

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Информационная система бронирования
деловых поездок

Студент _____ Буйняков Георгий Эдуардович
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(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
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Буйняков Георгий Эдуардович гр. 19ВП2 факультета ВТ.
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система бронирования деловых поездок.....

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

2. Руководитель работы к.т.н., доцент Балашова И.Ю.

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

1) *Назначение: сервис для оформления и согласования служебных командировок*

2) *Среда выполнения: операционная система Ubuntu 20.04, Web-браузер.*

Языковые средства разработки: C#, Go, JavaScript, UML.

Инструментальные средства: Jet Brains GoLand, Jet Brains Rider, Jet Brains DataGrip.

Технологии разработки ООП, сервисно-ориентированный подход.

3) *Основные функции приложения:*

а) авторизация;

б) добавление сотрудников;

в) поиск авиабилетов, железнодорожных билетов и услуг проживания;

г) наполнение корзины услугами разного типа;

д) покупка корзины с имеющимися в ней услугами;

е) отправка билетов и ваучеров на электронную почту;

ж) просмотр истории покупок;

з) просмотр аналитики по покупкам услуг разного типа.

4. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области и постановка задачи;
- 2) Проектирование;
- 3) Реализация;
- 4) Контроль качества программного обеспечения;
- 5) План разработки бюджета;
- 6) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова

Реферат

Пояснительная записка содержит 78 листов, 29 рисунков, 12 таблиц, 19 использованных источников и 5 приложений.

БРОНИРОВАНИЕ, MONGODB, MSSQL, СЕРВЕР БАЗЫ ДАННЫХ, UML, REST, MVC, ДЕЛОВЫЕ ПОЕЗДКИ, C#, DOCKER.

Объект исследования – системы автоматизации бронирования деловых поездок.

Цель выпускной квалификационной работы – разработать приложение для бронирования деловых поездок.

Технологии разработки – разработка сервера приложений проводилась на языке программирования C#, для создания сервера БД использовался MSSQL и MongoDB. В качестве среды разработки использовались интегрированные среды разработки Rider и DataGrip.

Среда исполнения – Linux, Web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы была создана серверная часть сервиса по бронированию деловых поездок. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04– 8BP192.05 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система бронирования деловых поездок. Пояснительная записка		
Разраб.		Буйняков Г.Э.					
Пров.		Балашова И.Ю.					
Н. Контр.		Такташкин Д.В					
Утв.							
					Лит.	Лист	Листов
						4	78
					Группа 19ВП2		

Содержание

Введение.....	7
1 Анализ систем бронирования деловых поездок	8
1.1 Анализ предметной области	8
1.2 Программные продукты для бронирования деловых поездок	9
1.2.1 Онлайн-система управления деловыми поездками «Авиацентр»	10
1.2.2 Сервис организации командировок «Ostrovok.ru Командировки».....	11
1.2.3 Платформа онлайн-бронирования «Вкомандировке».....	12
1.2.4 Система организации деловых поездок «Ozon Командировки».....	13
1.2.5 Сравнительный анализ платформ бронирования деловых поездок	15
1.3 Постановка задачи на разработку информационной системы бронирования деловых поездок.....	17
1.3.1 Общее описание системы.....	17
1.3.2 Роли системы	18
1.3.3 Основные бизнес-процессы	18
1.3.4 Интерфейсы личных кабинетов.....	19
1.3.5 Нефункциональные требования	19
1.3.6 Функциональные требования.....	19
2 Проектирование системы бронирования деловых поездок.....	23
2.1 Проектирование базы данных.....	23
2.2 Проектирование программных средств	24
3 Реализация программного обеспечения	27
3.1 Разработка обобщенной архитектуры системы	27
3.2 Разработка серверной части приложения.....	29
3.2.1 Реализация аналитических бизнес-процессов.....	31

3.2.2 Реализация автоматически выполняющихся событий.....	35
3.3 Инфраструктура развертывания приложения	37
4 Контроль качества программного обеспечения.....	39
4.1 Функциональное тестирование.....	39
4.2 Модульное тестирование.....	43
4.3 Метрики устойчивости и абстрактности компонент.....	44
4.4 Метрики кода.....	46
5 Планирование разработки и оценка бюджета.....	47
Заключение	49
Список использованных источников	50
Приложение А. Глоссарий	52
Приложение Б. Структура коллекций MongoDB	54
Приложение В. Листинг приложения	57
Приложение Г. Модульные тесты	73
Приложение Д. Справка о внедрении	77

Введение

Деловые поездки — неотъемлемая часть рабочего процесса любой компании. Процесс организации деловой поездки в компании выглядит следующим образом: секретарь или тревел-менеджер подбирает билеты, гостиницу, при необходимости согласовывает варианты и детали с сотрудником и руководителем, пока не будет одобрен конкретный вариант.

Когда деловых поездок много, риск ошибки при покупке билетов растет. Достаточно выбрать неверную дату или ввести данные старого паспорта сотрудника, и билеты придется переоформлять. Чем больше у компании поездок, тем выше расходы. Онлайн-сервис бронирования деловых поездок может помочь сэкономить средства компании. На рынке программного обеспечения представлено достаточно много подобных программных продуктов, однако большинство из них имеют ограниченный инструментарий для бронирования деловых поездок.

Таким образом, актуальной является задача разработки информационной системы, обеспечивающей полный набор инструментов для бронирования деловых поездок.

Целью выпускной квалификационной работы является разработка информационной системы бронирования деловых поездок.

1 Анализ систем бронирования деловых поездок

1.1 Анализ предметной области

Существует три способа организации деловых поездок: IT-платформа, агентство или самостоятельное бронирование [1].

Самостоятельное бронирование – самый распространенный способ организации деловых поездок. При таком способе сотрудники сами покупают билеты и бронируют гостиницы на B2C-сервисах, а компания возмещает расходы.

Можно выделить следующие преимущества самостоятельного бронирования – быстрая покупка и отсутствие необходимости в помощи тревел-менеджера. Из минусов можно выделить закрывающие документы в разных местах, сложность контроля покупок, отсутствие возможности зафиксировать цену за билет и отсутствие тревел-поддержки. Из-за сложностей контроля покупок сотрудников компании могут столкнуться с распространенными схемами обмана, когда стоимость проживания искусственно завышается, и разницу в цене сотрудник забирает себе. Использование сразу нескольких информационных систем для бронирования приводит к созданию документов в разных местах, что в свою очередь порождает проблемы с бухгалтерией.

Второй способ организации деловых поездок – бронирование через агентства. Тревел-менеджеры согласовывают билеты по телефону или электронной почте, относят документы на подпись, отправляют письмо с подтверждением и ждут бронь.

Плюсы такого подхода:

- возможность организации деловых поездок в странах без современной инфраструктуры;
- удобно для клиентов, которые неуверенно обращаются с интернетом;
- предоставление персонализированных услуг, например, частный самолет;
- не нужно обучать бронированию сотрудников компании.

Недостатки:

- длительное согласование услуг;
- сервисные сборы и наценки за билеты и гостиницы;
- нет бесплатной отмены услуг;
- бухгалтерия не автоматизирована, много времени тратится на ручной ввод закрывающих документов.

Третий способ – использование IT-платформ для организации деловых поездок. Программные продукты для организации деловых поездок представляют из себя привычные пользователям сервисы для покупки билетов и бронирования отелей. Пользователь вводит направление и даты деловой поездки и получает различные варианты услуг по своему запросу. Помимо этого, пользователь получает инструменты согласования и контроля расходов и автоматическое формирование закрывающих документов. Таким образом, онлайн-решения закрывают сразу несколько тревел-потребностей пользователей. Единый интерфейс также ускоряет и упрощает работу сотрудникам от тревел-менеджера до руководителя.

Учитывая все выше сказанное, онлайн-сервис организации деловых поездок может помочь сэкономить средства компании. С помощью автоматизированной системы бронирования деловых поездок сотрудник может сам выбрать авиабилеты, ж/д билеты, отели и трансферы онлайн на одном сервисе без необходимости использования отдельных сервисов на каждый тип услуг, добавляет необходимые услуги в корзину, после чего отправляет заполненную форму с приложенными услугами на согласование указанному ответственному лицу. После того, как поездка будет оплачена, пользователь получает все необходимые ваучеры, счета, а также бухгалтерские документы.

1.2 Программные продукты для бронирования деловых поездок

Программные продукты для бронирования деловых поездок – это специализированные программы, которые позволяют организовать и упростить

процесс бронирования билетов на самолеты, поезда, а также бронирования отелей и других услуг для деловых поездок.

Такие программы могут предоставляться как отдельными компаниями, так и в составе комплексных систем управления корпоративными поездками. Они обеспечивают возможность быстрого и удобного поиска и бронирования необходимых услуг, а также позволяют контролировать расходы на деловые поездки и управлять корпоративными счетами.

Рассмотрим наиболее популярные сервисы бронирования деловых поездок.

1.2.1 Онлайн-система управления деловыми поездками «Авиацентр»

«Авиацентр» – это онлайн-система бронирования туристических услуг для корпоративных клиентов. С ее помощью компании могут полностью автоматизировать и управлять всеми процессами по организации деловых поездок, контролировать соблюдение тревел-политики сотрудниками компании, а также экономить время и ресурсы на каждом этапе организации деловых поездок [2].

Онлайн-система (рисунок 1), позволяет управлять и организовывать деловые поездки в режиме онлайн. Интеграция с клиентскими системами, такими как 1С, позволяет автоматизировать процесс и сократить трудозатраты офис-менеджеров, бухгалтерии и руководителей на управление деловыми поездками.

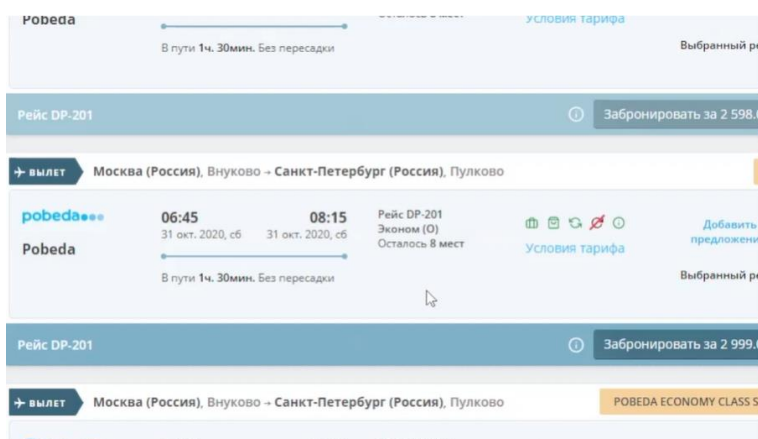


Рисунок 1 – Онлайн-система «Авиацентр»

Онлайн-система помогает эффективно управлять и решать все вопросы по организации деловых поездок в режиме онлайн. Интеграция с различными клиентскими системами, такими как 1С, позволяет полностью автоматизировать процесс и сократить время и трудозатраты офис-менеджеров, бухгалтерии и руководителей на управление деловыми поездками.

Автоматизация значительно упрощает процесс взаимодействия сотрудников, вовлеченных в процесс подбора, согласования и оформления деловых поездок. Все операции можно совершить удаленно, нужен только доступ к интернету и логин/пароль от учетной записи в онлайн-системе.

Онлайн-система позволяет полностью управлять бронированием сотрудников на каждом этапе организации деловых поездок – от бронирования авиа- и ж/д билетов и отелей до закрывающих документов по деловой поездке.

Недостатки данной системы заключаются в отсутствии автоматического бронирования таких услуг, как такси и трансферы, а также в устаревшем интерфейсе системы.

1.2.2 Сервис оформления командировок «Ostrovok.ru Командировки»

Сервис «Ostrovok.ru Командировки» (рисунок 2) разработан для тревел-координаторов, организующих деловые поездки для сотрудников компаний, самостоятельно бронирующих для себя все составляющие деловой поездки, а также для бухгалтеров, имеющих доступ к отчетности по всем поездкам [3].

The screenshot displays the Ostrovok.ru booking interface. At the top, it indicates the carrier is UTAir and the time limit is 10 minutes. The main section shows a round trip: 'Туда' (There) on Jan 13, 2022, from Moscow (VKO) to Istanbul (IST) on flight TK-4416, departing at 21:10 and arriving at 00:15. The return flight 'Обратно' (Back) is on Jan 20, 2022, from Istanbul (IST) to Moscow (SVO) on flight N4-704, departing at 13:55 and arriving at 17:35. Both are direct flights. Below the flight details, the fare conditions are shown: 'Условия тарифа' (Fare conditions), 'Класс обслуживания: Эконом' (Service class: Economy), and a price of 40 572 RUB. A 'Выбрать' (Select) button is visible. The bottom section lists additional conditions: 'Можно обменять' (Can be exchanged), 'Багаж не входит в стоимость' (Baggage not included in the price), 'Невозвратный' (Non-refundable), and 'Ручная кладь: 1 место 5 кг' (Hand luggage: 1 piece 5 kg).

Рисунок 2 – Сервис организации командировок «Ostrovok.ru Командировки»

Преимущества данной системы:

- отсутствие абонентской платы;
- помимо бронирования стандартных услуг, таких как авиабилеты, ж/д билеты и отели, в системе представлен функционал аренды автомобиля без водителя и страховка;
- более 180 поставщиков услуг;
- большое количество прямых контрактов с отелями;
- мультиредактирование учетных записей пользователей (возможность внести одновременные изменения для 500 пользователей);
- кешбэк за бронирования.

Недостатки системы:

- отсутствует услуга бронирования такси;
- отсутствует услуга бронирования аэроэкспресса;
- отсутствует услуга бронирования ж/д билетов;
- нельзя купить справку о перелете.

Несмотря на то, что в данной системе отсутствует множество видов услуг, собственный кешбек за бронирования является большим преимуществом над другими системами.

1.2.3 Платформа онлайн-бронирования «Вкомандировке»

«Вкомандировке» – это платформа онлайн-бронирования, которая построена по модульному принципу и может быть доработана под любые индивидуальные требования и запросы клиента, чтобы полностью соответствовать его бизнес-процессам [4]. Сервис автоматизирует покупку любых тревел-услуг для сотрудников и процесс документального сопровождения командировки, а именно предоставляет возможность закрывать документы в одном сервисе (рисунок 3).

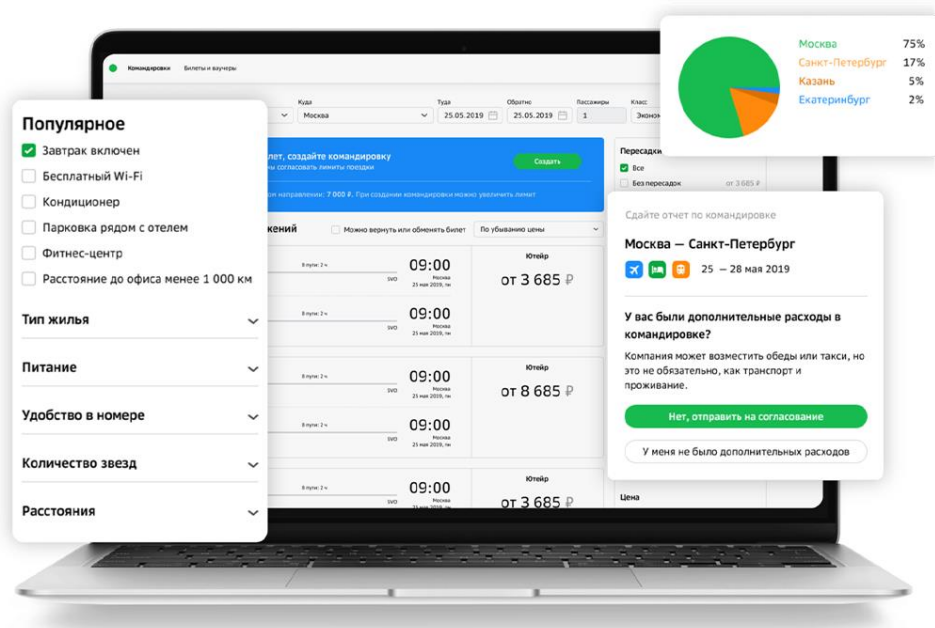


Рисунок 3 – Платформа онлайн-бронирования «Вкомандировке»

Сервис легко интегрируется во все основные ERP-системы (1С, Oracle, SAP и другие), что обеспечивает автоматическое оформление авансовых отчетов и закрывающих документов. При бронировании перелета и гостиницы система учитывает полученные от клиента актуальные данные о должности и статусе сотрудника, его принадлежности к тому или иному центру затрат, допустимых лимитах.

Недостатки системы:

- нет возможности бронирования такси;
- отсутствует поиск отелей на карте;
- нет услуги «Аэроэкспресс».

Среди преимуществ сервиса можно выделить автоматизацию процесса оформления командировок сотрудников, планирование и учет расходов, бронирование гостиниц и оформление билетов

1.2.4 Система организации деловых поездок «Ozon Командировки»

«Ozon Командировки» – это система организации деловых поездок для сотрудников, сделанная на базе интернет-магазина «Ozon» [5]. Отличительная

особенность данного сервиса заключается в том, что покупать товары для бизнеса и оформлять командировки можно на одной площадке (рисунок 4).

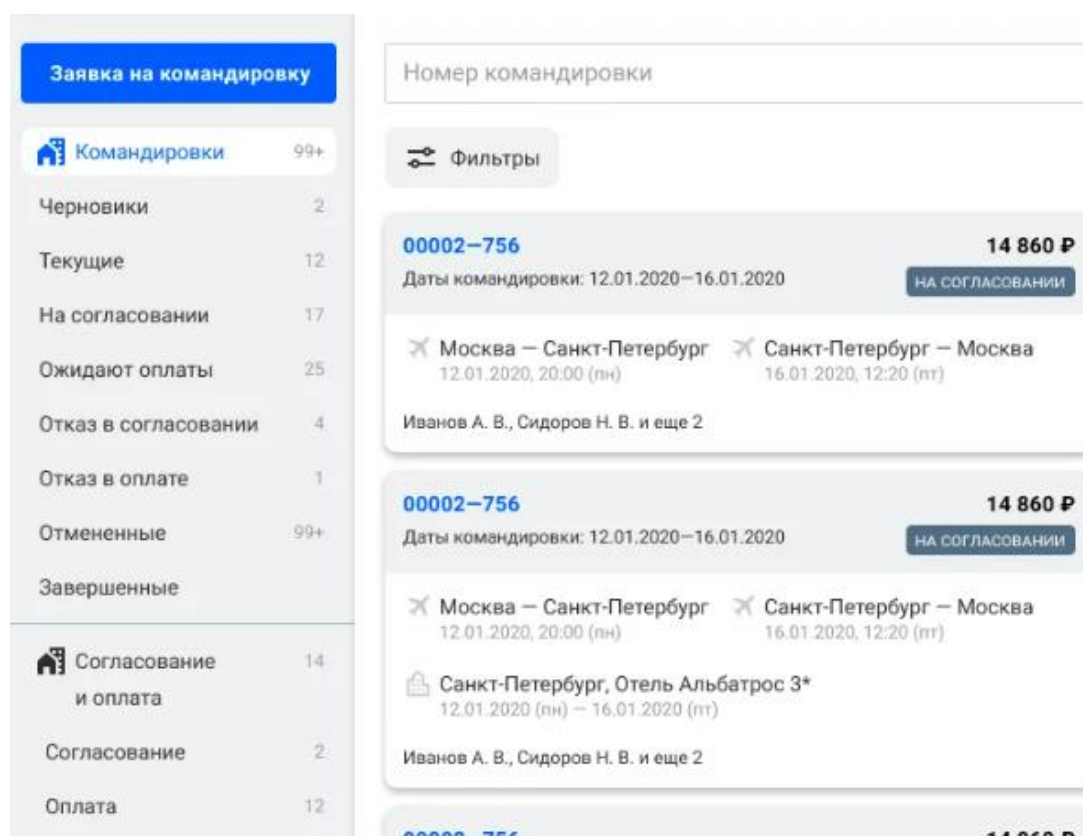


Рисунок 4 – Система организации деловых поездок «Ozon Командировки»

Сервис «Ozon Командировки» предлагает следующее:

- контроль тревел-расходов компании на единой платформе;
- заказ дополнительных услуг через выделенного менеджера;
- электронный документооборот и автоматизация процессов;
- выбор лучших предложений перевозчиков и отелей.

Можно выделить следующие недостатки системы:

- среди услуг, которые можно купить автоматически, представлены только авиабилеты, ж/д билеты и отели;
- нет функции автоматической отмены и обмена билетов;
- отсутствие аналитики по командировкам компании.

Отличительной особенностью «Ozon Командировки» является собственная система бонусных баллов.

1.2.5 Сравнительный анализ платформ бронирования деловых поездок

Исследовав бизнес-процессы, которые возникают во время использования систем бронирования деловых поездок, были составлены коэффициенты значимости критериев, которые представлены в таблице 1.

Таблица 1 – Критерии оценки систем бронирования деловых поездок

№	Наименование	Коэффициент значимости
1	Бронирование авиабилетов	1
2	Бронирование ж/д билетов	1
3	Бронирование отелей	1
4	Бронирование аэроэкспрессов	0,75
5	Бронирование такси	0,75
6	Бронирование кастомных услуг	0,5
7	Бронирование трансферов	0,5
8	Обмен и возврат авиабилетов	0,75
9	Автоматическое формирование закрывающих документов	0,75
10	Наличие бонусов или кешбека	0,5
11	Наличие аналитики по поездкам и тратам	0,25
12	Поиск отелей на карте	0,5

Для оценки описанных выше критериев используется трехбальная шкала реализации, которая отражена в таблице 2.

Таблица 2 – Шкала баллов

Балл	Характеристика оценки
0	Функция не реализована
1	Функция частично реализована
2	Функция полностью реализована

Сведем результаты анализа в таблицу 3.

Таблица 3 – Сравнительный анализ продуктов-аналогов

Критериальная оценка	Авиацентр	Ostrovok.ru Командировки	Вкомандиро вке	Ozon Командировки
Бронирование авиабилетов	2	2	2	2
Бронирование отелей	1	2	1	1
Бронирование ж/д билетов	0	0	2	2
Бронирование аэроэкспрессов	2	0	0	2
Бронирование трансферов	2	0	2	0
Бронирование такси	0	0	0	0
Обмен и возврат билетов	2	2	2	0
Бронирование кастомных услуг	1	1	1	1
Автоматическое формирование закрывающих документов	2	2	2	2
Наличие бонусов и кешбека	0	2	0	2
Наличие аналитики по поездкам и тратам	2	1	1	0
Поиск отелей на карте	0	2	0	0

После получения результатов оценивания рассматриваемых систем по указанным выше критериям, был произведен расчет аддитивной суммы интегральной оценки для каждого сравниваемого продукта по следующей формуле:

$$O_{пп} = \sum_{i=1}^n (Z_i * K_i),$$

где $O_{пп}$ – интегральная оценка программного продукта;

n – количество критериев сравнения заявленных продуктов согласно таблице критериев;

Z_i – значение выполнения критерия;

K_i – коэффициент значимости критерия сравнения в пределах от 0 до 1.

Результаты подсчета отражены в таблице 4.

Таблица 4 – Результаты подсчета интегральной оценки

Авиацентр	Ostrovok.ru Командировки	Вкомандировке	Ozon Командировки
8	10,25	9,75	9,5

Диаграмма с результатами подсчета изображена на рисунке 5.

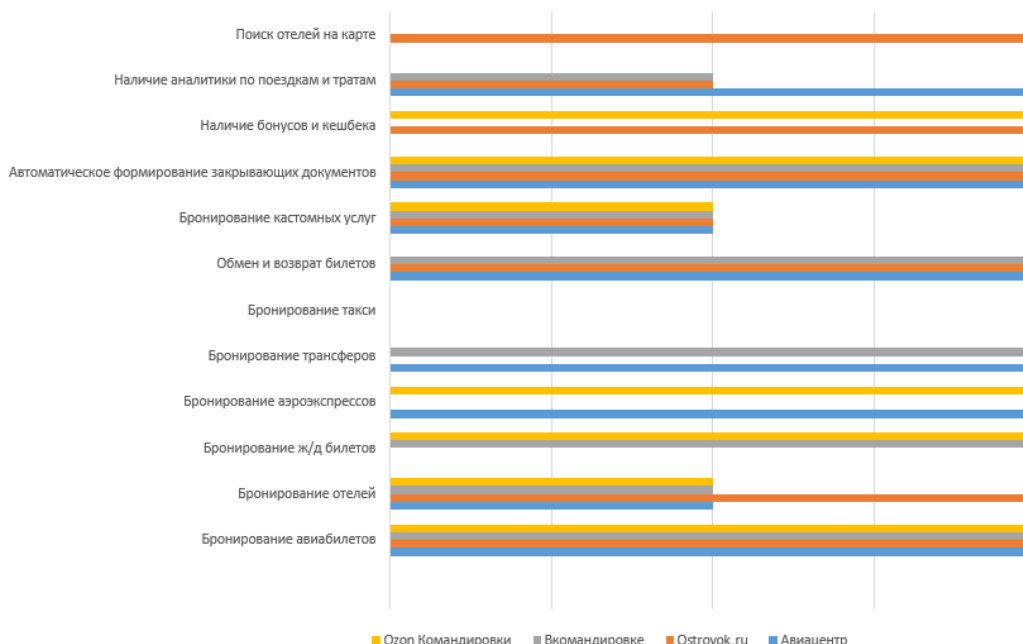


Рисунок 5 – Круговая диаграмма результатов подсчет интегральной оценки

Результаты проведенного анализа показывают, что все рассмотренные системы похожи между собой и почти одинаковы по критериальной оценке. Однако они имеют ограниченный инструментарий для бронирования деловых поездок. По итогам расчета интегральной оценки наиболее выигрышным оказался сервис «Ostrovok.ru Командировки». Таким образом, в выборе системы бронирования поездки решающую роль играет разнообразие услуг для автоматического бронирования.

1.3 Постановка задачи на разработку информационной системы бронирования деловых поездок

1.3.1 Общее описание системы

Разрабатываемое приложение автоматизирует бронирование услуг различного типа с помощью обращения к сторонним сервисам.

Система должна обеспечить следующий функционал, который сможет подстроиться под запросы любой компании. Сервис должен предоставлять возможности:

- найти и купить билеты для поездки в любую точку мира у авиакомпаний и железнодорожных поставщиков по самому выгодному тарифу;
- забронировать номер в отеле при выборе из сотен тысяч вариантов;
- предоставление аналитики по командировкам в графическом и текстовом виде.

Приложение работает в online-режиме и доступно пользователям в сети Интернет.

1.3.2 Роли системы

Администратор:

- создание ограничений на поездки на категории сотрудников;
- создание и редактирование сотрудников и их личных кабинетов;
- управление бюджетом;
- получение документов для бухгалтерии.

Сотрудник:

- бронирование поездок (транспорт и проживание).

Администратор одновременно может выполнять роль сотрудника.

1.3.3 Основные бизнес-процессы

Бизнес-процесс «Бронирование поездки»:

- сотрудник ищет авиабилеты, железнодорожные билеты или отели на подходящие даты в нужном направлении;
- сотрудник добавляет в корзину выбранные позиции;
- при выписке услуг из корзины происходит бронирование у сторонних поставщиков услуг.

Бизнес-процесс «Автоматическое получение билетов на почту»:

- после каждой покупки формируется ваучер на оказываемую услугу;

– ваучер автоматически отправляется на почту сотрудника и администратора личного кабинета.

Таким образом, в бизнес-процессах задействованы сторонние сервисы, такие как почтовый сервер и поставщики услуг.

1.3.4 Интерфейсы личных кабинетов

У каждой из ролей есть страницы «Главная», «Корзина», «Профиль» (личные данные), а также «Поиск авиабилетов, железнодорожных билетов и отелей».

Администратор имеет страницы, присущие только ему:

- панель с доступными средствами и задолженностью;
- страница управления сотрудниками;
- страница «Бухгалтерия»;
- аналитика.

Сотрудник не имеет страниц, присущих только его роли.

1.3.5 Нефункциональные требования

Локализация: сайт должен быть реализован на русском языке. Предусмотрена возможность перевода интерфейса программы на английский язык. Для английского языка не предусмотрен перевод бизнес-данных.

Дизайн: в цветовой схеме используются только четыре цвета: белый, черный, красный и серый. Стилизация сайта в едином формате.

Поддержка браузеров: разрабатываемое приложение должно корректно открываться и функционировать в текущих версиях (на момент разработки) основных популярных браузеров – Chrome, Firefox, Safari, Яндекс.Браузер, Safari.

1.3.6 Функциональные требования

Изобразим основные функции системы и пользователей при помощи языка UML на диаграмме вариантов использования (рисунок 6).

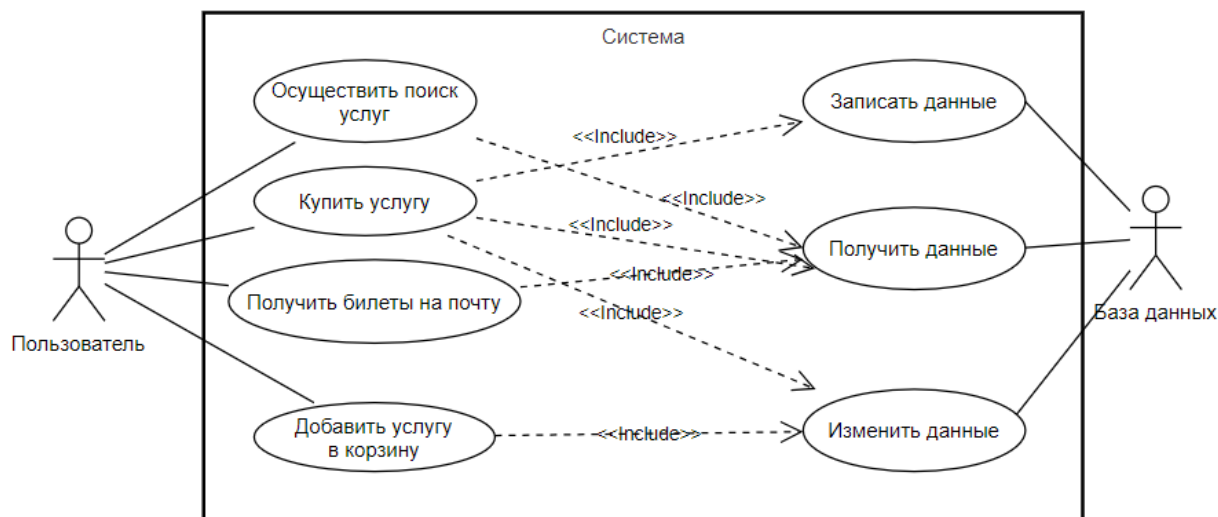


Рисунок 6 – Диаграмма вариантов использования сервиса

Подробное описание прецедента «Купить услугу» приведено на рисунке 7.

Прецедент: Купить услугу
ID: 1
Краткое описание: Пользователь бронирует услугу, предварительно добавленную в корзину
Главные актеры: Пользователь
Второстепенные актеры: Нет.
Предусловия: Пользователь авторизован. Услуга добавлена в корзину
Основной поток:
1. Пользователь открывает корзину
2. Include (Получить данные)
3. Система отображает данные по корзине пользователю
4. Пользователь оплачивает услугу
5. Система отправляет запрос провайдеру услуги на бронирование
6. Система ожидает ответа
7. Если пришел успешный ответ
7.1 Система сообщает пользователю о том, что услуга забронирована
7.2 Include (Изменить данные)
7.3 Система удаляет услугу из корзины
7.4 Система сохраняет услугу в поездках
7.5 Include (Записать данные)
8. Иначе
8.1 Система сообщает пользователю об ошибке
Постусловие: Система отправляет пользователю счета и ваучеры на почту
Альтернативные потоки: Нет

Рисунок 7 – Спецификация прецедента «Купить услугу»

Описание прецедента «Записать данные» приведено на рисунке 8.

Прецедент: Записать данные
ID: 2
Краткое описание: База данных сохраняет данные в таблицу
Главные актеры: База данных
Второстепенные актеры: Пользователь
Предусловия: Нет.
Основной поток: 1. Пользователь покупает услугу 2. Система обрабатывает данные и отправляет запрос на сохранение в БД забронированной услуги 3. БД записывает данные
Постусловие: Данные сохранены
Альтернативные потоки: Нет.

Рисунок 8 – Сценарий варианта использования «Записать данные»

Сценарий варианта использования «Получить данные» описан на рисунке 9.

Прецедент: Получить данные
ID: 3
Краткое описание: База данных получает данные
Главные актеры: База данных
Второстепенные актеры: Пользователь
Предусловия: Нет.
Основной поток: 1. Пользователь открывает корзину 2. Система отправляет запрос на получение в БД корзины по Id 3. БД получает данные
Постусловие: Система получает данные из БД и отдает их пользователю
Альтернативные потоки: Нет.

Рисунок 9 – Сценарий варианта использования «Получить данные»

Сценарий варианта использования «Изменить данные» описан на рисунке 10.

Прецедент: Изменить данные
ID: 4
Краткое описание: База данных изменяет данные
Главные актеры: База данных
Второстепенные актеры: Пользователь
Предусловия: Нет.
Основной поток: 1. Пользователь бронирует услугу 2. Система отправляет запрос на изменение статуса услуги 3. БД изменяет статус услуги на забронированную
Постусловие: Нет.
Альтернативные потоки: Нет.

Рисунок 10 – Сценарий варианта использования «Изменить данные»

Таким образом, были проанализированы основные функции системы и пользователей при помощи языка UML, сформированы требования к разрабатываемому программному продукту и выделены бизнес-процессы системы.

2 Проектирование системы бронирования деловых поездок

2.1 Проектирование базы данных

Для приложения была спроектирована концептуальная модель базы данных (рисунок 11). Под концептуальной моделью понимают отражение предметной области для разрабатываемой базы данных [6].

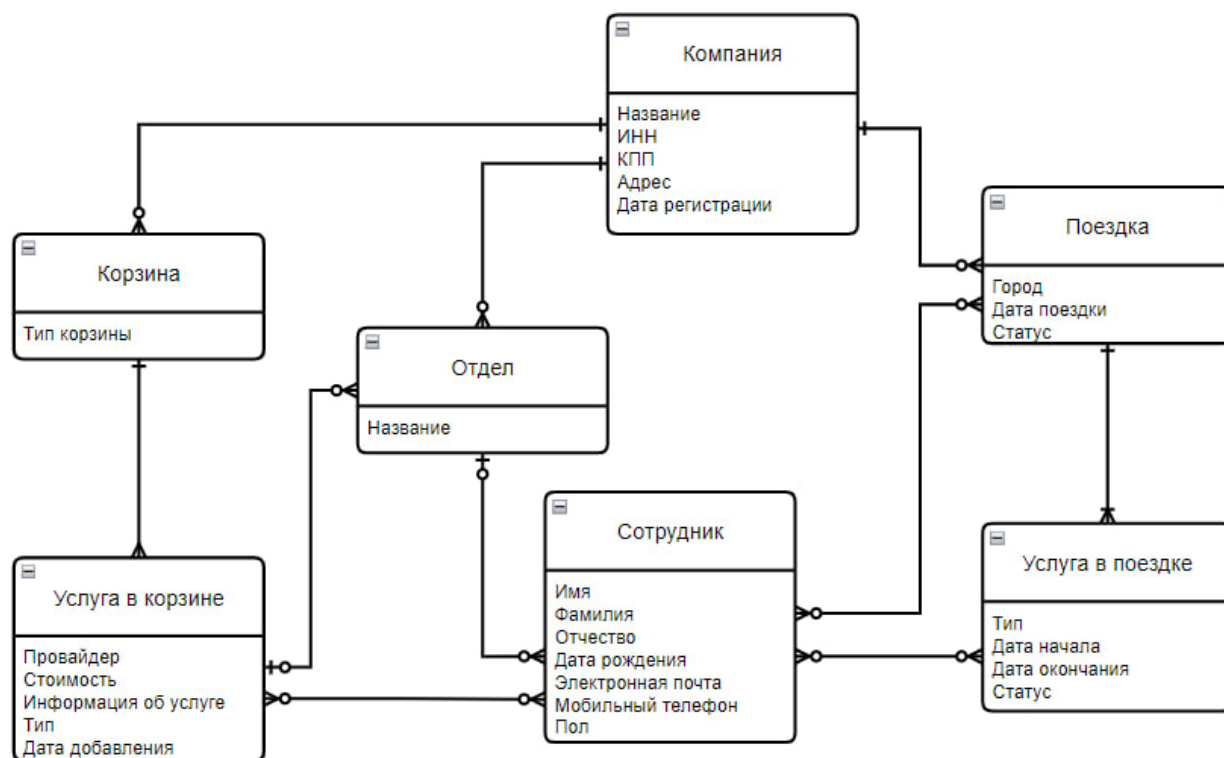


Рисунок 11 – Концептуальная модель базы данных

В MongoDB для данных от поставщиков авиаперелетов была спроектирована коллекция «airlineSearch». Пример записи из коллекции представлен на листинге в приложении Б.

Спецификация коллекции представлена в таблице 5.

Таблица 5 – Спецификация коллекции «airlineSearch»

Название поля	Описание
SearchId	Идентификатор поиска
Trips	Информация о найденном билете
Routes	Полный маршрут перелета. Содержит более одного элемента, если выбран сложный маршрут

Продолжение таблицы 5

Название поля	Описание
UserId	Идентификатор пользователя
SearchOptions	Параметры, использовавшиеся при этом поиске
ProviderName	Название провайдера
Price	Стоимость билета
Fares	Тарифы билета
Duration	Продолжительность перелета

Для создания коллекций в MongoDB не требуются скрипты и миграции, коллекции и базы данных создаются в момент обращения к ним, если их не существует.

2.2 Проектирование программных средств

Диаграмма классов сервиса, отвечающего за бронирование услуг, представлена на рисунке 12.

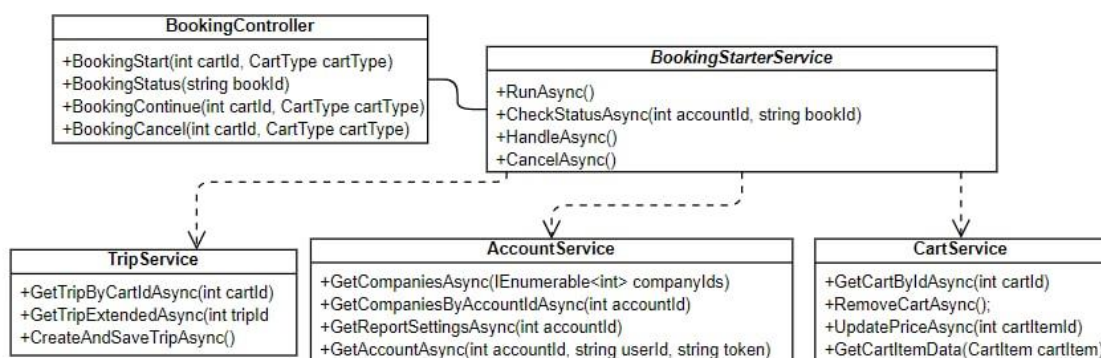


Рисунок 12 – Диаграмма классов микросервиса бронирования

Спецификация класса TripService приведена на рисунке 13.

Класс «TripService» - класс с функционалом предназначенным для взаимодействия с микросервисом поездок	
GetTripByCartIdAsync(int cartId)	Получение поездки по идентификатор корзины
GetTripExtendedAsync(int tripId)	Получение поездки по идентификатору
CreateAndSaveTripAsync()	Создание и сохранение поездки

Рисунок 13 – Спецификация класса TripService

Спецификация класса AccountService приведена на рисунке 14.

Класс «AccountService» - класс сервис для работы с данных аккаунта пользователя	
GetCompaniesAsync(IEnumerable<int> companyIds)	Получение компаний по списку идентификаторов компаний
GetCompaniesByAccountIdAsync(int accountId)	Получить все компании по идентификатору аккаунта
GetAccountAsync(int accountId)	Получение аккаунта по идентификатору

Рисунок 14 – Спецификация класса AccountService

На рисунке 15 изображен сценарий выполнения информационных задач (процессов) по бронированию поездки действующими лицами в виде диаграммы деятельности UML.

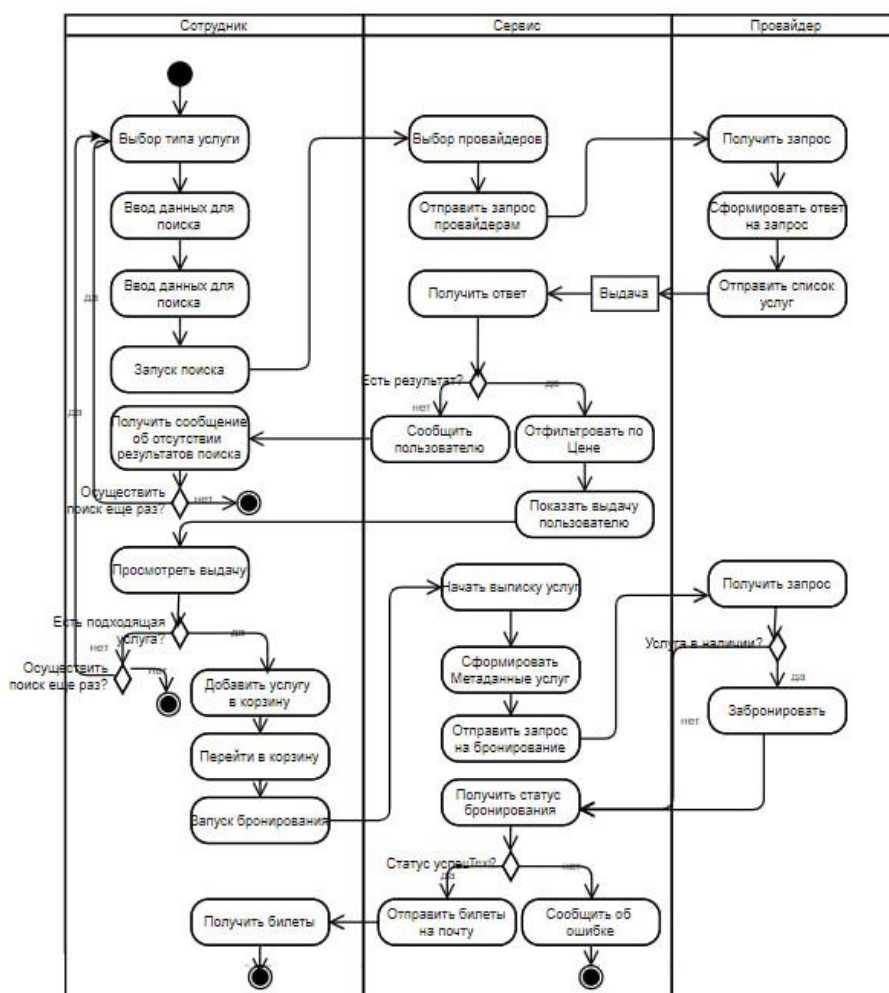


Рисунок 15 – Сценарий выполнения информационных процессов

Спецификации информационных процессов приведены в таблице 6.

Таблица 6 – Спецификации информационных процессов

Название информационного процесса	Описание	Действующие лица
Поиск услуг	Сотрудник может выполнить поиск по различным типам услуг: авиабилеты, железнодорожные билеты, гостиницы. Сервис должен осуществить запросы соответствующим провайдерам и отобразить полученные данные пользователю	Сотрудник Сервис
Выбор услуг	Сотрудник может выбрать подходящую услугу из списка услуг	Сотрудник
Пополнение корзины	Сотрудник может добавить услугу себе в корзину	Сотрудник
Получение доступа к покупке услуг	Сотрудник может получить доступ к покупке услуг, если корзина согласована, либо пользователь имеет право покупать без согласования	Сотрудник
Бронирование	Сотрудник может купить услуги из корзины, при этом со счета будут списаны средства Сервис должен осуществить выписку услуг у провайдера и уведомить пользователя об успешности выписки	Сотрудник
Отправка ваучеров на почту	После бронирования сотрудник получает все билеты и ваучеры на электронную почту	Сотрудник

Прежде чем выполнить любой бизнес-процесс в рамках системы, пользователь должен пройти авторизацию.

В результате проектирования были смоделированы бизнес-процессы бронирования деловых поездок.

3 Реализация программного обеспечения

3.1 Разработка обобщенной архитектуры системы

SOA (сервис-ориентированная архитектура) – архитектура ПО, при которой система представляет собой набор слабосвязанных автономных взаимодействующих модулей (сервисов), которые можно разрабатывать, вводить в эксплуатацию независимо друг от друга [7].

Микросервисная архитектура – это чистый SOA-подход, согласно которому сервисы должны в полной мере соответствовать следующим ключевым критериям:

- «stateless»: любой ответ сервера не зависит от какого-либо состояния;
- независимость: сервисы на протяжении всего жизненного цикла не зависят друг от друга и от разделяемых состояний;
- единственная ответственность (SRP): у сервиса лишь одно назначение;
- легковесность: зоны ответственности должны быть сравнительно малы;
- приложение должно быть небольшим, с быстрым временем старта;
- возможность увеличивать количество экземпляров каждого из сервисов для осуществления горизонтального масштабирования.

Для разрабатываемого приложения выделены дополнительные важные критерии, которые описывают концептуально более строгую микросервисную архитектуру:

- ответственность микросервиса определяется бизнес-требованиями таким образом, чтобы бизнес-транзакция целиком выполнялась одним сервисом и не распределялась между несколькими;
- сервисы должны оперировать собственными хранилищами данных и ни в коем случае не должны использовать чужие хранилища.

Реализованная архитектура сервиса изображена на крупноблочной схеме (рисунок 16).

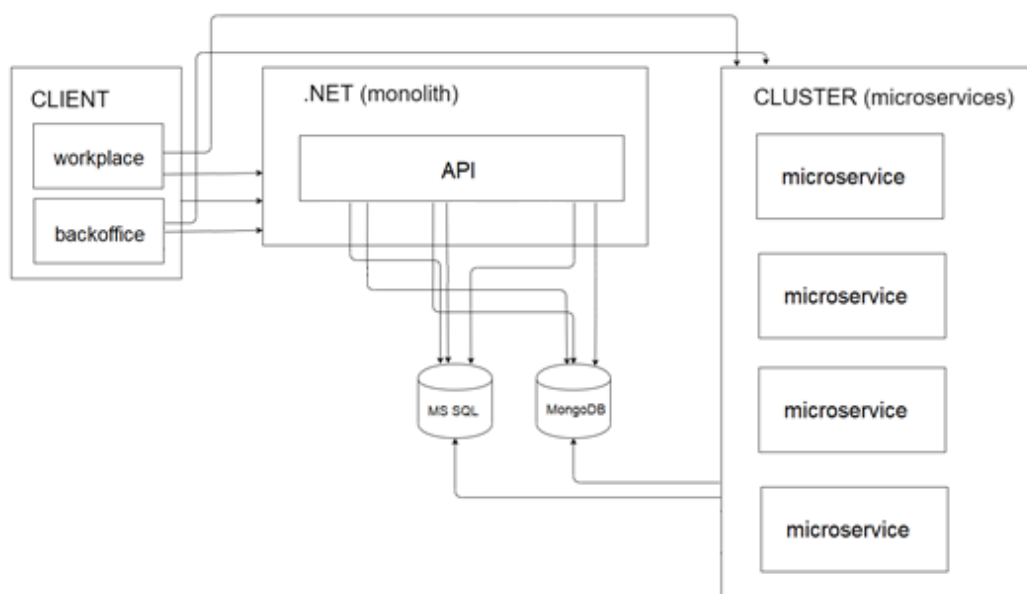


Рисунок 16 – Крупноблочная схема сервиса

Текущая сложность серверной части приложения – это следствие монолитной архитектуры, реализованной на старте проекта: API, решающие разные продуктовые задачи (обслуживающие разные части сервиса), оказались относительно тесно интегрированы. Они разделяют общий утилитарный код, общие способы взаимодействия с хранилищами данных и внешними системами; у них общая система управления зависимостями, сборки и доставки. Одним словом, они обладают общей инфраструктурой. Высокая сложность (количество строк кода) функционала всегда со временем перерастает в сложность организации инфраструктуры по естественным причинам.

При этом, в продукте существует четко осязаемое разделение между разделами: авиабилеты, поезда, гостиницы, трансфер, поездки, отчетность, настройки, корзина и т. д. Иногда их функционал пересекается (покупка билета приводит к отправке его в корзину), но, в основном, они существуют независимо.

Приняв во внимание поставленную задачу, выигрышной является стратегия, когда можно независимо вносить изменения в код различных компонентов системы, не затрагивая остальные. Так получаются меньшие по объему и сложности приложения, что уменьшает количество принимаемых разработчиком решений в ходе работы. Это достаточный мотив для разделения

единого API на набор непересекающихся сервисов. При этом инфраструктура изнутри приложения переносится наружу.

Таким образом, новые разрабатываемые компоненты приложения реализуются на базе микросервисной архитектуры.

3.2 Разработка серверной части приложения

Приложение разрабатывалось с применением паттерна проектирования MVC (Model-View-Controller). MVC – это шаблон программирования, который позволяет разделить логику приложения на три части:

- Model (модель): получает данные от контроллера, выполняет необходимые операции и передаёт их в вид;
- View (вид или представление): получает данные от модели и выводит их для пользователя;
- Controller (контроллер): обрабатывает действия пользователя, проверяет полученные данные и передаёт их модели.

Этот паттерн разработки нужен для того, чтобы разделить логические части приложения и создавать их отдельно друг от друга, то есть писать независимые блоки кода, которые можно как угодно менять, не затрагивая другие [8].

Диаграмма классов сервиса, отвечающего за бронирование услуг, представлена на рисунке 17.

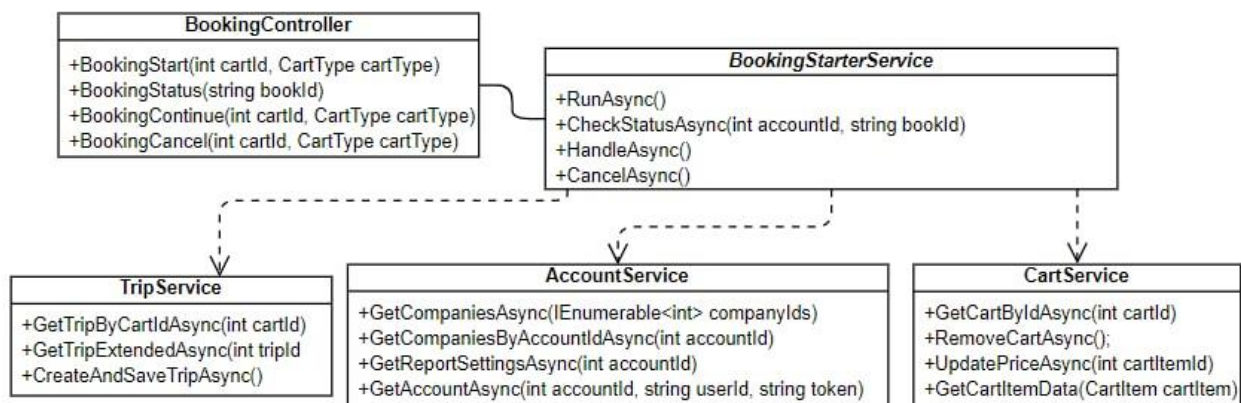


Рисунок 17 – Диаграмма классов

Спецификация класса BookingController приведена в таблице 7.

Таблица 7 – Спецификация класса BookingController

Класс «BookingController» – класс контроллера, на который приходят запросы от клиента	
BookingStart(int cartId, CartType cartType)	Запуск бронирования услуги в корзине
BookingStatus(string bookId)	Проверка статуса бронирования
BookingContinue(int cartId, CartType cartType)	Продолжить бронирование услуг
BookingCancel(int cartId, CartType cartType)	Приостановить бронирование услуг

В таблице 8 приведена спецификация класса TripService, предназначенного для управления поездками.

Таблица 8 – Спецификация класса TripService

Класс «TripService» – класс с функционалом, предназначенным для взаимодействия с микросервисом поездок	
GetTripByCartIdAsync(int cartId)	Получение поездки по идентификатор корзины
GetTripExtendedAsync(int tripId)	Получение поездки по идентификатору
CreateAndSaveTripAsync()	Создание и сохранение поездки

В таблице 9 приведена спецификация класса AccountService, предназначенного для данных аккаунта пользователей.

Таблица 9 – Спецификация класса AccountService

Класс «AccountService» – класс сервис для работы с данными аккаунта пользователя	
GetCompaniesAsync(IEnumerable<int> companyIds)	Получение компаний по списку идентификаторов компаний
GetCompaniesByAccountIdAsync(int accountId)	Получить все компании по идентификатору аккаунта
GetAccountAsync(int accountId)	Получение аккаунта по идентификатору

В таблице 10 приведена спецификация класса CartService, предназначенного для функционала корзины.

Таблица 10 – Спецификация класса CartService

Класс «CartService» – класс сервиса для взаимодействия с микросервисом корзины	
GetCartByIdAsync(int cartId)	Получение корзины по идентификатору
RemoveCartAsync(int cartId);	Удаление корзины по идентификатору
UpdatePriceAsync(int cartItemId)	Обновление стоимость услуги в корзине
GetCartItemData(CartItem cartItem))	Получение данных провайдера из услуги в корзине

На диаграмме и спецификациях рассмотрены лишь основные, самые важные с точки зрения бизнес-логики модели.

Диаграмма разработанных компонентов изображена на рисунке 18.

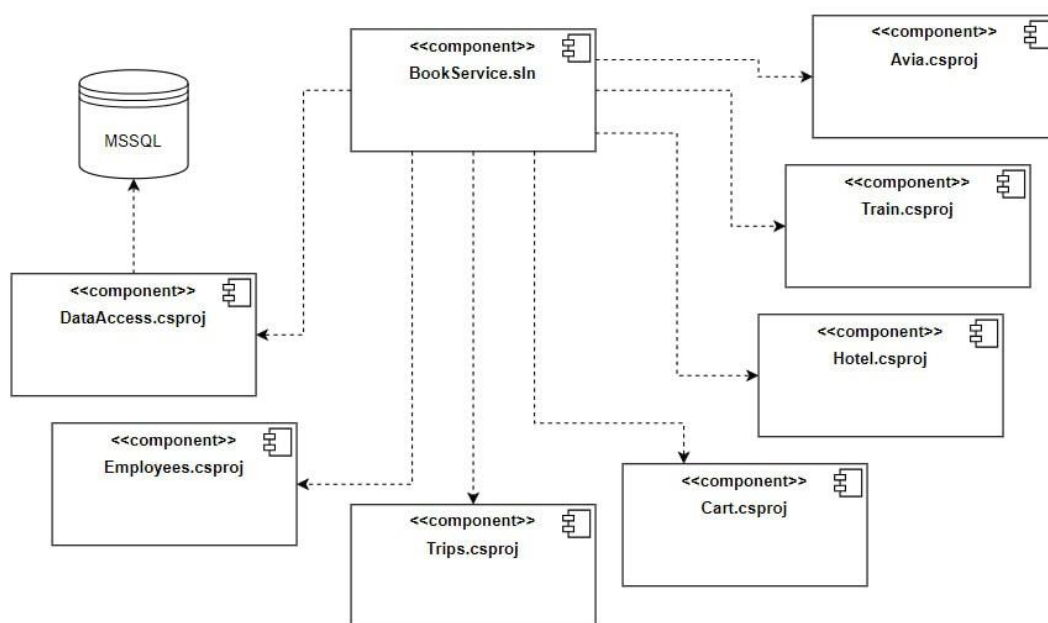


Рисунок 18 – Диаграмма компонентов приложения

На диаграмме компонентов можно увидеть, что каждый компонент соответствует отдельному типу услуг и отвечает за отдельную часть сервиса.

3.2.1 Реализация аналитических бизнес-процессов

Аналитика в приложении представляет собой сводку по бронированиям в виде круговых диаграмм и гистограмм, а также в текстовом (табличном)

формате. За аналитику отвечает отдельный микросервис «analytics-service», в обязанности которого входит сбор аналитики по базе данных MSSQL и запись ее в базу данных PostgreSQL.

Отображение аналитики по авиакомпаниям в пользовательском приложении представлено на рисунке 19.

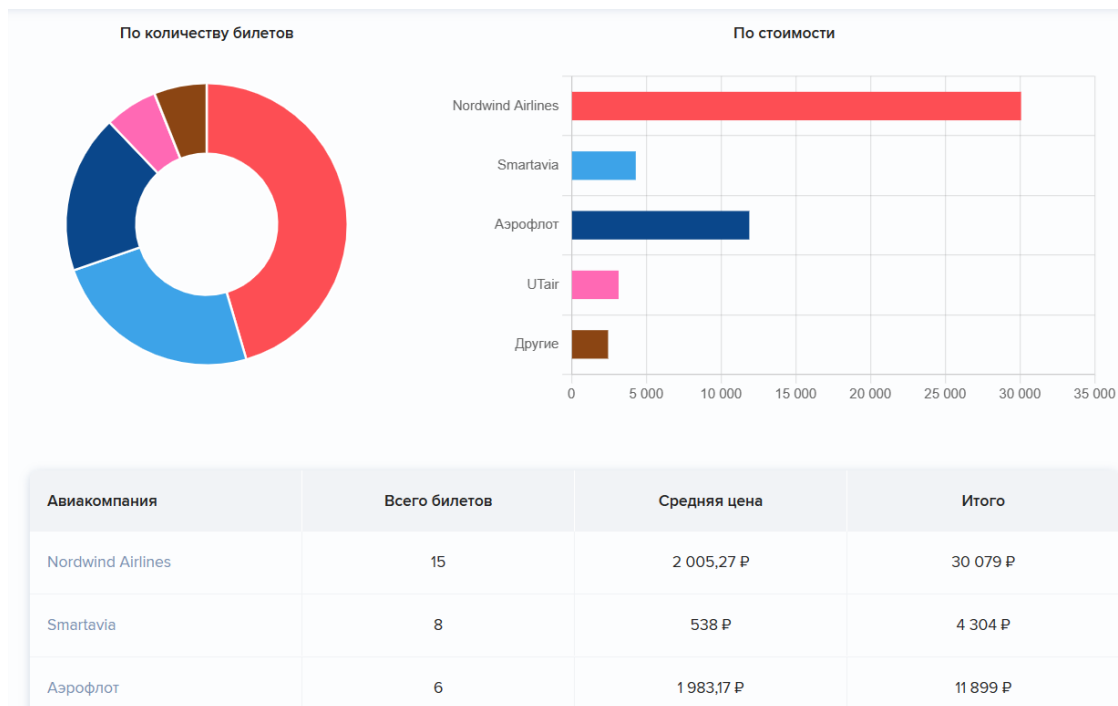


Рисунок 19 – Аналитика по авиакомпаниям

Отображение аналитики по отелям представлено на рисунке 20.

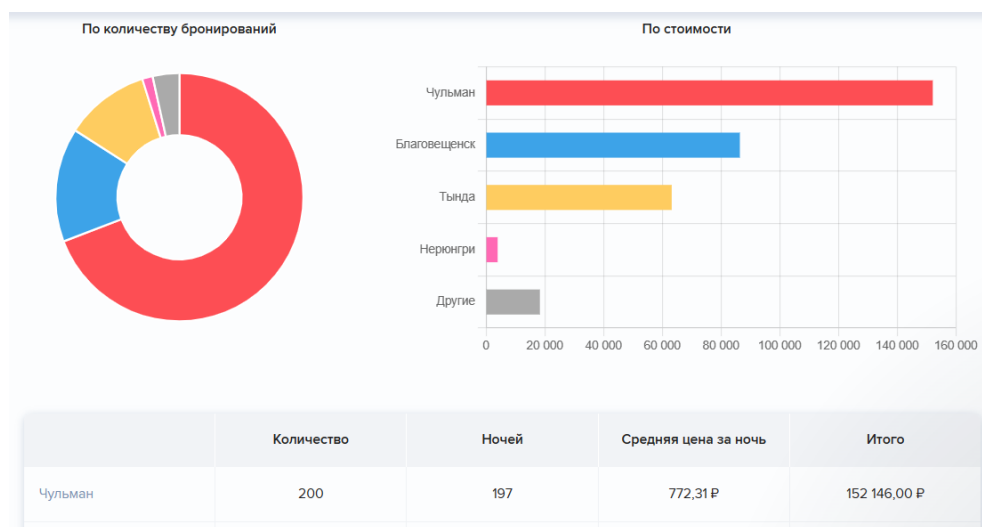


Рисунок 20 – Аналитика по авиакомпаниям

Все аналитические данные хранятся в одной таблице с большим количеством столбцов. Преимущество такого подхода – улучшение быстродействия за счет отсутствия дорогостоящих операций JOIN, а также ввиду наличия динамических данных – простота масштабирования.

Сценарий создания и получения аналитики изображен с помощью диаграммы последовательности на рисунке 21.

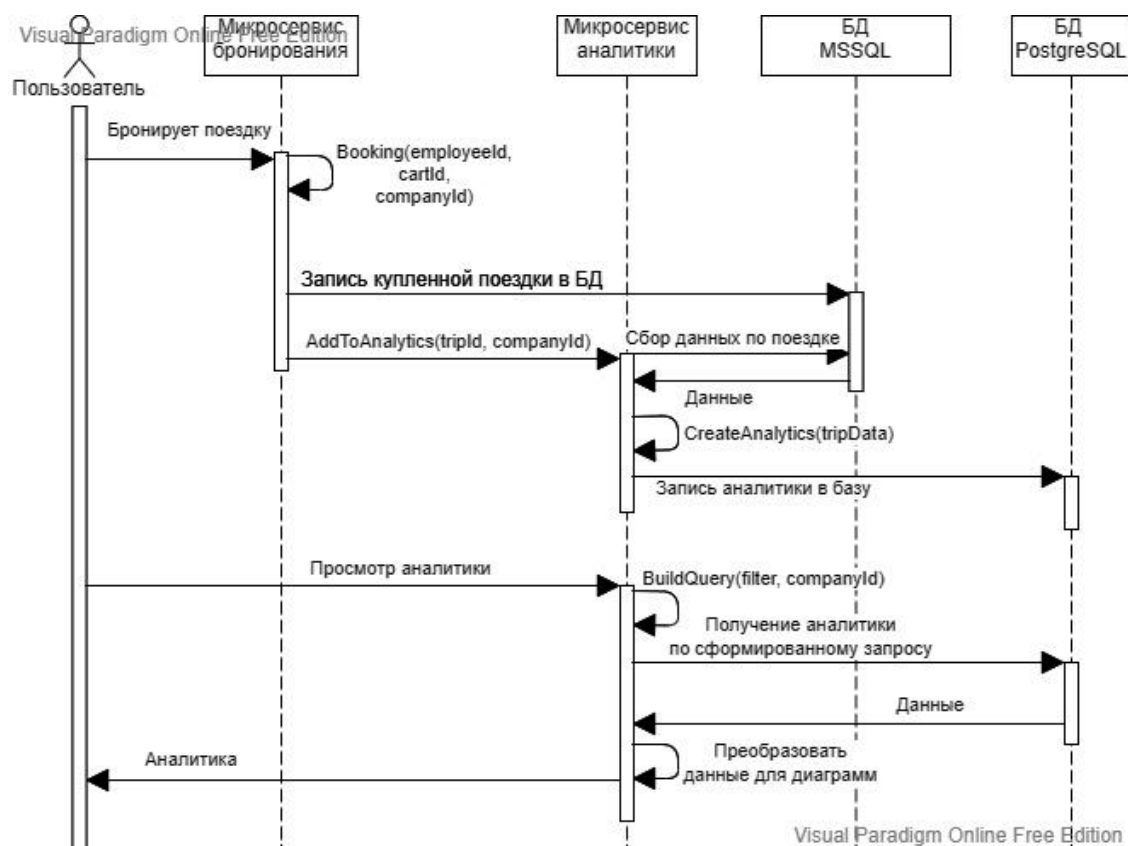


Рисунок 21 – Диаграмма последовательности функционирования аналитики

Для реализации аналитических бизнес-процессов был использован архитектурный паттерн Builder, с помощью которого динамически создается SQL-запрос.

Рассмотрим процесс получения аналитики на примере запроса на просмотр аналитики по отделам.

Модель, которая отправляется контроллеру в запросе, имеет следующую структуру:

– startDate – начало периода;

- endDate – конец периода;
- companyIds – список айди компаний;
- filter – фильтр по отделам, сотрудникам и т.д.

Рассмотрим класс `SqlBuilder`, с помощью которого реализуется формирование sql-запроса. Поля класса приведены на листинге 1.

Листинг 1 – Поля класса `SqlBuilder`

```
private string sql;
private List<string> selectList = new List<string>();
private List<string> whereList = new List<string>();
private List<string> orderList = new List<string>();
private List<string> groupList = new List<string>();
```

Поле `sql` предназначено для каркаса запроса, который имеет вид, приведенный на листинге 2.

Листинг 2 – Каркас запроса на получение авиакомпаний

```
select
    count(1)                as "Count",
    round(sum(amount), 2)    as "Amount",
    aircompany_code         as "Code",
    aircompany_name         as "Name"
from analytics
where /**where**/
group by
    aircompany_code,
    aircompany_name
order by
    count(1) desc,
    aircompany_name asc
```

Поля `selectList`, `whereList`, `orderList`, `GroupList` отвечают соответственно за параметры выборки, условия фильтрации, сортировку и группировку. Значения в списки добавляются соответствующими методами класса `SqlBuilder`: `Select(string sql)`, `Where(string sql, object params)`, `OrderBy(string sql)`, `GroupBy(string sql)`.

3.2.2 Реализация автоматически выполняющихся событий

Одно из требований к системе – поддержка актуальной информации в пользовательском приложении. Так как интерфейс имеет иконки со счетчиками, показывающими состояние, например, корзины, необходимо обновлять счетчик при изменении состояния. Чтобы этого достичь, был реализован отдельный микросервис, базирующийся на библиотеке SignalR.

SignalR Core представляет библиотеку от компании Microsoft, которая работает поверх ASP.NET и которая предназначена для создания приложений, работающих в режиме реального времени. В частности, ее можно использовать вместе с ASP.NET Core. SignalR использует двунаправленную связь для обмена сообщениями между клиентом и сервером, благодаря чему сервер может отправлять в режиме реального времени всем клиентам некоторые данные, а клиенты также в режиме реального времени могут данные серверу и другим клиентам.

SignalR используется в приложениях, которые получают данные в реальном режиме времени, например, чаты, социальные сети, игровые приложения, карты, приложения для аукционов, голосований и карт, панели управления, приложения для мониторинга данных и так далее.

Для обмена сообщениями между клиентом и сервером SignalR использует ряд механизмов:

- WebSockets;
- Server-Side Events;
- Long Polling.

Исходя из возможностей клиента и сервера инфраструктура SignalR выбирает наилучший механизм для взаимодействия. В частности, наиболее оптимальным является WebSockets [9].

На рисунке 22 с помощью диаграммы последовательности показана схема взаимодействия пользователя, клиентского приложения, микросервиса бронирования и микросервиса событий.

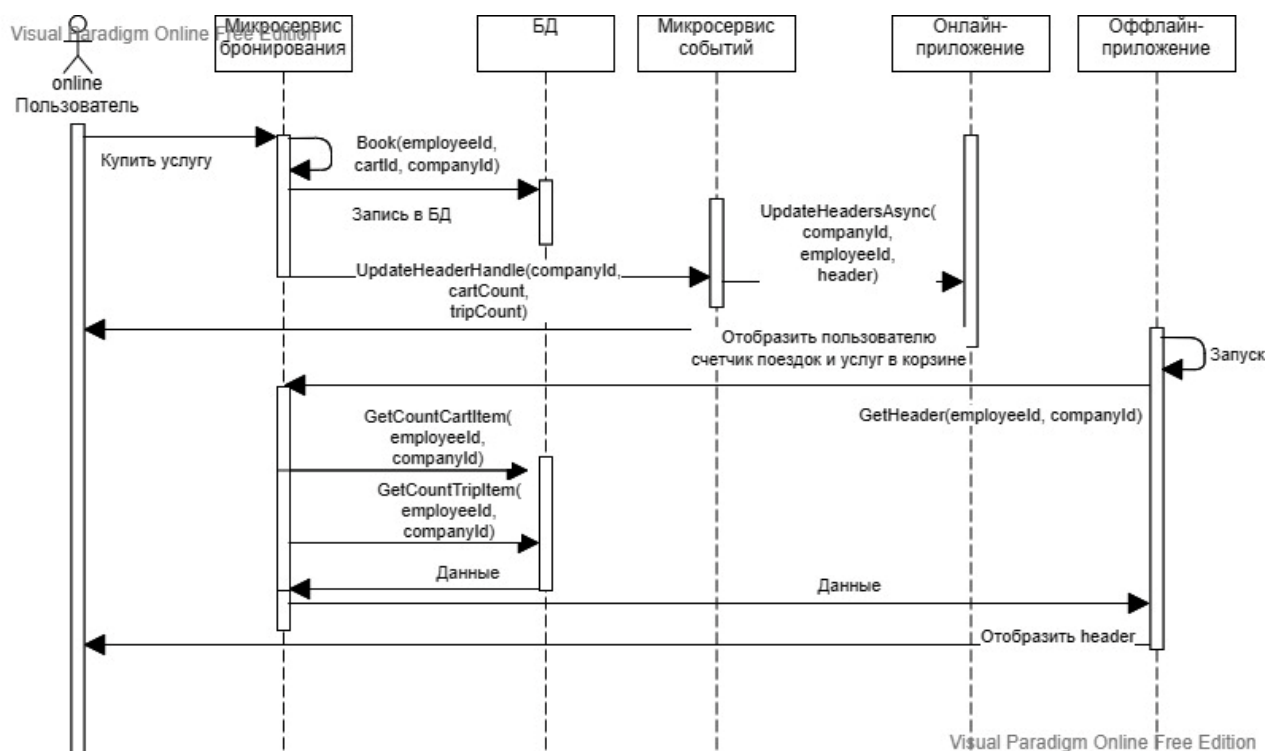


Рисунок 22 – Диаграмма последовательности выполнения автоматического события

На диаграмме отображены следующие методы:

1. `Book(cartId, employeeId, companyId)` – метод бронирования услуг. Изменяет статус услуг в корзине и создает поездку и купленные услуги;
2. `UpdateHeaderHandle(companyId, cartItemCount, tripCount)` – по данному событию осуществляется POST запрос в SignalR, в котором передаются компания, сотрудникам которой нужно выполнить обновление счетчика, счетчик услуг в корзине и счетчик поездок;
3. `UpdateHeadersAsync(companyId, employeeId, header)` – метод отправляет с помощью сокетов пользовательскому приложению данные, которые необходимо отобразить;
4. `GetCountCartItem(employeeId, companyId)` – метод запрашивает из базы данных количество услуг в корзине;
5. `GetCountTripItem(employeeId, companyId)` – метод запрашивает из базы данных количество поездок;

6. `GetHeader(employeeId, companyId)` – получение счетчика при открытии приложения, т. е. для оффлайн-приложения: данный метод запрашивает данные из базы данных и отдает их пользователю при открытии стартовой страницы.

Таким образом были реализованы автоматически выполняющиеся события. Код программы приведен в приложении В.

3.3 Инфраструктура развертывания приложения

Для развертывания приложения использовалась технология Docker – это наиболее популярная контейнерная система. Docker представляет собой комплекс ПО для создания, развертывания и исполнения контейнеров.

Среди преимуществ использования контейнеров выделяют следующие:

- в них можно размещать: код приложений, настройки приложений и код используемых библиотек;
- их использование гарантирует эффективность выполнения операций, возможность контроля версий и единообразие среды;
- они обеспечивают надежное и единообразное развертывание приложений независимо от среды развертывания;
- повышают эффективность ресурсов, позволяя в единой системе выполнять различные процессы [10].

Создать надёжное и качественное приложение, удовлетворяющее требованиям намного легче, если есть возможность тестировать и запускать один и тот же контейнер на разных этапах интеграции. Кроме этого, можно повышать доступность и оптимизировать использование ресурсов.

Немаловажным фактором является также гибкость Docker-контейнеров, обеспечиваемая благодаря их малому размеру, простоте развертывания и портативности. Сервисы упаковки позволяют с легкостью перемещать контейнеры между хостами, защищены от отказов смежных сервисов и ошибочных обновлений в системе хоста [11].

На рисунке 23 изображена диаграмма развертывания приложения.

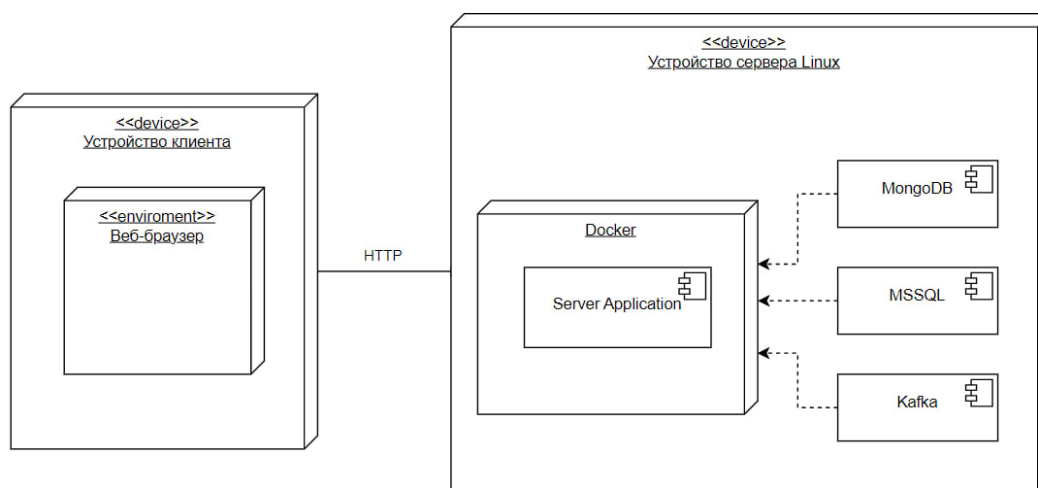


Рисунок 23 – Диаграмма развертывания

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их. Диаграммы развертывания обычно используются для визуализации физического аппаратного и программного обеспечения системы [12].

Как можно увидеть на диаграмме развертывания, серверное приложение развернуто на платформе автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации Docker [13].

4 Контроль качества программного обеспечения

4.1 Функциональное тестирование

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям компании клиента. То есть проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям [14].

Функциональное тестирование методом черного ящика – это тестирование функциональных возможностей ПО, в условиях, когда внутренняя структура и реализация неизвестна.

Произведено функциональное тестирование поиска авиабилетов. Пользователь должен иметь возможность ввести параметры «Город вылета», «Город прилета» и даты «Туда» и «Обратно» (необязательно). Дата «Туда» не может быть датой ранее, чем сегодня. Дата «Обратно» не может быть датой ранее, чем «Туда». «Город прилета» должен отличаться от «Город вылета», оба должны входить в список существующих городов.

Определены следующие граничные условия. Дата вылета должна быть в диапазоне [$<\text{сегодня}>$, ...], дата «обратно» должна быть либо не заполнена, либо лежать в диапазоне [$<\text{дата вылета}>$, ...]. Следовательно, дата «сегодня» и максимально возможная дата, поддерживаемая типом данных `DateTime`, находятся на границах допустимых значений. Эти значения следует проверить. Также имеет смысл проверить ближайшие неверные значения «Вчера» и дату, выходящую за диапазон допустимых значений.

Для того, чтобы построить эффективно-результативные тесты, можно воспользоваться методом функциональных диаграмм. Метод заключается в преобразовании входных описаний программы в функциональную диаграмму (диаграмму причинно – следственных связей) с помощью простейших булевских отношений, построения таблицы решений (методом обратной трассировки),

которая является основой для написания эффективных тестовых наборов данных [15].

Для процесса загрузки набора данных была построена функциональная диаграмма, представленная на рисунке 24.

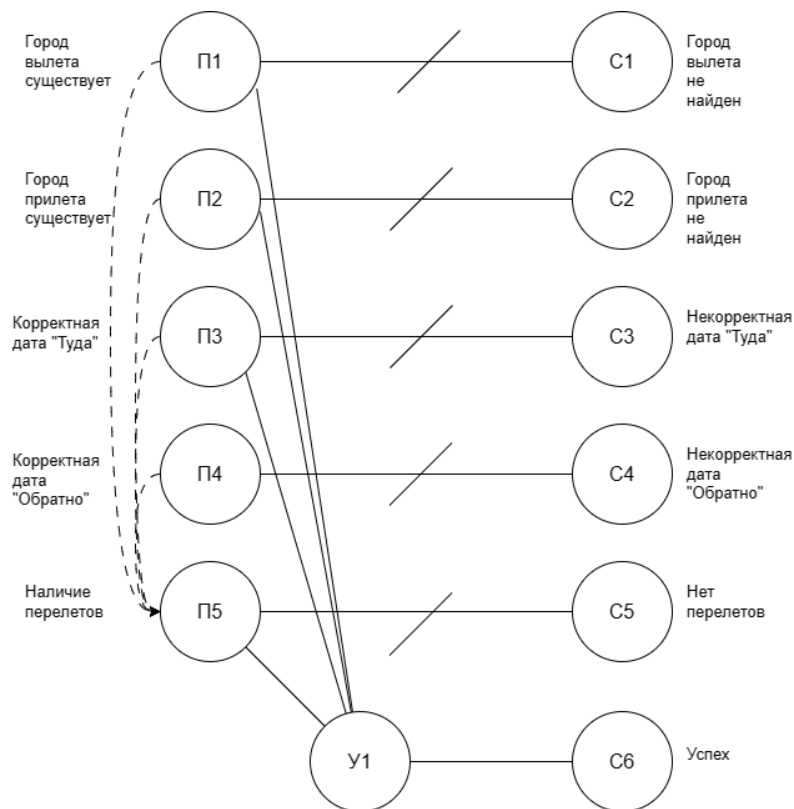


Рисунок 24 – Функциональная диаграмма

Причины:

- город вылета существует;
- город прилета существует;
- корректная дата «Туда»;
- корректная дата «Обратно»;
- по выбранному маршруту на выбранную дату есть перелеты.

Следствия:

- сообщение «Город вылета не найден»;
- сообщение «Город прилета не найден»;

– нельзя указать дату ранее, чем сегодня, и позднее, чем максимально поддерживаемая;

– нельзя указать дату «Обратно» ранее, чем «Туда», и позднее, чем максимально поддерживаемая;

– сообщение «Не найдены перелеты на указанную дату»;

– успех.

На основании функциональной диаграммы была построена таблица решений, представленная в таблице 11.

Таблица 11 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
П1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
П2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
П3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
П4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
П5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
С1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
С2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
С3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
С4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
С5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

В таблице 12 представлен набор тестов, составленных на основе таблицы решений.

Таблица 12 – Набор тестов

№	Входные данные	Выходные данные
1	Город вылета: Пензас Город прилета: Москвас Туда: 10.04.2003 Оттуда: 10.03.2003	Город «Пензас» не существует Город «Москвас» не существует Нельзя указать дату ранее, чем сегодня Дата «Оттуда» не может быть раньше, чем «Туда»
2	Город вылета: Пенза Город прилета: Москвас Туда: 10.04.2003 Оттуда: 10.03.2003	Город «Москвас» не существует Нельзя указать дату ранее, чем сегодня Дата «Оттуда» не может быть раньше, чем «Туда»

Продолжение таблицы 12

№	Входные данные	Выходные данные
3	Город вылета: Пензас Город прилета: Москва Туда: 10.04.2003 Оттуда: 10.03.2003	Город «Пензас» не существует Нельзя указать дату ранее, чем сегодня Дата «Оттуда» не может быть раньше, чем «Туда»
4	Город вылета: Пенза Город прилета: Москва Туда: 10.04.2003 Оттуда: 10.03.2003	Нельзя указать дату ранее, чем сегодня Дата «Оттуда» не может быть раньше, чем «Туда»
5	Город вылета: Пензас Город прилета: Москвас Туда: 10.04.2023 Оттуда: 10.03.2003	Город «Пензас» не существует Город «Москвас» не существует Дата «Оттуда» не может быть раньше, чем «Туда»
6	Город вылета: Пенза Город прилета: Москвас Туда: 10.04.2023 Оттуда: 10.03.2023	Город «Москвас» не существует Дата «Оттуда» не может быть раньше, чем «Туда»
7	Город вылета: Пензас Город прилета: Москва Туда: 10.04.2023 Оттуда: 10.03.2023	Город «Пензас» не существует Дата «Оттуда» не может быть раньше, чем «Туда»
8	Город вылета: Пенза Город прилета: Москва Туда: 10.04.2023 Оттуда: 10.03.2023	Дата «Оттуда» не может быть раньше, чем «Туда»
9	Город вылета: Пензас Город прилета: Москвас Туда: 10.04.2003 Оттуда: -	Город «Пензас» не существует Город «Москвас» не существует Нельзя указать дату ранее, чем сегодня
10	Город вылета: Пенза Город прилета: Москвас Туда: 10.04.2003 Оттуда: -	Город «Москвас» не существует Нельзя указать дату ранее, чем сегодня
11	Город вылета: Пензас Город прилета: Москва Туда: 10.04.2003 Оттуда: -	Город «Пензас» не существует Нельзя указать дату ранее, чем сегодня
12	Город вылета: Пенза Город прилета: Москва Туда: 10.04.2003 Оттуда: -	Нельзя указать дату ранее, чем сегодня
13	Город вылета: Пензас Город прилета: Москвас Туда: 10.04.2023 Оттуда: -	Город «Пензас» не существует Город «Москвас» не существует

Окончание таблицы 12

№	Входные данные	Выходные данные
14	Город вылета: Пенза Город прилета: Москвас Туда: 10.04.2023 Оттуда: -	Город «Москвас» не существует
15	Город вылета: Пензас Город прилета: Москва Туда: 10.04.2023 Оттуда: -	Город «Пензас» не существует
16	Город вылета: Пенза Город прилета: Москва Туда: 10.04.2023 Оттуда: - (на эту дату нет перелетов)	Не найдены перелеты
17	Город вылета: Пенза Город прилета: Москва Туда: 10.04.2023 Оттуда: -	Поиск успешен

Функциональное тестирование показало, что при загрузке набора данных программа работает правильно. Результаты тестирования представлены в приложении В.

4.2 Модульное тестирование

Модульное тестирование позволяет проверить в автоматическом режиме корректность работы отдельных модулей программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Также при модульном тестировании необходимо знать метрику покрытия кода. Она показывает процент исходного кода программы, который был выполнен в процессе тестирования [16].

Для валидации русского и иностранного паспорта при покупке были написаны unit-тесты. Для написания использовался nugget-пакет «NUnit» и «Mock».

«NUnit» – это открытая среда юнит-тестирования приложений для .NET.

При создании модульных тестов почти всегда необходимо использовать платформу «Mock Object». Платформа макета объектов позволяет создавать макеты и заглушки для классов в приложении [17].

Например, можно использовать платформу «Mock Object» для создания макетной версии класса репозитория. Таким образом, можно использовать макетный класс репозитория вместо реального класса репозитория в модульных тестах. Использование макетного репозитория позволяет избежать выполнения кода базы данных при выполнении модульного теста.

Исходный код unit-тестов приведен в приложении Г.

4.3 Метрики устойчивости и абстрактности компонент

Метрика устойчивости для компоненты отражает количественную меру работы, которую необходимо произвести при изменении других зависимых компонент.

Для вычисления позиционной устойчивости кода используется следующая формула:

$$I = \frac{Ce}{Ca + Ce},$$

где Ce – количество классов вне данного компонента, зависящих от классов внутри компонента;

Ca – количество классов внутри данного компонента, зависящих от классов вне компонента.

Величина меры неустойчивости изменяется в диапазоне от $[0, 1]$. Если $I = 0$, то компонент максимально устойчив, если $I = 1$, то компонент максимально неустойчив.

Для компонента TripService была рассчитана величина метрики неустойчивости:

$$I = \frac{2}{10 + 2} = 0.16.$$

Мерой абстрактности компонента служит метрика A . Ее значение определяется простым отношением количества интерфейсов и абстрактных классов к общему числу классов в компоненте [18].

$$A = \frac{Na}{Nc},$$

где Na – количество абстрактных классов в компоненте;

Nc – количество классов в нем же.

Значение метрики A изменяется в диапазоне от 0 до 1. 0 означает полное отсутствие абстрактных классов в компоненте, а 1 означает, что компонент не содержит ничего, кроме абстрактных классов.

Для компонента TripService была рассчитана метрика абстрактности A :

$$A = \frac{12}{20} = 0.6.$$

Вычислив меру устойчивости компоненты и ее абстрактность можно отобразить их на графике. На вертикальной оси необходимо отложить меру абстрактности A , а на горизонтальной оси – меру неустойчивости I . На практике часто случается, так что полученная точка лежит на диагонали между точкой $(1,0)$ и точкой $(0,1)$. Такая диагональ называется главная последовательность (рисунок 25).

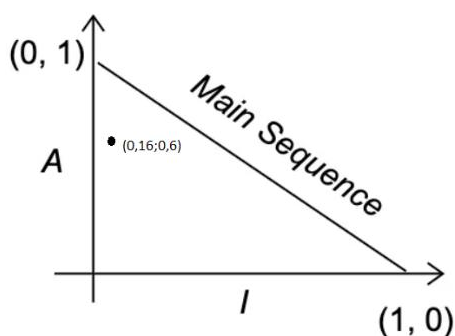


Рисунок 25 – Зоны исключения

Для вычисления количественной оценки используется формула вычисления расстояние до главной последовательности:

$$D = \left| \frac{A + I - 1}{\sqrt{2}} \right|.$$

Чем ближе точка располагается к главной последовательности, тем лучше.
Вычислено расстояние до главной последовательности:

$$D = \left| \frac{0.6 + 0.16 - 1}{\sqrt{2}} \right| = 0.17.$$

Расстояние отношений метрик первой компоненты до главной последовательности достаточно близко к нулю, а значит, ее отношение абстрактности к устойчивости является правильным.

4.4 Метрики кода

Был проведен анализ качества кода реализованной программы с помощью Visual Studio. Метрики кода приведены на рисунке 26.

	Индекс удобства поддержки	Сложность организации цик...	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
Booking.Common (Debug)	79	160	1	33	1 106	212
Booking.Common.DataAccess (Debug)	93	15	1	30	76	21
Booking.Common.Domain (Debug)	95	1 864	4	146	3 019	467
Booking.Common.Services (Debug)	93	176	2	63	565	104
Booking.Services.Aeroexpress.Domain (C	89	215	2	35	419	169
Booking.Services.Aeroexpress.Services (C	84	58	2	86	441	119
Booking.Services.Airline.DataAccess (Debug)	91	17	1	18	122	21
Booking.Services.Airline.Domain (Debug)	96	2 075	2	162	3 932	592
Booking.Services.Airline.Services (Debug)	89	418	2	221	2 439	598
Booking.Services.Custom.Domain (Debug)	87	44	2	11	81	22
Booking.Services.Custom.Services (Debug)	96	6	2	10	25	5
Services.FlightCertificate.Do	97	70	2	6	123	11
Services.FlightCertificate.Ser	92	21	2	29	110	23
Services.Hotel.DataAccess (Debug)	85	4	1	16	44	10
Services.Hotel.Domain (Debug)	97	848	2	62	1 511	262
Services.Hotel.Services (Debug)	82	303	2	137	2 151	523
Services.TaxiVoucher.Domain (D	96	178	2	27	266	32
Services.TaxiVoucher.Services (D	82	52	2	73	431	107
Services.Train.DataAccess (Debug)	88	21	1	32	180	39
Services.Train.Domain (Debug)	96	1 487	2	90	2 707	670
Services.Train.Services (Debug)	90	252	2	178	1 637	351
Services.Transfer.Domain (Debug)	97	489	3	52	1 007	100
Services.Transfer.Services (Debug)	86	102	2	92	761	151
Api (Debug)	75	33	3	76	292	136
DataAccess (Debug)	90	150	2	79	728	137
Domain (Debug)	96	1 210	1	89	2 507	330
Services (Debug)	86	2 307	2	508	10 251	3 279
Tests (Debug)	76	10	1	51	178	33

Рисунок 26 – Метрики кода

Индекс удобства поддержки показывает, насколько удобно и легко поддерживать данный код. Рассчитывается по формуле:

$$MI = MAX \left(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * \frac{100}{171} \right),$$

где HV – Halstead Volume, вычислительная сложность;

CC – Cyclomatic Complexity, структурная сложность кода;

LoC – количество строк исходного кода [19].

Нормой является значение от 20 до 100. В данном случае значения метрики для каждого модуля близки к 100, что означает, что код удобен в поддержке.

5 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project. Проект был начат 12 апреля 2023 года.

При планировании разработки было выделено 28 задач. Список задач и их описание представлено на рисунке 27.

	🔍	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1		Начало реализации проекта	0 дней	Ср 12.04.23	Ср 12.04.23		
2	☐	Техническое задание	10 дней	Ср 12.04.23	Вт 25.04.23		Интернет[200,00 Р]
3		Разработка технического задания	7 дней	Ср 12.04.23	Чт 20.04.23	1	Ведущий программист;Руководитель проекта
4		Согласование ТЗ с заказчиком	3 дней	Пт 21.04.23	Вт 25.04.23	3	Руководитель проекта
5		Завершение разработки ТЗ	0 дней	Вт 25.04.23	Вт 25.04.23	4	
6	☐	Проектирование	7 дней	Чт 27.04.23	Пт 05.05.23		Интернет[700,00 Р]
7	📄	Проектирование дизайна	1 день	Чт 27.04.23	Чт 27.04.23	5	Руководитель проекта;Frontend-разработчик
8		Согласование дизайна с заказчиком	1 день	Пт 28.04.23	Пт 28.04.23	7	Руководитель проекта
9	📄	Проектирование баз данных	1 день	Пт 28.04.23	Пт 28.04.23	7	Ведущий программист
10		Проектирование программного обеспечения	5 дней	Пн 01.05.23	Пт 05.05.23	5;9	Ведущий программист;Backend-разработчик
11		Завершение проектирования	0 дней	Пт 05.05.23	Пт 05.05.23	8;10	
12	☐	Реализация	18 дней	Пн 08.05.23	Ср 31.05.23		Интернет[1 500,00 Р]
13		Реализация базы данных	3 дней	Пн 08.05.23	Ср 10.05.23	11	Backend-разработчик
14		Реализация серверной части	10 дней	Чт 11.05.23	Ср 24.05.23	13	Ведущий программист;Backend-разработчик
15		Реализация клиентской части приложения	3 дней	Чт 25.05.23	Пн 29.05.23	14	Ведущий программист[50%];Frontend-разработчик
16		Внедрение клиентской и серверной части	1 день	Вт 30.05.23	Вт 30.05.23	15	Backend-разработчик;Frontend-разработчик
17		Разработка инфраструктуры	2 дней	Вт 30.05.23	Ср 31.05.23	14;15	Backend-разработчик[30%];Ведущий программист
18		Завершение реализации	0 дней	Ср 31.05.23	Ср 31.05.23	17	
19	☐	Тестирование	5 дней	Чт 01.06.23	Ср 07.06.23		Интернет[1 000,00 Р]
20		Модульное тестирование	1 день	Чт 01.06.23	Чт 01.06.23	18	Backend-разработчик;Frontend-разработчик
21		Функциональное тестирование	3 дней	Чт 01.06.23	Пн 05.06.23	18	Тестировщик
22		Исправление ошибок	1 день	Вт 06.06.23	Вт 06.06.23	20;21	Backend-разработчик;Frontend-разработчик
23		Регрессионное тестирование	1 день	Ср 07.06.23	Ср 07.06.23	22	Тестировщик
24		Завершение тестирования	0 дней	Ср 07.06.23	Ср 07.06.23	23	
25	☐	Внедрение	2 дней	Чт 08.06.23	Пт 09.06.23		Интернет[200,00 Р]
26		Запуск в эксплуатацию	2 дней	Чт 08.06.23	Пт 09.06.23	24	Ведущий программист;Тестировщик
27		Внедрение завершено	0 дней	Пт 09.06.23	Пт 09.06.23	26	
28		Завершение проекта	0 дней	Пт 09.06.23	Пт 09.06.23	27	

Рисунок 27 – Список задач

Среди ресурсов были выделены:

- ведущий программист;
- frontend-разработчик;
- backend-разработчик;
- руководитель проекта;
- тестировщик.

На рисунке 28 изображена диаграмма Ганта, показывающая зависимость и последовательность задач.

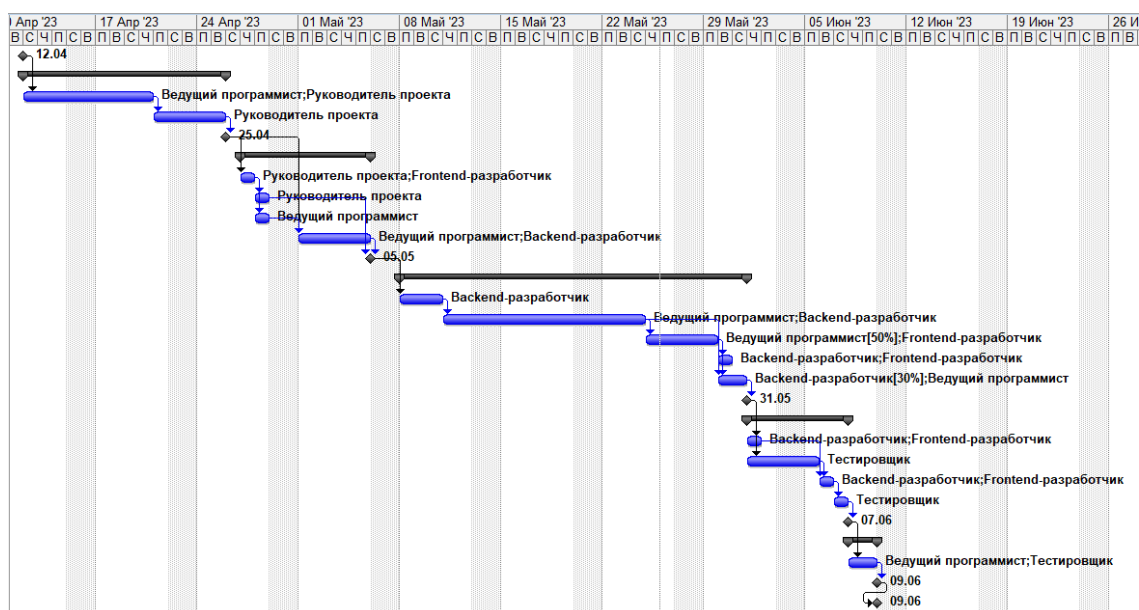


Рисунок 28 – Диаграмма Ганта

На рисунке 29 изображен отчет по проекту со всей необходимой информацией: бюджетом, сроками и затратами (финансовыми и трудовыми).

Даты			
Начало:	Ср 12.04.23	Окончание:	Пт 09.06.23
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	43 дней	Оставшаяся:	43 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	43 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	600,8 ч	Оставшиеся:	600,8 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	600,8 ч	Процент завершения:	0%
Затраты			
Текущий план:	261 400,00 Р	Оставшиеся:	261 400,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	261 400,00 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	28	Трудовые ресурсы:	5
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	1
Завершенные задачи:	0	Материальные ресурсы:	0
Итого задач:	28	Итого ресурсов:	6

Рисунок 29 – Отчет по проекту

Таким образом разработка проекта составит 43 полных рабочих дней с 12 апреля 2023 года по 9 июня 2023 года. Трудозатраты составят 600 человеко-часов. Оценка бюджета проекта составляет 261 400 рублей.

Заключение

В ходе выполнения выпускной квалификационной работы разработана информационная система бронирования деловых поездок.

Проведен сравнительный анализ сервисов бронирования деловых поездок и выделены основные особенности их реализации. На данной основе выделены функциональные и нефункциональные требования к системе.

Выбрана архитектура системы и основной шаблон проектирования, также спроектирована концептуальная и физическая схема БД, отображающие основные сущности и связи в выбранной предметной области. Для основных модулей построена диаграмма классов, а также диаграмма деятельности для основных бизнес-процессов.

Разработка программных средств проводилась в среде разработки JetBrains Rider на языке программирования C#. Разработанная программа функционирует на любом современном web-браузере, а разработанное API обладает быстрой переносимостью, благодаря использованию Docker-контейнеров.

Выполнено функциональное и модульное тестирование для разработанного программного обеспечения, все тесты пройдены успешно.

С помощью Microsoft Project было проведено планирование разработки и оценка бюджета. Разработка проекта составит 43 полных рабочих дней с 12 апреля 2023 года по 9 июня 2023 года. Трудозатраты составят 600 человеко-часов. Оценка бюджета проекта составляет 261 400 рублей.

Таким образом, в результате выполнения выпускной квалификационной работы разработан сервис для хранения, поиска и визуализации открытых данных, отвечающий всем требованиям технического задания.

Список использованных источников

1. Организация командировок [Электронный ресурс]. URL: <https://smartway.today/organizacia-komandirovok?ysclid=liag8o9l29693600940> (дата обращения: 19.04.2023).
2. Онлайн-система для организации и управления командировками Авиацентр [Электронный ресурс]. URL: <https://aviacenter.ru/corporate/online/> (дата обращения: 21.04.2023).
3. «Ostrovok.ru Командировки» [Электронный ресурс]. URL: <https://corp.ostrovok.ru> (дата обращения: 22.04.2023).
4. Сервис для автоматизации и управления деловыми поездками «Вкомандировке» [Электронный ресурс]. URL: <https://vkomandirovke.com/?ysclid=liagckwm5i97991511> (дата обращения: 22.04.2023).
5. «Ozon Travel для бизнеса» [Электронный ресурс]. URL: <https://www.ozon.ru/travel/corporate/> (дата обращения: 24.04.2023).
6. Реляционная база данных. Проектирование реляционных баз данных [Электронный ресурс]. URL: <https://otus.ru/nest/post/589/?ysclid=lik310uw5h79591120> (дата обращения: 24.04.2023).
7. Интеграция информационных систем предприятия [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/1055/271/lecture/6878?page=2&ysclid=lik3gtem5c442082823> (дата обращения: 25.04.2023).
8. ASP.NET и SignalR и C# [Электронный ресурс]. URL: <https://metanit.com/sharp/signalr/1.1.php?ysclid=lbxed8qufh595115211> (дата обращения: 10.05.2023).
9. Что такое MVC: базовые концепции [Электронный ресурс]. URL: https://skillbox.ru/media/code/chto_takoe_mvc_bazovye_kontseptsii_i_primer_prilozheniya/?ysclid=lik3l7397a77084909 (дата обращения: 12.05.2023).

10. Контейнеры как основа для совместной работы DevOps [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/containerized-lifecycle/docker-application-lifecycle/containers-foundation-for-devops-collaboration> (дата обращения: 13.05.2023).

11. AWS. Разбивка монолитного приложения на микросервисы [Электронный ресурс]. URL: <https://aws.amazon.com/ru/getting-started/hands-on/break-monolith-app-microservices-ecs-docker-ec2/module-one/> (дата обращения: 14.05.2023).

12. Простое руководство к диаграммам развертывания UML [Электронный ресурс]. URL: <https://creately.com/blog/ru/> (дата обращения: 15.05.2023).

13. Docker [Электронный ресурс]. URL: <https://www.docker.com/> (дата обращения: 15.05.2023).

14. Функциональное тестирование программного обеспечения [Электронный ресурс]. URL: <https://daglab.ru/funkcionalnoe-testirovanie-programmnogo-obespechenija/?ysclid=lik53mhvk3112279305> (дата обращения: 21.05.2023).

15. Понятие и классификация исключительных ситуаций [Электронный ресурс]. URL: https://lms2.sseu.ru/pluginfile.php/189317/mod_resource/content/1 (дата обращения: 21.05.2023).

16. Модульное тестирование, или юнит-тестирование [Электронный ресурс]. URL: <https://unetway.com/tutorial/modulnoe-testirovanie-ili-unit-testirovanie?ysclid=lik5tcvi4y396464525> (дата обращения: 23.05.2023).

17. Итерация 5. Создание модульных тестов (C#) [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/aspnet/mvc/overview/older-versions-1/contact-manager/iteration-5-create-unit-tests-cs> (дата обращения: 23.05.2023).

18. Метрика программного обеспечения [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Метрика_программного_обеспечения (дата обращения: 24.05.2023).

19. Средства оценки качества кода MS Visual Studio [Электронный ресурс]. URL: <https://habr.com/ru/post/111524/> (дата обращения: 19.05.2023).

Приложение А
ГЛОССАРИЙ
(справочное)

База данных (БД) – это информативная модель, позволяющая организованно поддерживать данные о группе объектов, имеющих один и тот же набор параметров.

Unified Modeling Language (UML) – графический унифицированный язык моделирования, использующий объектно-ориентированный подход для моделирования в сфере разработки программного обеспечения.

Микросервис – это веб-сервис, отвечающий за один элемент логики в определенной предметной области.

Docker – это проект с открытым исходным кодом для автоматизации развертывания приложений в виде переносимых автономных контейнеров, выполняемых в облаке или локальной среде.

Приложение Б
СТРУКТУРА КОЛЛЕКЦИЙ MONGODB
(рекомендуемое)

Листинг Б.1 – Пример записи из коллекции airlineSearch

```

{
  "Trips": [
    {
      "Routes": [
        {
          "Segments": [
            {
              "DepartureCountry": "Россия",
              "DepartureCountryCode": "RU",
              "DepartureCity": "Москва",
              "DepartureCityCode": "MOW",
              "ArrivalCountry": "Россия",
              "ArrivalCountryCode": "RU",
              ...
            }, ...
          ],
          ...
        },
        ...
      ],
      "Fares": [
        {
          "Name": "Эконом Лайт",
          "CarryonPlaces": "1 место не более 10 кг 55x40
x25см",
          "BaggagePlaces": "Платный",
          "AirlineBonusInformation": "Мили «Аэрофлот Бон
ус» – 75-125%",
          "Price": {
            "Base": "3150.00",
            "Tax": "547.00",
            ...
          }, ...
        },
        {
          "Name": "Эконом Оптимум",
          "CarryonPlaces": "1 место не более 10 кг 55x40
x25см",
          "BaggagePlaces": "1 место не более 23 кг",
          "AirlineBonusInformation": "100-150% миль",
          "Price": {
            "Base": "5150.00",
            "Tax": "547.00",
            ...
          }, ...
        },
        ...
      ],
      "ProviderName": "Sirena",
      ...
    },
    ...
  ],
  "SearchId": "b64b8fa0-7098-11ed-a9d7-024200",

```

```

"UserId": "41c739a1-ce4f-4fab-a9ca2d7a6",
"SearchOptions": {
  "Routes": [
    {
      "Date": "2022-11-30T00:00:00",
      "DepartureCode": "MOW",
      "DepartureName": "Москва (MOW)",
      "ArrivalCode": "LED",
      "ArrivalName": "Санкт-Петербург (LED)"
    }
  ],
  ...
}

```

Приложение В
ЛИСТИНГ ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг В.1 – Методы работы с корзиной

```

using System;
using System.Collections.Generic;
using System.Linq;
using DataAccess.Cart.Interfaces;
using DataAccess.Interfaces;
using DataAccess.SQL.Models;
using Logs.Common;
using Logs.Common.Enums;
using Logs.Common.Messages;

namespace DataAccess.Cart
{
    public class CartRepository : ICartRepository
    {
        private readonly IDataContext _dataContext;
        private readonly ICommonLog _commonLog;

        public CartRepository(IDataContext dataContext, ICommonLog
commonLog)
        {
            _dataContext = dataContext;
            _commonLog = commonLog;
        }

        public bool IsEmpty(string userId)
        {
            if (string.IsNullOrEmpty(userId))
            {
                _commonLog.Write(MessageLevel.Error, new
ArgumentNullMessage(nameof(userId)));
                return true;
            }

            return !_dataContext.GetQueryable<DbCartItem>().Any(i
=> i.UserId == userId);
        }

        public ICollection<DbCart> GetCarts(ICollection<int>
cartIds)
        {
            var carts =
_dataContext.GetQueryable<DbCart>().Where(a =>
cartIds.Contains(a.Id)).ToList();
            if (carts.Any())
            {
                return carts;
            }

            return null;
        }
    }
}

```

```

        public DbCartItem GetCartItem(string userId, int
cartItemId, bool trackChanges = false)
        {
            return

_dataContext.GetQueryable<DbCartItem>(trackChanges)
                .FirstOrDefault(i =>
                    (i.UserId == userId ||
i.Cart.Approve.Any(a => a.UserId == userId)) && i.Id ==
cartItemId);
        }

        public DbCartItem GetCartItemById(int cartItemId)
        {
            return
_dataContext.GetQueryable<DbCartItem>(false).FirstOrDefault(i =>
i.Id == cartItemId);
        }

        public DbCartItem GetCartItemByRequest(string userId, int
requestItemId)
        {
            return
_dataContext.GetQueryable<DbCartItem>().FirstOrDefault(x =>
                x.UserId == userId && x.RequestItemId.HasValue &&
x.RequestItemId.Value == requestItemId);
        }

        public IEnumerable<DbCart> GetCarts(string userId, int
type)
        {
            if (string.IsNullOrEmpty(userId))
            {
                return null;
            }

            return _dataContext.GetQueryable<DbCart>().Where(c =>
c.UserId == userId && c.Type == type);
        }

        public int Count(string userId, bool all)
        {
            return all
                ? _dataContext.GetQueryable<DbCartItem>().Count(i
=> i.UserId == userId && i.Cart.Type == 0)
                : _dataContext.GetQueryable<DbCartItem>().
                    .Count(i => i.UserId == userId && i.IsBooking
== false && i.Cart.Type == 0);
        }

        public void RemoveWithBooking(string userId,
IEnumerable<int> cartItemIds)

```

```

{
    if (cartItemIds == null)
    {
        return;
    }

    foreach (int id in cartItemIds)
    {
        DbCartItem item =
        _dataContext.GetQueryable<DbCartItem>()
            .FirstOrDefault(
                i =>
                    (i.UserId == userId ||
i.Cart.Approve.Any(a => a.UserId.Equals(userId)))
                    && i.Id == id);

        if (item != null)
        {
            _dataContext.Delete(item);
        }
    }

    _dataContext.Save();
}

public void Remove(string userId, int cartItemId)
{
    DbCartItem item = _dataContext
        .GetQueryable<DbCartItem>()
        .FirstOrDefault(i =>
            (i.UserId == userId || i.Cart.Approve.Any(a =>
a.UserId == userId)) &&
            i.Id == cartItemId &&
            i.IsBooking == false);

    if (item == null)
    {
        return;
    }

    _dataContext.Delete(item);

    DbCart cart = _dataContext.GetQueryable<DbCart>()
        .FirstOrDefault(c => c.UserId == userId && c.Id ==
item.CartId);

    if (cart != null && (cart.Items == null ||
!cart.Items.Any()))
    {
        cart.Name = string.Empty;
        _dataContext.Update(cart);
    }
}

```

```

        _dataContext.Save();
    }

    public DbCart AddCart(DbCart cart)
    {
        if (cart == null)
        {
            return null;
        }

        _dataContext.Insert(cart);
        _dataContext.Save();

        return cart;
    }

    public DbCartItem AddCartItem(DbCartItem cartItem)
    {
        if (cartItem == null)
        {
            return null;
        }

        _dataContext.Insert(cartItem);
        _dataContext.Save();

        return cartItem;
    }

    public DbCartItem UpdateCartItem(DbCartItem cartItem)
    {
        if (cartItem == null)
        {
            return null;
        }

        _dataContext.Update(cartItem);
        _dataContext.Save();

        return cartItem;
    }

    public void Clear(string userId, int cartId, int cartType)
    {
        IEnumerable<DbCartItem> carItems = _dataContext
            .GetQueryable<DbCartItem>()
            .Where(i => i.UserId == userId &&
                i.IsBooking == false &&
                i.Cart.Type == cartType &&
                i.CartId == cartId);
    }

```

```

        foreach (var item in carItems)
        {
            _dataContext.Delete(item);
        }

        SetCartName(userId, cartId, string.Empty);
        _dataContext.Save();
    }

    public void SetCartName(string userId, int cartId, string
newName)
    {
        DbCart cart = _dataContext.GetQueryable<DbCart>()
            .FirstOrDefault(c =>
                (c.UserId.Equals(userId) || c.Approve.Any(a =>
a.UserId.Equals(userId))) && c.Id == cartId);

        if (cart == null)
        {
            cart = new DbCart {UserId = userId, Name =
newName};

            _dataContext.Insert(cart);
        }
        else
        {
            cart.Name = newName;

            _dataContext.Update(cart);
        }

        _dataContext.Save();
    }

    public void RemoveCartItemEmployee(int cartItemId, int
employeeId)
    {
        DbCartItemEmployee cartItemEmployee =
_dataContext.GetQueryable<DbCartItemEmployee>()
            .FirstOrDefault(e => e.CartItemId == cartItemId &&
e.EmployeeId == employeeId);

        if (cartItemEmployee == null)
        {
            return;
        }

        _dataContext.Delete(cartItemEmployee);
        _dataContext.Save();
    }

    public void SetCartItemEmployee(DbCartItemEmployee
cartItemEmployee)

```

```

        {
            if (cartItemEmployee == null)
            {
                _commonLog.Write(MessageLevel.Error, new
ArgumentNullException(nameof(cartItemEmployee)));
                return;
            }

            DbCartItemEmployee currentCartItemEmployee =
_dataContext.GetQueryable<DbCartItemEmployee>()
                .FirstOrDefault(e =>
                    e.CartItemId == cartItemEmployee.CartItemId &&
e.EmployeeId == cartItemEmployee.EmployeeId);

            if (currentCartItemEmployee == null)
            {
                _dataContext.Insert(cartItemEmployee);
            }
            else
            {
                currentCartItemEmployee.DocuemntId =
cartItemEmployee.DocuemntId;

                _dataContext.Update(currentCartItemEmployee);
            }

            _dataContext.Save();
        }

        public IEnumerable<DbCartItemEmployee>
GetEmployeeCartItem(int employeeId)
        {
            return
_dataContext.GetQueryable<DbCartItemEmployee>().Where(e =>
e.EmployeeId == employeeId);
        }

        public IEnumerable<DbCartItemEmployee>
GetEmployeeCartItem(int cartItemId, string userId)
        {
            return _dataContext.GetQueryable<DbCartItemEmployee>()
                .Where(a => a.CartItemId == cartItemId &&
a.CartItem.UserId.Equals(userId));
        }

        public void ClearCartItemEmployeeDocuments(int companyId,
string userId, int employeeId)
        {
            var currentEmployeeCartItems =
_dataContext.GetQueryable<DbCartItemEmployee>()
                .Where(e => e.EmployeeId == employeeId);

```

```

        foreach (DbCartItemEmployee cartItemEmployee in
currentEmployeeCartItems)
        {
            cartItemEmployee.DocuemntId = null;
            _dataContext.Update(cartItemEmployee);
        }

        _dataContext.Save();
    }

    public void SetCartItemBookingStatus(string userId,
IEnumerable<int> cartIds, bool status)
    {
        IEnumerable<DbCartItem> cartItems =
            _dataContext.GetQueryable<DbCartItem>()
                .Where(
                    i =>
                        (i.UserId == userId ||
i.Cart.Approve.Any(a => a.UserId.Equals(userId)))
                        && cartIds.Contains(i.CartId));

        foreach (var item in cartItems)
        {
            item.IsBooking = status;
            _dataContext.Update(item);
        }

        _dataContext.Save();
    }

    public void SetCartItemEmployeeCompany(int companyId, int
employeeId, int cartItemId)
    {
        var cartItemEmployee =
_dataContext.GetQueryable<DbCartItemEmployee>()
            .FirstOrDefault(a => a.EmployeeId == employeeId &&
a.CartItemId == cartItemId);
        if (cartItemEmployee == null)
        {
            _commonLog.Write(MessageLevel.Debug, new
ArgumentNullMessage(nameof(cartItemEmployee)));
            return;
        }

        if (companyId == 0)
        {
            cartItemEmployee.CompanyId = null;
        }
        else
        {
            cartItemEmployee.CompanyId = companyId;
        }
    }

```

```

        _dataContext.Save();
    }

    public void SetCartItemEmployeeDepartment(int cartItemId,
int employeeId, int? departmentId)
    {
        var cartItemEmployee =
_dataContext.GetQueryable<DbCartItemEmployee>()
                .FirstOrDefault(i => i.EmployeeId == employeeId &&
i.CartItemId == cartItemId);
        if (cartItemEmployee == null)
        {
            _commonLog.Write(MessageLevel.Debug, new
ArgumentNullMessage(nameof(cartItemEmployee)));
            return;
        }

        cartItemEmployee.DepartmentId = departmentId == 0 ?
null : departmentId;
        _dataContext.Save();
    }

    public void SetProjectInCartItem(int accountId, int
itemId, int employeeId, int projectId)
    {
        var cartItemEmployee =
_dataContext.GetQueryable<DbCartItemEmployee>()
                .FirstOrDefault(a => a.EmployeeId == employeeId &&
a.CartItemId == itemId);

        cartItemEmployee.ProjectId = projectId;
        _dataContext.Save();
    }

    public void RemoveProjectInCartItem(int accountId, int
itemId, int employeeId)
    {
        var cartItemEmployee =
_dataContext.GetQueryable<DbCartItemEmployee>()
                .FirstOrDefault(a =>
                    a.EmployeeId == employeeId && a.CartItemId ==
itemId && a.Project.AccountId == accountId);
        if (cartItemEmployee == null)
        {
            _commonLog.Write(MessageLevel.Error, new
ArgumentNullMessage(nameof(cartItemEmployee)));
            return;
        }

        cartItemEmployee.ProjectId = null;
        _dataContext.Save();
    }

```

```

        public int CreateCart(string userId, int cartType)
        {
            var newCart = new DbCart {UserId = userId, Type =
cartType, Name = string.Empty};
            _dataContext.Insert(newCart);
            _dataContext.Save();
            return newCart.Id;
        }

        public DbCart GetCart(string userId, int cartType, int
cartId)
        {
            return _dataContext.GetQueryable<DbCart>(false)
                .FirstOrDefault(
                    a => (a.UserId.Equals(userId) ||
a.Approve.Any(c => c.UserId.Equals(userId))
                        && a.Type == cartType
                        && a.Id == cartId);
        }

        public void DeleteCart(string userId, int cartType, int
cartId)
        {
            try
            {
                var dbCart = _dataContext.GetQueryable<DbCart>()
                    .FirstOrDefault(a =>
                        (a.UserId.Equals(userId) ||
a.Approve.Any(b => b.UserId.Equals(userId)) ||
a.ApproversCustom.Any(ac => ac.UserId.Equals(userId))
                            && a.Type == cartType
                            && a.Id == cartId);

                if (dbCart == null)
                {
                    return;
                }

                if (dbCart.Approve.Any())
                {
                    var approveIds = dbCart.Approve.Select(a =>
a.Id).ToList();
                    foreach (var approveId in approveIds)
                    {
                        var dbApprove =
_dataContext.GetQueryable<DbApprove>().FirstOrDefault(a => a.Id ==
approveId);
                        if (dbApprove != null)
                        {
                            _dataContext.Delete(dbApprove);
                        }
                    }
                }
            }
        }

```

```

        if (dbCart.ApproversCustom.Any())
        {
            var customApproverIds =
dbCart.ApproversCustom.Select(a => a.Id).ToList();
            foreach (var id in customApproverIds)
            {
                var dbCustomApprover =
_dataContext.GetQueryable<DbStepApproverCustoms>()
                    .FirstOrDefault(a => a.Id == id);
                if (dbCustomApprover != null)
                {
                    _dataContext.Delete(dbCustomApprover);
                }
            }

            _dataContext.Delete(dbCart);
            _dataContext.Save();
        }
        catch (Exception e)
        {
            _commonLog.Write(e);
        }
    }

    public void SetReasonMessage(string userId, int
cartItemId, string message)
    {
        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
            .FirstOrDefault(a => a.UserId.Equals(userId) &&
a.Id == cartItemId);
        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.Reason = message;
        _dataContext.Save();
    }

    public void SetTravelPolicy(string userId, int cartItemId,
string travelPolicy)
    {
        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
            .FirstOrDefault(a => a.UserId.Equals(userId) &&
a.Id == cartItemId);
        if (dbCartItem == null)
        {
            return;
        }
    }

```

```

        dbCartItem.TravelPolicy = travelPolicy;

        _dataContext.Save();
    }

    public void CreateCart(string userId)
    {
        DbCart currentCart =
        _dataContext.GetQueryable<DbCart>(false).FirstOrDefault(a =>
        a.UserId == userId);
        if (currentCart != null)
        {
            return;
        }

        DbCart newCart = new DbCart {Name = string.Empty, Type
= 0, UserId = userId};
        _dataContext.Insert(newCart);
        _dataContext.Save();
    }

    public ICollection<DbContract3D> GetContracts3D(int
companyId)
    {
        return
        _dataContext.GetQueryable<DbContract3D>().Where(a => a.CompanyId
== companyId).ToList();
    }

    public void SaveBookData(string userId, int cartItemId,
string bookData, DateTimeOffset? deadline)
    {
        DbCartItem dbCartItem =
        _dataContext.GetQueryable<DbCartItem>()
            .FirstOrDefault(a =>
                a.Id == cartItemId && (a.UserId == userId ||
a.Cart.Approve.Any(b => b.UserId == userId)));

        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.BookData = bookData;
        dbCartItem.BookDeadline = deadline;
        dbCartItem.IsReserved = true;
        _dataContext.Save();
    }

    public void UpdatePriceOrigin(string userId, int
cartItemId)
    {

```

```

        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
                .FirstOrDefault(a =>
                    a.Id == cartItemId && (a.UserId == userId ||
a.Cart.Approve.Any(b => b.UserId == userId)));

        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.PriceOrigin = dbCartItem.Price;
        _dataContext.Save();
    }

    public void UpdatePrice(string userId, int cartItemId,
decimal newPrice)
    {
        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
                .FirstOrDefault(a =>
                    a.Id == cartItemId && (a.UserId == userId ||
a.Cart.Approve.Any(b => b.UserId == userId)));

        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.Price = newPrice;
        _dataContext.Save();
    }

    public void UpdatePrice(string userId, int cartItemId)
    {
        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
                .FirstOrDefault(a =>
                    a.Id == cartItemId && (a.UserId == userId ||
a.Cart.Approve.Any(b => b.UserId == userId)));

        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.Price = dbCartItem.PriceOrigin;
        _dataContext.Save();
    }

    public void ClearBookData(string userId, int cartItemId)
    {

```

```

        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
                .FirstOrDefault(a =>
                    a.Id == cartItemId && (a.UserId == userId ||
a.Cart.Approve.Any(b => b.UserId == userId)));

        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.BookData = null;
        dbCartItem.BookDeadline = null;
        dbCartItem.IsReserved = false;
        _dataContext.Save();
    }

    public void RemoveApprove(string userId, int cartId)
    {
        var approveIds =
_dataContext.GetQueryable<DbApprove>()
                .Where(a => a.CartId.HasValue && a.CartId.Value ==
cartId).Select(b => b.Id).ToList();
        foreach (var approveId in approveIds)
        {
            var dbApprove =
_dataContext.GetQueryable<DbApprove>().FirstOrDefault(a => a.Id ==
approveId);
            if (dbApprove != null)
            {
                _dataContext.Delete(dbApprove);
            }
        }

        _dataContext.Save();
    }

    public void SetErrorMessage(string userId, int cartItemId,
string errorMessage)
    {
        DbCartItem dbCartItem =
_dataContext.GetQueryable<DbCartItem>()
                .FirstOrDefault(a =>
                    a.Id == cartItemId && (a.UserId == userId ||
a.Cart.Approve.Any(b => b.UserId == userId)));

        if (dbCartItem == null)
        {
            return;
        }

        dbCartItem.Error = errorMessage;
        _dataContext.Save();
    }

```

```

    }

    public void SetVerificationCode(string userId, int cartId,
string code)
    {
        var dbCart = _dataContext.GetQueryable<DbCart>()
            .FirstOrDefault(a =>
                (a.UserId == userId || a.Approve.Any(b =>
b.UserId == userId)) && a.Id == cartId);
        if (dbCart == null)
        {
            return;
        }

        dbCart.VerificationCode = code;
        _dataContext.Update(dbCart);
        _dataContext.Save();
    }

    public bool HasReservedItems(int employeeId)
    {
        IQueryable<DbCartItem> result =
_dataContext.GetQueryable<DbCartItem>()
            .Where(a => (a.IsReserved || a.IsBooking) &&
a.EmployeeCartItems.Any(b => b.EmployeeId == employeeId));
        return result.Any();
    }

    public void RemoveVerificationCode(string userId, int
cartItemId)
    {
        var dbCart = _dataContext.GetQueryable<DbCart>()
            .FirstOrDefault(a => a.UserId == userId &&
a.Items.Any(b => b.Id == cartItemId));

        if (dbCart == null)
        {
            return;
        }

        dbCart.VerificationCode = null;
        _dataContext.Update(dbCart);
        _dataContext.Save();
    }

    public DateTimeOffset? GetMaxDeadline(int employeeId)
    {
        var cartItems =
_dataContext.GetQueryable<DbCartItem>()
            .Where(a => a.EmployeeCartItems.Any(b =>
b.EmployeeId == employeeId))
            .ToList();
    }

```

```

        return cartItems.Any(a => a.BookDeadline.HasValue)
            ? cartItems.Select(a => a.BookDeadline).Max()
            : null;
    }

    public void ClearBookData(int employeeId)
    {
        var cartItems =
            _dataContext.GetQueryable<DbCartItem>()
                .Where(a => a.EmployeeCartItems.Any(b =>
                    b.EmployeeId == employeeId));

        foreach (var item in cartItems)
        {
            item.IsReserved = false;
            item.IsBooking = false;
            item.BookDeadline = null;
            item.BookData = null;
        }

        _dataContext.Save();
    }
}

```

Приложение Г
МОДУЛЬНЫЕ ТЕСТЫ
(рекомендуемое)

Листинг Г.1 – Класс модульных тестов валидации паспорта

```

public class ValidatorServiceTests
{
    private IValidatorService validatorService;

    [SetUp]
    public void Setup()
    {
        validatorService = new ValidatorService();
    }

    [Test]
    public void ValidateRuPassport_WhenNumberHasManyNumbers()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", Number =
"324834829104"};
        var isValid = validatorService.ValidateRuPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateRuPassport_WhenNumberHasNoDigit()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", Number =
"324834104f"};
        var isValid = validatorService.ValidateRuPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateRuPassport_WhenIsNotRussian()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "KZ", Number =
"3248341041"};
        var isValid = validatorService.ValidateRuPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateRuPassport_WhenIsValid()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", Number =
"3248341041"};
        var isValid = validatorService.ValidateRuPassport(employee,
validEmployee);
        Assert.True(isValid);
    }

    [Test]
    public void ValidateRuPassport_WhenIsEmpty()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", Number =
string.Empty};
    }
}

```

```

        var isValid = validatorService.ValidateRuPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateForeignPassport_WhenIsEmpty()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", ZNumber =
string.Empty, DueDate = null};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateForeignPassport_WhenIncorrectCountOfNumbers()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", ZNumber =
"234458233482", DueDate = DateTime.Now};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateForeignPassport_WhenZNumberHasNoDigit()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", ZNumber =
"23334825g", DueDate = DateTime.Now};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void
ValidateForeignPassport_WhenIsNotRussianAndIncorrectZNumber()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "KZ", ZNumber =
string.Empty, DueDate = DateTime.Now};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateForeignPassport_WhenIncorrectDueDate()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "KZ", ZNumber =
"574839405", DueDate = new DateTime()};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.False(isValid);

        employee = new PreviewEmployee();
        validEmployee = new ValidImportEmployee {Code = "KZ", ZNumber =
"574839405", DueDate = null};

```

```

        isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.False(isValid);
    }

    [Test]
    public void ValidateForeignPassport_WhenIsValidIsNotRussian()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "KZ", ZNumber =
"567443238340", DueDate = DateTime.Now.AddMonths(1)};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.True(isValid);
    }

    [Test]
    public void ValidateForeignPassport_WhenIsValidIsRussian()
    {
        var employee = new PreviewEmployee();
        var validEmployee = new ValidImportEmployee {Code = "RU", ZNumber =
"567438340", DueDate = DateTime.Now.AddMonths(1)};
        var isValid = validatorService.ValidateForeignPassport(employee,
validEmployee);
        Assert.True(isValid);
    }
}

```

Приложение Д
СПРАВКА О ВНЕДРЕНИИ
(рекомендуемое)

УТВЕРЖДАЮ

руководитель отдела разработки

ООО «Смартвэй»

Мурзаев А. А.

23 мая 2023 года

СПРАВКА

о результатах внедрения решений,
разработанных в выпускной квалификационной работе

Настоящая справка подтверждает то, что результаты выпускной квалификационной работы студента ФГБОУ ВО «Пензенский государственный университет» факультета ФВТ группы 19ВП2 очной формы обучения Буйнякова Г. Э. на тему «Программные средства для бронирования деловых поездок» внедрены в онлайн-сервис организации командировок «Смартвэй».

Руководитель отдела разработки

ООО «Смартвэй»

Мурзаев А. А.



Мур