МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Программные средства поддержки процессов резервирования и продажи авиабилетов

Студент _____ Староверов Илья Михайлович
(подпись, дата) (ФИО полностью)

Руководитель _____ Князев В.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

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

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

Пенза, 2023

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«__»_____20__ г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Староверов Илья Михайлович гр. 19ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства поддержки процессов резервирования и продажи авиабилетов

Тема утверждена приказом ПГУ №_____ от "____" _____ 20__

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать программные средства для мобильных устройств, обеспечивающие поддержку резервирования и продажи авиабилетов.
2) Среда выполнения приложения: операционная система Android 9 и выше
Среда выполнения сервера: операционная система Windows 10 и выше
Языковые средства разработки: C#, UML, SQL, Linq
Инструментальные средства разработки: Microsoft SQL Server 2019, Microsoft Visual Studio 2019, ArgoUML, Visual Paradigm, Microsoft Project 2007
Технологии разработки: RUP, ООП, MVC, Entity Framework
3) Основные функции мобильного приложения:
а) оформление бронирования и покупки авиабилета;
б) оформление выкупа и снятия брони;
в) оформление выкупа и снятия множества броней;
г) оформление возврата билета;
д) оформление возврата множества билетов;
е) изменение данных пассажира или покупателя.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	*Анализ предметной области постановка задачи*	10%	12.04.2023–16.04.2023	
2	*Проектирование*	20%	17.04.2023–26.04.2023	
3	*Реализация*	30%	27.04.2023–11.05.2023	
4	*Контроль качества программного обеспечения*	20%	12.05.2023–21.05.2023	
5	*Оформление пояснительной записки*	20%	22.05.2023–01.06.2023	

Дата выдачи задания "22" декабря 2022

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____ Л.Р. Фионова

(подпись, дата)

Реферат

Пояснительная записка содержит 137 листов, 78 рисунков, 30 таблиц, 32 использованных источника, 7 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, C#, RUP, SQL, LINQ, UML, MS SQL SERVER, РЕЗЕРВИРОВАНИЕ АВИАБИЛЕТА, ПРОДАЖА АВИАБИЛЕТА

Объектом разработки является мобильное приложение для поддержки процессов резервирования и продажи авиабилетов.

Цель работы – разработка мобильного приложения, позволяющего бронировать и покупать авиабилеты, выкупать и снимать брони, возвращать билеты, получать аналитическую информацию по рейсам и пассажирам.

Разработка серверной части приложения производилась на языке программирования C#. В качестве среды разработки использовалась Microsoft Visual Studio 2019. В качестве СУБД была использована СУБД MS SQL Server 2019.

Разработка мобильного приложения производилась на языке программирования C#. В качестве среды разработки использовалась Microsoft Visual Studio 2019.

Разработка проведена с использованием операционной системы Windows 10. Приложение предназначено для использования под управлением ОС Android версии 9.0 и выше, а его серверная часть работает под управлением операционной системы Windows 10 и выше.

Содержание

Введение 7
1 Анализ предметной области и требований 9
1.1 Анализ предметной области и постановка задачи 9
1.2 Анализ программных продуктов аналогичного назначения 10
1.2.1 «Аэрофлот» 10
1.2.2 «Aviasales» 12
1.2.3 «Яндекс.авиабилеты» 13
1.2.4 Сравнение аналогов 14
1.3 Анализ целевой аудитории 15
1.4 Анализ функциональных требований 17
1.5 Анализ нефункциональных требований 21
1.6 Планирование разработки и оценка бюджета 22
2 Проектирование программного обеспечения 33
2.1 Выбор технологий и инструментальных средств разработки 33
2.2 Проектирование архитектуры программного обеспечения 34
2.3 Проектирование базы данных 35
2.3.1 Концептуальное проектирование базы данных 35
2.3.2 Логическое проектирование базы данных 36
2.4 Проектирование серверной части приложения 38
2.5 Проектирование интерфейса пользователя 39
3 Реализация программного обеспечения 50
3.1 Реализация базы данных 50
3.2 Реализация серверной части приложения 54
3.3 Реализация мобильного приложения 56

3.4 Тестирование программного обеспечения 58
3.4.1 Тестирование серверной части приложения 58
3.4.1.1 Функциональное тестирование 58
3.4.1.2 Модульное тестирование 60
3.4.2 Тестирование клиентской части приложения 61
3.4.3 Метрики кода 62
3.4.3.1 Метрики размера программы 64
3.4.3.2 Метрики Холстеда 64
Заключение 67
Список использованных источников 68
Приложение А. Глоссарий 71
Приложение Б. Исходный код скриптов базы данных 73
Приложение В. Исходный код серверной части приложения 84
Приложение Г. Исходный код мобильного приложения 100
Приложение Д. Результаты тестирования серверной части приложения 120
Приложение Е. Исходный код функций тестирования 126
Приложение Ж. Результаты тестирования мобильного приложения 129

Введение

В настоящее время информационные ресурсы общества являются одним из самых главных факторов его развития. Они задают направления в различных сферах общества и оказывают огромное влияние на массовое сознание людей. В начале XXI века происходило развитие цифровых технологий. Ученые были уверены, что развитие технологий в будущем повлияет на все отношения между людьми, на экономику, на общество, на весь мир в целом. Так оно и произошло. Сегодня цифровая среда проникла во все сферы, как в область профессиональной деятельности, так и в повседневные практики.

Цифровизацию можно определить, как внедрение информационных технологий во все сферы деятельности в системах разного уровня, в том числе и в экономику [1].

Процесс внедрения информационных технологий во все сферы деятельности активно происходит и в России. Данный процесс поддерживается, в том числе, и на государственном уровне. Об этом может говорить программа «Цифровая экономика Российской Федерации», которая была утверждена 9 мая 2017 года правительством Российской Федерации. Данный документ определяет стратегию развития «…цифровой экономики Российской Федерации, в которой данные в цифровой форме являются ключевым фактором производства во всех сферах социально-экономической деятельности, что повышает конкурентоспособность страны, качество жизни граждан, обеспечивает экономический рост» [2].

Также 9 мая 2017 года был подписан Указ Президента Российской Федерации «О стратегии развития информационного общества в Российской Федерации на 2017 – 2030 годы». В нем были поставлены основные цели и задачи политики Российской Федерации в сфере информационных технологий, а также объявлен курс на формирование национальной цифровой экономики [3].

Одной из важнейших особенностей развития информационных технологий в нашей стране и во всем мире является повсеместное распространение мобильных устройств с доступом к Интернету. При этом

мобильный Интернет постепенно оттесняет на второй план традиционный доступ к сети с использованием стационарных компьютеров по причине своего существенно большего удобства. Так, например, как отмечено в государственной программе «Цифровая экономика», в 2016 году в нашей стране «на 100 человек приходилось 159,95 мобильного телефона и из 100 человек 71,29 человека использовали мобильный доступ к сети Интернет» [2]. В настоящее время эта тенденция лишь усиливается.

Сегодня авиаперевозки как в мире, так и в России, являются одним из самых быстро развивающихся способов передвижения. Люди пользуются ими для быстрого перемещения на большие расстояния по разным целям. Кто-то перемещается с целью туризма, кто-то отправляется в командировку, кто-то навещает своих родственников. В связи с этим авиаперелеты для многих людей являются очень важной частью их жизни.

Как было сказано выше, в нашей стране почти у каждого человека есть мобильный телефон, который имеет выход в сеть Интернет. Поэтому его использование является привычным для многих граждан. Это касается и работников аэропорта, которые часто используют различные девайсы в своей работе. Например, работникам кассы намного удобнее было бы выполнять свои обязанности именно через телефон, не используя компьютер и другие устройства.

Исходя из всего вышеперечисленного следует, что проект разработки программных средств поддержки процессов резервирования и продажи авиабилетов для мобильного устройства является актуальным и перспективным.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Разрабатываемое мобильное приложение должно обеспечить его пользователям удобный и простой доступ к покупке или бронированию авиабилета на определенный рейс. С помощью приложения пользователи смогут просто и удобно просматривать необходимую информацию по рейсу, а именно дату и время вылета и прилета, количество свободных мест, цену, тип самолета и другое. Пользователи смогут через свое мобильное устройство оформить бронирование или продажу билетов, что намного удобнее, нежели через компьютер. Необходимо лишь будет иметь выход в сеть Интернет. Для поиска списка рейсов необходимо будет указать направление авиаперелета и желаемую дату вылета, и приложение само подберет подходящие рейсы.

Помимо удобного нахождения подходящих рейсов приложение будет предоставлять функцию повторного использования данных пассажиров. То есть если пассажир уже был ранее учтен системой, то не нужно будет заполнять все его данные заново. Еще нужно будет предоставить возможность изменения данных пассажира или покупателя. Также в приложении необходимо будет реализовать функцию выкупа брони или ее снятия. Иногда пассажиры приобретают билеты или брони на рейс на несколько человек сразу, например, если осуществляют перелет семьей. В связи с этим также нужно реализовать функцию выкупа или снятия множества броней за одну операцию. Также пассажиры иногда отказываются от перелета даже после покупки билета. Поэтому нужно еще добавить функцию возврата билета и возврата множества билетов. Также должна присутствовать возможность просмотра пользователем информации о загруженности совершенных рейсов и количестве совершенных полетов пассажирами.

Таким образом, в приложении должны быть представлены следующие основные функции:

– аутентификация пользователя;

– оформление бронирования или покупки билета;
– оформление выкупа или снятия брони;
– оформление выкупа или снятия множества броней;
– оформление возврата билета;
– оформление возврата множества билетов;
– просмотр информации о загруженности совершенных рейсов;
– просмотр информации о количестве совершенных полетов пассажирами;
– изменение данных пассажира или покупателя.

1.2 Анализ программных продуктов аналогичного назначения

В настоящее время существует большое множество приложений, позволяющих бронировать и выкупать билеты на авиарейсы. Рассмотрим наиболее известные из них и проведем их сравнительный анализ.

1.2.1 «Аэрофлот»

Компания «Аэрофлот – Российские авиалинии» была основана в 1992 году для выполнения международных и внутренних пассажирских и грузовых воздушных перевозок. В настоящее время в группу «Аэрофлот» входят одноименная компания и несколько дочерних компаний, таких как «Донавиа», «Россия», «Победа» и другие [4]. Также у компании «Аэрофлот» имеется мобильное приложение. Главная страница данного приложения представлена на рисунке 1.

Через мобильное приложение «Аэрофлот» продает билеты на сумму более 1 млрд рублей в месяц. Общее число пользователей мобильного приложения «Аэрофлот» на всех платформах по состоянию на середину 2017 года составляет около 1,5 миллионов человек, которое продолжает расти [5]. При этом соотношение онлайн продаж билетов, включая продажи через мобильное приложение, на 2021 год составляло более 44 процентов [6].

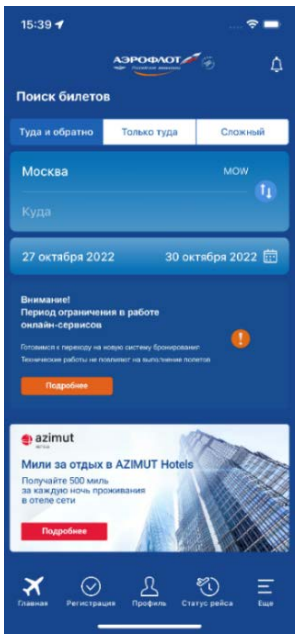

Рисунок 1 – Главная страница мобильного приложения «Аэрофлот»

На первой вкладке данного приложения расположена страница поиска авиабилетов на различные направления. Присутствует возможность выбора рейсов как в одну сторону, так и в обе сразу. Также имеется возможность поиска сложного маршрута, то есть с пересадками.

На второй вкладке имеется возможность пройти онлайн-регистрацию на рейс. В третьей вкладке можно посмотреть или изменить профиль пользователя. На четвертой вкладке можно узнать статус рейса, на который куплены или забронированы билеты. На последней вкладке представлена информация о приложении.

К достоинствам данного приложения можно отнести удобный интерфейс, позволяющий искать авиарейсы по разным направлениям. Также можно быстро проходить регистрацию на рейс и просматривать статус рейса. Еще через рассматриваемое приложение имеется возможность приобретать билеты на рейсы, проводящие другие авиакомпании.

К недостаткам приложения можно отнести медленную работу самого приложения при поиске рейсов, постоянную повторную авторизацию при выходе из приложения. Также при изменении личных данных пассажира купленный или забронированный билет можно будет только вернуть обратно,

так как изменения в нем произвести нельзя. Следовательно, пассажира не пустят на рейс.

1.2.2 «Aviasales»

Aviasales – одно из самых популярных в России приложений для поиска авиабилетов. Данная платформа является лишь агрегатором, то есть на ней нельзя купить билеты, а только найти их и перейти на сайты авиакомпаний. На данный момент в базе агрегатора 728 авиакомпаний и около 200 агентств. В день совершается около 1.4 млн поисков [7]. У данного сервиса также существует мобильное приложение, главный экран которого приведен на рисунке 2.

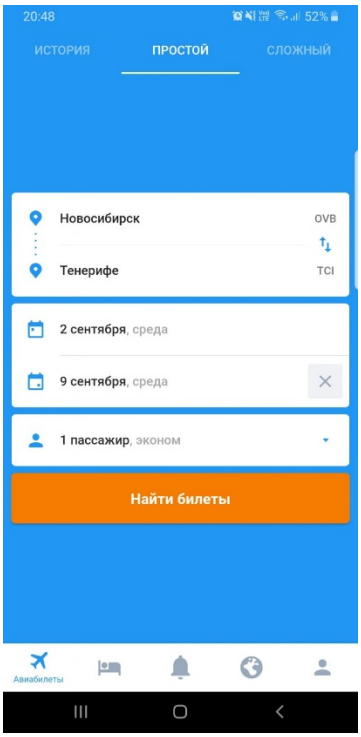

Рисунок 2 – Главная страница мобильного приложения «Aviasales»

В данном приложении имеется несколько вкладок. На первой вкладке можно осуществлять поиск авиарейсов (как простой, так и с пересадками) и просматривать историю предыдущих поисков. Также имеется возможность нахождения рейсов в обе стороны.

На второй вкладке также имеется возможность просмотра имеющихся в различных городах отелей и гостиниц и цен на них. На третьей вкладке можно включить или выключить напоминания о предстоящих рейсах и уведомления об

изменении рейсов, на которые куплены билеты. Четвертая вкладка служит для поиска различных путешествий. На последней вкладке находится информация о пользователе, которую можно изменять.

Достоинством данного приложения является просмотр всех рейсов от разных авиакомпаний. Также плюсом приложения является быстрый поиск рейсов от большого множества авиакомпаний, что позволяет пользователю сэкономить время и деньги.

Минусом является то, что именно данный сервис определяет, рейсы каких компании он будет показывать. На данный фактор пользователь не может никак повлиять. Также явным недостатком является то, что в приложении нельзя купить или забронировать билет: пользователю лишь предлагается перейти в приложение авиакомпании, которая выполняет выбранный рейс. Еще в приложении имеется платный контент, который позволяет получить лучшую обратную связь и другие преимущества.

1.2.3 «Яндекс.авиабилеты»

Яндекс.авиабилеты – продукт одной из самых известных поисковых систем «Яндекс». Яндекс.авиабилеты начал работать в 2009 году. При этом количество посетителей данного метапоисковика постоянно растет [8]. Данный сервис так же, как и Aviasales, позволяет только просматривать авиарейсы и осуществлять переход на приложения авиакомпаний, непосредственно осуществляющих рейсы. В 2016 году появилась мобильная версия данного продукта [9]. Главная страница приложения представлена на рисунке 3.

Как можно увидеть из рисунка 3, в приложении имеется 4 вкладки. На первой вкладке осуществляется поиск рейсов по направлению и дате вылета. Рейсы можно добавлять в избранное для отслеживания динамики цен. Избранные пользователем рейсы отображаются во второй вкладке. На третьей вкладке можно просмотреть информацию о рейсах, на которые были куплены или забронированы билеты. На четвертой вкладке можно изменять параметры

пользователя приложения, а также настраивать уведомления пассажиров об изменении в расписании.

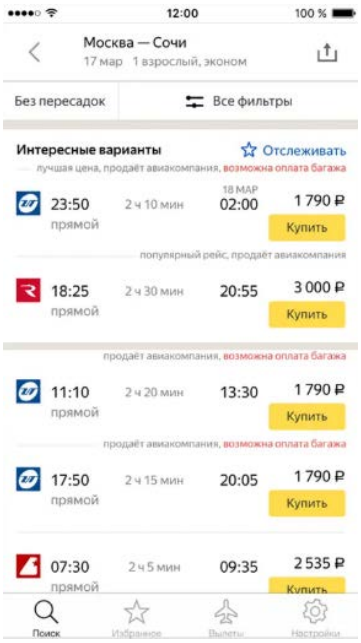

Рисунок 3 – Главная страница мобильного приложения «Яндекс.авиабилеты»

Плюсом приложения является довольно быстрый поиск рейсов от большого количества авиакомпаний.

Таким же минусом, как и у компании «Aviasales», является то, что осуществляется просмотр всех рейсов от разных авиакомпаний, на выбор которых пользователь никак не может повлиять. Вследствие этого, пользователь не может купить или забронировать билет из данного мобильного приложения. Также минусом в данном сервисе является то, что нельзя искать сложные (с пересадками) маршруты. Еще отсутствует функция автозаполнения документов, а также возможность возвратить купленный билет через приложение.

1.2.4 Сравнение аналогов

На данном этапе был произведен сравнительный анализ рассмотренных ранее мобильных приложений. В результате анализа была сформирована таблица 1.

Таблица 1 – Сравнение аналогов

Мобильное приложение	Бронирование билетов	Покупка билетов	Автозаполнение данных	Платный функционал	Функция возврата билета
Аэрофлот	Есть	Есть	Нет	Нет	Есть
Aviasales	Нет	Нет	Нет	Есть	Есть
Яндекс.авиабилеты	Нет	Нет	Нет	Нет	Нет
Разрабатываемое приложение	Есть	Есть	Есть	Нет	Есть

Из данного сравнения можно сделать вывод, что в большинстве из рассмотренных приложений недоступна функция бронирования или покупки билетов. Также ни в одном из приложений нет возможности автозаполнения данных. В одном из сервисов имеется платный функционал, а в другом невозможно из приложения осуществить возврат билета.

Таким образом, в результате анализа программных продуктов аналогичного назначения было принято решение о разработке собственного приложения, которое будет иметь весь необходимый пользователю функционал и будет лишено недостатков продуктов-аналогов.

1.3 Анализ целевой аудитории

Целевой аудиторией проекта являются кассиры аэропорта, использующие для своей работы мобильные устройства и обслуживающие людей, которым необходимо осуществить перелет из одного города в другой по различным целям, например, командировка, туризм, посещение родственников и другое. В настоящее время количество авиаперевозок имеет тренд на повышение. Этот вывод можно сделать на основании данных, предоставляемых сайтом Aviastat, который берет статистику из заключений Росстата и предоставляет ее в виде графиков [10, 11]. На рисунке 4 представлен график, на котором показаны выполненный пассажирооборот, тренд, а также прогноз в мире на пассажирские авиаперевозки за 1980 – 2025 годы.

Как видно из рисунка, пассажирооборот в интервале с 1980 по 2020 годы постоянно растет с небольшими падениями в 2001 году после трагедии 11 сентября и в 2008 году в связи с мировым финансовым кризисом.

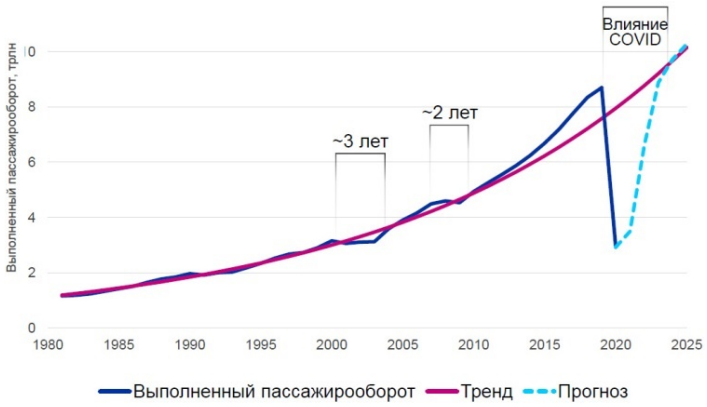

Рисунок 4 – Динамика пассажирооборота в 1980 – 2025 годах

В 2020 году произошло сильное падение пассажирооборота почти в 3 раза, связанное с пандемией COVID-19. Но, как можно увидеть на графике, уже к 2025 году по прогнозам пассажирооборот вернется к прежнему уровню и даже возрастет. Пассажирские перевозки в России по данным сайта Aviastat имеют схожий тренд.

На рисунке 5 представлены доли основных производителей смартфонов на мировом рынке за последние 2 года [12].

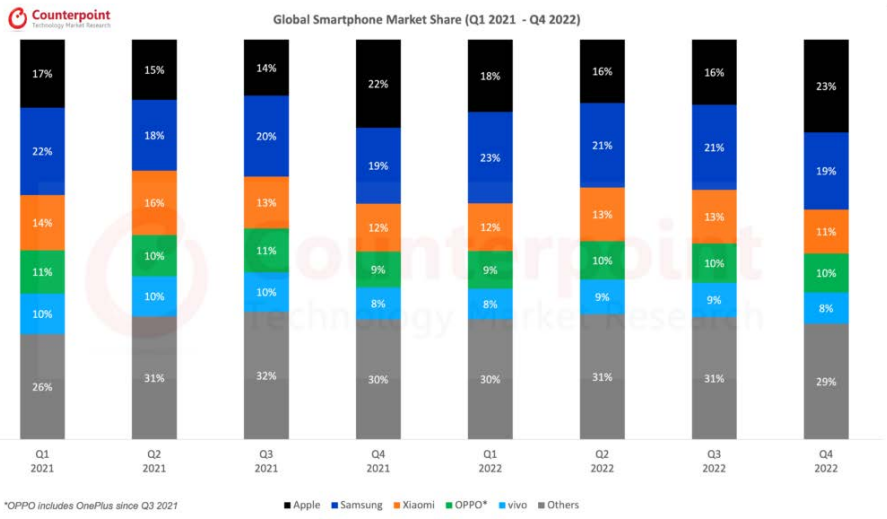

Рисунок 5 – Доли производителей смартфонов на мировом рынке

Из графика можно увидеть, что основную долю (около 70%) на мировом рынке производителей смартфонов занимаю смартфоны с операционной системой Android, такие как Samsung, Xiaomi, vivo и другие. Компания же Apple, использующая операционную систему iOS, хоть и увеличила количество своих

смартфонов на рынке до 23%, но даже не приблизилась ко своим конкурентам. Поэтому с целью увеличения количества пользователей, которые смогут использовать разрабатываемое мобильное приложение, было решено создавать его именно для операционной системы Android.

На рисунке 6 приведена статистика распространения разных версий системы Android [13].

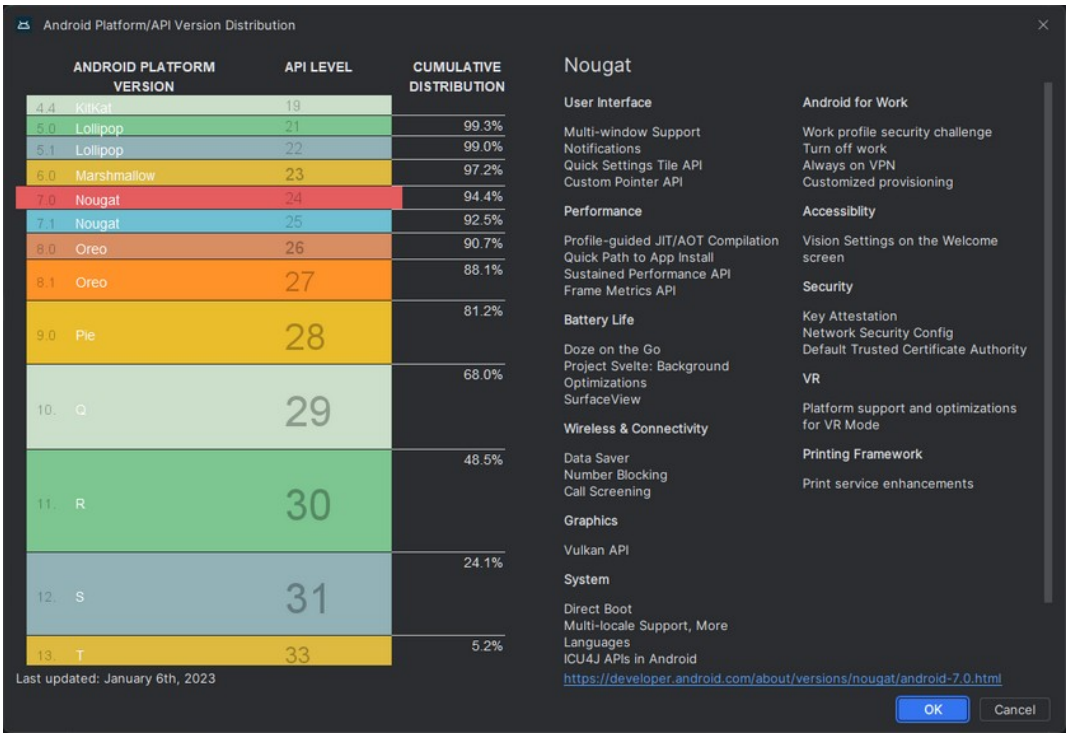

Рисунок 6 – Статистика распространения разных версий системы Android

Из рисунка можно заметить, что 81,2% пользователей операционной системы Android используют версию не ниже 9.0. Поэтому было принято решение разрабатывать приложение, которое будет работать на версии 9.0 или выше.

1.4 Анализ функциональных требований

Основным пользователем разрабатываемого приложения является кассир аэропорта. Он, работая с мобильным приложением, должен, в первую очередь, иметь возможность оформления бронирования и продажи билета и, при необходимости, заносить информацию о пассажирах и покупателе в информационную систему. Также ему должна быть предоставлена возможность

входа в систему и выхода из нее. Помимо этого, кассиру должна быть предоставлена возможность оформления как одиночного, так и множественного выкупа и снятия броней и возврата билетов.

На основе всего вышесказанного была создана диаграмма вариантов использования, представленная на рисунке 7 [14].

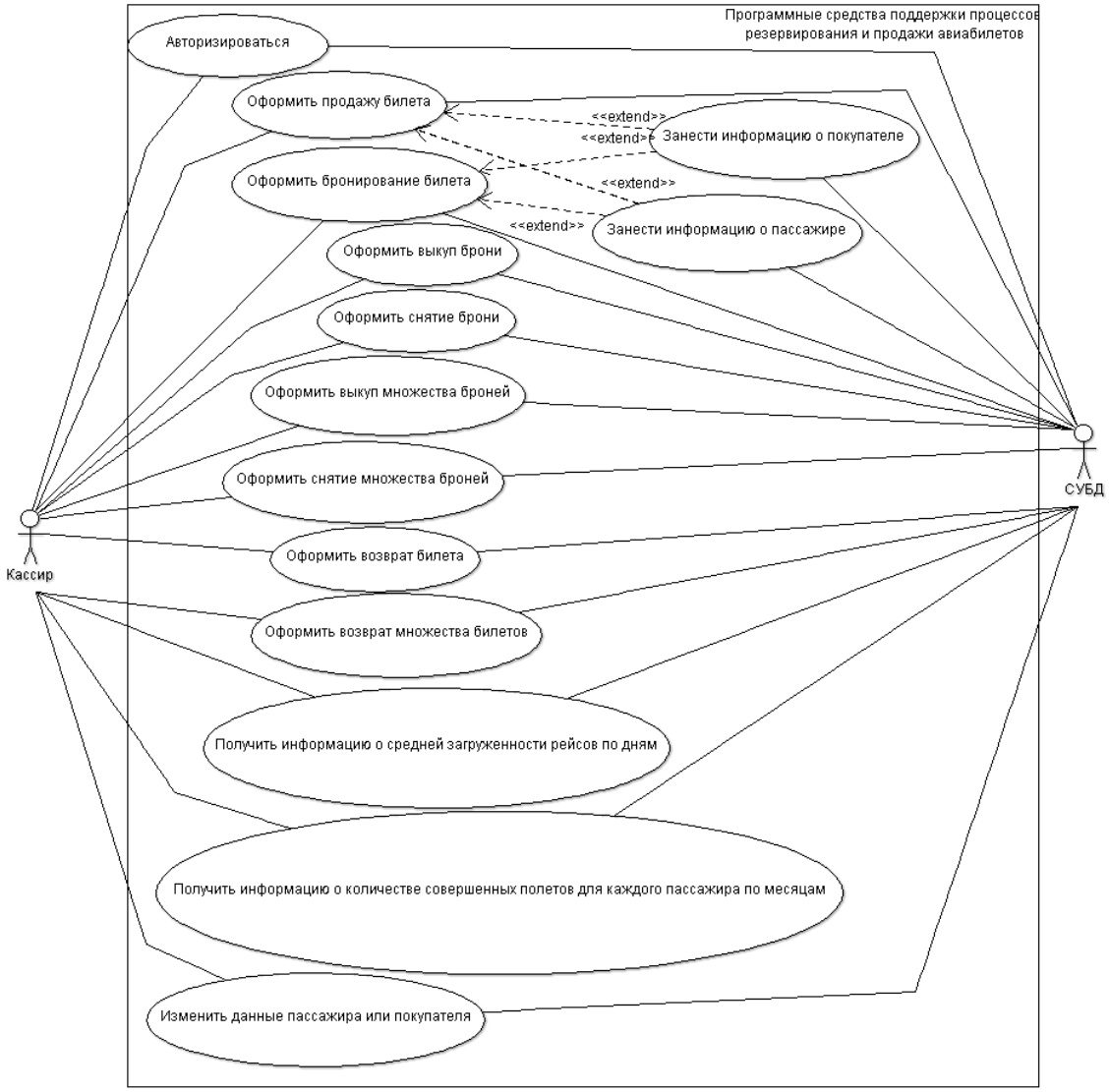

Рисунок 7 – Диаграмма вариантов использования мобильного приложения

Составим спецификации для основных вариантов использования: «Оформить продажу билета» (рисунок 8), «Оформить выкуп множества броней» (рисунок 9), «Оформить возврат билета» (рисунок 10).

Наименование: оформить продажу билета
ID: 1
Краткое описание: кассир продает билет на выбранный рейс
Действующие лица: кассир, СУБД
Предусловие: покупатель обратился к кассиру для покупки билета
Основной поток: 1. Кассир вводит дату рейса 2. Кассир вводит направление перелета 3. Система ищет информацию о рейсах по дате и направлению перелета 4. Система отображает найденную информацию о рейсах 5. Кассир выбирает рейс 6. ЕСЛИ информации о покупателе нет в БД 6.1. Кассир вводит информацию о покупателе 7. ИНАЧЕ 7.1. Кассир вводит номер паспорта покупателя 8. Система заносит информацию о покупателе в БД 9. ЕСЛИ информации о пассажирах нет в БД 9.1. Кассир вводит информацию о пассажирах 10. ИНАЧЕ 10.1. Кассир вводит номер паспорта пассажира 11. Кассир вводит номер места пассажира в самолете 12. Система заносит информацию о пассажире в БД 13. Кассир инициирует операцию продажи билета 14. Система заносит информацию о продаже в БД и сообщает об этом кассиру
Постусловие: билет был продан и занесен в БД

Рисунок 8 – Спецификация варианта использования «Оформить продажу билета»

Наименование: оформить выкуп множества броней
ID: 2
Краткое описание: кассир оформляет выкуп нескольких броней
Действующие лица: кассир, СУБД
Предусловие: покупатель обратился к кассиру для выкупа нескольких броней
Основной поток: 1. Кассир вводит номер рейса 2. Кассир вводит номер паспорта покупателя броней 3. Система ищет информацию по номеру рейса и номеру паспорта покупателя броней в БД 4. ЕСЛИ брони найдены 4.1. Система отображает найденную информацию кассиру 4.2. Кассир инициирует операцию выкупа броней 4.3. Система корректирует информацию в БД 5. ИНАЧЕ 5.1. Система сообщает кассиру, что броней с заданным параметрами в БД не найдены
Постусловие: множество броней было выкуплено

Рисунок 9 – Спецификация варианта использования «Оформить выкуп множества броней»

Наименование: оформить возврат билета
ID: 3
Краткое описание: кассир оформляет возврат билета
Действующие лица: кассир, СУБД
Предусловие: покупатель обратился к кассиру для возврата билета
Основной поток: 1. Кассир вводит номер билета 2. Система ищет информацию по номеру билета в БД 3. ЕСЛИ билет найден 3.1. Система отображает найденную информацию кассиру 3.2. Кассир инициирует операцию возврата билета 3.3. Система корректирует информацию в БД 4. ИНАЧЕ 4.1. Система сообщает кассиру, что билет с заданным номером в БД не найден
Постусловие: билет был возвращен

Рисунок 10 – Спецификация варианта использования «Оформить возврат билета»

Помимо спецификаций для некоторых прецедентов были составлены диаграмма деятельности и диаграмма последовательности. Для варианта использования «Авторизоваться» составлена диаграмма деятельности, представленная на рисунке 11.

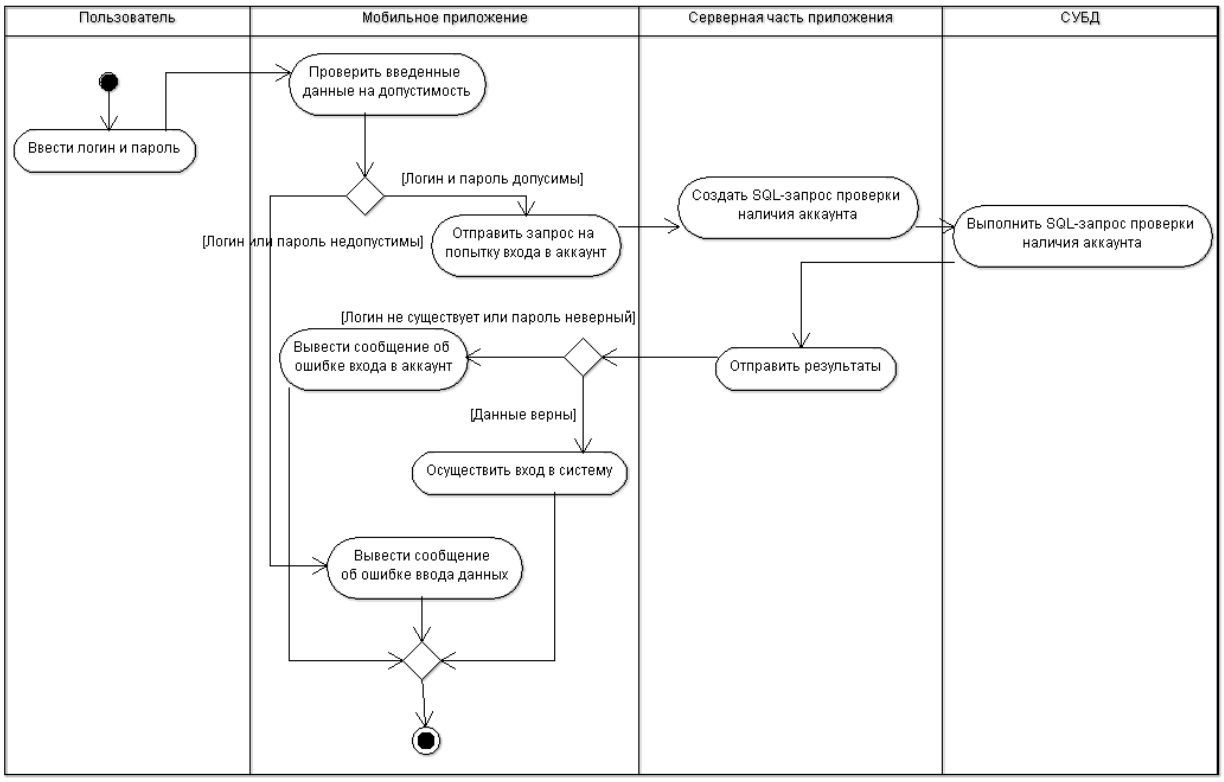

Рисунок 11 – Диаграмма деятельности варианта использования «Авторизоваться»

Для варианта использования «Занести информацию о пассажире» была составлена диаграмма последовательности, приведенная на рисунке 12.

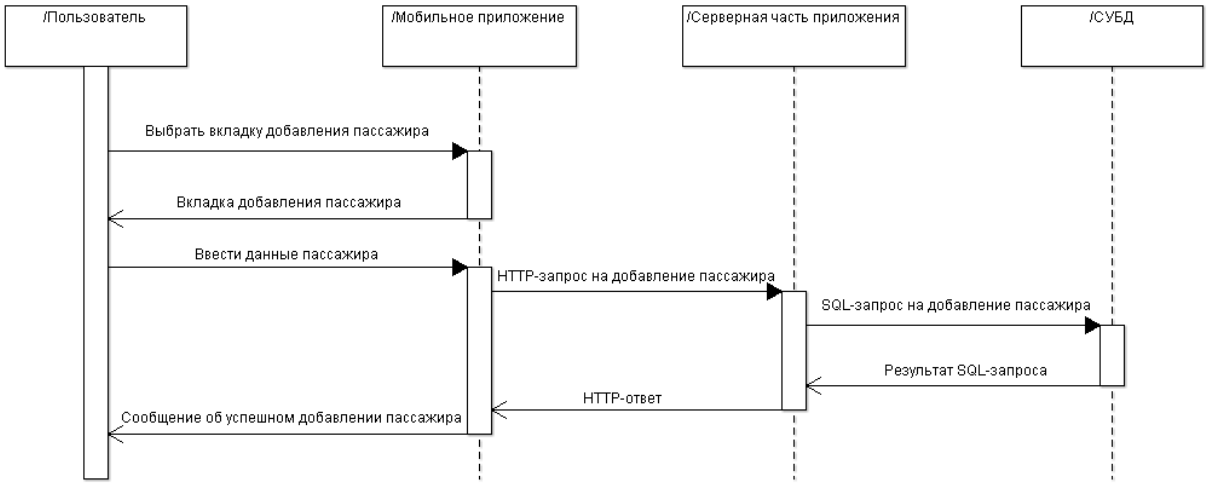

Рисунок 12 – Диаграмма последовательности варианта использования «Занести информацию о пассажире»

На данном этапе были определены основные функции разрабатываемого мобильного приложения. При этом для данного приложения были построены диаграммы вариантов использования, деятельности и последовательности, а также для некоторых вариантов использования были составлены спецификации.

1.5 Анализ нефункциональных требований

На данном этапе были определены нефункциональные требования к разрабатываемому программному обеспечению. Должны быть использованы такие технологии разработки, как RUP, ООП, MVC [15, 16].

Серверная часть приложения должна быть разработана на языке программирования C# на платформе ASP.NET в среде программирования Microsoft Visual Studio 2019 [17]. Данные на сервере должны храниться в базе данных, реализованной с использованием СУБД MS SQL Server 2019, запросы к базе данных должны осуществляться на языке SQL с использованием фреймворка Entity Framework Core [18]. Серверная часть должна работать под управлением операционной системы Windows 10 и выше.

Клиентская часть приложения должна быть реализована на языке программирования C# в среде программирования Microsoft Visual Studio 2019. Она должна представлять собой мобильное приложение с удобным и интуитивно понятным пользовательским интерфейсом. Клиентское приложение должно быть отлажено и должно успешно работать под управлением операционной системы Android 9.0 и выше.

1.6 Планирование разработки и оценка бюджета

Без правильного планирования невозможно качественно и в установленные сроки выполнить работы по любому проекту. Это касается и сферы информационных технологий. Планирование предполагает построение графика выполнения работ проекта таким образом, чтобы выполнить все поставленные задачи с приемлемым качеством за минимально возможное время с использованием минимума затрат [19 – 23].

Задачи, представляющие собой этапы работы над проектом, представлены в таблице 2.

Таблица 2 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ предметной области и постановка задачи	1 день	1
4	Анализ программных продуктов аналогичного назначения	3 дня	1
5	Анализ функциональных требований	2 дня	3;4
6	Анализ требований к интерфейсу	2 дня	5
7	Анализ целевой аудитории	1 день	5
8	Анализ требований завершен	-	6;7
9	Выбор решения	-	-
10	Анализ существующих технологий разработки ПО	2 дня	8
11	Выбор технологий разработки ПО	1 день	10
12	Выбор решения завершен	-	11
13	Проектирование программных средств	-	-
14	Проектирование архитектуры приложения	5 дней	12
15	Проектирование базы данных	3 дня	12
16	Проектирование серверной части приложения	3 дня	14;15
17	Проектирование мобильного приложения	4 дня	14
18	Проектирование интерфейса пользователя	2 дня	14
19	Проектирование завершено	-	16;17;18
20	Реализация программных средств	-	-

Продолжение таблицы 2

№	Название задачи	Длительность	Предшественники
21	Реализация информационной системы	5 дней	14;15
22	Реализация серверной части приложения	19 дней	19;21
23	Реализация мобильного приложения	21 день	19;21
24	Реализация завершена	-	22;23
25	Тестирование	-	-
26	Изучение основных видов тестирования	1 день	24
27	Функциональное тестирование	2 дня	26
28	Модульное тестирование	3 дня	26
29	Тестирование завершено	-	27;28
30	Составление и оформление документации	7 дней	29
31	Конец проекта	-	30

Ниже в таблице 3 представлено описание затрачиваемых ресурсов.

Таблица 3 – Перечень ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	400 руб/ч	600 руб/ч
Программист 2	Трудовой	350 руб/ч	550 руб/ч
Аналитик	Трудовой	350 руб/ч	550 руб/ч
Дизайнер	Трудовой	300 руб/ч	400 руб/ч
Тестировщик	Трудовой	300 руб/ч	400 руб/ч
Компьютер	Материальный	2000 руб	-
Бумага	Материальный	500 руб	-
Интернет	Затраты	-	-

В таблице 4 приведено распределение ресурсов по задачам.

Таблица 4 – Назначение ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ предметной области и постановка задачи	Аналитик Компьютер Интернет	40% 3 1500 руб
4	Анализ программных продуктов аналогичного назначения	Аналитик	50%
5	Анализ функциональных требований	Программист 1 Программист 2 Аналитик	100% 100% 50%
6	Анализ требований к интерфейсу	Дизайнер Аналитик	100% 30%
7	Анализ целевой аудитории	Аналитик	50%
8	Анализ требований завершен	-	-
9	Выбор решения	-	-
10	Анализ существующих технологий разработки ПО	Программист 2	30%
11	Выбор технологий разработки ПО	Программист 1 Программист 2	50% 50%

Продолжение таблицы 4

№	Название задачи	Ресурс	Единицы
12	Выбор решения завершен	-	-
13	Проектирование программных средств	-	-
14	Проектирование архитектуры приложения	Программист 1 Программист 2 Аналитик	100% 100% 50%
15	Проектирование базы данных	Программист 2 Аналитик	70% 50%
16	Проектирование серверной части приложения	Программист 1 Аналитик	100% 30%
17	Проектирование мобильного приложения	Программист 2 Аналитик	100% 30%
18	Проектирование интерфейса пользователя	Дизайнер Аналитик	60% 50%
19	Проектирование завершено	-	-
20	Реализация программных средств	-	-
21	Реализация информационной системы	Программист 2	100%
22	Реализация серверной части приложения	Программист 1	100%
23	Реализация мобильного приложения	Программист 2	100%
24	Реализация завершена	-	-
25	Тестирование	-	-
26	Изучение основных видов тестирования	Программист 2	30%
27	Функциональное тестирование	Тестировщик	70%
28	Модульное тестирование	Тестировщик Программист 2	70% 50%
29	Тестирование завершено	-	-
30	Составление и оформление документации	Аналитик Программист 1 Программист 2 Бумага	100% 50% 50% 2
31	Конец проекта	-	-

Таким образом, реализацией проекта будут заниматься программист 1, программист 2, аналитик и дизайнер, которые в процессе работы используют три компьютера и доступ в интернет. Для вычисления длительности реализации проекта и его критического пути был построен сетевой график, представленный на рисунке 13.

Теперь необходимо рассчитать критический путь проекта. Для этого необходимо высчитать раннее и позднее времена начала работ для всех задач проекта, и на основе данных сведений определить резерв времени для каждой из задач.

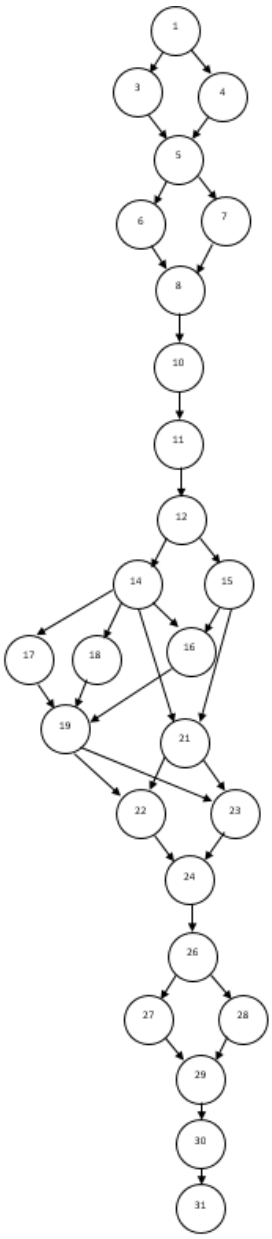

Рисунок 13 – Сетевой график проекта

В таблице 5 представлены расчеты раннего времени начала работ для его задач.

Таблица 5 – Расчеты раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
1	-	-	-	-	0
3	1	0	0	0+0	0
4	1	0	0	0+0	0
5	3 4	0 0	1 3	Max(0+1;0+3)	3
6	5	3	2	3+2	5
7	5	3	2	3+2	5

Продолжение таблицы 5

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
8	6 7	5 5	2 1	Max(5+2;5+1)	7
10	8	7	0	7+0	7
11	10	7	2	7+2	9
12	11	9	1	9+1	10
14	12	10	0	10+0	10
15	12	10	0	10+0	10
16	14 15	10 10	5 3	Max(10+5;10+3)	15
17	14	10	5	10+5	15
18	14	10	5	10+5	15
19	16 17 18	15 15 15	3 4 2	Max(15+3;15+4;15+2)	19
21	14 15	10 10	5 3	Max(10+5;10+3)	15
22	19 21	19 15	0 5	Max(19+0;15+5)	20
23	19 21	19 15	0 5	Max(19+0;15+5)	20
24	22 23	20 20	19 21	Max(20+19;20+21)	41
26	24	41	0	41+0	41
27	26	41	1	41+1	42
28	26	41	1	41+1	42
29	27 28	42 42	2 3	Max(42+2;42+3)	45
30	29	45	0	45+0	45
31	30	45	7	45+7	52

Также было рассчитано позднее время начала работ для задач проекта (таблица 6).

Таблица 6 – Расчеты позднего времени начала работ

Работа	Последователи			Вычисление	Позднее время начала
	№	Позднее время начала	Длительность		
31	-	-	-	-	52
30	31	52	7	52-7	45
29	30	45	0	45-0	45
28	29	45	3	45-3	42
27	29	45	2	45-2	43
26	27 28	43 42	1	Min(43-1;42-1)	41
24	26	41	0	41-0	41
23	24	41	21	41-21	20
22	24	41	19	41-19	22
21	22 23	22 20	5	Min(22-5;20-5)	15
19	22 23	22 20	0	Min(22-0;20-0)	20
18	19	20	2	20-2	18
17	19	20	4	20-4	16

Продолжение таблицы 6

Работа	Последователи			Вычисление	Позднее время начала
	№	Позднее время начала	Длительность		
16	19	20	3	20-3	17
15	16 21	17 15	3	Min(17-3;15-3)	12
14	16 17 18 21	17 16 18 15	5	Min(17-5;16-5;18-5;15-5)	10
12	14 15	10 12	0	Min(10-0;12-0)	10
11	12	10	1	10-1	9
10	11	9	2	9-2	7
8	10	7	0	7-0	7
7	8	7	1	7-1	6
6	8	7	2	7-2	5
5	6 7	5 6	2	Min(5-2;6-2)	3
4	5	3	3	3-3	0
3	5	3	1	3-1	2
1	3 4	2 0	0	Min(2-0;0-0)	0

По результатам нахождения раннего и позднего времени начала работ можно найти критический путь проекта. Вычисление критического пути представлено в таблице 7.

Таблица 7 – Вычисление критического пути проекта

Номер работы	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
3	0	2	2
4	0	0	0
5	3	3	0
6	5	5	0
7	5	6	1
8	7	7	0
10	7	7	0
11	9	9	0
12	10	10	0
14	10	10	0
15	10	12	2
16	15	17	2
17	15	16	1
18	15	18	3
19	19	20	1
21	15	15	0
22	20	22	2
23	20	20	0
24	41	41	0
26	41	41	0

Продолжение таблицы 7

Номер работы	Раннее время начала	Позднее время начала	Резерв времени
27	42	43	1
28	42	42	0
29	45	45	0
30	45	45	0
31	52	52	0

На основе полученных данных можно сделать вывод, что критический путь проходит через работы с номерами 1, 4, 5, 6, 8, 10, 11, 12 14, 21, 23, 24, 26, 28, 29, 30, 31, а общее время выполнения проекта составляет 52 дня. Граф проекта, где пунктирными линиями указан критический путь, приведен на рисунке 14.

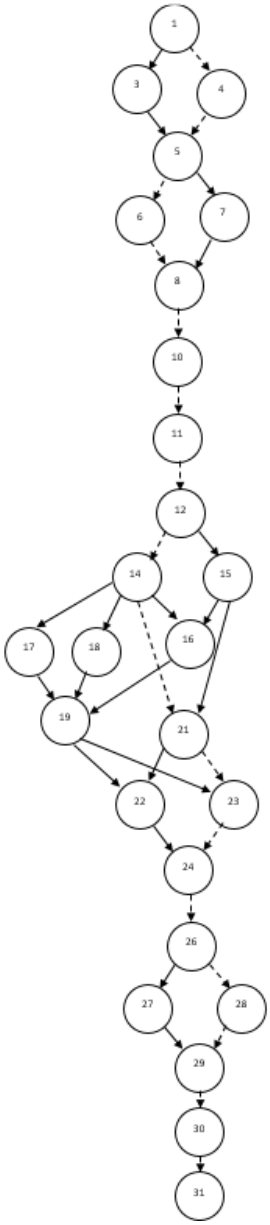

Рисунок 14 – Критический путь проекта

Теперь проверим правильность расчетов продолжительности проекта и вычислим величину его бюджета. Для этого воспользуемся системой управления проектами Microsoft Project 2007, которая позволяет автоматически произвести все необходимые вычисления [24]. Список составленных задач в системе Microsoft Project представлен на рисунке 15. В качестве времени начала проекта была выбрана дата 10.03.2023.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало проекта	0 дней	Пт 10.03.23	Пт 10.03.23		
2	**Анализ требований**	**7 дней**	**Пт 10.03.23**	**Пн 20.03.23**		
3	Анализ предметной области и постановка задачи	1 день	Пт 10.03.23	Пт 10.03.23	1	Аналитик[40%];Компьютер[3];Интернет[1 500,00 ₽]
4	Анализ программных продуктов аналогичного назначения	3 дней	Пт 10.03.23	Вт 14.03.23	1	Аналитик[50%]
5	Анализ функциональных требований	2 дней	Ср 15.03.23	Чт 16.03.23	3;4	Программист 1;Программист 2;Аналитик[50%]
6	Анализ требований к интерфейсу	2 дней	Пт 17.03.23	Пн 20.03.23	5	Дизайнер;Аналитик[30%]
7	Анализ целевой аудитории	1 день	Пт 17.03.23	Пт 17.03.23	5	Аналитик[50%]
8	Анализ требований завершен	0 дней	Пн 20.03.23	Пн 20.03.23	6;7	
9	**Выбор решения**	**3 дней**	**Вт 21.03.23**	**Чт 23.03.23**		
10	Анализ существующих технологий разработки ПО	2 дней	Вт 21.03.23	Ср 22.03.23	8	Программист 2[30%]
11	Выбор технологий разработки ПО	1 день	Чт 23.03.23	Чт 23.03.23	10	Программист 1[50%];Программист 2[50%]
12	Выбор решения завершен	0 дней	Чт 23.03.23	Чт 23.03.23	11	
13	**Проектирование программных средств**	**9 дней**	**Пт 24.03.23**	**Ср 05.04.23**		
14	Проектирование архитектуры приложения	5 дней	Пт 24.03.23	Чт 30.03.23	12	Программист 1;Программист 2;Аналитик[50%]
15	Проектирование базы данных	3 дней	Пт 24.03.23	Вт 28.03.23	12	Программист 2[70%];Аналитик[50%]
16	Проектирование серверной части приложения	3 дней	Пт 31.03.23	Вт 04.04.23	14;15	Программист 1;Аналитик[30%]
17	Проектирование мобильного приложения	4 дней	Пт 31.03.23	Ср 05.04.23	14	Программист 2;Аналитик[30%]
18	Проектирование интерфейса пользователя	2 дней	Пт 31.03.23	Пн 03.04.23	14	Дизайнер[60%];Аналитик[50%]
19	Проектирование завершено	0 дней	Ср 05.04.23	Ср 05.04.23	16;17;18	
20	**Реализация программных средств**	**26 дней**	**Пт 31.03.23**	**Ср 10.05.23**		
21	Реализация информационной системы	5 дней	Пт 31.03.23	Чт 06.04.23	14;15	Программист 2
22	Реализация серверной части приложения	19 дней	Пт 07.04.23	Чт 04.05.23	19;21	Программист 1
23	Реализация мобильного приложения	21 дней	Пт 07.04.23	Ср 10.05.23	19;21	Программист 2
24	Реализация завершена	0 дней	Ср 10.05.23	Ср 10.05.23	22;23	
25	**Тестирование**	**4 дней**	**Чт 11.05.23**	**Вт 16.05.23**		
26	Изучение основных видов тестирования	1 день	Чт 11.05.23	Чт 11.05.23	24	Программист 2[30%]
27	Функциональное тестирование	2 дней	Пт 12.05.23	Пн 15.05.23	26	Тестировщик[70%]
28	Модульное тестирование	3 дней	Пт 12.05.23	Вт 16.05.23	26	Тестировщик[70%];Программист 2[50%]
29	Тестирование завершено	0 дней	Вт 16.05.23	Вт 16.05.23	27;28	
30	Составление и оформление документации	7 дней	Ср 17.05.23	Чт 25.05.23	29	Аналитик;Программист 1[50%];Программист 2[50%];Бумага[2]
31	Конец проекта	0 дней	Чт 25.05.23	Чт 25.05.23	30	

Рисунок 15 – Список задач проекта в системе Microsoft Project

Построенная системой Microsoft Project диаграмма Ганта приведена на рисунке 16.

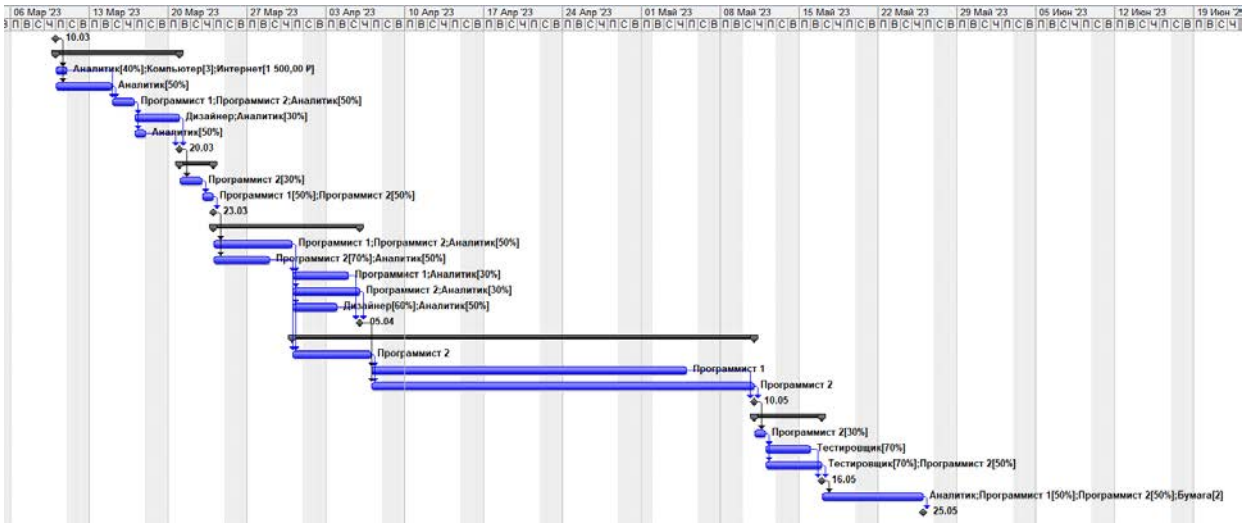

Рисунок 16 – Диаграмма Ганта

График загруженности ресурса «Программист 1» представлен на рисунке 17.

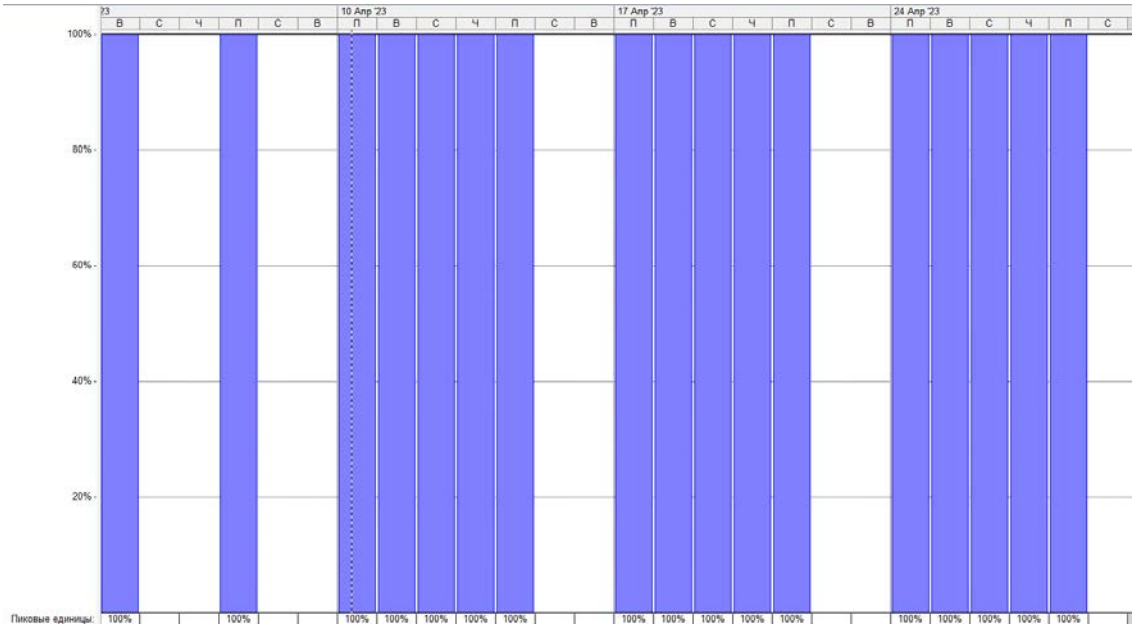

Рисунок 17 – График загруженности ресурса «Программист 1»

График загруженности ресурса «Программист 2» представлен на рисунке 18.

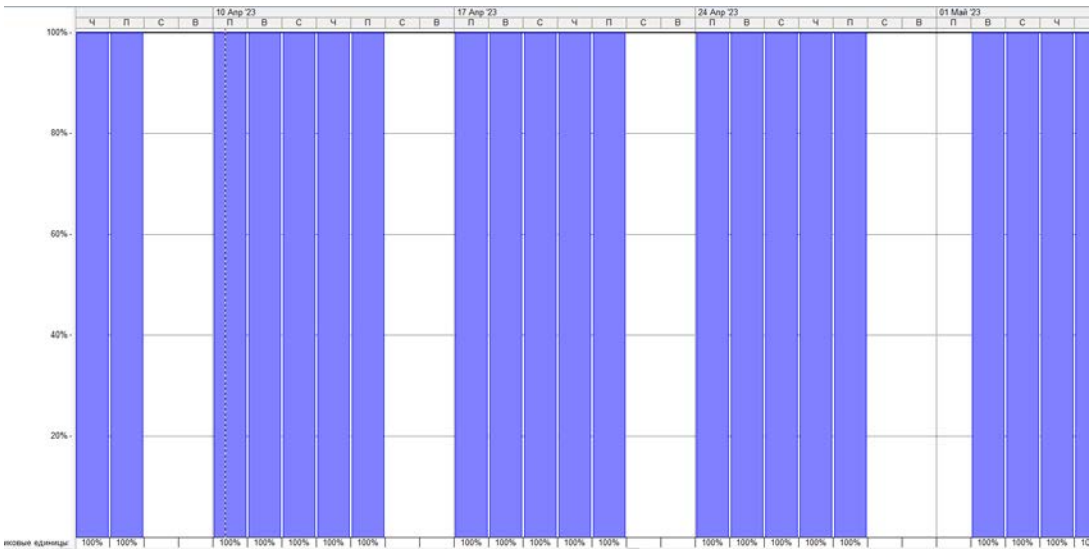

Рисунок 18 – График загруженности ресурса «Программист 2»

График загруженности ресурса «Аналитик» представлен на рисунке 19.

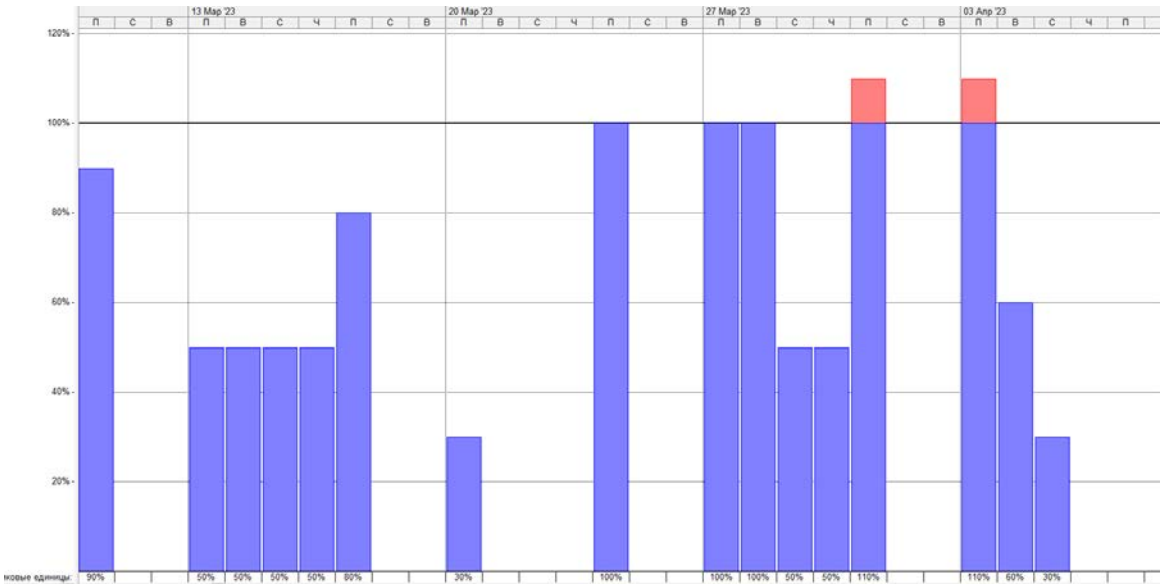

Рисунок 19 – График загруженности ресурса «Аналитик»

График загруженности ресурса «Дизайнер» представлен на рисунке 20.

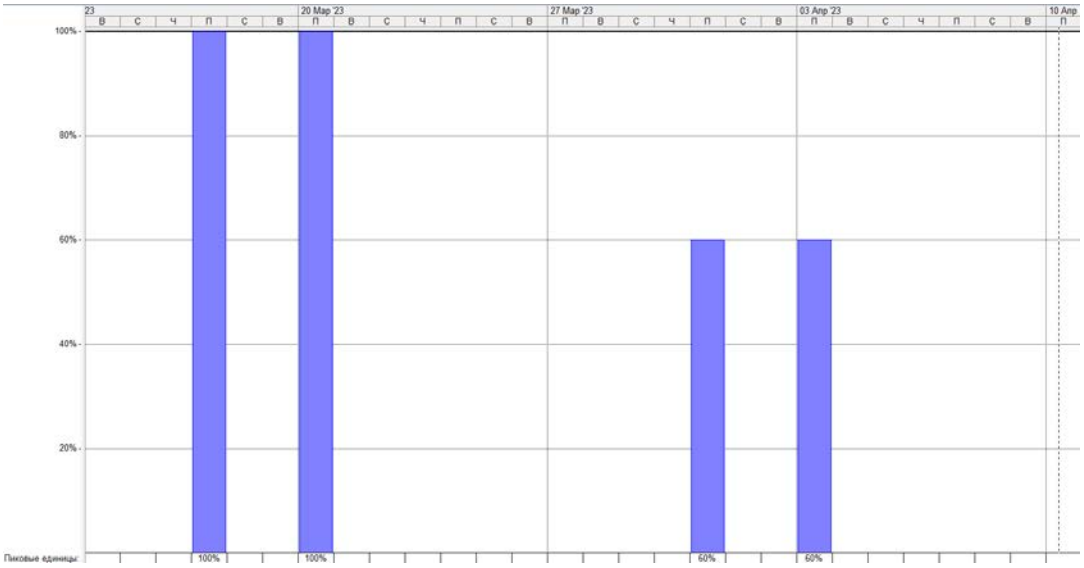

Рисунок 20 – График загруженности ресурса «Дизайнер»

График загруженности ресурса «Тестировщик» представлен на рисунке 21.

На рисунке 22 представлен обзорный отчет проекта, составленный системой Microsoft Project. Из рисунка видно, что общая длительность проекта составляет 52 дня (совпадает со значением, вычисленным вручную), а предполагаемые затраты – 303760 рублей.

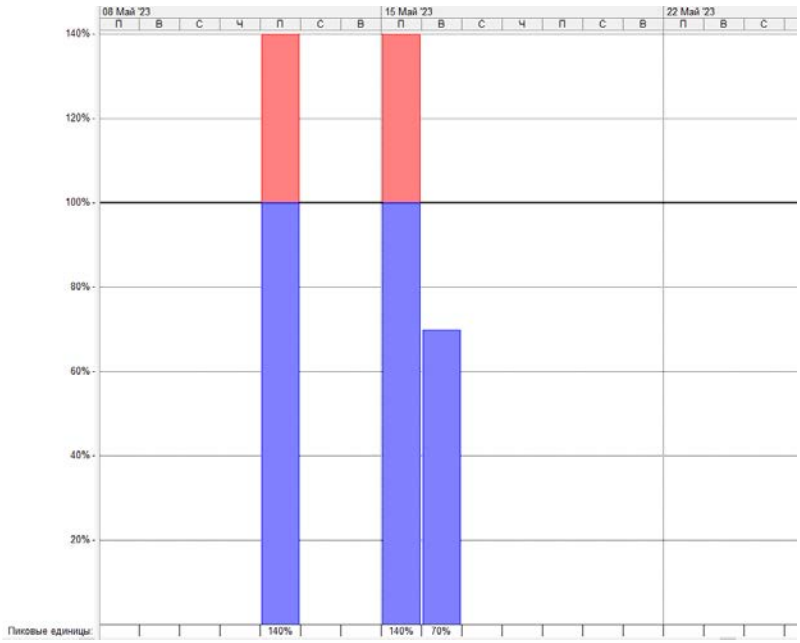

Рисунок 21 – График загруженности ресурса «Тестировщик»

Даты			
Начало:	Пт 10.03.23	Окончание:	Чт 25.05.23
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней

Длительность			
Текущий план:	52 дней	Оставшаяся:	52 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	52 дней	Процент завершени:	0%

Трудозатраты			
Текущий план:	826,4 ч	Оставшиеся:	826,4 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	826,4 ч	Процент завершени:	0%

Затраты			
Текущий план:	303 760,00 ₽	Оставшиеся:	303 760,00 ₽
Базовый план:	0,00 ₽	Фактические:	0,00 ₽
Отклонение:	303 760,00 ₽		

Состояние задачи	
Еще не начатые задачи	31
Выполняющиеся задачи	0
Завершенные задачи:	0
Итого задач:	31

Состояние ресурса	
Трудовые ресурсы:	3
Трудовые ресурсы с прев. дост	3
Материальные ресурсы:	2
Итого ресурсов:	8

Рисунок 22 – Обзорный отчет проекта

На основе произведенных оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности.

2 Проектирование программного обеспечения

2.1 Выбор технологий и инструментальных средств разработки

В результате проектирования программного обеспечения было решено использовать одну из самых известных методологий разработки Rational Unified Process (RUP). Данная модель описывает общий процесс, которым должна руководствоваться проектная команда, чтобы создать программный продукт. В первую очередь, данная методология предполагает итеративную разработку. То есть вся разработка делится на четыре основных этапа, каждый из которых включает в себя одну или несколько итераций, продолжающихся от 2 до 6 недель [15]:

– начальная стадия – определяются границы проекта, требования, приводится экономическое обоснование;

– стадия уточнения – производится уточнение требований и обновление экономического обоснования, проектирование архитектуры;

– стадия построения – реализуется основная часть функциональности продукта;

– стадия внедрения – продукт дорабатывается до окончательной версии и передается заказчику, производится обучение пользователей.

Технология RUP была выбрана, так как обладает рядом преимуществ относительно других методологий. Во-первых, RUP обеспечивает гибкость процесса разработки. То есть в проект можно относительно быстро вносить корректировки в случае выявления ошибок на предыдущих стадиях или изменений требований заказчиком. Также в данной методологии производится оценка рисков уже на первых стадиях разработки, поэтому в случае его экономической нецелесообразности можно завершить проект еще на начальной стадии. Помимо этого, в отличии от других методологий, например, eXtreme Programming (XP), постоянное взаимодействие между заказчиком не требуется.

Технология RUP для анализа требований, предполагает использование прецедентов использования (use case). В связи с этим было решено использовать

диаграммы на языке моделирования UML. Для построения диаграмм использовалось средство UML-моделирования ArgoUML.

В качестве системы управления базами данных была выбрана Microsoft SQL Server 2019, являющаяся одной из самых популярных СУБД [25]. Серверная часть взаимодействует с базой данных с помощью объектно-реляционного модуля Entity Framework Core 6. Данный модуль позволяет управлять сущностями базы данных через их проекцию – классы C#, описанные с использованием специальных атрибутов, а также предоставляет возможность выполнения SQL запросов через язык запросов Linq [26].

Серверная часть приложения будет реализована при помощи платформы ASP.NET, которая предоставляет широкий функционал для обработки HTTP-запросов и методы управления поведением сервера. Сервер будет работать под управлением операционной системы Windows 10.

Клиентская часть приложения будет реализована при помощи платформы Xamarin, предназначенной для построения современных производительных приложений для iOS, Android и Windows [27]. Целевой платформой в данной разработке будет являться Android.

2.2 Проектирование архитектуры программного обеспечения

В данной работе было принято решение использовать трехуровневую архитектуру «клиент-сервер». Ее схема представлена на рисунке 23.

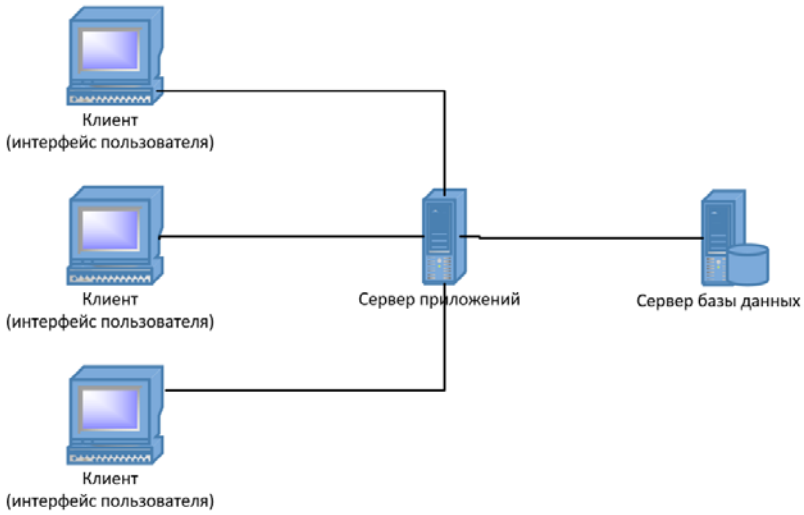

Рисунок 23 – Трехуровневая архитектура «клиент-сервер»

В данной архитектуре на сервере базы данных осуществляется управлением данными. Сервер приложений реализует функционал всего проекта и отвечает за взаимодействие с базой данных. Клиентская же часть, в первую очередь, обеспечивает интерфейс, позволяющий просматривать информацию, полученную с сервера приложений, и взаимодействовать с данными путем удаленного обращения к серверу. В данной работе в качестве клиентов будут выступать мобильные устройства.

Основным преимуществом трехуровневой архитектуры «клиент-сервер» является то, что каждый уровень в ней независим и может разрабатываться параллельно с разработкой других уровней. Также клиент в данной архитектуре не работает напрямую с базой данных, в связи с чем повышается безопасность данных.

2.3 Проектирование базы данных

2.3.1 Концептуальное проектирование базы данных

На основе анализа предметной области была построена концептуальная модель базы данных. Графическое представление концептуальной модели БД в виде диаграммы «сущность-связь» представлено на рисунке 24.

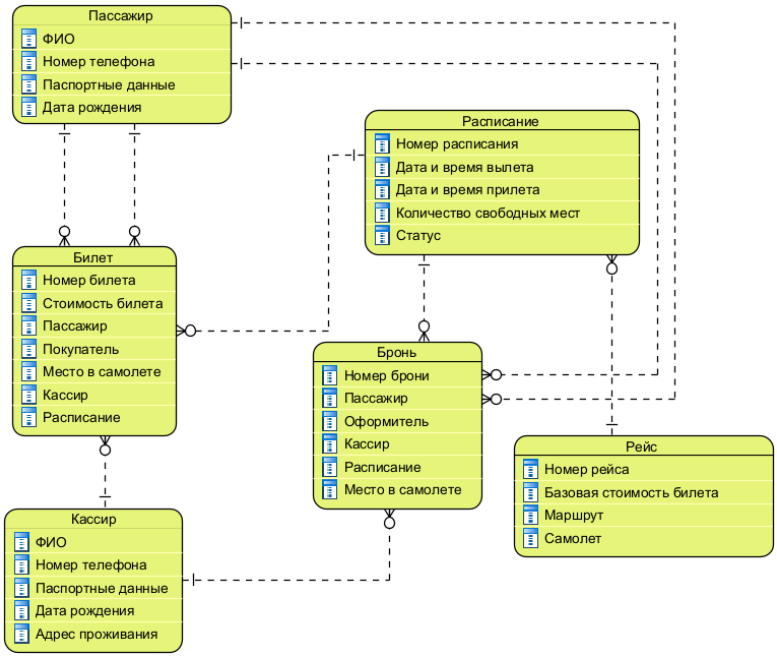

Рисунок 24 – Графическое представление концептуальной модели базы данных

Сущности концептуальной модели данных представлены в таблице 8.

Таблица 8 – Сущности концептуальной модели

№	Название	Описание	Количество экземпляров	Режим обновления
1	Пассажир	Список пассажиров и покупателей	10000	При покупке билетов или оформлении брони клиентом, информация о котором отсутствует в системе
2	Рейс	Список рейсов	500	При добавлении нового рейса
3	Расписание	Список расписаний рейсов	10000	При добавлении нового расписания рейса
4	Кассир	Список кассиров	20	При принятии на работу нового кассира
5	Билет	Список билетов	300000	При продаже билета
6	Бронь	Список броней	300000	При бронировании билета

В таблице 9 представлено описание связей между сущностями.

Таблица 9 – Описание связей между сущностями концептуальной модели

№	Связь	Описание	Мощность	Обязательность
1	Пассажир – Билет	Определяет билеты пассажира	1:M	Да
2	Пассажир – Билет	Определяет купленные билеты покупателя	1:M	Да
3	Пассажир – Бронь	Определяет брони пассажира	1:M	Да
4	Пассажир – Бронь	Определяет оформленные брони оформителя	1:M	Да
5	Рейс – Расписание	Определяет расписания для рейса	1:M	Нет
6	Расписание – Билет	Определяет билеты, проданные на расписание	1:M	Нет
7	Расписание – Бронь	Определяет брони, оформленные на расписание	1:M	Нет
8	Кассир – Билет	Определяет билеты, проданные кассиром	1:M	Нет
9	Кассир – Бронь	Определяет брони, оформленные кассиром	1:M	Нет

Таким образом, были спроектированы основные сущности с атрибутами и связями между ними, без включения атрибутов первичных и внешних ключей.

2.3.2 Логическое проектирование базы данных

На основании концептуальной модели была спроектирована логическая модель базы данных, представленная на рисунке 25.

На данной логической модели структура базы данных отображена уже более подробно. Здесь определены первичные и внешние ключи, ограничения атрибутов сущностей, а также уточнен их состав и связи между ними.

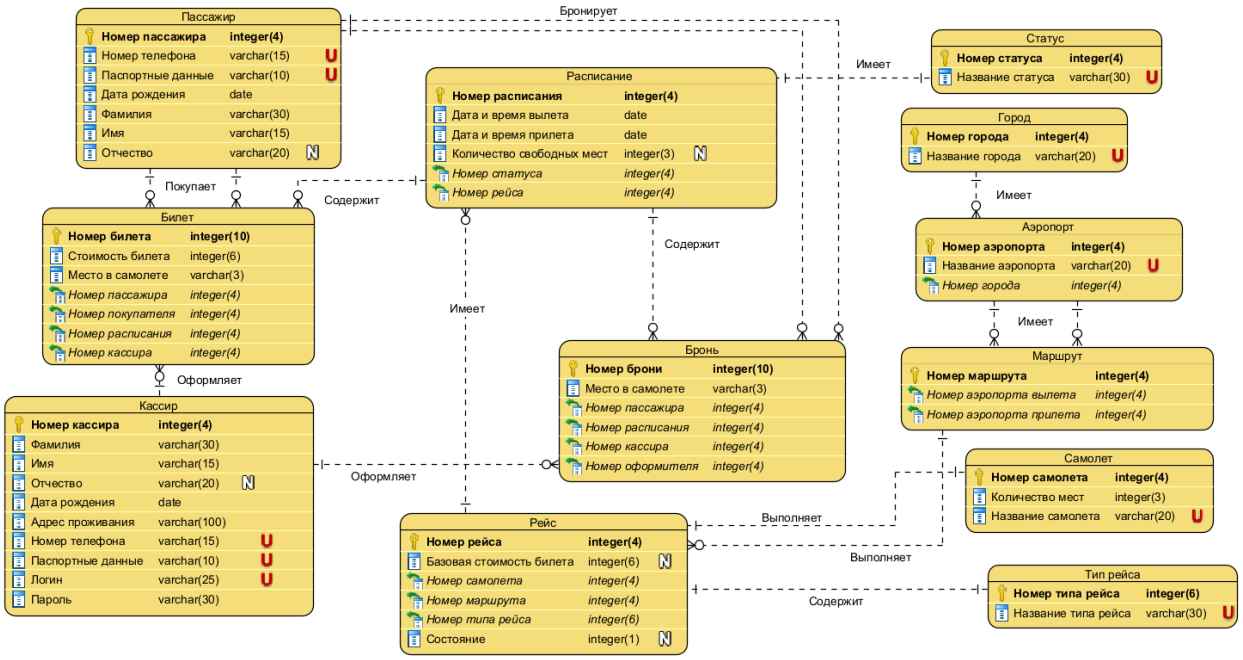

Рисунок 25 – Графическое представление логической модели базы данных

Сущности логической модели данных, не определенные в концептуальной модели, представлены в таблице 10.

Таблица 10 – Сущности логической модели

№	Название	Описание	Количество экземпляров	Режим обновления
1	Роль продавца	Список ролей продавцов	10	При добавлении новых типов продавцов
2	Тип рейса	Список типов рейса	20	При добавлении новых типов рейса
3	Самолет	Список самолетов	100	При добавлении нового самолета
4	Маршрут	Список авиамаршрутов	500	При добавлении нового маршрута
5	Аэропорт	Список аэропортов	200	При добавлении нового аэропорта
6	Город	Список городов	150	При добавлении нового города
7	Статус	Список статусов авиарейса	10	При добавлении нового статуса

Также в таблице 11 представлено описание связей между сущностями логической модели, не определенных в результате концептуального моделирования.

Таблица 11 – Описание связей между сущностями логической модели

№	Связь	Описание	Мощность	Обязательность
1	Рейс – Тип рейса	Определяет тип для рейса	1:1	Да
2	Рейс – Самолет	Определяет самолет, которым выполняется рейс	1:1	Да
3	Рейс – Маршрут	Определяет маршрут рейса	M:1	Да
4	Аэропорт – Маршрут	Определяет аэропорт вылета для маршрута	1:M	Да
5	Аэропорт – Маршрут	Определяет аэропорт прилета для маршрута	1:M	Да
6	Город – Аэропорт	Определяет аэропорты в городе	1:M	Да
7	Расписание – Статус	Определяет статус расписания	1:1	Да

Несмотря на достаточную содержательность данной модели, сущности все еще представлены в абстрактной форме, без привязки к конкретной реализации базы данных.

2.4 Проектирование серверной части приложения

Серверная часть приложения выполнена на основе архитектуры REST API. В данном случае взаимодействие с сервером осуществляется по протоколу HTTP при помощи стандартных методов, таких как GET (получение ресурса), PUT (обновление существующего ресурса), POST (создание нового ресурса) и DELETE (удаление ресурса). При этом обращение к ресурсам сервера происходит по их уникальным идентификаторам, называемым URI.

Исходя из выбранных технологий разработки серверной части приложения, для ее реализации было решено использовать шаблон MVC (Model-View-Controller). Схема данного шаблона приведена на рисунке 26.

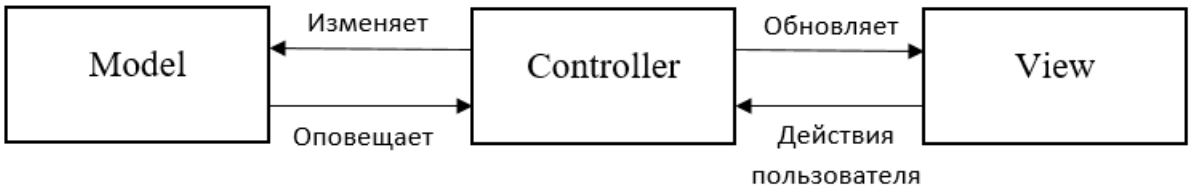

Рисунок 26 – Схема шаблона MVC

Как видно из данной схемы, в системе имеется 3 компонента: модель, представление и контроллер. Классы уровня модели отвечают за организацию доступа к данным (за непосредственное взаимодействие с базой данных). Классы контроллера отвечают за организацию взаимодействия пользователя с клиентом.

Они обрабатывают HTTP-запросы клиента и формируют ответы на них. Уровень представления отвечает за взаимодействие с пользователем.

Выбор данного шаблона обусловлен использованием в работе технологии ASP.NET, которая нацелена на применение данного шаблона.

2.5 Проектирование интерфейса пользователя

На основе выявленных функциональных требований был спроектирован интерфейс мобильного приложения. Он состоит из 15 страниц, 4 из которых являются основными вкладками. Они располагаются на странице входа: «Продажа» для оформления бронирования и продажи билетов; «Выкуп брони» для оформления снятия и выкупа броней; «Возврат билет» для возврата купленных билетов; «Другое» для получения аналитической информации и изменения данных клиентов.

Макет интерфейса страницы авторизации представлен на рисунке 27.

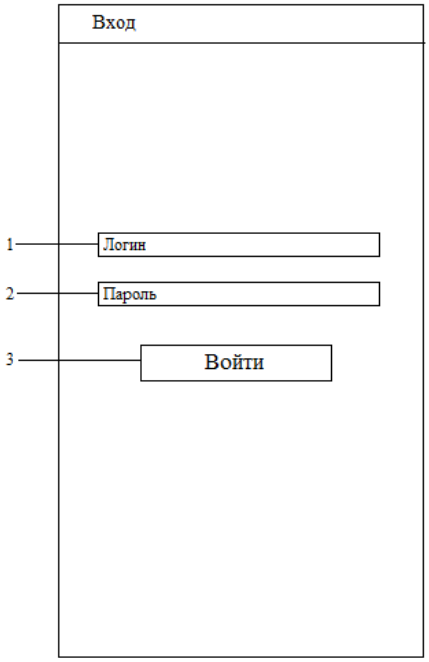

Рисунок 27 – Страница авторизации

На макете страницы авторизации имеются следующие элементы:

1 – поле ввода логина;

2 – поле ввода пароля;

3 – кнопка входа, по нажатию на которую осуществляется попытка авторизации.

Макет интерфейса страницы продажи представлен на рисунке 28.

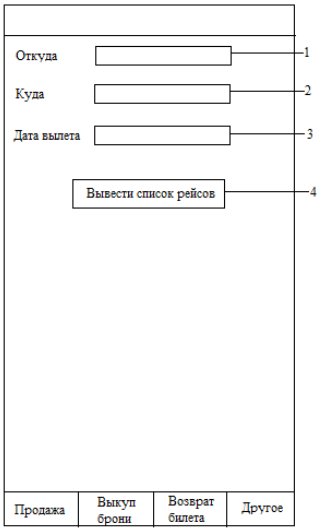

Рисунок 28 – Страница продажи

На макете страницы продажи имеются следующие элементы:

1 – поле выбора города отправления рейса;

2 – поле выбора города прибытия рейса;

3 – поле выбора даты рейса;

4 – кнопка вывода списка рейсов, удовлетворяющих заданным критериям.

Макет интерфейса страницы выбора рейса представлен на рисунке 29.

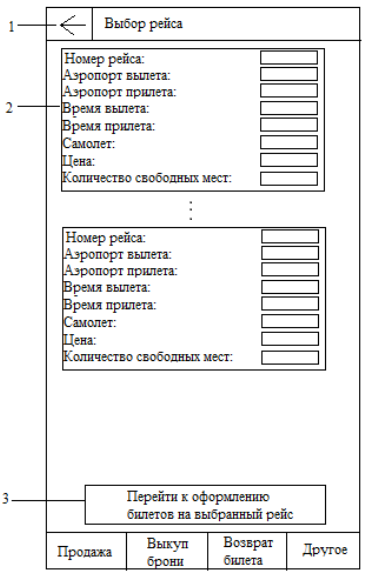

Рисунок 29 – Страница выбора рейса

На макете страницы авторизации имеются следующие элементы:

1 – кнопка «Назад», по нажатию на которую осуществляется возврат на предыдущую страницу;

2 – результаты поиска рейсов по заданным критериям;

3 – кнопка перехода к оформлению билетов, по нажатию на которую осуществляется переход на страницу покупки/бронирования билетов.

Макет интерфейса страницы покупки/бронирования билетов представлен на рисунке 30.

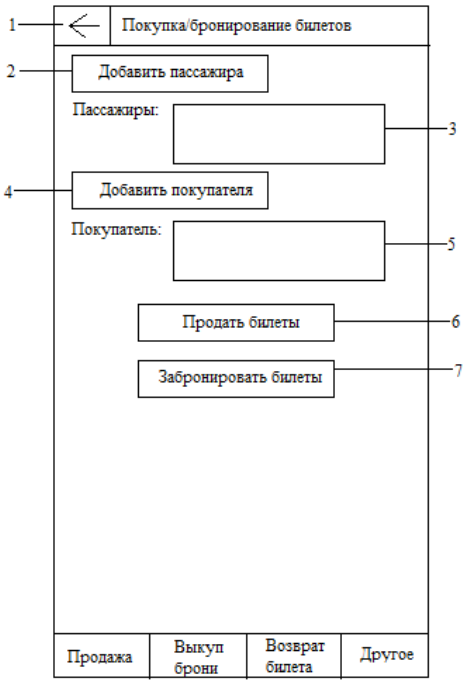

Рисунок 30 – Страница покупки/бронирования билетов

На макете страницы покупки/бронирования билетов имеются следующие элементы:

1 – кнопка «Назад», по нажатию на которую осуществляется возврат на предыдущую страницу;

2 – кнопка добавления пассажира, по нажатию на которую осуществляется переход на страницу добавления пассажира;

3 – список с информацией о добавленных пассажирах;

4 – кнопка добавления покупателя, по нажатию на которую осуществляется переход на страницу добавления покупателя;

5 – надпись с информацией о добавленном покупателе;

6 – кнопка продажи билетов, по нажатию на которую осуществляется попытка оформления продажи билетов с введенными параметрами и происходит переход на предыдущую страницу;

7 – кнопка бронирования билетов, по нажатию на которую осуществляется попытка оформления бронирования билетов с введенными параметрами и происходит переход на предыдущую страницу.

Макет интерфейса страницы добавления ранее учтенного пассажира представлен на рисунке 31.

Рисунок 31 – Страница добавления ранее учтенного пассажира

На макете страницы добавления ранее учтенного пассажира имеются следующие элементы:

1 – поле ввода паспортных данных пассажира;

2 – поле выбора места в самолете;

3 – кнопка добавления пассажира, по нажатию на которую осуществляется поиск пассажира и добавление его к списку добавленных пассажиров;

4 – кнопка «Назад», по нажатию на которую осуществляется возврат на страницу покупки/бронирования билетов.

Макет интерфейса страницы добавления нового пассажира представлен на рисунке 32.

Рисунок 32 – Страница добавления нового пассажира

На макете страницы добавления нового пассажира имеются следующие элементы:

1 – поле ввода фамилии пассажира;

2 – поле ввода имени пассажира;

3 – поле ввода отчества пассажира;

4 – поле ввода даты рождения пассажира;

5 – поле ввода номера телефона пассажира;

6 – поле ввода паспортных данных пассажира;

7 – поле выбора места в самолете;

8 – кнопка добавления пассажира, по нажатию на которую осуществляется добавление пассажира в информационную систему и занесение его в список добавленных пассажиров;

9 – кнопка «Назад», по нажатию на которую осуществляется возврат на страницу покупки/бронирования билетов.

Макет интерфейса страницы добавления ранее учтенного покупателя представлен на рисунке 33.

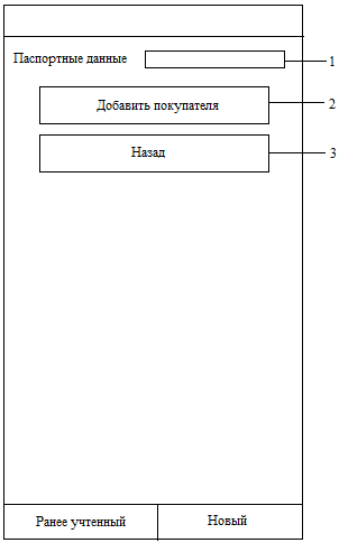

Рисунок 33 – Страница добавления ранее учтенного покупателя

На макете страницы добавления ранее учтенного покупателя имеются следующие элементы:

1 – поле ввода паспортных данных пассажира;

2 – кнопка добавления покупателя, по нажатию на которую осуществляется поиск покупателя и его выбор;

3 – кнопка «Назад», по нажатию на которую осуществляется возврат на страницу покупки/бронирования билетов.

Макет интерфейса страницы добавления нового покупателя представлен на рисунке 34.

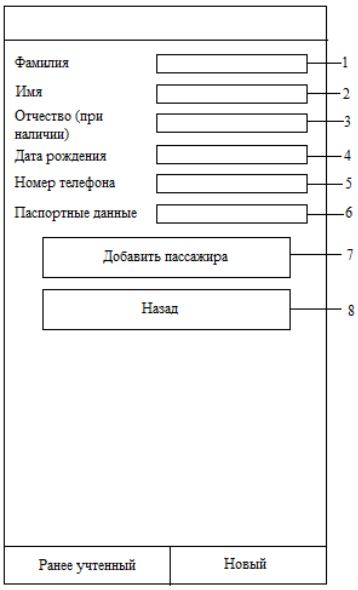

Рисунок 34 – Страница добавления нового покупателя

На макете страницы добавления нового покупателя имеются следующие элементы:

1 – поле ввода фамилии покупателя;

2 – поле ввода имени покупателя;

3 – поле ввода отчества покупателя;

4 – поле ввода даты рождения покупателя;

5 – поле ввода номера телефона покупателя;

6 – поле ввода паспортных данных покупателя;

7 – кнопка добавления покупателя, по нажатию на которую осуществляется добавление покупателя в информационную систему и его выбор;

8 – кнопка «Назад», по нажатию на которую осуществляется возврат на страницу покупки/бронирования билетов.

Макет интерфейса страницы выкупа брони представлен на рисунке 35.

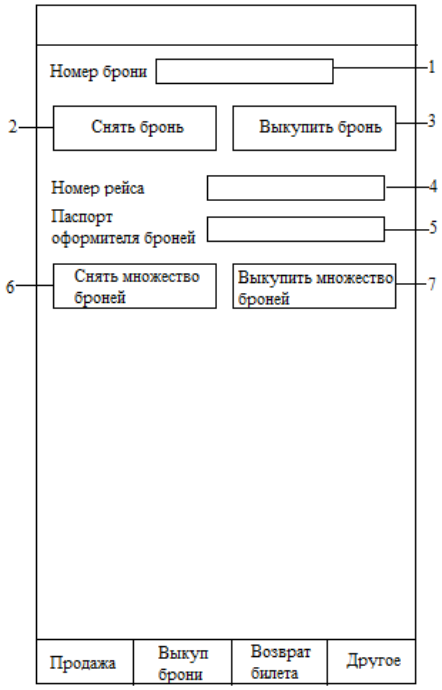

Рисунок 35 – Страница выкупа брони

На макете страницы выкупа брони имеются следующие элементы:

1 – поле ввода номера брони;

2 – кнопка снятия брони, по нажатию на которую осуществляется сброс брони с введенным номером;

3 – кнопка выкупа брони, по нажатию на которую осуществляется выкуп брони с введенным номером;

4 – поле ввода номера рейса;

5 – поле ввода паспортных данных оформителя броней;

6 – кнопка снятия множества броней, по нажатию на которую осуществляется сброс одной или нескольких броней на основе введенных данных;

7 – кнопка выкупа множества броней, по нажатию на которую осуществляется выкуп одной или нескольких броней на основе введенных данных.

Макет интерфейса страницы возврата билета представлен на рисунке 36.

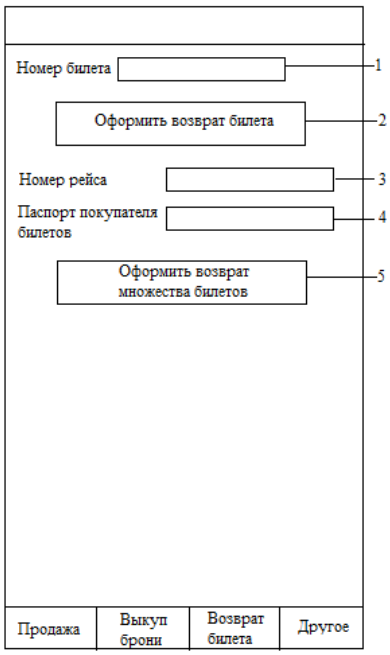

Рисунок 36 – Страница возврата билета

На макете страницы возврата билета имеются следующие элементы:

1 – поле ввода номера билета;

2 – кнопка оформления возврата билета, по нажатию на которую осуществляется обнуление билета;

3 – поле ввода номера рейса;

4 – поле ввода паспортных данных покупателя билетов;

5 – кнопка оформления возврата множества билетов, по нажатию на которую осуществляется обнуление одного или нескольких билетов на основе введенных данных.

Макет интерфейса страницы «Другое» представлен на рисунке 37.

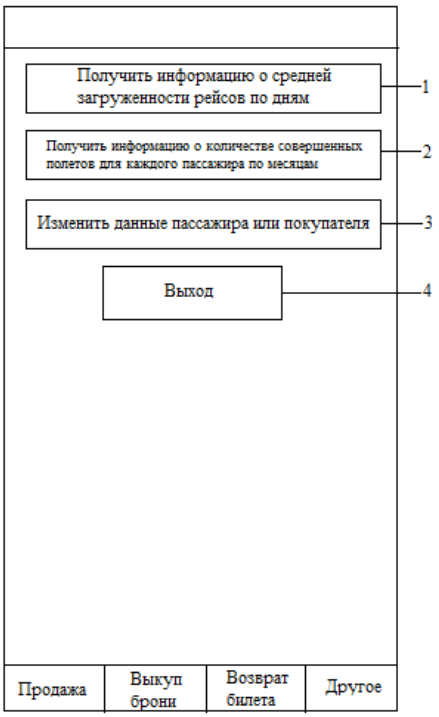

Рисунок 37 – Страница «Другое»

На макете страницы «Другое» имеются следующие элементы:

1 – кнопка, по нажатию на которую открывается страница с информацией о средней загруженности рейсов по дням;

2 – кнопка, по нажатию на которую открывается страница с информацией о количестве совершенных полетов для каждого пассажира по месяцам;

3 – кнопка, по нажатию на которую открывается страница выбора пассажира или покупателя;

4 – кнопка выхода из аккаунта, по нажатию на которую происходит переход на страницу авторизации.

Макет интерфейса страницы выбора пассажира или покупателя представлен на рисунке 38.

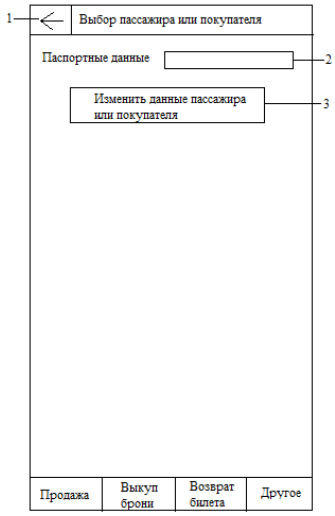

Рисунок 38 – Страница выбора пассажира или покупателя

На макете страницы выбора пассажира или покупателя имеются следующие элементы:

1 – кнопка «Назад», по нажатию на которую осуществляется возврат на предыдущую страницу;

2 – поле ввода паспортных данных пассажира или покупателя;

3 – кнопка, по нажатию на которую происходит поиск пассажира или покупателя и осуществляется переход на страницу изменения данных пассажира или покупателя.

Макет интерфейса страницы изменения данных пассажира или покупателя представлен на рисунке 39.

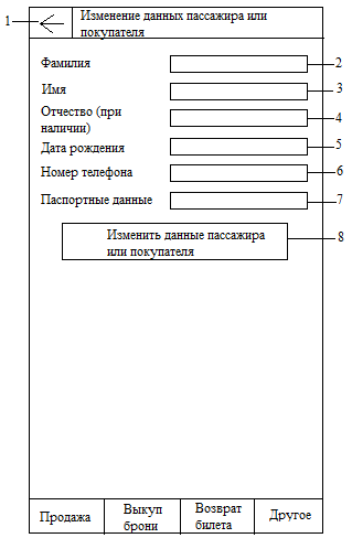

Рисунок 39 – Страница изменения данных пассажира или покупателя

На макете страницы изменения данных пассажира или покупателя имеются следующие элементы:

1 – кнопка «Назад», по нажатию на которую осуществляется возврат на предыдущую страницу;

2 – поле ввода фамилии пассажира или покупателя;

3 – поле ввода имени пассажира или покупателя;

4 – поле ввода отчества пассажира или покупателя;

5 – поле выбора даты рождения пассажира или покупателя;

6 – поле ввода номера телефона пассажира или покупателя;

7 – поле ввода паспортных данных пассажира или покупателя;

8 – кнопка, по нажатию на которую осуществляется сохранение введенных данных возврат на предыдущую страницу.

3 Реализация программного обеспечения

3.1 Реализация базы данных

На основании концептуальной и логической модели для СУБД Microsoft SQL Server 2019 была разработана физическая модель базы данных (рисунок 40).

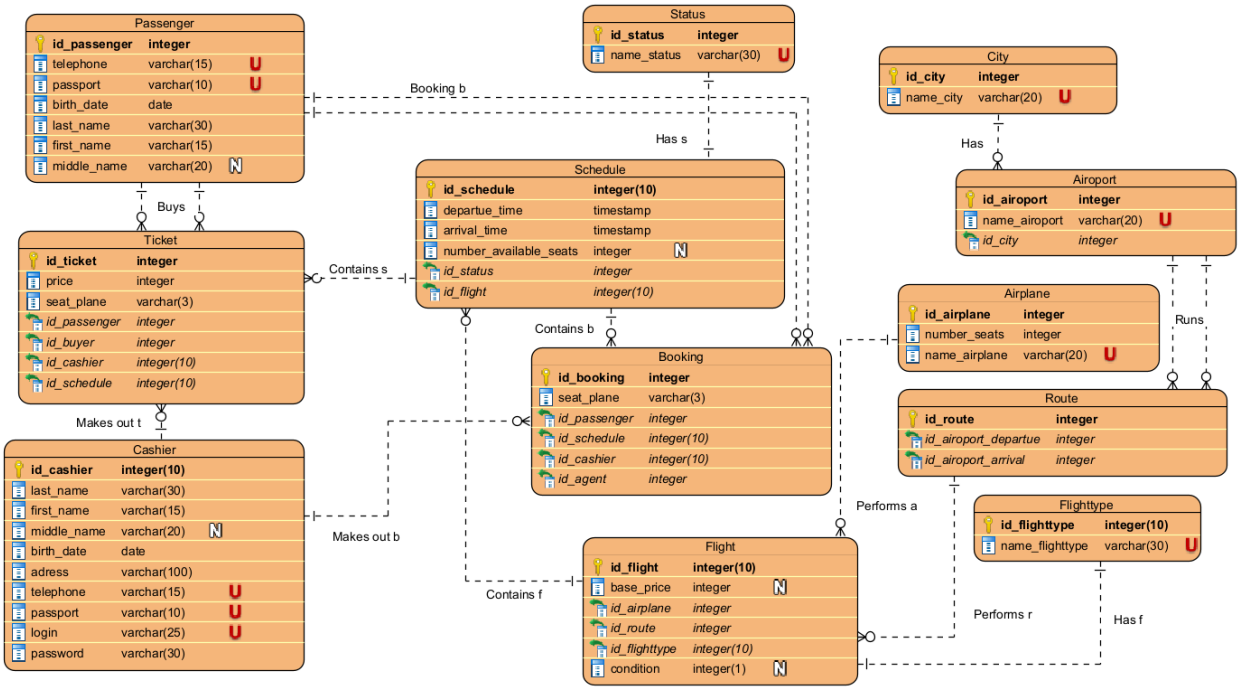

Рисунок 40 – Графическое представление физической модели базы данных

Описание атрибутов сущности «Пассажир» представлено в таблице 12.

Таблица 12 – Описание атрибутов сущности «Пассажир»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_passenger	Идентификатор	integer	Нет	Да	PK
telephone	Номер телефона	varchar(15)	Нет	Да	-
passport	Номер паспорта	varchar(10)	Нет	Да	-
birth_date	Дата рождения	date	Нет	Нет	-
last_name	Фамилия	varchar(30)	Нет	Нет	-
first_name	Имя	varchar(15)	Нет	Нет	-
middle_name	Отчество	varchar(20)	Да	Нет	-

Описание атрибутов сущности «Кассир» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «Кассир»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_cashier	Идентификатор	integer	Нет	Да	PK
telephone	Номер телефона	varchar(15)	Нет	Да	-
passport	Номер паспорта	varchar(10)	Нет	Да	-
birth_date	Дата рождения	date	Нет	Нет	-
last_name	Фамилия	varchar(30)	Нет	Нет	-
first_name	Имя	varchar(15)	Нет	Нет	-
middle_name	Отчество	varchar(20)	Да	Нет	-
adress	Адрес проживания	varchar(100)	Нет	Нет	-
login	Логин	varchar(25)	Нет	Да	-
password	Пароль	varchar(30)	Нет	Нет	-

Описание атрибутов сущности «Статус» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «Статус»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_status	Идентификатор	integer	Нет	Да	PK
name_status	Наименование статуса рейса	varchar(30)	Нет	Да	-

Описание атрибутов сущности «Город» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Город»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_city	Идентификатор	integer	Нет	Да	PK
name_city	Наименование города	varchar(20)	Нет	Да	-

Описание атрибутов сущности «Аэропорт» представлено в таблице 16.

Таблица 16 – Описание атрибутов сущности «Аэропорт»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_airoport	Идентификатор	integer	Нет	Да	PK
name_airoport	Наименование аэропорта	varchar(20)	Нет	Да	-
id_city	Идентификатор города	integer	Нет	Нет	FK

Описание атрибутов сущности «Самолет» представлено в таблице 17.

Таблица 17 – Описание атрибутов сущности «Самолет»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_airplane	Идентификатор	integer	Нет	Да	PK
number_seats	Количество мест	integer	Нет	Нет	-
name_airplane	Наименование самолета	varchar(20)	Нет	Да	-

Описание атрибутов сущности «Маршрут» представлено в таблице 18.

Таблица 18 – Описание атрибутов сущности «Маршрут»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_route	Идентификатор	integer	Нет	Да	PK
id_airoport_departue	Идентификатор аэропорта вылета	integer	Нет	Нет	FK
id_airoport_arrival	Идентификатор аэропорта прилета	integer	Нет	Нет	FK

Описание атрибутов сущности «Тип рейса» представлено в таблице 19.

Таблица 19 – Описание атрибутов сущности «Тип рейса»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_flighttype	Идентификатор	integer	Нет	Да	PK
name_flighttype	Наименование типа рейса	varchar(30)	Нет	Да	-

Описание атрибутов сущности «Рейс» представлено в таблице 20.

Таблица 20 – Описание атрибутов сущности «Рейс»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_flight	Идентификатор	integer	Нет	Да	PK
base_price	Базовая стоимость	integer	Да	Нет	-
id_airplane	Идентификатор самолета	integer	Нет	Нет	FK
id_route	Идентификатор маршрута	integer	Нет	Нет	FK
id_flighttype	Идентификатор типа рейса	integer	Нет	Нет	FK
condition	Состояние рейса	integer	Да	Нет	-

Описание атрибутов сущности «Расписание» представлено в таблице 21.

Таблица 21 – Описание атрибутов сущности «Расписание»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_schedule	Идентификатор	integer	Нет	Да	PK
departure_time	Дата и время вылета	timestamp	Нет	Нет	-
arrival_time	Дата и время прилета	timestamp	Нет	Нет	-
number_available_seats	Количество свободных мест	integer	Да	Нет	-
id_status	Идентификатор статуса рейса	integer	Нет	Нет	FK
id_flight	Идентификатор рейса	integer	Нет	Нет	FK

Описание атрибутов сущности «Билет» представлено в таблице 22.

Таблица 22 – Описание атрибутов сущности «Билет»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_ticket	Идентификатор	integer	Нет	Да	PK
price	Стоимость	integer	Нет	Нет	-
seat_plane	Место в самолете	varchar(3)	Нет	Нет	-
id_passenger	Идентификатор пассажира	integer	Нет	Нет	FK
id_buyer	Идентификатор покупателя	integer	Нет	Нет	FK
id_cashier	Идентификатор кассира	integer	Нет	Нет	FK
id_schedule	Идентификатор расписания	integer	Нет	Нет	FK

Описание атрибутов сущности «Бронь» представлено в таблице 23.

Таблица 23 – Описание атрибутов сущности «Бронь»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_booking	Идентификатор	integer	Нет	Да	PK
seat_plane	Место в самолете	varchar(3)	Нет	Нет	-
id_passenger	Идентификатор пассажира	integer	Нет	Нет	FK
id_agent	Идентификатор оформителя	integer	Нет	Нет	FK
id_cashier	Идентификатор кассира	integer	Нет	Нет	FK
id_schedule	Идентификатор расписания	integer	Нет	Нет	FK

На данной физической модели структура базы данных отображена с учетом спецификации типов СУБД Microsoft SQL Server 2019.

На основе физической модели БД был сгенерирован скрипт на языке SQL для создания базы данных. Его код представлен в приложении Б. Развертывание спроектированной базы данных было выполнено путем запуска данного скрипта в СУБД Microsoft SQL Server 2019.

3.2 Реализация серверной части приложения

Реализация серверной части приложения была произведена в среде программирования Microsoft Visual Studio 2019 на языке C# с применением технологии Web API на платформе ASP.NET [17, 28]. Для взаимодействия с созданной базой данных была использована объектно-ориентированная технология Entity Framework [18, 29].

На основании используемой архитектуры MVC все классы серверной части делятся на два слоя: слой модели, предназначенный для взаимодействия с базой данных, и слой контроллеров, предназначенный для обработки HTTP-запросов к серверу.

Диаграмма классов слоя модели представлена на рисунке 41.

Представленные на данной диаграмме классы были автоматически сгенерированы средой программирования Visual Studio с помощью технологии Entity Framework. Генерация производилась на основе реализованной до этого базы данных.

Как можно увидеть из диаграммы, классы Passenger, Ticket, Schedule, Cashier, Status, Booking, Flight, City, Airoport, Route, Airplane и Flighttype были созданы на основе одноименных таблиц реализованной базы данных. Классы же PassengersFlightsCount и AverageFlightLoad основываются на одноименных представлениях и позволяют облегчить выборку нужных данных из базы.

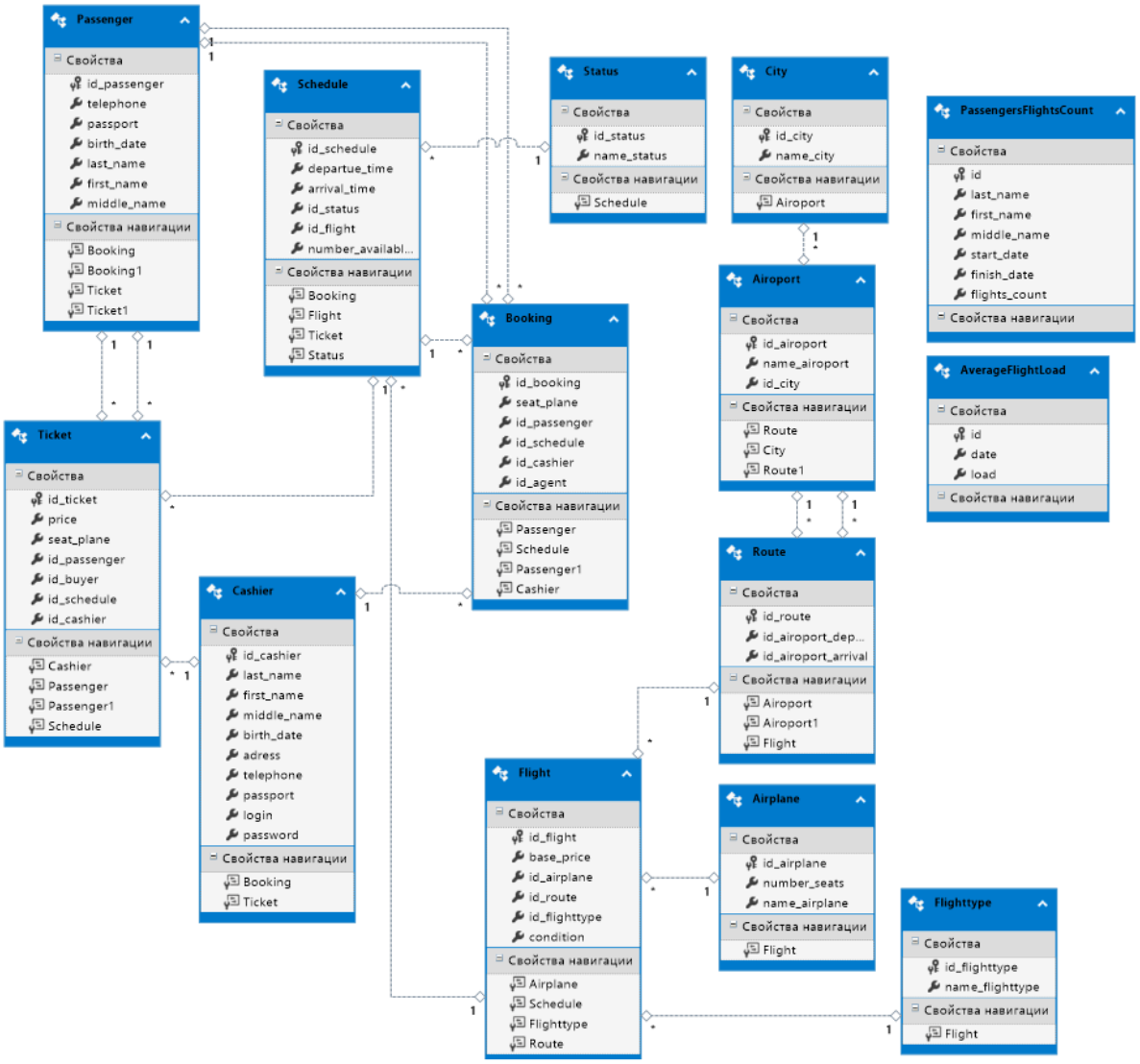

Рисунок 41 – Диаграмма классов серверной части слоя модели

Диаграмма классов слоя контроллеров представлена на рисунке 42.

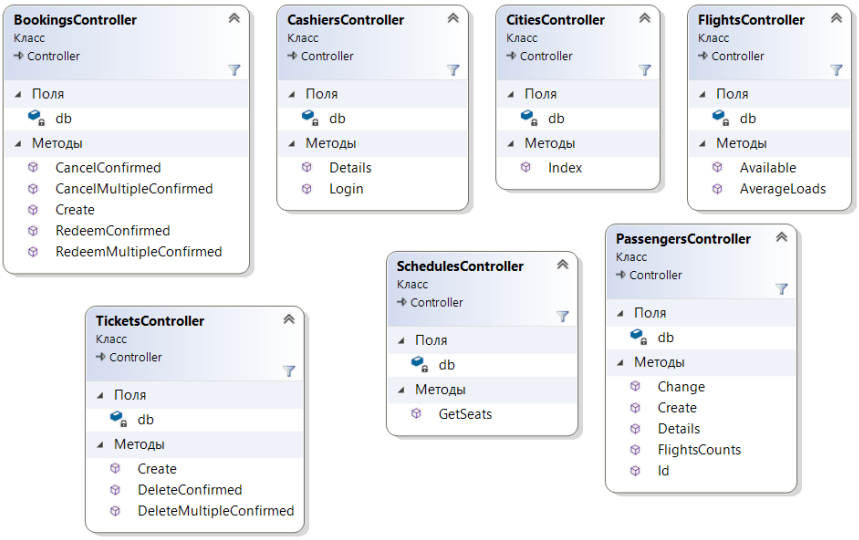

Рисунок 42 – Диаграмма классов серверной части слоя контроллеров

Описание классов слоя контроллеров приведено в таблице 24.

Таблица 24 – Описание классов слоя контроллеров

Класс	Назначение
BookingsController	Обрабатывает запросы добавления новых броней, выкупа и снятия одной и множества броней
CashiersController	Обрабатывает запросы авторизации кассира и получения о нем информации
CitiesController	Обрабатывает запросы получения списка городов
FlightsController	Обрабатывает запросы получения доступных рейсов на определенное направление и дату, а также получения информации о средней загруженности рейсов по дням
TicketsController	Обрабатывает запросы добавления новых билетов, возврата одного и множества билетов
SchedulesController	Обрабатывает запросы получения списка свободных мест на расписание
PassengersController	Обрабатывает запросы добавления нового пассажира или покупателя, изменения его данных, получения информации о пассажирах или покупателях и их номеров, а также информации о количестве совершенных полетов по месяцам

Исходный код серверной части приведен в приложении В.

3.3 Реализация мобильного приложения

На данном этапе было реализовано мобильное приложение на языке C# в среде Visual Studio 2019. Исходный код представлен в приложении Г. Диаграмма компонентов, отражающая структуру программы, представлена на рисунке 43.

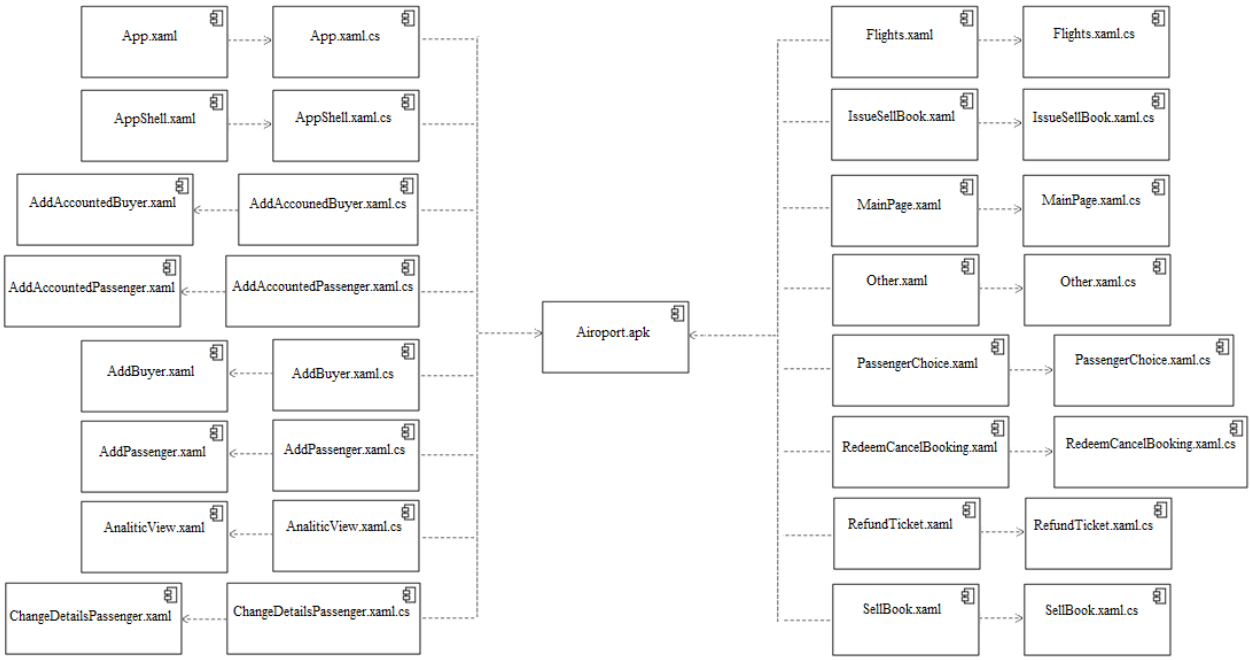

Рисунок 43 – Диаграмма компонентов мобильного приложения

Описание компонентов представлено в таблице 25.

Таблица 25 – Описание компонентов мобильного приложения

Компонент	Назначение
App.xaml	XAML-файл описания подключаемых ресурсов к приложению
App.xaml.cs	Начальная точка загрузки и закрытия приложения
AppShell.xaml	XAML-файл описания структуры интерфейса приложения
AppShell.xaml.cs	Регистрирует страницы приложения, не вошедшие в основную структуру
AddAccountedBuyer.xaml	XAML-файл описания интерфейса страницы добавления ранее учтенного покупателя
AddAccountedBuyer.xaml.cs	Управляет логикой и данными страницы добавления ранее учтенного покупателя
AddAccountedPassenger.xaml	XAML-файл описания интерфейса страницы добавления ранее учтенного пассажира
AddAccountedPassenger.xaml.cs	Управляет логикой и данными страницы добавления ранее учтенного пассажира
AddBuyer.xaml	XAML-файл описания интерфейса страницы добавления нового покупателя
AddBuyer.xaml.cs	Управляет логикой и данными страницы добавления нового покупателя
AddPassenger.xaml	XAML-файл описания интерфейса страницы добавления нового пассажира
AddPassenger.xaml.cs	Управляет логикой и данными страницы добавления нового пассажира
AnaliticView.xaml	XAML-файл описания интерфейса страницы с аналитической информацией
AnaliticView.xaml.cs	Управляет логикой и данными страницы с аналитической информацией
ChangeDetailsPassenger.xaml	XAML-файл описания интерфейса страницы изменения данных пассажира или покупателя
ChangeDetailsPassenger.xaml.cs	Управляет логикой и данными страницы изменения данных пассажира или покупателя
Flights.xaml	XAML-файл описания интерфейса страницы вывода списка рейсов
Flights.xaml.cs	Управляет логикой и данными страницы вывода списка рейсов
IssueSellBook.xaml	XAML-файл описания интерфейса страницы покупки/бронирования билетов
IssueSellBook.xaml.cs	Управляет логикой и данными страницы покупки/бронирования билетов
MainPage.xaml	XAML-файл описания интерфейса страницы авторизации
MainPage.xaml.cs	Управляет логикой и данными страницы авторизации
Other.xaml	XAML-файл описания интерфейса страницы «Другое»
Other.xaml.cs	Управляет логикой и данными страницы «Другое»
PassengerChoice.xaml	XAML-файл описания интерфейса страницы выбора пассажира или покупателя при изменении его данных
PassengerChoice.xaml.cs	Управляет логикой и данными страницы выбора пассажира или покупателя при изменении его данных
RedeemCancelBooking.xaml	XAML-файл описания интерфейса страницы выкупа и снятия броней
RedeemCancelBooking.xaml.cs	Управляет логикой и данными страницы выкупа и снятия броней
RefundTicket.xaml	XAML-файл описания интерфейса страницы возврата билетов
RefundTicket.xaml.cs	Управляет логикой и данными страницы возврата билетов

Продолжение таблицы 25

Компонент	Назначение
SellBook.xaml	XAML-файл описания интерфейса страницы продажи и бронирования билетов
SellBook.xaml.cs	Управляет логикой и данными страницы продажи и бронирования билетов
Airoport.apk	Исполняемый файл мобильного приложения

Диаграмма развертывания, на которой описывается физическое расположение компонентов системы, представлена на рисунке 44.

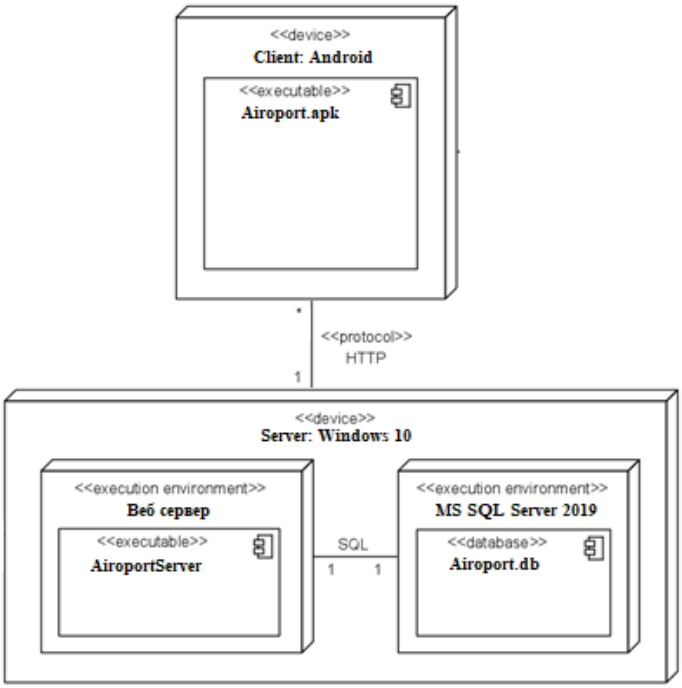

Рисунок 44 – Диаграмма развертывания программного обеспечения

Из данной диаграммы можно увидеть, что программное обеспечение состоит из трех уровней: мобильное приложение, работающее под управлением ОС Android, серверная часть и СУБД, работающие под управлением ОС Windows 10.

3.4 Тестирование программного обеспечения

3.4.1 Тестирование серверной части приложения

3.4.1.1 Функциональное тестирование

На данном этапе было выполнено функциональное тестирование в целях проверки правильности работы серверной части приложения. Результаты

функционального тестирования серверной части приложения представлены в таблице 26.

Таблица 26 – Результаты функционального тестирования серверной части

№	Описание теста	Структура запроса	Результат
1	Авторизация кассира	Метод: GET URL: Cashiers/Login?login=Matveev &password=Matveevich1983	Тест пройден. Кассир авторизирован (рисунок Д.1).
2	Получение информации о кассире по его идентификатору	Метод: GET URL: Cashiers/Details/4	Тест пройден. Информация получена (рисунок Д.2).
3	Получение списка городов	Метод: GET URL: Cities	Тест пройден. Список получен (рисунок Д.3).
4	Получение списка рейсов на определенное направление и дату	Метод: GET URL: Flights/Available?city_from=Пенза &city_to=Москва&date=2023-01-04	Тест пройден. Список получен (рисунок Д.4).
5	Получение списка свободных мест на определенный рейс по его идентификатору	Метод: GET URL: Schedules/GetSeats/3	Тест пройден. Список получен (рисунок Д.5).
6	Получение информации о пассажире по его идентификатору	Метод: GET URL: Passengers/Details/6	Тест пройден. Информация получена (рисунок Д.6).
7	Изменение данных пассажира	Метод: PUT URL: Passengers/Change Body: { "id_passenger": 3, "telephone": "+79375091284", "passport": "5299100322", "birth_date": "1992-09-27", "last_name": "Богданов", "first_name": "Богдан", "middle_name": "Богданович" }	Тест пройден. Данные изменены (рисунок Д.7).
8	Добавление пассажира	Метод: POST URL: Passengers/Create Body: { "telephone": "+79378410924", "passport": "9288160325", "birth_date": "1993-08-14", "last_name": "Захаров", "first_name": "Захар", "middle_name": "Захарович" }	Тест пройден. Пассажир добавлен (рисунок Д.8).
9	Добавление брони	Метод: POST URL: Bookings/Create Body: { "seat_plane": "08c", "id_passenger": 10, "id_buyer": 10, "id_schedule": 5, "id_cashier": 9 }	Тест пройден. Бронь добавлена (рисунок Д.9).
10	Снятие брони по ее идентификатору	Метод: DELETE URL: Bookings/Cancel/114	Тест пройден. Бронь снята (рисунок Д.10).
11	Выкуп брони	Метод: POST URL: Bookings/Redeem Body: { "id": 115 }	Тест пройден. Бронь выкуплена (рисунок Д.11).
12	Снятие множества броней по номеру рейса и номеру паспорта оформителя броней	Метод: DELETE URL: Bookings/CancelMultiple? id_schedule=5&passport=9090761536	Тест пройден. Множество броней снято (рисунок Д.12).

Продолжение таблицы 26

№	Описание теста	Структура запроса	Результат
13	Выкуп множества броней	Метод: POST URL: Bookings/RedeemMultiple Body: { "id_schedule": 5, "passport": "9090761536" }	Тест пройден. Множество броней выкуплено (рисунок Д.13).
14	Добавление билета	Метод: POST URL: Tickets/Create Body: { "seat_plane": "01b", "id_passenger": 3, "id_buyer": 4, "id_schedule": 13, "id_cashier": 2 }	Тест пройден. Билет добавлен (рисунок Д.14).
15	Возврат билета по его идентификатору	Метод: DELETE URL: Tickets/Refund/113	Тест пройден. Возврат билета осуществлен (рисунок Д.15).
16	Возврат множества билетов	Метод: DELETE URL: Tickets/RefundMultiple?id_schedule=5&passport=9090761536	Тест пройден. Возвращено множество билетов (рисунок Д.16).
17	Получение средней загруженности рейсов по дням за последнюю неделю	Метод: GET URL: Flights/AverageLoads	Тест пройден. Информация получена (рисунок Д.17).
18	Получение количества полетов пассажиров по месяцам за последний год	Метод: GET URL: Passengers/FlightsCounts	Тест пройден. Информация получена (рисунок Д.18).

Все тесты тестирования были пройдены. Значит можно говорить о правильности работы серверной части приложения.

3.4.1.2 Модульное тестирование

В дополнении к функциональному тестированию было проведено модульное тестирование серверной части для запроса получения списка рейсов на заданное направление и дату, а также для запроса авторизации кассира. Оно выполнялось на языке программирования C# в среде разработки Visual Studio 2019. Исходный код тестов приведен в приложении Е, а его результаты – на рисунке 45.

Рисунок 45 – Результаты модульного тестирования

Все тесты модульного тестирования прошли успешно. Следовательно, можно говорить о корректной работе данных функций.

3.4.2 Тестирование клиентской части приложения

Также было произведено функциональное тестирование мобильного приложения, результаты которого приведены в таблице 27.

Таблица 27 – Результаты функционального тестирования мобильного приложения

Вариант использования	Параметры теста	Результат
Авторизоваться	Авторизация кассира с корректными данными. Логин: Fedorov Пароль: Fedorovich1960	Тест пройден. Авторизация выполнена (рисунок Ж.1).
	Авторизация кассира с некорректными данными. Логин: Fed Пароль: Fed196	Тест пройден. Показано сообщение с ошибкой (рисунок Ж.2).
Оформить продажу билетов	Оформление продажи билетов	Тест пройден. Продажа оформлена (рисунок Ж.3).
Оформить бронирование билетов	Оформление бронирования билетов	Тест пройден. Бронирование оформлено (рисунок Ж.4).
Занести информацию о покупателе	Добавление нового покупателя	Тест пройден. Информация занесена (рисунок Ж.5).
Занести информацию о пассажире	Добавление нового пассажира	Тест пройден. Информация занесена (рисунок Ж.6).
Оформить выкуп брони	Оформление выкупа брони	Тест пройден. Выкуп брони оформлен (рисунок Ж.7).
Оформить снятие брони	Оформление снятия брони	Тест пройден. Снятие брони оформлено (рисунок Ж.8).
Оформить выкуп множества броней	Оформление выкупа множества броней	Тест пройден. Выкуп множества броней оформлен (рисунок Ж.9).
Оформить снятие множества броней	Оформление снятия множества броней	Тест пройден. Снятие множества броней оформлено (рисунок Ж.10).
Оформить возврат билета	Оформление возврата билета	Тест пройден. Возврат билета оформлен (рисунок Ж.11).
Оформить возврат множества билетов	Оформление возврата множества билетов	Тест пройден. Возврат множества билетов оформлен (рисунок Ж.12).
Получить информацию о средней загруженности рейсов по дням	Получение информации о средней загруженности рейсов по дням	Тест пройден. Информация о средней загруженности получена (рисунок Ж.13).
Получить информацию о количестве совершенных полетов для каждого пассажира по месяцам	Получение информации о количестве совершенных полетов для каждого пассажира по месяцам	Тест пройден. Информация о количестве полетов получена (рисунок Ж.14).
Изменить данные пассажира или покупателя	Изменение данных пассажира или покупателя	Тест пройден. Данные изменены (рисунок Ж.15).

Все тесты были выполнены успешно. Следовательно, можно говорить о корректной работе мобильного приложения.

3.4.3 Метрики кода

Метрики кода позволяют получить количественные величины, характеризующее программное обеспечение по тем или иным свойствам. Существует большое количество разнообразных метрик. Одной из самых простых метрик кода является количество строк кода (SLOC-метрики). Также одной из самых распространенных метрик, основанных на подсчете некоторых единиц в коде программы, относят метрики Холстеда.

Рассмотрим 3 метода класса Service, отвечающие за авторизацию кассира в системе (LoginCashierAsync), получение списка рейсов (GetFlightsAsync) и получения списка наименований городов (GetCitiesAsync). Коды данных методов приведены в листинге 1.

Листинг 1 – Коды методов класса Service

```
private static string url = "http://192.168.0.227:44307/";
private static HttpClientHandler ClientHandler = new HttpClientHandler();
private static HttpClient Client = new HttpClient(ClientHandler);

/// <summary>
/// Функция для получения кассира по номеру паспорта
/// </summary>
/// <param name="login">Логин</param>
/// <param name="password">Пароль</param>
/// <returns>Объект типа Cashier или null, если номер паспорта не найден</returns>
public async static Task<Cashier> LoginCashierAsync(string login, string password)
{
    Cashier cashier = null;
    var response = await Client.GetAsync(url + "Cashiers/Login?login=" + login +
"&password=" + password);

    if (response.IsSuccessStatusCode) {
        string responseText = await response.Content.ReadAsStringAsync();
        cashier = JsonConvert.DeserializeObject<Cashier>(responseText);
    }
    return cashier;
}

/// <summary>
/// Функция для получения списка рейсов
/// </summary>
/// <param name="city_from">Город отправления</param>
/// <param name="city_to">Город прилета</param>
/// <param name="date">Дата отправления</param>
/// <returns>Список объектов типа Flight или null, если рейсы не найдены</returns>
public async static Task<List<Flight>> GetFlightsAsync(string city_from, string
city_to, DateTime date)
{
    List<Flight> flights = null;
    string date_str = date.Date.ToString("yyyy-MM-dd");
```

```
    var response = await Client.GetAsync(url + "Flights/Available?city_from=" +
city_from + "&city_to=" + city_to + "&date=" + date_str);

    if (response.IsSuccessStatusCode)
    {
        string responseText = await response.Content.ReadAsStringAsync();
        flights = JsonConvert.DeserializeObject<List<Flight>>(responseText);
    }
    return flights;
}

/// <summary>
/// Функция для получения списка наименований имеющихся городов
/// </summary>
///  <returns>Список объектов типа string или пустой список, если города не
найдены</returns>
public static async Task<List<string>> GetCitiesAsync()
{
    List<City> cities = new List<City>();
    List<string> name_cities = new List<string>();
    var response = await Client.GetAsync(url + "Cities");

    if (response.IsSuccessStatusCode)
    {
        string responseText = await response.Content.ReadAsStringAsync();
        cities = JsonConvert.DeserializeObject<List<City>>(responseText);
        foreach(City city in cities)
        {
             name_cities.Add(city.name_city);
        }
    }
    return name_cities;
}
```

Данные методы находятся в клиентской части приложения. Функция для получения кассира запускается при попытке авторизации кассира в мобильном приложении. В ней выполняется HTTP-запрос к серверной части приложения для получения информации о кассире. Если данный запрос успешный, то возвращается данная информация вызываемому методу, иначе возвращается null. Функция для получения списка рейсов вызывается при поиске рейсов и работает аналогично функции получении информации о кассире. Функция получения списка наименований имеющихся городов вызывается после авторизации кассира и работает также, как и предыдущие, только возвращает не полную информацию о городах, а лишь их названия.

Для данных методов будут вычислены две группы метрик: метрики размера программы и метрики Холстеда.

3.4.3.1 Метрики размера программы

Данная группа метрик используется для измерения объема программы с помощью подсчета количества строк в тексте исходного кода. При этом часто выделяют методику подсчета физических строк кода и строк комментариев. Вычислим метрики размера программы для кода, приведенного в листинге 1 (таблица 28).

Таблица 28 – Метрики размера программы

Метрика	Количество
Количество строк кода	64
Количество строк комментариев	17

Теперь вычислим процент комментариев. Он равен (17/64)*100%=26%. Это означает, что количество комментариев в коде достаточно для его хорошего понимания, и при этом не слишком велико и не переполняет программу.

3.4.3.2 Метрики Холстеда

Метрика Холстеда вычисляется на основе анализа числа строк и синтаксических элементов исходного кода программы, а именно используются характеристики: число уникальных операторов, число уникальных операндов, общее число операторов и общее число операндов [30 – 32]. Для измерения данных характеристик необходимо составить словарь операторов (таблица 29) и словарь операндов (таблица 30).

Таблица 29 – Словарь операторов

Оператор	Число вхождений
=	16
.	19
If	3
foreach	1
return	3
HttpClientHandler	1
HttpClient	1
Client.GetAsync	3
response.Content.ReadAsStringAsync	3
JsonConvert.DeserializeObject	3
date.Date.ToString	1
List	2
name_cities.Add	1
;	21

Продолжение таблицы 29

Оператор	Число вхождений
,	2
{}	7
+	9

Таблица 30 – Словарь операндов

Операнд	Число вхождений
url	4
ClientHandler	2
Client	4
login	2
password	2
cashier	3
response	9
response.IsSuccessStatusCode	3
responseText	6
city_from	2
city_to	2
date	2
flights	3
date_str	2
cities	3
name_cities	3
city	2
"http://192.168.0.227:44307/"	1
null	2
"Cashiers/Login?passport="	1
"yyyy-MM-dd"	1
"Flights/Available?city_from="	1
"&city_to="	1
"&date="	1
"Cities"	1

Из вышеописанных таблиц получаем следующие данные: число уникальных операторов программы n_1 = 17, число уникальных операндов программы n_2 = 25, общее число операторов в программе N_1 = 96, общее число операндов в программе N_2 = 63.

Вычислим некоторые из существующих характеристик на основании полученных данных:

- длина программы (N):

$$N = N_1 + N_2 = 159;$$

- словарь программы (n):

$$n = n_1 + n_2 = 42;$$

- теоретический объем программы (V):

$$V = N \cdot \log_2 N \approx 1162{,}5;$$

- сложность программы (D):

$$D = \left(\frac{n_1}{2}\right) \cdot \left(\frac{N_2}{n_2}\right) \approx 21.4;$$

- оценка интеллектуальных усилий (E):

$$E = D \cdot V = 24877.$$

Вычисленные характеристики показывают, что программный код имеет достаточно высокий уровень качества и удобства для дальнейшего усовершенствования поддержки.

Заключение

В ходе выполнения выпускной квалификационной работы были созданы программные средства поддержки процессов резервирования и продажи авиабилетов.

На этапе анализа предметной области и требований был выполнен анализ программного обеспечения аналогичного назначения, в результате которого были выявлены их основные достоинства и недостатки, произведен анализ целевой аудитории и определены функциональные и нефункциональные требования.

Также было осуществлено планирование разработки о оценки бюджета. По результатам планирования было определено, что реализация проекта займет 52 дня, а предполагаемые затраты будут составлять 303760 рублей, откуда можно сделать вывод об его экономической целесообразности.

На этапе проектирования были спроектированы концептуальная и логическая модели базы данных. Было определено, что база данных будет развернута под управлением СУБД Microsoft SQL Server 2019, серверная часть будет реализована на платформе ASP.NET Core, а мобильное приложение будет написано на языке C# на платформе Xamarin.

На этапе реализации было произведено физическое проектирование базы данных, развертывание базы данных в СУБД Microsoft SQL Server 2019. Также был реализован весь спроектированный функционал.

В перспективе планируется опубликовать разработанное приложение в магазине мобильных приложений для системы Android Google Play, а серверную часть развернуть на облачном сервисе.

Таким образом, в ходе выполнения выпускной квалификационной работы были успешно выполнены все поставленные задачи.

Список использованных источников

1. Демура, Н.А. Цифровизация: сущность и роль в развитии национальной экономики / Н.А. Демура, Н.П. Путинцева // Научный результат. Экономические исследования. – 2021. – №1. – С. 22 – 30.

2. Программа «Цифровая экономика Российской Федерации». Утверждена распоряжением Правительства Российской Федерации от 28 июля 2017 г. № 1632-р. [Электронный ресурс]. URL: http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf (дата обращения: 03.04.2023).

3. Распоряжение Президента Российской Федерации от 9 мая 2017 г. № 203 «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы». [Электронный ресурс]. URL: http://zmedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 03.04.2023).

4. История компании Аэрофлот. [Электронный ресурс]. URL: https://www.aeroflot.ru/ru-ru/about/aeroflot_today/aeroflot_history (дата обращения: 28.03.2023).

5. Мобильное приложение Аэрофлота признано лучшим в мире. [Электронный ресурс]. URL: https://www.aeroflot.ru/ru-ru/news/60445 (дата обращения: 28.03.2023).

6. Годовой отчет Аэрофлота. [Электронный ресурс]. URL: https://ir.aeroflot.ru/fileadmin/user_upload/files/rus/common_info/gosa_doc_2022/Aeroflot_AR21_RUS_book_web.pdf (дата обращения: 28.03.2023).

7. Aviasales – отзывы и описание принципа работы. [Электронный ресурс]. URL: https://miroskop.ru/aviabilety/50-aviasales (дата обращения: 28.03.2023).

8. Авиакасса Яндекс.авиабилеты. Обзор и отзывы. [Электронный ресурс]. URL: https://kajftur.ru/aviakassa/avia.yandex.ru.html (дата обращения: 28.03.2023).

9. Сервис «Яндекс.авиабилеты» стал доступен через мобильное приложение. [Электронный ресурс]. URL: https://3dnews.ru/928991/

servisyandeksaviabileti-stal-dostupen-cherez-mobilnoe-prilogenie (дата обращения: 28.03.2023).

10. Особенности рынка пассажирских и грузовых авиаперевозок в период пандемии. [Электронный ресурс]. URL: https://www.aviastat.ru/statistics/108-osobennosti-rynka-passazhirskih-i-gruzovyh-aviaperevozok-v-period-pandemii (дата обращения: 28.03.2023).

11. Росстат. [Электронный ресурс]. URL: https://rosstat.gov.ru/ (дата обращения: 28.03.2023).

12. Global smartphone shipments market data. [Электронный ресурс]. URL: https://www.counterpointresearch.com/global-smartphone-share/ (дата обращения: 28.03.2023).

13. Статистика версий Android. [Электронный ресурс]. URL: https://4pda.to/2023/01/19/408761/vsyo_grustno_google_obnarodovala_statistiku_versij_android/ (дата обращения: 28.03.2023).

14. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка, 2-е издание / Дж. Рамбо, М. Блаха // СПб.: Питер, 2007. – 544 с.: ил.

15. Крачтен, Ф. Введение в Rational Unified Process, 2-е издание / Ф. Крачтен // М.: Вильямс, 2002. – 240 с.

16. Шаблон MVC. [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/aspnet/core/mvc (дата обращения: 11.04.2023).

17. Документация по ASP.NET. [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/ (дата обращения: 11.04.2023).

18. Документация по Entity Framework. [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/ef (дата обращения: 11.04.2023).

19. Беркун, С. Искусство управления IT-проектами / С. Беркун // СПб.: Питер, 2007. – 400 с.: ил.

20. Разу, М.Л. Управление проектом. Основы проектного управления: учебник, 4-е издание / М.Л. Разу, Т.М. Бронникова, А.М. Лялин // М.: КНОРУС, 2015. – 756 с.

21. Гвоздев, В.Е. Управление программными проектами: учебное пособие для вузов / В.Е. Гвоздев // Москва: Издательство Юрайт, 2023. – 167 с.

22. Ехлаков, Ю.П. Управление программными проектами. Стандарты, модели: Учебное пособие / Ю.П. Ехлаков // СПб.: Лань, 2019. – 244 с.

23. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов // Рн/Д: Феникс, 2018. – 102 с.

24. Документация по Project. [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/project/ (дата обращения: 11.04.2023).

25. Документация по Microsoft SQL. [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/sql/ (дата обращения: 04.05.2023).

26. Основы Linq. [Электронный ресурс]. URL: https://metanit.com/sharp/tutorial/15.1.php (дата обращения: 04.05.2023).

27. Документация по Xamarin. [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/xamarin/ (дата обращения: 04.05.2023).

28. Введение в Web API. [Электронный ресурс]. URL: https://metanit.com/sharp/mvc/12.1.php (дата обращения: 05.05.2023).

29. Руководство по Entity Framework. [Электронный ресурс]. URL: https://metanit.com/sharp/efcore/ (дата обращения: 05.05.2023).

30. Программный код и его метрики. [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/106082/ (дата обращения: 05.05.2023).

31. Аверьянов, А.В. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ / А.В. Аверьянов, И.Н. Кошель, В.В. Кузнецов // Известия высших учебных заведений. Приборостроение. – 2019. – №11. – С. 970 – 975.

32. Полякова, Э.Р. Исследование программного обеспечения встраиваемой вычислительной системы метриками Холстеда / Э.Р. Полякова, А.А. Поляков, А.Л. Шустов, Д.Г. Придачкин // Cloud of Science. – 2020. – №4. – С. 950 – 959.

Приложение А

ГЛОССАРИЙ

(справочное)

Android – операционная система для смартфонов, планшетов, ноутбуков, телевизоров и других устройств, используемая на подавляющем большинстве современных мобильных устройств.

RUP (Rational Unified Process) – методология разработки программного обеспечения, в основе которой лежит итеративная модель разработки.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

СУБД (Система Управления Базой Данных) – специализированное программное обеспечение, предоставляющее программный интерфейс, который дает возможность управлять созданием и использованием баз данных.

SQL (Structured Query Language) – декларативный язык программирования, применяемый для создания и модификации данных в реляционной базе данных.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных, основанный на технологии «клиент-сервер».

REST API – архитектурный стиль взаимодействия компонентов распределенного приложения сети.

Приложение Б

ИСХОДНЫЙ КОД СКРИПТОВ БАЗЫ ДАННЫХ

(рекомендуемое)

Листинг Б.1 – Исходный код скрипта создания таблиц базы данных

```
CREATE TABLE Passenger (
  id_passenger int IDENTITY NOT NULL CHECK(id_passenger > 0 and id_passenger <=
9999),
  telephone varchar(15) NOT NULL UNIQUE CHECK(telephone like replicate('[0-9+]',
len(telephone)) and len(telephone) >= 12 and telephone like '+%'),
  passport varchar(10) NOT NULL UNIQUE CHECK(passport like replicate('[IVXА-Я0-
9]', len(passport))),
  birth_date date NOT NULL CHECK(birth_date between '01.01.1910' and
'31.12.2030'),
  last_name varchar(30) NOT NULL CHECK(last_name like replicate('[а-яА-Я-]',
len(last_name))),
  first_name varchar(15) NOT NULL CHECK(first_name like replicate('[а-яА-Я-]',
len(first_name))),
  middle_name varchar(20) NULL CHECK(middle_name like replicate('[а-яА-Я-]',
len(middle_name))),
  PRIMARY KEY (id_passenger));

CREATE TABLE Ticket (
  id_ticket int IDENTITY NOT NULL CHECK(id_ticket > 0 and id_ticket <= 9999),
  price int NOT NULL CHECK(price > 0 and price <= 999999),
  seat_plane varchar(3) NOT NULL CHECK(seat_plane like replicate('[a-z0-9]',
len(seat_plane))),
  id_passenger int NOT NULL,
  id_buyer int NOT NULL,
  id_schedule int NOT NULL,
  id_cashier int NOT NULL,
  PRIMARY KEY (id_ticket));

CREATE TABLE Booking (
  id_booking int IDENTITY NOT NULL CHECK(id_booking > 0 and id_booking <= 9999),
  seat_plane varchar(3) NOT NULL CHECK(seat_plane like replicate('[a-z0-9]',
len(seat_plane))),
  id_passenger int NOT NULL,
  id_schedule int NOT NULL,
  id_cashier int NOT NULL,
  id_agent int NOT NULL,
  PRIMARY KEY (id_booking));

CREATE TABLE Schedule (
  id_schedule int IDENTITY NOT NULL CHECK(id_schedule > 0 and id_schedule <=
9999),
  departue_time datetime NOT NULL CHECK(departue_time between '01.01.2010' and
'31.12.2040'),
  arrival_time datetime NOT NULL CHECK(arrival_time between '01.01.2010' and
'31.12.2040'),
  number_available_seats int CHECK(number_available_seats > 0 and
number_available_seats <= 999),
  id_status int NOT NULL,
  id_flight int NOT NULL,
  PRIMARY KEY (id_schedule));

CREATE TABLE Flight (
  id_flight int IDENTITY NOT NULL CHECK(id_flight > 0 and id_flight <= 9999),
  base_price int CHECK(base_price > 0 and base_price <= 999999),
  condition int,
  id_airplane int NOT NULL,
  id_route int NOT NULL,
  id_flighttype int NOT NULL,
  PRIMARY KEY (id_flight));
```

```
CREATE TABLE Flighttype (
  id_flighttype int IDENTITY NOT NULL CHECK(id_flighttype > 0 and id_flighttype
<= 9999),
  name_flighttype  varchar(20)  NOT  NULL  UNIQUE  CHECK(name_flighttype  like
replicate('[-а-яА-Яa-zA-Z ]', len(name_flighttype))),
  PRIMARY KEY (id_flighttype));

CREATE TABLE Airoport (
  id_airoport int IDENTITY NOT NULL CHECK(id_airoport > 0 and id_airoport <=
9999),
  name_airoport varchar(50) NOT NULL UNIQUE CHECK(name_airoport like replicate('[-
а-яА-Яa-zA-Z0-9. ]', len(name_airoport))),
  id_city int NOT NULL,
  PRIMARY KEY (id_airoport));

CREATE TABLE City (
  id_city int IDENTITY NOT NULL CHECK(id_city > 0 and id_city <= 9999),
  name_city varchar(20) NOT NULL UNIQUE CHECK(name_city like replicate('[-а-яА-
Яa-zA-Z ]', len(name_city))),
  PRIMARY KEY (id_city));

CREATE TABLE Cashier (
  id_cashier int IDENTITY NOT NULL CHECK(id_cashier > 0 and id_cashier <= 9999),
  last_name  varchar(30)  NOT  NULL  CHECK(last_name  like  replicate('[а-яА-Я-]',
len(last_name))),
  first_name varchar(15) NOT NULL CHECK(first_name like replicate('[а-яА-Я-]',
len(first_name))),
  middle_name  varchar(20)  NULL  CHECK(middle_name  like  replicate('[а-яА-Я-]',
len(middle_name))),
  birth_date  date  NOT  NULL  CHECK(birth_date  between  '01.01.1940'  and
'31.12.2030'),
  adress varchar(100) NOT NULL CHECK(adress like replicate('[а-яА-Я0-9-., ]',
len(adress))),
  telephone varchar(15) NOT NULL UNIQUE CHECK(telephone like replicate('[0-9+]',
len(telephone)) and len(telephone) >= 12 and telephone like '+%'),
  passport varchar(13) NOT NULL UNIQUE CHECK(passport like replicate('[-0-9A-Z]',
len(passport))),
  login varchar(25) NOT NULL UNIQUE CHECK(login like replicate('[-a-zA-Z0-9!@#%]',
len(login))),
  password varchar(30) NOT NULL CHECK(password like replicate('[-a-zA-Z0-9!@#%]',
len(password))),
  PRIMARY KEY (id_cashier));

CREATE TABLE Airplane (
  id_airplane  int  IDENTITY  NOT  NULL  CHECK(id_airplane  >  0  and  id_airplane  <=
9999),
  number_seats int NOT NULL CHECK(number_seats > 0 and number_seats <= 999),
  name_airplane varchar(20) NOT NULL UNIQUE CHECK(name_airplane like replicate('[-
а-яА-Яa-zA-Z0-9 ]', len(name_airplane))),
  PRIMARY KEY (id_airplane));

CREATE TABLE Route (
  id_route int IDENTITY NOT NULL CHECK(id_route > 0 and id_route <= 9999),
  id_airoport_departure int NOT NULL,
  id_airoport_arrival int NOT NULL,
  PRIMARY KEY (id_route));

CREATE TABLE Status (
  id_status int IDENTITY NOT NULL CHECK(id_status > 0 and id_status <= 99),
  name_status varchar(30) NOT NULL UNIQUE CHECK(name_status like replicate('[а-
яА-Я- ]', len(name_status))),
  PRIMARY KEY (id_status));
```

```
ALTER TABLE Ticket ADD CONSTRAINT Buys FOREIGN KEY (id_passenger) REFERENCES
Passenger (id_passenger);
ALTER TABLE Booking ADD CONSTRAINT [Booking b] FOREIGN KEY (id_passenger)
REFERENCES Passenger (id_passenger);
ALTER TABLE Ticket ADD CONSTRAINT [Contains t] FOREIGN KEY (id_schedule)
REFERENCES Schedule (id_schedule);
ALTER TABLE Booking ADD CONSTRAINT [Contains b] FOREIGN KEY (id_schedule)
REFERENCES Schedule (id_schedule);
ALTER TABLE Route ADD CONSTRAINT FKRoute832681 FOREIGN KEY (id_airoport_arrival)
REFERENCES Airoport (id_airoport);
ALTER TABLE Airoport ADD CONSTRAINT Has FOREIGN KEY (id_city) REFERENCES City
(id_city);
ALTER TABLE Ticket ADD CONSTRAINT [Makes out t] FOREIGN KEY (id_cashier) REFERENCES
Cashier (id_cashier);
ALTER TABLE Booking ADD CONSTRAINT [Makes out b] FOREIGN KEY (id_cashier)
REFERENCES Cashier (id_cashier);
ALTER TABLE Flight ADD CONSTRAINT [Performs a] FOREIGN KEY (id_airplane)
REFERENCES Airplane (id_airplane);
ALTER TABLE Route ADD CONSTRAINT Runs FOREIGN KEY (id_airoport_departure)
REFERENCES Airoport (id_airoport);
ALTER TABLE Ticket ADD CONSTRAINT FKTicket963229 FOREIGN KEY (id_buyer) REFERENCES
Passenger (id_passenger);
ALTER TABLE Booking ADD CONSTRAINT FKBooking113727 FOREIGN KEY (id_agent)
REFERENCES Passenger (id_passenger);
ALTER TABLE Flight ADD CONSTRAINT [Performs r] FOREIGN KEY (id_route) REFERENCES
Route (id_route);
ALTER TABLE Flight ADD CONSTRAINT [Has f] FOREIGN KEY (id_flighttype) REFERENCES
Flighttype (id_flighttype);
ALTER TABLE Schedule ADD CONSTRAINT [Has s] FOREIGN KEY (id_status) REFERENCES
Status (id_status);
ALTER TABLE Schedule ADD CONSTRAINT [Contains f] FOREIGN KEY (id_flight)
REFERENCES Flight (id_flight);
```

Листинг Б.2 – Исходный код создания представления PassengersFlightsCount (информация о количестве совершенных полетов пассажиров по месяцам)

```
create view PassengersFlightsCount as
with tbl (num)
as
(select 0 union select 1 union select 2 union select 3 union select 4
union select 5 union select 6 union select 7 union select 8 union
select 9 union select 10 union select 11)
select (tbl.num + 1 + (p.id_passenger - 1) * 12) as "id",
p.last_name "last_name",
p.first_name "first_name",
p.middle_name "middle_name",
DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num, GETDATE())), 0)
"start_date",
DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num + 1, GETDATE())), 0) -
1 "finish_date",
count (t.id_ticket) "flights_count"
from Ticket t
join Schedule s
on s.id_schedule = t.id_schedule
join Status st
on st.id_status = s.id_status
right join Passenger p
on p.id_passenger = t.id_passenger
cross join tbl
where st.name_status = 'Выполнен'
```

```
and s.departue_time >= DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num,
GETDATE())), 0)
and s.departue_time < DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num +
1, GETDATE())), 0)
group by tbl.num, p.id_passenger, p.last_name, p.first_name, p.middle_name,
DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num, GETDATE())), 0),
DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num + 1, GETDATE())), 0)
union
select (tbl.num + 1 + (p.id_passenger - 1) * 12),
p.last_name,
p.first_name,
p.middle_name,
DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num, GETDATE())), 0),
DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num + 1, GETDATE())), 0) -
1,
0
from Ticket t
join Schedule s
on s.id_schedule = t.id_schedule
join Status st
on st.id_status = s.id_status
right join Passenger p
on p.id_passenger = t.id_passenger
cross join tbl
where not exists
(select 1
from Ticket t
join Schedule s
on s.id_schedule = t.id_schedule
join Status st
on st.id_status = s.id_status
right join Passenger p0
on p.id_passenger = t.id_passenger
cross join tbl tb
where st.name_status = 'Выполнен'
and s.departue_time >= DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num,
GETDATE())), 0)
and s.departue_time < DATEADD(month, DATEDIFF(month, 0, DATEADD(month, -tbl.num +
1, GETDATE())), 0)
and p0.id_passenger = p.id_passenger
and tb.num = tbl.num);
```

Листинг Б.3 – Исходный код создания представления AverageFlightLoad (информация о средней загруженности рейсов по дням)

```
create view AverageFlightLoad as
with tbl (num)
as
(select 0 union select 1 union select 2 union select 3 union select 4
union select 5 union select 6)
select tbl.num as "id",
convert (date, GETDATE() - tbl.num) "date",
1 - convert (float, sum(s.number_available_seats)) / sum(a.number_seats) "load"
from Schedule s join Flight f
on s.id_flight = f.id_flight
join Airplane a
on f.id_airplane = a.id_airplane
right join tbl
on convert (date, s.departue_time) = convert (date, GETDATE() - tbl.num)
and s.number_available_seats is not null
group by tbl.num, convert (date, GETDATE() - tbl.num);
```

Листинг Б.4 – Исходный код создания хранимой функции fullAge (получение целого числа лет, прошедшего между двумя датами)

```
create function fullAge
(@startdate date, @enddate datetime)
returns int
as
begin
return (datediff(year, @startdate, @enddate) -
      case
            when month(@startdate) < month(@enddate)
            then 0
            when month(@startdate) > month(@enddate)
            then 1
            when day(@startdate) > day(@enddate)
            then 1
            else 0
      end)
end;
```

Листинг Б.5 – Исходный код создания хранимой процедуры sell_ticket_new (продажа билета)

```
create procedure sell_ticket_new
(@seat_plane varchar(3), @id_passenger int, @id_buyer int, @id_schedule int,
@id_cashier int, @price int output)
as
begin
      declare @count_seats int;
      select @count_seats = number_available_seats from Schedule where id_schedule
= @id_schedule;
      if @count_seats < 1
      begin
            set @price = -1;
            return;
      end;

      select @price = case when dbo.fullAge (p.birth_date, getdate()) between 0
and 1 then 0
      when dbo.fullAge (p.birth_date, getdate()) between 2 and 11 then 0.5 *
f.base_price
      else f.base_price end
      from Schedule s join Flight f
      on f.id_flight = s.id_flight
      join Passenger p
      on p.id_passenger = @id_passenger
      where @id_schedule = s.id_schedule;

      insert into Ticket (price, seat_plane, id_passenger, id_buyer, id_schedule,
id_cashier)
      values
      (@price, @seat_plane, @id_passenger, @id_buyer, @id_schedule, @id_cashier);
end;
```

Листинг Б.6 – Исходный код создания хранимой процедуры book_ticket (бронирование билета)

```
create procedure book_ticket
(@seat_plane varchar(3), @id_passenger int, @id_agent int, @id_schedule int,
@id_cashier int)
as
begin
	declare @count_seats int;
	select @count_seats = number_available_seats from Schedule where id_schedule
= @id_schedule;
	if @count_seats < 1
		return -1;
	insert into Booking (seat_plane, id_passenger, id_agent, id_schedule,
id_cashier)
	values
	(@seat_plane, @id_passenger, @id_agent, @id_schedule, @id_cashier);
	return @count_seats - 1;
end;
```

Листинг Б.7 – Исходный код создания хранимой процедуры add_passenger (добавление пассажира или покупателя)

```
create procedure add_passenger
(@telephone varchar(15), @passport varchar(10), @birth_date date, @last_name
varchar(30), @first_name varchar(15), @middle_name varchar(20))
as
begin
	insert into Passenger (telephone, passport, birth_date, last_name,
first_name, middle_name)
	values
	(@telephone, @passport, @birth_date, @last_name, @first_name,
@middle_name);
end;
```

Листинг Б.8 – Исходный код создания хранимой процедуры change_details_passenger (изменение данных пассажира или покупателя)

```
create procedure change_details_passenger
(@id_passenger int, @telephone varchar(15), @passport varchar(10), @birth_date
date, @last_name varchar(30), @first_name varchar(15), @middle_name varchar(20))
as
begin
	update Passenger
	set telephone = @telephone,
	passport = @passport,
	birth_date = @birth_date,
	last_name = @last_name,
	first_name = @first_name,
	middle_name = @middle_name
	where id_passenger = @id_passenger;
end;
```

Листинг Б.9 – Исходный код создания хранимой процедуры redeem_booking_new (выкуп брони)

```
create procedure redeem_booking_new
(@id_booking int,
@price int output)
as
begin
      declare  @seat_plane  varchar(3),  @id_passenger  int,  @id_buyer  int,
@id_schedule int, @id_cashier int;
      select @seat_plane = b.seat_plane,
      @id_passenger = b.id_passenger,
      @id_buyer = b.id_agent,
      @id_schedule = b.id_schedule,
      @id_cashier = b.id_cashier,
      @price = case when dbo.fullAge (p.birth_date, getdate()) between 0 and 1
then 0
      when dbo.fullAge (p.birth_date, getdate()) between 2 and 11 then 0.5 *
f.base_price
      else f.base_price end
      from Booking b join Schedule s
      on b.id_schedule = s.id_schedule
      join Flight f
      on f.id_flight = s.id_flight
      join Passenger p
      on p.id_passenger = b.id_passenger
      where @id_booking = b.id_booking;

      insert into Ticket(price, seat_plane, id_passenger, id_buyer, id_schedule,
id_cashier)
      values
      (@price, @seat_plane, @id_passenger, @id_buyer, @id_schedule, @id_cashier);

      delete from Booking
      where id_booking = @id_booking;
end;
```

Листинг Б.10 – Исходный код создания хранимой процедуры cancel_booking (снятие брони)

```
create procedure cancel_booking
(@id_booking int)
as
begin
      delete from Booking
      where id_booking = @id_booking;
end;
```

Листинг Б.11 – Исходный код создания хранимой процедуры redeem_multiple_booking_new (выкуп множества билетов)

```
create procedure redeem_multiple_booking_new
(@id_schedule int, @id_agent int, @sum_price int output)
as
begin
```

```
      declare @seat_plane varchar(3), @id_passenger int, @id_buyer int,
@id_cashier int, @id_booking int, @prev_id_booking int, @price int;
      set @id_booking = 0;
      set @prev_id_booking = 0;
      set @sum_price = 0;

      while (1 = 1)
      begin
      set @prev_id_booking = @id_booking;
      select top 1 @id_booking = b.id_booking,
      @seat_plane = b.seat_plane,
      @id_passenger = b.id_passenger,
      @id_buyer = b.id_agent,
      @id_cashier = b.id_cashier,
      @price = case when dbo.fullAge (p.birth_date, getdate()) between 0 and 1
then 0
      when dbo.fullAge (p.birth_date, getdate()) between 2 and 11 then 0.5 *
f.base_price
      else f.base_price end
      from Booking b join Schedule s
      on b.id_schedule = s.id_schedule
      join Flight f
      on f.id_flight = s.id_flight
      join Passenger p
      on p.id_passenger = b.id_passenger
      where b.id_schedule = @id_schedule
      and b.id_agent = @id_agent;

      if (@id_booking = @prev_id_booking)
            return @sum_price;

      insert into Ticket(price, seat_plane, id_passenger, id_buyer, id_schedule,
id_cashier)
      values
      (@price, @seat_plane, @id_passenger, @id_buyer, @id_schedule, @id_cashier);

      delete from Booking
      where id_booking = @id_booking;
      set @sum_price = @sum_price + @price;
      end
end;
```

Листинг Б.12 – Исходный код создания хранимой процедуры cancel_multiple_booking (снятие множества броней)

```
create procedure cancel_multiple_booking
(@id_schedule int, @id_agent int)
as
begin
      delete from Booking
      where id_schedule = @id_schedule
      and id_agent = @id_agent;
end;
```

Листинг Б.13 – Исходный код создания хранимой процедуры refund_ticket_new (возврат билета)

```
create procedure refund_ticket_new
(@id_ticket int,
```

```
@price int output)
as
begin
      select @price = price from Ticket where id_ticket = @id_ticket;
      delete from Ticket
      where id_ticket = @id_ticket;
end;
```

Листинг Б.14 – Исходный код создания хранимой процедуры refund_multiple_ticket (возврат множества билетов)

```
create procedure refund_multiple_ticket
(@id_schedule int, @id_buyer int, @sum_price int output)
as
begin
      declare @id_ticket int, @prev_id_ticket int, @price int;
      set @id_ticket = 0;
      set @prev_id_ticket = 0;
      set @sum_price = 0;

      while (1 = 1)
      begin
      set @prev_id_ticket = @id_ticket;
      select top 1 @id_ticket = id_ticket,
      @price = price
      from Ticket
      where id_schedule = @id_schedule
      and id_buyer = @id_buyer;

      if (@id_ticket = @prev_id_ticket)
            return @sum_price;

      delete from Ticket
      where id_ticket = @id_ticket;
      set @sum_price = @sum_price + @price;
      end
end;
```

Листинг Б.15 – Исходный код создания хранимой процедуры get_flights (получение списка рейсов по заданным параметрам)

```
create procedure get_flights
@city_from varchar(20),
@city_to varchar(20),
@date date
as
select s.id_schedule,
ad.name_airoport name_airoport_from,
aa.name_airoport name_airoport_to,
s.departue_time,
s.arrival_time,
air.name_airplane,
f.base_price,
s.number_available_seats
from Schedule s join Flight f
on s.id_flight = f.id_flight
join Route r
on r.id_route = f.id_route
join Airoport ad
```

```
on ad.id_airoport = r.id_airoport_departure
join Airoport aa
on aa.id_airoport = r.id_airoport_arrival
join Airplane air
on air.id_airplane = f.id_airplane
join City cd
on cd.id_city = ad.id_city
join City ca
on ca.id_city = aa.id_city
join Flighttype ft
on ft.id_flighttype = f.id_flighttype
join Status st
on st.id_status = s.id_status
where  ft.name_flighttype not in ('Грузоперевозки', 'Специальный рейс')
and (st.name_status = 'По расписанию' or st.name_status = 'Задерживается')
and cd.name_city = @city_from
and ca.name_city = @city_to
and convert(date, s.departue_time) = @date;
```

Приложение В

ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

(рекомендуемое)

Листинг В.1 – Файл Cashier.cs

```
namespace Airoport.Models
{
    using System.Collections.Generic;

    public partial class Cashier
    {
        public Cashier()
        {
            this.Booking = new HashSet<Booking>();
            this.Ticket = new HashSet<Ticket>();
        }

        public int id_cashier { get; set; }
        public string last_name { get; set; }
        public string first_name { get; set; }
        public string middle_name { get; set; }
        public System.DateTime birth_date { get; set; }
        public string adress { get; set; }
        public string telephone { get; set; }
        public string passport { get; set; }
        public string login { get; set; }
        public string password { get; set; }
        public virtual ICollection<Booking> Booking { get; set; }
        public virtual ICollection<Ticket> Ticket { get; set; }
    }
}
```

Листинг В.2 – Файл Ticket.cs

```
namespace Airoport.Models
{
    public partial class Ticket
    {
        public int id_ticket { get; set; }
        public int price { get; set; }
        public string seat_plane { get; set; }
        public int id_passenger { get; set; }
        public int id_buyer { get; set; }
        public int id_schedule { get; set; }
        public int id_cashier { get; set; }

        public virtual Cashier Cashier { get; set; }
        public virtual Passenger Passenger { get; set; }
        public virtual Passenger Passenger1 { get; set; }
        public virtual Schedule Schedule { get; set; }
    }
}
```

Листинг В.3 – Файл Booking.cs

```
namespace Airoport.Models
{
    public partial class Booking
    {
        public int id_booking { get; set; }
        public string seat_plane { get; set; }
        public int id_passenger { get; set; }
        public int id_schedule { get; set; }
```

```
        public int id_cashier { get; set; }
        public int id_agent { get; set; }

        public virtual Passenger Passenger { get; set; }
        public virtual Schedule Schedule { get; set; }
        public virtual Passenger Passenger1 { get; set; }
        public virtual Cashier Cashier { get; set; }
    }
}
```

Листинг В.4 – Файл AverageFlightLoad.cs

```
namespace Airoport.Models
{
    using System;

    public partial class AverageFlightLoad
    {
        public int id { get; set; }
        public Nullable<System.DateTime> date { get; set; }
        public Nullable<double> load { get; set; }
    }
}
```

Листинг В.5 – Файл PassengersFlightsCount.cs

```
namespace Airoport.Models
{
    using System;

    public partial class PassengersFlightsCount
    {
        public int id { get; set; }
        public string last_name { get; set; }
        public string first_name { get; set; }
        public string middle_name { get; set; }
        public Nullable<System.DateTime> start_date { get; set; }
        public Nullable<System.DateTime> finish_date { get; set; }
        public Nullable<int> flights_count { get; set; }
    }
}
```

Листинг В.6 – Файл BookingsController.cs

```
using System.Data.Entity.Core.Objects;
using System.Linq;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class BookingsController : Controller
    {
        private AiroportEntities db = new AiroportEntities();

        // POST: Bookings/Create
        [HttpPost]
        public ActionResult Create(string seat_plane, int id_passenger, int
id_buyer, int id_schedule, int id_cashier)
        {
```

```
            int count = db.book_ticket(seat_plane, id_passenger, id_buyer,
id_schedule, id_cashier);
            if (count > 0)
            {
                return new EmptyResult();
            }
            else
            {
                return HttpNotFound();
            }
        }

        // DELETE: Bookings/Cancel/5
        [HttpDelete, ActionName("Cancel")]
        public ActionResult CancelConfirmed(int id)
        {
            int count = db.cancel_booking(id);
            if (count > 0)
            {
                return new EmptyResult();
            }
            else
            {
                return HttpNotFound();
            }
        }

        // POST: Bookings/Redeem
        [HttpPost, ActionName("Redeem")]
        public ActionResult RedeemConfirmed(int id)
        {
            try
            {
                ObjectParameter price = new ObjectParameter("price",
typeof(int));
                int count = db.redeem_booking_new(id, price);
                if (count > 0)
                {
                    return Json(price.Value, JsonRequestBehavior.AllowGet);
                }
                else
                {
                    return HttpNotFound();
                }
            }
            catch
            {
                return HttpNotFound();
            }
        }

        // DELETE: Bookings/CancelMultiple/
        [HttpDelete, ActionName("CancelMultiple")]
        public ActionResult CancelMultipleConfirmed(int id_schedule, string
passport)
        {
            Passenger passenger = db.Passenger.FirstOrDefault(p => p.passport ==
passport);
            int count = db.cancel_multiple_booking(id_schedule,
passenger.id_passenger);
            if (count > 0)
            {
                return Json(count - 1, JsonRequestBehavior.AllowGet);
            }
```

```
            else
            {
                return HttpNotFound();
            }
        }

        // POST: Bookings/RedeemMultiple
        [HttpPost, ActionName("RedeemMultiple")]
        public ActionResult RedeemMultipleConfirmed(int id_schedule, string
passport)
        {
            try
            {
                Passenger passenger = db.Passenger.FirstOrDefault(p =>
p.passport == passport);
                ObjectParameter sum_price = new ObjectParameter("sum_price",
typeof(int));
                int count = db.redeem_multiple_booking_new(id_schedule,
passenger.id_passenger, sum_price);
                if (count > 0)
                {
                    return Json((count / 4, (int)sum_price.Value),
JsonRequestBehavior.AllowGet);
                }
                else
                {
                    return HttpNotFound();
                }
            }
            catch
            {
                return HttpNotFound();
            }
        }
    }
}
```

Листинг В.7 – Файл CashiersController.cs

```
using System.Linq;
using System.Net;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class CashiersController : Controller
    {
        private AiroportEntities db = new AiroportEntities();

        // GET: Cashiers/Login
        public ActionResult Login(string login, string password)
        {
            Cashier cas = db.Cashier.FirstOrDefault(c => c.login == login &&
c.password == password);
            if (cas != null)
            {
                cas.Booking.Clear();
                cas.Ticket.Clear();
                return Json(cas, JsonRequestBehavior.AllowGet);
            }
            return new HttpUnauthorizedResult();
```

```
        }

        // GET: Cashiers/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Cashier cashier = db.Cashier.Find(id);
            if (cashier == null)
            {
                return HttpNotFound();
            }
            cashier.Booking.Clear();
            cashier.Ticket.Clear();
            return Json(cashier, JsonRequestBehavior.AllowGet);
        }
    }
}
```

Листинг В.8 – Файл CitiesController.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class CitiesController : Controller
    {
        private AiroportEntities db = new AiroportEntities();

        // GET: Cities
        public ActionResult Index()
        {
            List<City> cities = db.City.ToList();
            foreach(City city in cities)
            {
                city.Airoport.Clear();
            }
            return Json(db.City.ToList(), JsonRequestBehavior.AllowGet);
        }
    }
}
```

Листинг В.9 – Файл FlightsController.cs

```
using System;
using System.Linq;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class FlightsController : Controller
    {
        private AiroportEntities db = new AiroportEntities();

        // GET: Flights/Available/
```

```
        public ActionResult Available(string city_from, string city_to, DateTime
date)
        {
            var flights = db.get_flights(city_from, city_to, date);
            var listFlights = flights.ToList();
            if (listFlights.Count() == 0)
            {
                return HttpNotFound();
            }
            return Json(listFlights, JsonRequestBehavior.AllowGet);
        }

        // GET: Flights/AverageLoads
        public ActionResult AverageLoads()
        {
            return Json(db.AverageFlightLoad.ToList(),
JsonRequestBehavior.AllowGet);
        }
    }
}
```

Листинг В.10 – Файл PassengersController.cs

```
using System;
using System.Linq;
using System.Net;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class PassengersController : Controller
    {
        private AiroportEntities db = new AiroportEntities();

        // GET: Passengers/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Passenger passenger = db.Passenger.Find(id);
            if (passenger == null)
            {
                return HttpNotFound();
            }
            passenger.Ticket.Clear();
            passenger.Ticket1.Clear();
            passenger.Booking.Clear();
            passenger.Booking1.Clear();
            return Json(passenger, JsonRequestBehavior.AllowGet);
        }

        // GET: Passengers/Id/1234567890
        public ActionResult Id(string passport)
        {
            Passenger passenger = db.Passenger.FirstOrDefault(p => p.passport ==
passport);
            if (passenger == null)
            {
                return HttpNotFound();
```

```
            }
            return Json(passenger.id_passenger, JsonRequestBehavior.AllowGet);
        }

        // POST: Passengers/Create
        [HttpPost]
        public ActionResult Create(string telephone, string passport, DateTime
birth_date, string last_name, string first_name, string middle_name)
        {
            db.add_passenger(telephone, passport, birth_date, last_name,
first_name, middle_name);
            return new EmptyResult();
        }

        // PUT: Passengers/Change
        [HttpPut]
        public ActionResult Change(int id_passenger, string telephone, string
passport, DateTime birth_date, string last_name, string first_name, string
middle_name)
        {
            db.change_details_passenger(id_passenger, telephone, passport,
birth_date, last_name, first_name, middle_name);
            return new EmptyResult();
        }

        // GET: Passengers/FlightsCounts
        public ActionResult FlightsCounts()
        {
            return Json(db.PassengersFlightsCount.ToList(),
JsonRequestBehavior.AllowGet);
        }
    }
}
```

Листинг В.11 – Файл SchedulesController.cs

```
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Net;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class SchedulesController : Controller
    {
        private AiroportEntities db = new AiroportEntities();

        // GET: Schedules/GetSeats/5
        public ActionResult GetSeats(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            var numberSeats = from s in db.Schedule
                              join f in db.Flight
                              on s.id_flight equals f.id_flight
                              join air in db.Airplane
                              on f.id_airplane equals air.id_airplane
                              where (s.id_schedule == id)
```

```
                                select air.number_seats;
            List<string> seats = new List<string>();
            for (int i = 0; i < numberSeats.First(); i++)
            {
                char letter;
                if (i % 4 == 0)
                {
                    letter = 'a';
                }
                else
                    if (i % 4 == 1)
                {
                    letter = 'b';
                }
                else
                    if (i % 4 == 2)
                {
                    letter = 'c';
                }
                else
                {
                    letter = 'd';
                }

                string number = (i / 4 + 1).ToString();
                if (number.Length < 2)
                {
                    number = '0' + number;
                }

                seats.Add(number + letter);
            }

            var occupiedSeats = (from t in db.Ticket
                                 where (t.id_schedule == id)
                                 select t.seat_plane)
                                 .Union
                                 (from b in db.Booking
                                  where (b.id_schedule == id)
                                  select b.seat_plane);
            foreach (var occupiedSeat in occupiedSeats)
            {
                seats.Remove(occupiedSeat);
            }
            return Json(seats, JsonRequestBehavior.AllowGet);
        }
    }
}
```

Листинг В.12 – Файл TicketsController.cs

```
using System.Data.Entity.Core.Objects;
using System.Linq;
using System.Web.Mvc;
using Airoport.Models;

namespace Airoport.Controllers
{
    public class TicketsController : Controller
    {
        private AiroportEntities db = new AiroportEntities();
```

```
        // POST: Tickets/Create
        [HttpPost]
        public ActionResult Create(string seat_plane, int id_passenger, int
id_buyer, int id_schedule, int id_cashier)
        {
            ObjectParameter price = new ObjectParameter("price", typeof(int));
            int count = db.sell_ticket_new(seat_plane, id_passenger, id_buyer,
id_schedule, id_cashier, price);
            if (count > 0)
            {
                return Json(price.Value, JsonRequestBehavior.AllowGet);
            }
            else
            {
                return HttpNotFound();
            }
        }

        // DELETE: Tickets/Refund/5
        [HttpDelete, ActionName("Refund")]
        public ActionResult DeleteConfirmed(int id)
        {
            try
            {
                ObjectParameter price = new ObjectParameter("price",
typeof(int));
                int count = db.refund_ticket_new(id, price);
                if (count > 0)
                {
                    return Json(price.Value, JsonRequestBehavior.AllowGet);
                }
                else
                {
                    return HttpNotFound();
                }
            }
            catch
            {
                return HttpNotFound();
            }
        }

        // DELETE: Tickets/RefundMultiple/5
        [HttpDelete, ActionName("RefundMultiple")]
        public ActionResult DeleteMultipleConfirmed(int id_schedule, string
passport)
        {
            try
            {
                Passenger passenger = db.Passenger.FirstOrDefault(p =>
p.passport == passport);
                ObjectParameter sum_price = new ObjectParameter("sum_price",
typeof(int));
                int count = db.refund_multiple_ticket(id_schedule,
passenger.id_passenger, sum_price);
                if (count > 0)
                {
                    return Json((count / 2, (int)sum_price.Value),
JsonRequestBehavior.AllowGet);
                }
                else
                {
                    return HttpNotFound();
                }
```

```
            }
            catch
            {
                return HttpNotFound();
            }
        }
    }
}
```

Листинг В.13 – Файл MsContext.context.cs

```
namespace Airoport.Models
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Data.Entity.Core.Objects;

    public partial class AiroportEntities : DbContext
    {
        public AiroportEntities()
            : base("name=AiroportEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Airoport> Airoport { get; set; }
        public virtual DbSet<Airplane> Airplane { get; set; }
        public virtual DbSet<Booking> Booking { get; set; }
        public virtual DbSet<Cashier> Cashier { get; set; }
        public virtual DbSet<City> City { get; set; }
        public virtual DbSet<Flight> Flight { get; set; }
        public virtual DbSet<Flighttype> Flighttype { get; set; }
        public virtual DbSet<Passenger> Passenger { get; set; }
        public virtual DbSet<Route> Route { get; set; }
        public virtual DbSet<Schedule> Schedule { get; set; }
        public virtual DbSet<Status> Status { get; set; }
        public virtual DbSet<Ticket> Ticket { get; set; }
        public virtual DbSet<PassengersFlightsCount> PassengersFlightsCount {
get; set; }
        public virtual DbSet<AverageFlightLoad> AverageFlightLoad { get; set; }

        public virtual int add_passenger(string telephone, string passport,
Nullable<System.DateTime> birth_date, string last_name, string first_name,
string middle_name)
        {
            var telephoneParameter = telephone != null ?
                new ObjectParameter("telephone", telephone) :
                new ObjectParameter("telephone", typeof(string));

            var passportParameter = passport != null ?
                new ObjectParameter("passport", passport) :
                new ObjectParameter("passport", typeof(string));

            var birth_dateParameter = birth_date.HasValue ?
                new ObjectParameter("birth_date", birth_date) :
                new ObjectParameter("birth_date", typeof(System.DateTime));
```

```
            var last_nameParameter = last_name != null ?
                new ObjectParameter("last_name", last_name) :
                new ObjectParameter("last_name", typeof(string));

            var first_nameParameter = first_name != null ?
                new ObjectParameter("first_name", first_name) :
                new ObjectParameter("first_name", typeof(string));

            var middle_nameParameter = middle_name != null ?
                new ObjectParameter("middle_name", middle_name) :
                new ObjectParameter("middle_name", typeof(string));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("add_passenger",
telephoneParameter, passportParameter, birth_dateParameter, last_nameParameter,
first_nameParameter, middle_nameParameter);
        }

        public virtual int book_ticket(string seat_plane, Nullable<int>
id_passenger, Nullable<int> id_agent, Nullable<int> id_schedule, Nullable<int>
id_cashier)
        {
            var seat_planeParameter = seat_plane != null ?
                new ObjectParameter("seat_plane", seat_plane) :
                new ObjectParameter("seat_plane", typeof(string));

            var id_passengerParameter = id_passenger.HasValue ?
                new ObjectParameter("id_passenger", id_passenger) :
                new ObjectParameter("id_passenger", typeof(int));

            var id_agentParameter = id_agent.HasValue ?
                new ObjectParameter("id_agent", id_agent) :
                new ObjectParameter("id_agent", typeof(int));

            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
                new ObjectParameter("id_schedule", typeof(int));

            var id_cashierParameter = id_cashier.HasValue ?
                new ObjectParameter("id_cashier", id_cashier) :
                new ObjectParameter("id_cashier", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("book_ticket",
seat_planeParameter, id_passengerParameter, id_agentParameter,
id_scheduleParameter, id_cashierParameter);
        }

        public virtual int cancel_booking(Nullable<int> id_booking)
        {
            var id_bookingParameter = id_booking.HasValue ?
                new ObjectParameter("id_booking", id_booking) :
                new ObjectParameter("id_booking", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("cancel_booking",
id_bookingParameter);
        }

        public virtual int cancel_multiple_booking(Nullable<int> id_schedule,
Nullable<int> id_agent)
        {
            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
```

```
                new ObjectParameter("id_schedule", typeof(int));

            var id_agentParameter = id_agent.HasValue ?
                new ObjectParameter("id_agent", id_agent) :
                new ObjectParameter("id_agent", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("cancel_multiple_boo
king", id_scheduleParameter, id_agentParameter);
        }

        public virtual int redeem_booking(Nullable<int> id_booking)
        {
            var id_bookingParameter = id_booking.HasValue ?
                new ObjectParameter("id_booking", id_booking) :
                new ObjectParameter("id_booking", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("redeem_booking",
id_bookingParameter);
        }

        public virtual int redeem_multiple_booking(Nullable<int> id_schedule,
Nullable<int> id_agent)
        {
            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
                new ObjectParameter("id_schedule", typeof(int));

            var id_agentParameter = id_agent.HasValue ?
                new ObjectParameter("id_agent", id_agent) :
                new ObjectParameter("id_agent", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("redeem_multiple_boo
king", id_scheduleParameter, id_agentParameter);
        }

        public virtual int refund_ticket(Nullable<int> id_ticket)
        {
            var id_ticketParameter = id_ticket.HasValue ?
                new ObjectParameter("id_ticket", id_ticket) :
                new ObjectParameter("id_ticket", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("refund_ticket",
id_ticketParameter);
        }

        public virtual int sell_ticket(string seat_plane, Nullable<int>
id_passenger, Nullable<int> id_buyer, Nullable<int> id_schedule, Nullable<int>
id_cashier)
        {
            var seat_planeParameter = seat_plane != null ?
                new ObjectParameter("seat_plane", seat_plane) :
                new ObjectParameter("seat_plane", typeof(string));

            var id_passengerParameter = id_passenger.HasValue ?
                new ObjectParameter("id_passenger", id_passenger) :
                new ObjectParameter("id_passenger", typeof(int));

            var id_buyerParameter = id_buyer.HasValue ?
                new ObjectParameter("id_buyer", id_buyer) :
                new ObjectParameter("id_buyer", typeof(int));
```

```
            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
                new ObjectParameter("id_schedule", typeof(int));

            var id_cashierParameter = id_cashier.HasValue ?
                new ObjectParameter("id_cashier", id_cashier) :
                new ObjectParameter("id_cashier", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sell_ticket",
seat_planeParameter, id_passengerParameter, id_buyerParameter,
id_scheduleParameter, id_cashierParameter);
        }

        public virtual int redeem_booking_new(Nullable<int> id_booking,
ObjectParameter price)
        {
            var id_bookingParameter = id_booking.HasValue ?
                new ObjectParameter("id_booking", id_booking) :
                new ObjectParameter("id_booking", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("redeem_booking_new"
, id_bookingParameter, price);
        }

        public virtual int redeem_multiple_booking_new(Nullable<int>
id_schedule, Nullable<int> id_agent, ObjectParameter sum_price)
        {
            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
                new ObjectParameter("id_schedule", typeof(int));

            var id_agentParameter = id_agent.HasValue ?
                new ObjectParameter("id_agent", id_agent) :
                new ObjectParameter("id_agent", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("redeem_multiple_boo
king_new", id_scheduleParameter, id_agentParameter, sum_price);
        }

        public virtual int refund_ticket_new(Nullable<int> id_ticket,
ObjectParameter price)
        {
            var id_ticketParameter = id_ticket.HasValue ?
                new ObjectParameter("id_ticket", id_ticket) :
                new ObjectParameter("id_ticket", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("refund_ticket_new",
id_ticketParameter, price);
        }

        public virtual ObjectResult<get_flights_Result> get_flights(string
city_from, string city_to, Nullable<System.DateTime> date)
        {
            var city_fromParameter = city_from != null ?
                new ObjectParameter("city_from", city_from) :
                new ObjectParameter("city_from", typeof(string));

            var city_toParameter = city_to != null ?
                new ObjectParameter("city_to", city_to) :
```

```
                new ObjectParameter("city_to", typeof(string));

            var dateParameter = date.HasValue ?
                new ObjectParameter("date", date) :
                new ObjectParameter("date", typeof(System.DateTime));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<get_flights_Result>(
"get_flights", city_fromParameter, city_toParameter, dateParameter);
        }

        public virtual int sell_ticket_new(string seat_plane, Nullable<int>
id_passenger, Nullable<int> id_buyer, Nullable<int> id_schedule, Nullable<int>
id_cashier, ObjectParameter price)
        {
            var seat_planeParameter = seat_plane != null ?
                new ObjectParameter("seat_plane", seat_plane) :
                new ObjectParameter("seat_plane", typeof(string));

            var id_passengerParameter = id_passenger.HasValue ?
                new ObjectParameter("id_passenger", id_passenger) :
                new ObjectParameter("id_passenger", typeof(int));

            var id_buyerParameter = id_buyer.HasValue ?
                new ObjectParameter("id_buyer", id_buyer) :
                new ObjectParameter("id_buyer", typeof(int));

            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
                new ObjectParameter("id_schedule", typeof(int));

            var id_cashierParameter = id_cashier.HasValue ?
                new ObjectParameter("id_cashier", id_cashier) :
                new ObjectParameter("id_cashier", typeof(int));
                return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sell_ticket_new",
seat_planeParameter, id_passengerParameter, id_buyerParameter,
id_scheduleParameter, id_cashierParameter, price);
        }

        public virtual int change_details_passenger(Nullable<int> id_passenger,
string telephone, string passport, Nullable<System.DateTime> birth_date, string
last_name, string first_name, string middle_name)
        {
            var id_passengerParameter = id_passenger.HasValue ?
                new ObjectParameter("id_passenger", id_passenger) :
                new ObjectParameter("id_passenger", typeof(int));

            var telephoneParameter = telephone != null ?
                new ObjectParameter("telephone", telephone) :
                new ObjectParameter("telephone", typeof(string));

            var passportParameter = passport != null ?
                new ObjectParameter("passport", passport) :
                new ObjectParameter("passport", typeof(string));

            var birth_dateParameter = birth_date.HasValue ?
                new ObjectParameter("birth_date", birth_date) :
                new ObjectParameter("birth_date", typeof(System.DateTime));

            var last_nameParameter = last_name != null ?
                new ObjectParameter("last_name", last_name) :
                new ObjectParameter("last_name", typeof(string));
```

```
            var first_nameParameter = first_name != null ?
                new ObjectParameter("first_name", first_name) :
                new ObjectParameter("first_name", typeof(string));

            var middle_nameParameter = middle_name != null ?
                new ObjectParameter("middle_name", middle_name) :
                new ObjectParameter("middle_name", typeof(string));
                return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("change_details_pass
enger", id_passengerParameter, telephoneParameter, passportParameter,
birth_dateParameter, last_nameParameter, first_nameParameter,
middle_nameParameter);
        }

        public virtual int refund_multiple_ticket(Nullable<int> id_schedule,
Nullable<int> id_buyer, ObjectParameter sum_price)
        {
            var id_scheduleParameter = id_schedule.HasValue ?
                new ObjectParameter("id_schedule", id_schedule) :
                new ObjectParameter("id_schedule", typeof(int));

            var id_buyerParameter = id_buyer.HasValue ?
                new ObjectParameter("id_buyer", id_buyer) :
                new ObjectParameter("id_buyer", typeof(int));

            return
((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("refund_multiple_tic
ket", id_scheduleParameter, id_buyerParameter, sum_price);
        }
    }
}
```

Приложение Г

ИСХОДНЫЙ КОД МОБИЛЬНОГО ПРИЛОЖЕНИЯ

(рекомендуемое)

Листинг Г.1 – Файл AppShell.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:view="clr-namespace:Airoport.Views"
             mc:Ignorable="d"
             x:Class="Airoport.AppShell">
    <TabBar>
        <ShellContent Title="Продажа"
                      Icon="sell.png"
                      Route="SellBook"
                      ContentTemplate="{DataTemplate view:SellBook}" />
        <ShellContent Title="Выкуп брони"
                      Icon="redeem.png"
                      Route="RedeemCancelBooking"
                      ContentTemplate="{DataTemplate view:RedeemCancelBooking}"
/>
        <ShellContent Title="Возврат билета"
                      Icon="ticket.png"
                      Route="RefundTicket"
                      ContentTemplate="{DataTemplate view:RefundTicket}" />
        <ShellContent Title="Другое"
                      Icon="other.png"
                      Route="Other"
                      ContentTemplate="{DataTemplate view:Other}" />
    </TabBar>

    <TabBar>
        <ShellContent Title="Ранее учтенный"
                      Icon="old_passenger.png"
                      Route="AddAccountedPassenger"
                      ContentTemplate="{DataTemplate
view:AddAccountedPassenger}" />
        <ShellContent Title="Новый"
                      Icon="new_passenger.png"
                      Route="AddPassenger"
                      ContentTemplate="{DataTemplate view:AddPassenger}" />
    </TabBar>

    <TabBar>
        <ShellContent Title="Ранее учтенный"
                      Icon="old_passenger.png"
                      Route="AddAccountedBuyer"
                      ContentTemplate="{DataTemplate view:AddAccountedBuyer}" />
        <ShellContent Title="Новый"
                      Icon="new_passenger.png"
                      Route="AddBuyer"
                      ContentTemplate="{DataTemplate view:AddBuyer}" />
    </TabBar>

    <ShellContent Route="MainPage"
                  ContentTemplate="{DataTemplate view:MainPage}" />
</Shell>
```

Листинг Г.2 – Файл AppShell.xaml.cs

```
using Airoport.Views;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace Airoport
{
    public partial class AppShell : Shell
    {
        public AppShell()
        {
            InitializeComponent();

            //Регистрация не вошедших в основную структуру страниц
            Routing.RegisterRoute(nameof(Flights), typeof(Flights));
            Routing.RegisterRoute(nameof(IssueSellBook), typeof(IssueSellBook));
            Routing.RegisterRoute(nameof(AnaliticView), typeof(AnaliticView));
            Routing.RegisterRoute(nameof(PassengerChoice),
typeof(PassengerChoice));
            Routing.RegisterRoute(nameof(ChangeDetailsPassenger),
typeof(ChangeDetailsPassenger));
        }
    }
}
```

Листинг Г.3 – Файл Service.cs

```
using Airoport.Models;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace Airoport.Services
{
    public static class Service
    {
        private static string url = "http://192.168.0.227:44307/";
        private static HttpClientHandler ClientHandler = new HttpClientHandler();
        private static HttpClient Client = new HttpClient(ClientHandler);

        //Авторизация кассира
        public async static Task<Cashier> LoginCashierAsync(string login, string
password)
        {
            Cashier cashier = null;
            var response = await Client.GetAsync(url + "Cashiers/Login?login=" +
login + "&password=" + password); //GET-запрос

            if (response.IsSuccessStatusCode) {
               string responseText = await response.Content .ReadAsStringAsync();
               cashier = JsonConvert.DeserializeObject<Cashier>(responseText);
            }
            return cashier;
        }

        //Получение кассира по его идентификатору
        public async static Task<Cashier> GetCashierAsync(int id_cashier)
```

```
        {
            Cashier cashier = null;
            var response = await Client.GetAsync(url + "Cashiers/Details/" +
id_cashier); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
               string responseText = await response.Content .ReadAsStringAsync();
               cashier = JsonConvert.DeserializeObject<Cashier>(responseText);
            }
            return cashier;
        }

        //Получение рейсов по направлению и дате
        public async static Task<List<Flight>> GetFlightsAsync(string city_from,
string city_to, DateTime date)
        {
            List<Flight> flights = null;
            string date_str = date.Date.ToString("yyyy-MM-dd");
            var response = await Client.GetAsync(url +
"Flights/Available?city_from=" + city_from + "&city_to=" + city_to + "&date=" +
date_str); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
                string responseText = await response.Content
.ReadAsStringAsync();
                flights = JsonConvert.DeserializeObject<List<Flight>>
(responseText);
            }
            return flights;
        }

        //Получение списка имеющихся городов
        public static async Task<List<string>> GetCitiesAsync()
        {
            List<City> cities = new List<City>();
            List<string> name_cities = new List<string>();
            var response = await Client.GetAsync(url + "Cities"); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
                string responseText = await response.Content
.ReadAsStringAsync();
                cities = JsonConvert.DeserializeObject<List<City>>
(responseText);
                foreach (City city in cities)
                {
                    name_cities.Add (city.name_city);
                }
            }
            return name_cities;
        }

        //Получение списка свободных мест на расписание
        public static async Task<List<string>> GetAvailableSeatsAsync(int
id_schedule, string[] occupidSeats)
        {
            List<string> availableSeats = new List<string>();
            var response = await Client.GetAsync(url + "Schedules/GetSeats/" +
id_schedule); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
```

```
                string      responseText      =      await      response.Content
.ReadAsStringAsync();
                availableSeats   =   JsonConvert.DeserializeObject<List<string>>
(responseText);
                foreach (string occupidSeat in occupidSeats)
                {
                    availableSeats.Remove (occupidSeat);
                }
            }
            return availableSeats;
        }

        //Добавление нового пассажира или покупателя
        public static async Task<int> AddPassengerAsync(string telephone, string
pasport,  DateTime  birth_date,  string  last_name,  string  first_name,  string
middle_name)
        {
            string date_str = birth_date.Date.ToString("yyyy-MM-dd");
            var content = new StringContent("{\"telephone\":\"" + telephone + "\""
                + ", \"passport\":\"" + pasport + "\""
                + ", \"birth_date\":\"" + date_str + "\""
                + ", \"last_name\":\"" + last_name + "\""
                + ", \"first_name\":\"" + first_name + "\""
                + ", \"middle_name\":\"" + middle_name + "\"}",
                Encoding.UTF8, "application/json");
            var  response  =  await  Client.PostAsync(url  +  "Passengers/Create",
content); //POST-запрос

            if (response.IsSuccessStatusCode)
            {
                return 0;
            }
            return -1;
        }

        //Получение идентификатора пассажира или покупателя по его номеру паспорта
        public static async Task<int> GetPassengerIdAsync(string pasport)
        {
            int id_passenger = 0;
            var response = await Client.GetAsync(url + "Passengers/Id? passport="
+ pasport); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
                string      responseText      =      await      response.Content
.ReadAsStringAsync();
                id_passenger = JsonConvert.DeserializeObject<int> (responseText);
            }
            return id_passenger;
        }

        //Получение пассажира или покупателя по его идентификатору
        public static async Task<Passenger> GetPassengerAsync(int id_passenger)
        {
            Passenger passenger = null;
            var  response  =  await  Client.GetAsync(url  +  "Passengers/Details/"  +
id_passenger); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
                string      responseText      =      await      response.Content
.ReadAsStringAsync();
                passenger       =       JsonConvert.DeserializeObject<Passenger>
(responseText);
```

```
            }
            return passenger;
        }



        //Продажа билета
        public static async Task<int> SellTicketsAsync()
        {
            int id_buyer = Purchase.id_buyer, id_cashier = Purchase.id_cashier,
id_schedule = Purchase.id_schedule;
            Dictionary<int, string> passengers = Purchase.passengers;
            int sumPrice = 0;
            if (id_buyer == 0 || id_cashier == 0 || id_schedule == 0 ||
passengers.Count == 0)
            {
                return -1;
            }

            foreach (int id in passengers.Keys)
            {
                var content = new StringContent("{\"seat_plane\":\"" +
passengers[id] + "\""
                + ", \"id_passenger\":\"" + id + "\""
                + ", \"id_buyer\":\"" + id_buyer + "\""
                + ", \"id_schedule\":\"" + id_schedule + "\""
                + ", \"id_cashier\":\"" + id_cashier + "\"}",
                Encoding.UTF8, "application/json");
                var response = await Client.PostAsync(url + "Tickets/Create",
content); //POST-запрос

                if (!response.IsSuccessStatusCode)
                {
                    return -1;
                }
                string responseText = await response.Content
.ReadAsStringAsync();
                int price = JsonConvert.DeserializeObject<int>(responseText);
                sumPrice += price;
            }
            return sumPrice;
        }

        //Бронирование билета
        public static async Task<bool> BookTicketsAsync()
        {
            int id_buyer = Purchase.id_buyer, id_cashier = Purchase.id_cashier,
id_schedule = Purchase.id_schedule;
            Dictionary<int, string> passengers = Purchase.passengers;
            if (id_buyer == 0 || id_cashier == 0 || id_schedule == 0 ||
passengers.Count == 0)
            {
                return false;
            }

            foreach (int id in passengers.Keys)
            {
                var content = new StringContent("{\"seat_plane\":\"" +
passengers[id] + "\""
                + ", \"id_passenger\":\"" + id + "\""
                + ", \"id_buyer\":\"" + id_buyer + "\""
                + ", \"id_schedule\":\"" + id_schedule + "\""
                + ", \"id_cashier\":\"" + id_cashier + "\"}",
                Encoding.UTF8, "application/json");
```

```
                var response = await Client.PostAsync(url + "Bookings/Create",
content); //POST-запрос

                if (!response.IsSuccessStatusCode)
                {
                    return false;
                }
            }
            return true;
        }

        //Получение средней загруженности рейсов по дням
        public async static Task<List<AverageFlightLoad>>
GetAverageFlightLoadAsync()
        {
            List<AverageFlightLoad> loads = new List<AverageFlightLoad>();
            var response = await Client.GetAsync(url + "Flights/AverageLoads");
//GET-запрос

            if (response.IsSuccessStatusCode)
            {
                string responseText = await response.Content
.ReadAsStringAsync();
                loads = JsonConvert.DeserializeObject<List<AverageFlightLoad>>
(responseText);
            }
            return loads;
        }

        //Получение количества совершенных рейсов пассажирами по месяцам
        public async static Task<List<PassengersFlightsCount>>
GetPassengersFlightsCountAsync()
        {
            List<PassengersFlightsCount> passengersFlights = new
List<PassengersFlightsCount>();
            var response = await Client.GetAsync(url + "Passengers/
FlightsCounts"); //GET-запрос

            if (response.IsSuccessStatusCode)
            {
                string responseText = await response.Content
.ReadAsStringAsync();
                passengersFlights = JsonConvert.DeserializeObject<List
<PassengersFlightsCount>>(responseText);
            }
            return passengersFlights;
        }

        //Выкуп брони
        public async static Task<int> RedeemBookingAsync(int id_booking)
        {
            var content = new StringContent("{\"id\":" + id_booking + "}",
Encoding.UTF8, "application/json");
            var response = await Client.PostAsync(url + "Bookings/Redeem",
content); //POST-запрос

            if (response.IsSuccessStatusCode)
            {
                int price;
                string responseText = await response.Content
.ReadAsStringAsync();
                price = JsonConvert.DeserializeObject<int>(responseText);
                return price;
            }
```

```
            return -1;
        }

        //Снятие брони
        public async static Task<bool> CancelBookingAsync(int id_booking)
        {
            var response = await Client.DeleteAsync(url + "Bookings/Cancel/" +
id_booking); //DELETE-запрос

            if (response.IsSuccessStatusCode)
            {
                return true;
            }
            return false;
        }

        //Выкуп множества броней
        public async static Task<(int, int)> RedeemMultipleBookingAsync(int
id_schedule, string passport)
        {
            var content = new StringContent("{\"id_schedule\":" + id_schedule +
", \"passport\":" + passport + "}", Encoding.UTF8, "application/json");
            var response = await Client.PostAsync(url + "Bookings
/RedeemMultiple", content); //POST-запрос

            if (response.IsSuccessStatusCode)
            {
                int count, price;
                string responseText = await response.Content
.ReadAsStringAsync();
                (count, price) = JsonConvert.DeserializeObject<(int, int)>
(responseText);
                return (count, price);
            }
            return (0, -1);
        }

        //Снятие множества броней
        public async static Task<int> CancelMultipleBookingAsync(int id_schedule,
string passport)
        {
            var response = await Client.DeleteAsync(url +
"Bookings/CancelMultiple?id_schedule=" + id_schedule + "&passport=" + passport);
//DELETE-запрос

            if (response.IsSuccessStatusCode)
            {
                string responseText = await response.Content
.ReadAsStringAsync();
                int count = JsonConvert.DeserializeObject<int>(responseText);
                return count;
            }
            return 0;
        }

        //Возврат билета
        public async static Task<int> RefundTicketAsync(int id_ticket)
        {
            var response = await Client.DeleteAsync(url + "Tickets/Refund/" +
id_ticket); //DELETE-запрос

            if (response.IsSuccessStatusCode)
            {
                int price;
```

```
                string responseText = await response.Content
.ReadAsStringAsync();
                price = JsonConvert.DeserializeObject<int>(responseText);
                return price;
            }
            return -1;
        }

        //Возврат множества билетов
        public async static Task<(int, int)> RefundMultipleTicketAsync(int
id_schedule, string passport)
        {
            var response = await Client.DeleteAsync(url +
"Tickets/RefundMultiple?id_schedule=" + id_schedule + "&passport=" + passport);
//DELETE-запрос

            if (response.IsSuccessStatusCode)
            {
                int count, price;
                string responseText = await response.Content
.ReadAsStringAsync();
                (count, price) = JsonConvert.DeserializeObject<(int, int)>
(responseText);
                return (count, price);
            }
            return (0, -1);
        }

        //Изменение данных пассажира или покупателя
        public static async Task<int> ChangePassengerAsync(int id_passenger,
string telephone, string pasport, DateTime birth_date, string last_name, string
first_name, string middle_name)
        {
            string date_str = birth_date.Date.ToString("yyyy-MM-dd");
            var content = new StringContent("{\"id_passenger\":\"" + id_passenger
+ "\""
                + ", \"telephone\":\"" + telephone + "\""
                + ", \"passport\":\"" + pasport + "\""
                + ", \"birth_date\":\"" + date_str + "\""
                + ", \"last_name\":\"" + last_name + "\""
                + ", \"first_name\":\"" + first_name + "\""
                + ", \"middle_name\":\"" + middle_name + "\"}",
                Encoding.UTF8, "application/json");
            var response = await Client.PutAsync(url + "Passengers/Change",
content); //PUT-запрос

            if (response.IsSuccessStatusCode)
            {
                return 0;
            }
            return -1;
        }
    }
}
```

Листинг Г.4 – Файл MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:d="http://xamarin.com/schemas/2014/forms/design"
```

```
            xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
            mc:Ignorable="d"
            x:Class="Airoport.Views.MainPage"
            Title="Вход">
    <StackLayout VerticalOptions="Center">
        <Entry x:Name="loginEntry"
               WidthRequest="350"
               Placeholder = "Логин"
               FontSize="Large"
               VerticalOptions="Center"
               HorizontalOptions="Center"/>
        <Entry x:Name="passwordEntry"
               WidthRequest="350"
               Placeholder = "Пароль"
               FontSize="Large"
               IsPassword ="True"
               VerticalOptions="Start"
               HorizontalOptions="Center" />
        <Button Text="Войти"
                VerticalOptions="Start"
                HorizontalOptions="Center"
                Clicked="EnterClicked" />
    </StackLayout>
</ContentPage>
```

Листинг Г.5 – Файл MainPage.xaml.cs

```
using System;
using System.ComponentModel;
using Xamarin.Forms;
using Airoport.Models;
using Airoport.Services;

namespace Airoport.Views
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        //Нажатие кнопки входа в аккаунт
        private async void EnterClicked(object sender, EventArgs e)
        {
            string login = loginEntry.Text.Trim(),
                password = passwordEntry.Text.Trim();
            Cashier cashier = await Service.LoginCashierAsync(login, password);
            if (cashier == null)
            {
                await DisplayAlert("Вход", "Неверный логин или пароль", "OK");
                return;
            }
            else
            {
                loginEntry.Text = "";
                passwordEntry.Text = "";
                await     Shell.Current.GoToAsync     ($"//SellBook?id_cashier=
{cashier.id_cashier}");
            }
        }
```

```
    }
}
```

Листинг Г.6 – Файл Flights.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="Airoport.Views.Flights"
             Title="Выбор рейса">
    <ContentPage.Content>
        <StackLayout>
            <ListView x:Name="FlightsView" IsVisible="True" RowHeight="220">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Orientation="Vertical">
                                <Label     Text="{Binding     Path=id_schedule,
StringFormat='Номер рейса: {0}'}" />
                                <Label  Text="{Binding  Path=name_airoport_from,
StringFormat='Аэропорт вылета: {0}'}" />
                                <Label   Text="{Binding   Path=name_airoport_to,
StringFormat='Аэропорт прилета: {0}'}" />
                                <Label    Text="{Binding    Path=departue_time,
StringFormat='Время вылета: {0}'}" />
                                <Label     Text="{Binding     Path=arrival_time,
StringFormat='Время прилета: {0}'}" />
                                <Label    Text="{Binding    Path=name_airplane,
StringFormat='Самолет: {0}'}" />
                                <Label      Text="{Binding      Path=base_price,
StringFormat='Цена: {0}'}" />
                                <Label                                  Text="{Binding
Path=number_available_seats, StringFormat='Количество свободных мест: {0}'}" />
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
            <Button Text="Перейти к оформлению билетов на выбранный рейс"
                    VerticalOptions="Start"
                    HorizontalOptions="Center"
                    Clicked="IssueClicked" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Листинг Г.7 – Файл Flights.xaml.cs

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

```
namespace Airoport.Views
{
    //Список передаваемых параметров
    [QueryProperty(nameof(Id_cashier), "id_cashier")]
    [QueryProperty(nameof(City_from), "city_from")]
    [QueryProperty(nameof(City_to), "city_to")]
    [QueryProperty(nameof(Date), "date")]
    public partial class Flights : ContentPage
    {
        Cashier cashier;
        string city_from, city_to;
        DateTime date;
        List<Flight> flights;
        public string Id_cashier
        {
            set
            {
                GetCashier(Int32.Parse(value));
            }
        }

        public string City_from
        {
            set
            {
                city_from = HttpUtility.UrlDecode(value);
            }
        }
        public string City_to
        {
            set
            {
                city_to = HttpUtility.UrlDecode(value);
            }
        }
        public string Date
        {
            set
            {
                date = DateTime.Parse(value);
            }
        }

        public Flights()
        {
            InitializeComponent();
        }

        //При открытии страницы
        protected async override void OnAppearing()
        {
            flights = await Service.GetFlightsAsync(city_from, city_to, date);
            FlightsView.ItemsSource = flights;
        }

        //Получение кассира по его идентификатору
        private async void GetCashier(int id_cashier)
        {
            cashier = await Service.GetCashierAsync(id_cashier);
        }

        //Нажатие кнопки перехода к оформлению билетов
        private async void IssueClicked(object sender, EventArgs e)
```

```
        {
            if (FlightsView.SelectedItem == null)
            {
                await DisplayAlert("Выбор рейса", "Не выбран рейс", "OK");
                return;
            }
            int id_schedule = (FlightsView.SelectedItem as Flight).id_schedule;
            Purchase.id_cashier = cashier.id_cashier;
            Purchase.id_schedule = id_schedule;
            Purchase.id_buyer = 0;
            Purchase.passengers = new Dictionary<int, string>();
            await Shell.Current.GoToAsync($"./IssueSellBook");
        }
    }
}
```

Листинг Г.8 – Файл SellBook.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="Airoport.Views.SellBook">
    <ContentPage.Content>
        <StackLayout>
            <StackLayout VerticalOptions="Start"
                HorizontalOptions="Start"
                Orientation="Horizontal">
                <Label x:Name="labelFrom"
                   Text="Откуда"
                   HorizontalOptions="Start" />
                <Picker x:Name="FromPicker"
                        WidthRequest="350"
                        FontSize="Large"
                        HorizontalOptions="End" />
            </StackLayout>
            <StackLayout VerticalOptions="Start"
                HorizontalOptions="Start"
                Orientation="Horizontal">
                <Label x:Name="labelTo"
                   Text="Куда"
                   HorizontalOptions="Start" />
                <Picker x:Name="ToPicker"
                        WidthRequest="350"
                        FontSize="Large"
                        HorizontalOptions="End" />
            </StackLayout>
            <StackLayout VerticalOptions="Start"
                HorizontalOptions="Start"
                Orientation="Horizontal">
                <Label x:Name="labelDate"
                   Text="Дата вылета"
                   HorizontalOptions="Start" />
                <DatePicker x:Name="FlightDate"
                   HorizontalOptions="End" />
            </StackLayout>
            <Button Text="Вывести список рейсов"
                    VerticalOptions="Start"
                    HorizontalOptions="Center"
```

```
                    Clicked="FlightsClicked" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Листинг Г.9 – Файл SellBook.xaml.cs

```
using Airoport.Models;
using Airoport.Services;
using System;
using System.Collections.Generic;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace Airoport.Views
{
    //Список передаваемых параметров
    [QueryProperty(nameof(Id_cashier), "id_cashier")]
    public partial class SellBook : ContentPage
    {
        Cashier cashier;
        public string Id_cashier
        {
            set
            {
                GetCashier(Int32.Parse(value));
            }
        }

        public SellBook()
        {
            InitializeComponent();
            GetCities();
        }

        //Получение кассира по его идентификатору
        private async void GetCashier(int id_cashier)
        {
            cashier = await Service.GetCashierAsync(id_cashier);
        }

        //Получение списка городов и запись их в поля выбора
        private async void GetCities()
        {
            List<string> cities = await Service.GetCitiesAsync();
            foreach (string name_city in cities)
            {
                FromPicker.Items.Add(name_city);
                ToPicker.Items.Add(name_city);
            }
        }

        //Нажатие кнопки выбора рейса
        private async void FlightsClicked(object sender, EventArgs e)
        {
            if (FromPicker.SelectedIndex < 0 || ToPicker.SelectedIndex < 0)
            {
                await DisplayAlert("Продажа/бронирование билетов", "Не выбрано
направление авиаперелета", "OK");
                return;
            }
            string city_from = FromPicker.Items[FromPicker.SelectedIndex],
```

```
                city_to = ToPicker.Items[ToPicker.SelectedIndex];
            await Shell.Current.GoToAsync($"Flights?id_cashier
={cashier.id_cashier}&city_from={city_from}&city_to={city_to}&date={FlightDate.D
ate.ToString("d")}");
        }
    }
}
```

Листинг Г.10 – Файл AnaliticView.xaml.cs

```
using Airoport.Models;
using Airoport.Services;
using System;
using System.Collections.Generic;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace Airoport.Views
{
    //Список передаваемых параметров
    [QueryProperty(nameof(TypeAn), "type")]
    public partial class AnaliticView : ContentPage
    {
        public string TypeAn {
            set
            {

                int type = Int32.Parse(value);
                if (type == 1)
                {
                    LoadForm1();
                }
                if (type == 2)
                {
                    LoadForm2();
                }
            }
        }

        public AnaliticView()
        {
            InitializeComponent();
        }

        //Добавление таблицы с информацией о загруженности рейсов по дням
        private async void LoadForm1()
        {
            var tblSec = new TableSection("Данные")
            {
                new ViewCell
                {
                    View = new StackLayout {
                        Orientation = StackOrientation.Horizontal,
                        Children =
                        {
                            new   Label   {Text   =   "Дата",   HorizontalOptions
=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new     Label     {Text     =     "Загруженность",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black}
                        }
                    }
                }
```

```
            };
            var table = new TableView
            {
                Intent = TableIntent.Form,
                Root = new TableRoot("Ввод данных")
                {
                    tblSec
                }
            };
            this.Content = table;
            List<AverageFlightLoad>       loads       =       await       Service
.GetAverageFlightLoadAsync();
            foreach (AverageFlightLoad load in loads)
            {
                tblSec.Insert (1, new ViewCell
                {
                    View = new StackLayout
                    {
                        Orientation = StackOrientation.Horizontal,
                        Children =
                        {
                            new    Label    {Text    =    TimeZoneInfo.ConvertTime
(load.date,       TimeZoneInfo.Local).Date.ToString("d"),       HorizontalOptions
=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new    Label    {Text    =    load.load.ToString(),
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black}
                        }
                    }
                });
            }
        }

        //Добавление таблицы с информацией о количестве совершенных полетов
пассажирами по месяцам
        private async void LoadForm2()
        {
            var tblSec = new TableSection("Данные")
            {
                new ViewCell
                {
                    View = new StackLayout {
                        Orientation = StackOrientation.Horizontal,
                        Children =
                        {
                            new       Label       {Text       =       "Фамилия",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new        Label        {Text        =        "Имя",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new       Label       {Text       =       "Отчество",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new     Label     {Text     =     "Нач.     дата",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new     Label     {Text     =     "Кон.     дата",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new     Label     {Text     =     "Кол.     рейсов",
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black}
                        }
                    }
                }
            };
            var table = new TableView
            {
                Intent = TableIntent.Form,
                Root = new TableRoot("Ввод данных")
```

```
                {
                    tblSec
                }
            };
            this.Content = table;
            List<PassengersFlightsCount> passengersFlights = await Service
.GetPassengersFlightsCountAsync();
            foreach (PassengersFlightsCount passengersFlight in
passengersFlights)
            {
                tblSec.Insert(1, new ViewCell
                {
                    View = new StackLayout
                    {
                        Orientation = StackOrientation.Horizontal,
                        Children =
                        {
                            new Label {Text = passengersFlight.last_name,
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new Label {Text = passengersFlight.first_name,
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new Label {Text = passengersFlight.middle_name,
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new Label {Text = TimeZoneInfo.ConvertTime
(passengersFlight.start_date, TimeZoneInfo.Local).Date.ToString("d"),
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new Label {Text = TimeZoneInfo.ConvertTime
(passengersFlight.finish_date, TimeZoneInfo.Local).Date.ToString("d"),
HorizontalOptions=LayoutOptions.CenterAndExpand, TextColor=Color.Black},
                            new Label {Text = passengersFlight
.flights_count.ToString(), HorizontalOptions=LayoutOptions.CenterAndExpand,
TextColor=Color.Black }
                        }
                    }
                });
            }
        }
    }
}
```

Листинг Г.11 – Файл IssueSellBook.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="Airoport.Views.IssueSellBook"
             Title="Покупка/бронирование билетов">
    <ContentPage.Content>
        <StackLayout>
            <StackLayout VerticalOptions="Start" Orientation="Vertical">
                <Button Text="Добавить пассажира"
                VerticalOptions="Start"
                HorizontalOptions="Start"
                Clicked="AddPassengerClicked" />
                <Label x:Name="label1"
               Text="Пассажиры:"
               HorizontalOptions="Start" />
                <Label x:Name="passengersLabel"
```

```
                Text=""
                HorizontalOptions="Center" />
            </StackLayout>
            <StackLayout VerticalOptions="Start" Orientation="Vertical">
                <Button Text="Добавить покупателя"
                VerticalOptions="Start"
                HorizontalOptions="Start"
                Clicked="AddBuyerClicked" />
                <Label x:Name="label2"
                Text="Покупатель:"
                HorizontalOptions="Start" />
                <Label x:Name="buyerLabel"
                Text=""
                HorizontalOptions="Center" />
            </StackLayout>
            <Button Text="Продать билеты"
                VerticalOptions="End"
                HorizontalOptions="Center"
                Clicked="SellTicketsClicked" />
            <Button Text="Забронировать билеты"
                VerticalOptions="End"
                HorizontalOptions="Center"
                Clicked="BookTicketsClicked" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Листинг Г.12 – Файл IssueSellBook.xaml.cs

```
using Airoport.Models;
using Airoport.Services;
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;


namespace Airoport.Views
{
    //Список передаваемых параметров
    [QueryProperty(nameof(Id_passenger), "id_passenger")]
    [QueryProperty(nameof(Id_buyer), "id_buyer")]
    public partial class IssueSellBook : ContentPage
    {
        Cashier cashier;
        public string Id_passenger
        {
            set
            {
                AddPassengerLabel(Int32.Parse(value));
            }
        }
        public string Id_buyer
        {
            set
            {
                SetBuyerLabel(Int32.Parse(value));
            }
        }

        public IssueSellBook()
        {
            InitializeComponent();
```

```
            GetCashier(Purchase.id_cashier);
        }

        //Получение кассира по его идентификатору
        private async void GetCashier(int id_cashier)
        {
            cashier = await Service.GetCashierAsync(id_cashier);
        }

        //Добавление пассажира на надпись с информацией о пассажирах
        private async void AddPassengerLabel(int id_passenger)
        {
            Passenger passenger = await Service.GetPassengerAsync(id_passenger);
            passengersLabel.Text    +=    passenger.last_name    +    "    "    +
passenger.first_name + " " + passenger.middle_name + "\n";
        }

        //Установление покупателя на надписи с информацией о покупателе
        private async void SetBuyerLabel(int id_buyer)
        {
            Passenger buyer = await Service.GetPassengerAsync(id_buyer);
            buyerLabel.Text = buyer.last_name + " " + buyer.first_name + " " +
buyer.middle_name;
        }

        //Нажатие на кнопку добавления пассажира
        private async void AddPassengerClicked(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync($"//AddAccountedPassenger");
        }

        //Нажатие на кнопку добавления покупателя
        private async void AddBuyerClicked(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync($"//AddAccountedBuyer");
        }



        //Нажатие на кнопку продажи билетов
        private async void SellTicketsClicked(object sender, EventArgs e)
        {
            int sumPrice;
            if (await DisplayAlert("Продажа билетов", "Вы точно хотите продать
билеты на выбранный рейс?", "OK", "Отмена"))
            {
                sumPrice = await Service.SellTicketsAsync();
                if (sumPrice < 0)
                {
                    await DisplayAlert("Продажа билетов", "Не все параметры были
выбраны", "OK");
                    return;
                }
            }
            else
            {
                return;
            }
            await DisplayAlert("Продажа билетов", "Все билеты успешно выкуплены
на общую сумму " + sumPrice + " рублей", "OK");
            await Navigation.PopAsync();
        }
        //Нажатие на кнопку бронирования билетов
        private async void BookTicketsClicked(object sender, EventArgs e)
```

```
        {
            if (await DisplayAlert("Бронирование билетов", "Вы точно хотите забронировать билеты на выбранный рейс?", "OK", "Отмена"))
            {
                if (!await Service.BookTicketsAsync())
                {
                    await DisplayAlert("Бронирование билетов", "Не все параметры были выбраны", "OK");
                    return;
                }
            }
            else
            {
                return;
            }
            await DisplayAlert("Бронирование билетов", "Все билеты успешно забронированы", "OK");
            await Navigation.PopAsync();
        }
    }
}
```

Приложение Д

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

(рекомендуемое)

Рисунок Д.1 – Авторизация кассира

Рисунок Д.2 – Получение информации о кассире по его идентификатору

Рисунок Д.3 – Получение списка городов

Рисунок Д.4 – Получение списка рейсов на определенное направление и дату

```
GET /Schedules/GetSeats/3 HTTP/1.1
Host: 192.168.0.227
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 685
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:06:58 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

["01a","01b","01c","01d","02a","02b","02d","03a","03b","0
```

Рисунок Д.5 – Получение списка свободных мест на определенный рейс по его идентификатору

```
GET /Passengers/Details/6 HTTP/1.1
Host: 192.168.0.227
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 250
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:07:47 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

{"Booking":[],"Booking1":[],"Ticket":[],"Ticket1":[],"id_p
```

Рисунок Д.6 – Получение информации о пассажире по его идентификатору

```
PUT /Passengers/Change HTTP/1.1
Host: 192.168.0.227
Content-Type: application/json
Content-Length: 195

{ "id_passenger": 3,
 "telephone": "+79375091284",
 "passport": "5299100322",
 "birth_date": "1992-09-27",
 "last_name": "Богданов",
 "first_name": "Богдан",
 "middle_name": "Богданович" }
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 0
date: Fri, 28 Apr 2023 07:09:49 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET
```

Рисунок Д.7 – Изменение данных пассажира

```
POST /Passengers/Create HTTP/1.1
Host: 192.168.0.227
Content-Type: application/json
Content-Length: 170

{ "telephone": "+79378410924",
"passport": "9288160325",
 "birth_date": "1993-08-14",
 "last_name": "Захаров",
 "first_name": "Захар",
 "middle_name": "Захарович" }
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 0
date: Fri, 17 Mar 2023 12:10:52 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET
```

Рисунок Д.8 – Добавление пассажира

```
POST /Bookings/Create HTTP/1.1
Host: 192.168.0.227
Content-Type: application/json
Content-Length: 102

{ "seat_plane": "08c",
"id_passenger": 10,
 "id_buyer": 10,
 "id_schedule": 5,
 "id_cashier": 9 }
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 0
date: Fri, 17 Mar 2023 12:12:22 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET
```

Рисунок Д.9 – Добавление брони

```
DELETE /Bookings/Cancel/114 HTTP/1.1
Host: 192.168.0.227
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 0
date: Fri, 17 Mar 2023 12:14:16 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET
```

Рисунок Д.10 – Снятие брони по ее идентификатору

```
POST /Bookings/Redeem HTTP/1.1
Host: 192.168.0.227
Content-Type: application/json
Content-Length: 15

{ "id": 115 }
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 5
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:15:46 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

32000
```

Рисунок Д.11 – Выкуп брони

```
DELETE /Bookings/CancelMultiple?id_schedule=5&passport=90
Host: 192.168.0.227
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 1
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:19:12 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

2
```

Рисунок Д.12 – Снятие множества броней по номеру рейса и номеру паспорта оформителя броней

```
POST /Bookings/RedeemMultiple HTTP/1.1
Host: 192.168.0.227
Content-Type: application/json
Content-Length: 48

{ "id_schedule": 5,
 "passport": "9090761536" }
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 25
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:22:25 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

{"Item1":2,"Item2":64000}
```

Рисунок Д.13 – Выкуп множества броней

```
POST /Tickets/Create HTTP/1.1
Host: 192.168.0.227
Content-Type: application/json
Content-Length: 99

{ "seat_plane": "01b",
"id_passenger": 3,
 "id_buyer": 4,
 "id_schedule": 13,
 "id_cashier": 2 }
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 5
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:23:33 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

11000
```

Рисунок Д.14 – Добавление билета

```
DELETE /Tickets/Refund/113 HTTP/1.1
Host: 192.168.0.227
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 5
content-type: application/json; charset=utf-8
date: Fri, 17 Mar 2023 12:24:40 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

11000
```

Рисунок Д.15 – Возврат билета по его идентификатору

```
DELETE /Tickets/RefundMultiple?id_schedule=5&passport=9096
Host: 192.168.0.227
```

```
HTTP/1.1 200 OK
cache-control: private
content-length: 25
content-type: application/json; charset=utf-8
date: Fri, 28 Apr 2023 07:21:34 GMT
server: Microsoft-IIS/10.0
x-aspnet-version: 4.0.30319
x-aspnetmvc-version: 5.2
x-powered-by: ASP.NET

{"Item1":2,"Item2":64000}
```

Рисунок Д.16 – Возврат множества билетов

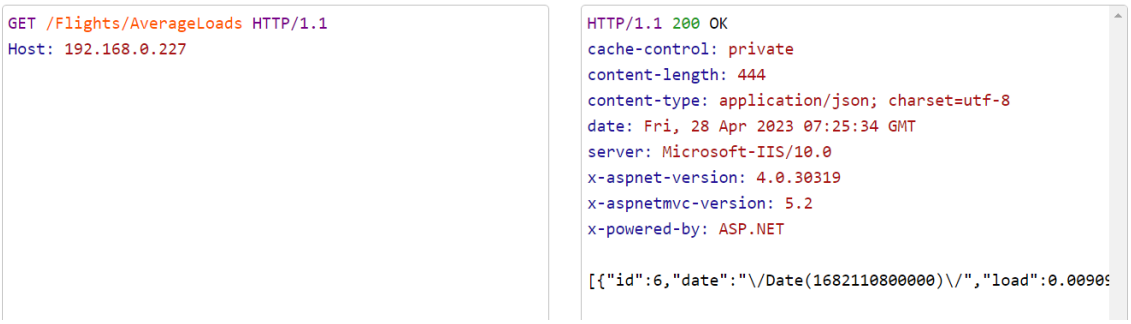

Рисунок Д.17 – Получение средней загруженности рейсов по дням за последнюю неделю

Рисунок Д.18 – Получение количества полетов пассажиров по месяцам за последний год

Приложение Е

ИСХОДНЫЙ КОД ФУНКЦИЙ ТЕСТИРОВАНИЯ

(рекомендуемое)

Листинг Е.1 – Файл ControllerTests.cs

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using ex.Controllers;
using System.Collections.Generic;
using System.Web.Mvc;
using ex.Models;

namespace Tests
{
    [TestClass]
    public class ControllerTests
    {
        public static IEnumerable<object[]> GetFlights()
        {
            yield return new object[] { true, "Пенза", "Москва",
DateTime.Parse("2023-05-03") };
            yield return new object[] { false, "Москва", "Пенза",
DateTime.Parse("2023-05-03") };
            yield return new object[] { false, "Пенза", "Москва",
DateTime.Parse("2023-05-05") };
            yield return new object[] { false, "Москва", "Пенза",
DateTime.Parse("2023-05-05") };
        }

        [DataTestMethod]
        [DynamicData(nameof(GetFlights), DynamicDataSourceType.Method)]
        public void Flights(bool isExists, string city_from, string city_to,
DateTime date)
        {
            var controller = new FlightsController();
            var result = controller.Available(city_from, city_to, date);
            if (!isExists)
            {
                Assert.IsInstanceOfType(result, typeof(HttpNotFoundResult));
                Assert.AreEqual(404, ((HttpNotFoundResult)result).StatusCode);
            }
            else
            {
                Assert.IsInstanceOfType(result, typeof(JsonResult));
                var r = ((JsonResult)result).Data;
                Assert.AreNotEqual(0, ((List<get_flights_Result>)r).Count);
            }
        }

        public static IEnumerable<object[]> GetLogins()
        {
            yield return new object[] { true, "Igorev", "Igorevich1998" };
            yield return new object[] { false, "Igorev", "Igorevich1999" };
            yield return new object[] { false, "Igore", "Igorevich1998" };
            yield return new object[] { false, "", "Igorevich1998" };
            yield return new object[] { false, "Igorev", "" };
            yield return new object[] { false, "", "" };
        }

        [DataTestMethod]
        [DynamicData(nameof(GetLogins), DynamicDataSourceType.Method)]
        public void Logins(bool isExists, string login, string password)
        {
            var controller = new CashiersController();
            var result = controller.Login(login, password);
```

```
            if (!isExists)
            {
                Assert.IsInstanceOfType(result, typeof(HttpUnauthorizedResult));
                Assert.AreEqual(401,
((HttpUnauthorizedResult)result).StatusCode);
            }
            else
            {
                Assert.IsInstanceOfType(result, typeof(JsonResult));
                var r = ((JsonResult)result).Data;
                Assert.IsNotNull((Cashier)r);
            }
        }
    }
}
```

Приложение Ж

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

(рекомендуемое)

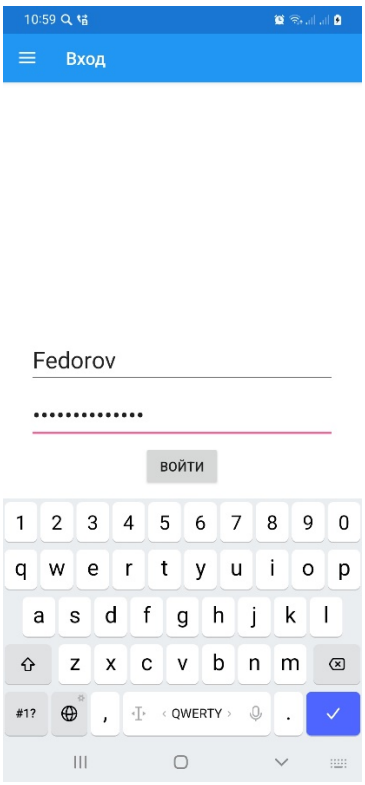

Рисунок Ж.1 – Авторизация кассира с корректными данными

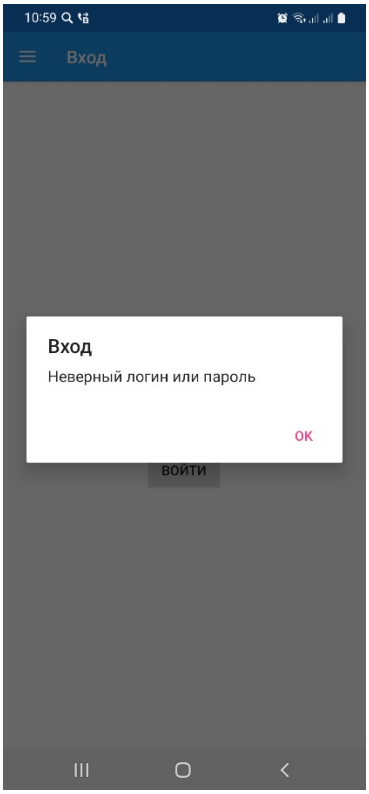

Рисунок Ж.2 – Авторизация кассира с некорректными данными

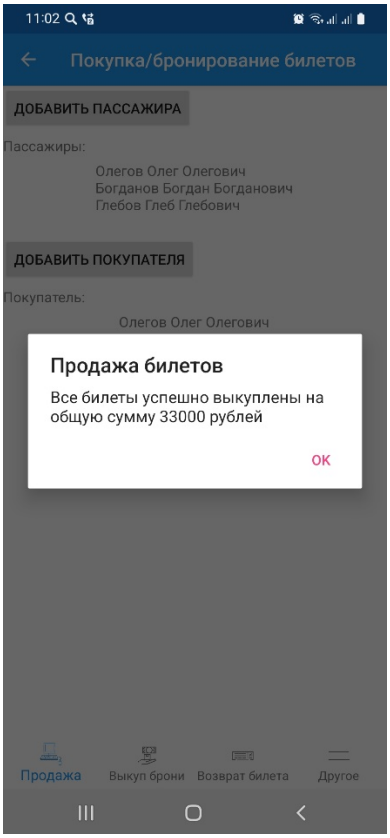

Рисунок Ж.3 – Оформление продажи билетов

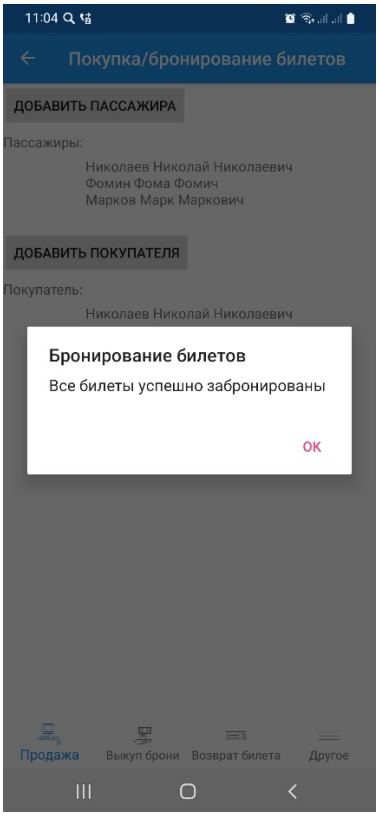

Рисунок Ж.4 – Оформление бронирования билетов

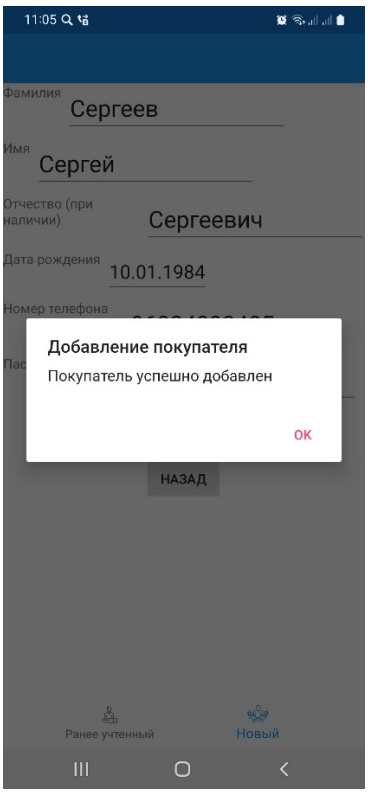

Рисунок Ж.5 – Добавление нового покупателя

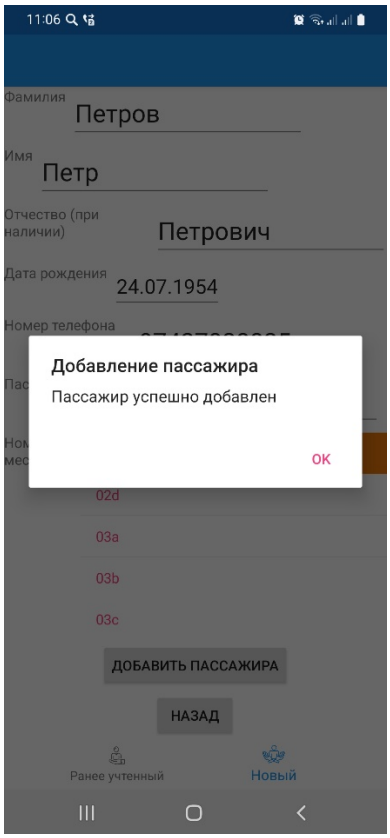

Рисунок Ж.6 – Добавление нового пассажира

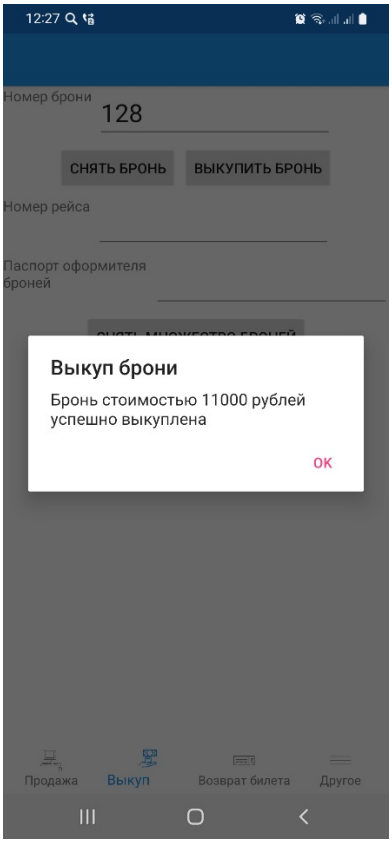

Рисунок Ж.7 – Осуществление выкупа брони

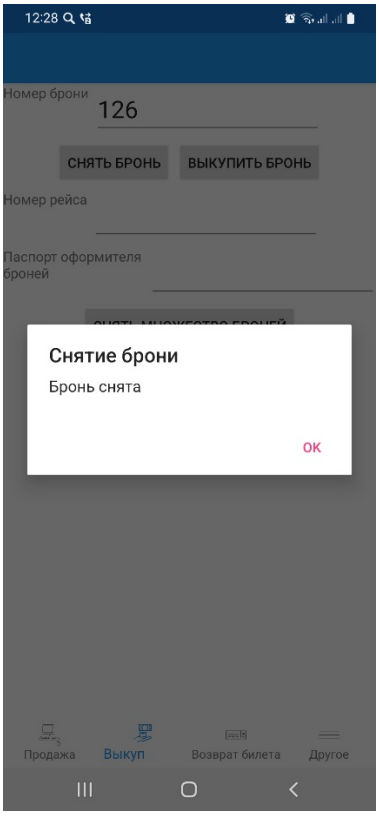

Рисунок Ж.8 – Осуществление снятия брони

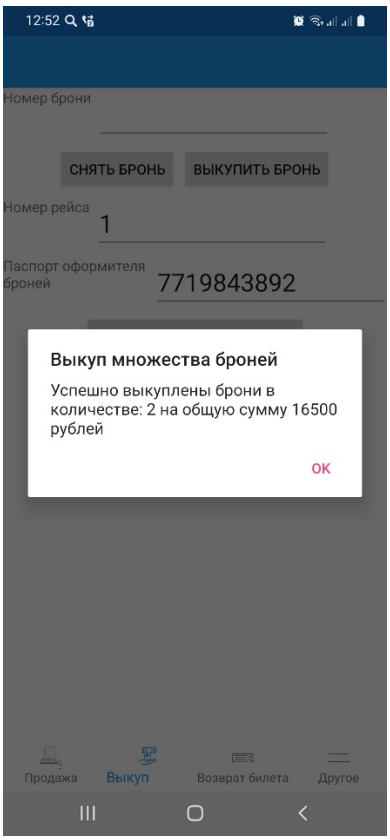

Рисунок Ж.9 – Осуществление выкупа множества броней

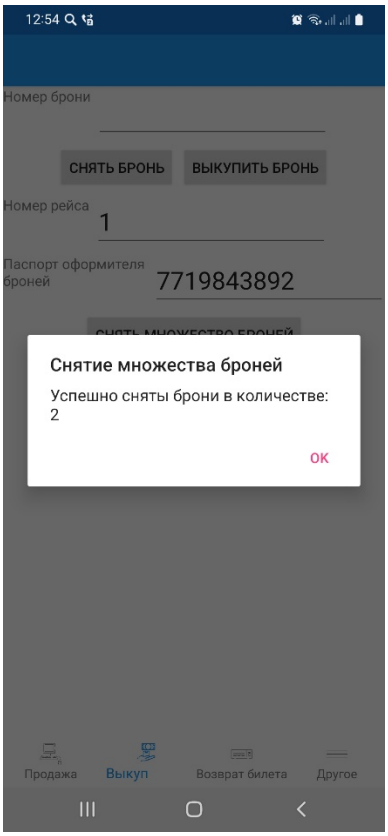

Рисунок Ж.10 – Осуществление снятия множества броней

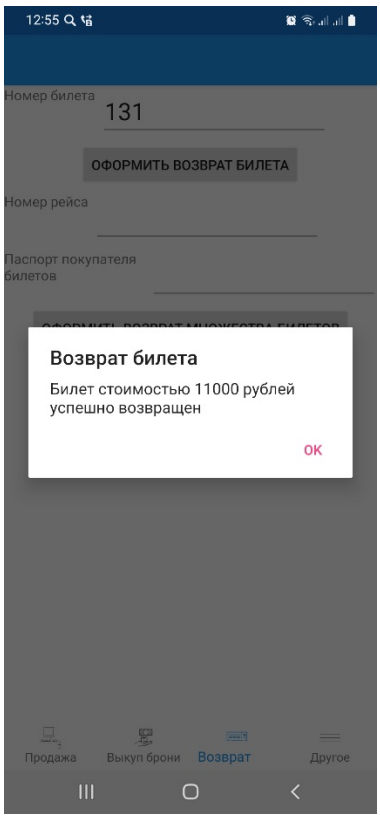

Рисунок Ж.11 – Оформление возврата билета

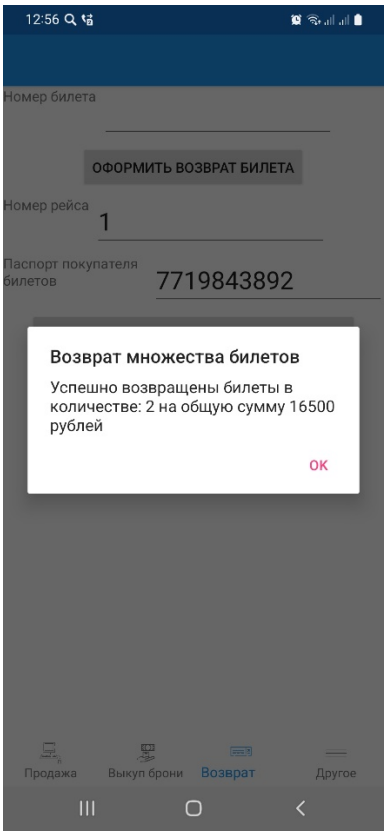

Рисунок Ж.12 – Оформление возврата множества билетов

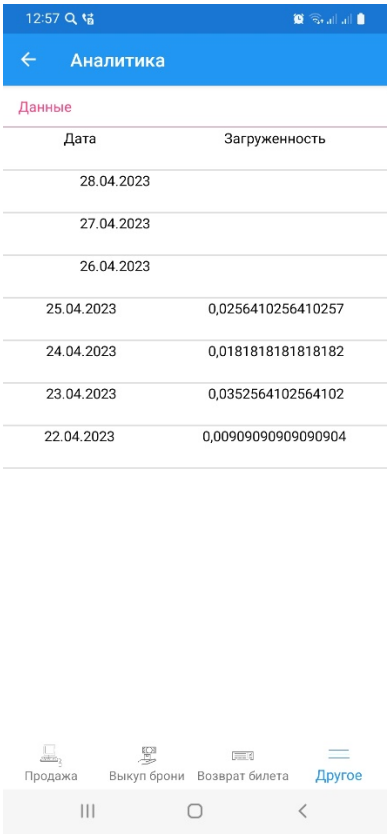

Рисунок Ж.13 – Получение информации о средней загруженности рейсов по дням

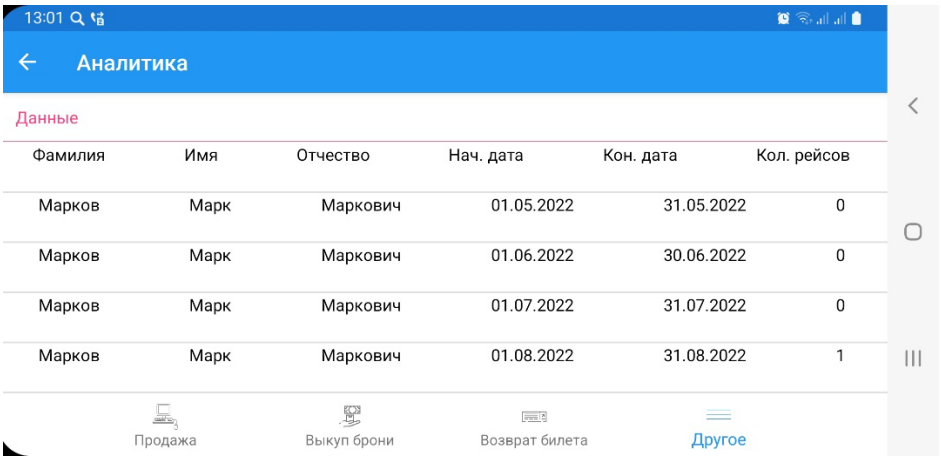

Рисунок Ж.14 – Получение информации о количестве совершенных полетов для каждого пассажира по месяцам

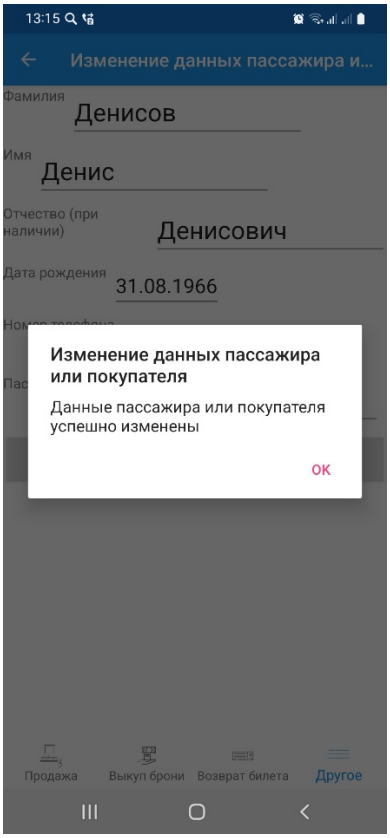

Рисунок Ж.15 – Изменение данных пассажира или покупателя