration()
        {
            if
(((ObservableCollection<Пользователи>)UserCollection.Source).Any(x => x.Логин
== Login) || ((ObservableCollection<Пользователи>)UserCollection.Source).Any(x
=> x.Электронная_почта_пользователя == Email))
                Message = string.Format("Не удалось зарегистрировать
пользователя. Такой логин или пароль уже есть в системе. ");
            else if (string.IsNullOrEmpty(Login) ||
string.IsNullOrEmpty(Password) || string.IsNullOrEmpty(Email))
                Message = string.Format("Не удалось зарегистрировать
пользователя. Одно из полей не заполнено. ");
            else if (Password.Count() < 6)
                Message = string.Format("Не удалось зарегистрировать
пользователя. Длина пароля должна привышать 6 символов. ");
            else
            {
                Пользователи user = new Пользователи() { Логин = Login, Пароль
= Password, Тип_пользователя = "Пользователь", Электронная_почта_пользователя =
Email };
                _entities.Пользователи.Add(user);
                _entities.SaveChanges();
                Message = string.Format("Регистрация пользователя {0} прошла
успешно. ", user.Логин);
                CurrentUserControl = _flightsSearchPanel;
                CurrentUser = user;
            }

            var view = new MessageDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
        }
```

```
        private async void SaveChangesDatabase()
        {
            try
            {
                _entities.SaveChanges();
                GetIdsTables();
                Message = string.Format("Изменения были успешно сохранены в БД.
");
            }
            catch (Exception e)
            {
                Message = string.Format("Не удалось сохранить изменения в БД.
Исключение: {0}. ", e.Message);
                YoungCollection.View.Refresh();
            }

            var view = new MessageDialog { DataContext = this };
            var result = await DialogHost.Show(view, "RootDialog");
        }

        private void GetIdsTables()
        {
            CompanyIds = new
ObservableCollection<long>(((ObservableCollection<Авиакомпании>)AviaCompanyColl
ection.Source).Select(x => x.Идентификатор_авиакомпании));
            MarshrutIds = new
ObservableCollection<long>(((ObservableCollection<Маршруты>)MarshrutCollection.
Source).Select(x => x.Идентификатор_маршрута));
            TicketIds = new
ObservableCollection<long>(((ObservableCollection<Билеты>)TicketCollection.Sour
ce).Select(x => x.Идентификатор_билета));
            PassagerIds = new
ObservableCollection<long>(((ObservableCollection<Пассажиры>)PassagerCollection
.Source).Select(x => x.Идентификатор_пассажира));
            ReisIds = new
ObservableCollection<long>(((ObservableCollection<Рейсы>)ReisCollection.Source)
.Select(x => x.Идентификатор_рейса));
            UserIds = new
ObservableCollection<long>(((ObservableCollection<Пользователи>)UserCollection.
Source).Select(x => x.Идентификатор_пользователя));
        }

        private void LoadDataFromDatabase()
        {
            _entities.Авиакомпании.Load();
            _entities.Маршруты.Load();
            _entities.Рейсы.Load();
            _entities.Билеты.Load();
            _entities.Пассажиры.Load();
            _entities.Пользователи.Load();
            _entities.Совершеннолетний.Load();
            _entities.Несовершеннолетний.Load();

            AviaCompanyCollection.Source = _entities.Авиакомпании.Local;
            MarshrutCollection.Source = _entities.Маршруты.Local;
            TicketCollection.Source = _entities.Билеты.Local;
            PassagerCollection.Source = _entities.Пассажиры.Local;
            ReisCollection.Source = _entities.Рейсы.Local;
            UserCollection.Source = _entities.Пользователи.Local;
            OldCollection.Source = _entities.Совершеннолетний.Local;
            YoungCollection.Source = _entities.Несовершеннолетний.Local;
        }
```

```
}
    }
```