

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

Курсовой проект

По дисциплине «Проектирование и конструирование программного
обеспечения»

на тему «Конструирование ПС учета и анализа клиентов информационной
системы туристической фирмы»

ПГУ 09.03.04 – 07КП201.14 ПЗ

Направление подготовки - 09.03.04 Программная инженерия

Выполнил студент: Медведь Левин М.В.

Группа: 20ВП1

Руководитель:

к.т.н., доцент Иванчук Иванчуков А.Г.

Работа защищена с оценкой хорошо

Преподаватели

Иванчук
Иванчуков А.Г.

Дата защиты

16.04.2024 г.

2024.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Пензенский государственный университет»
(ФГБОУ ВО «Пензенский государственный университет»)

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

Зав. кафедрой

Козлов А.Ю.

"__" _____ 2023 г.

ЗАДАНИЕ
на курсовую работу по дисциплине
«Проектирование и конструирование программного обеспечения»

Студенту Левину Максиму Вячеславовичу **Группа** 20ВП1.

Тема проекта: Конструирование ПС учета и анализа клиентов информационной системы туристической фирмы

Исходные данные (технические требования) на проектирование

1. Разработать микросервис, обеспечивающий взаимодействие с клиентами по технологии REST API.
2. Разработать БД, которая отвечает за хранение модели данных микросервиса.
3. Разработать клиентское приложение с графическим интерфейсом пользователя, которое осуществляет взаимодействие с микросервисом.
4. Обязательные требования к программе: многомодульность, использование сложных типов данных, использование коллекции для организации базы данных. Старт приложения должен сопровождаться всплывающим окном с информацией об авторе и темой проекта.
5. Среда разработки ПО: Microsoft Visual Studio 2022
6. Язык программирования: C#
7. Программное обеспечение должно быть полностью отлажено и протестировано, и должно функционировать под управлением ОС Windows 10 и выше.

Объём работ по курсу

1. Расчётная часть

- 1) Анализ требований.
- 2) Выбор и освоение инструментальных средств анализа и проектирования
- 3) Определение структуры и функций приложения
- 4) Разработка диаграмм описания приложения
- 5) Реализация приложения на языке C# в среде Microsoft Visual Studio
- 6) Отладка и тестирование приложения

2. Графическая часть

- 1) Схема базы данных
- 2) Диаграмма вариантов использования
- 3) Диаграмма классов
- 4) Диаграмма компонентов
- 5) Диаграмма последовательности

3. Экспериментальная часть

- 1) Отладка компонентов приложения и их взаимодействия
- 2) Функциональное тестирование приложения

Срок выполнения проекта по разделам

1	Анализ требований	к	22 сентября	2023 г.
2	Определение структуры и функций приложения	к	1 октября	2023 г.
3	Разработка UML диаграмм приложения	к	15 октября	2023 г.
4	Реализация приложения	к	12 ноября	2023 г.
5	Отладка и тестирование приложения	к	29 ноября	2023 г.
6	Оформление пояснительной записки проекта	к	10 декабря	2023 г.
7	Защита курсового проекта	к	15 декабря	2023 г.

Дата выдачи задания « 13 » сентября 2023 г.

Руководитель  /к.т.н. А.Г. Иванчуков/

Задание получил « 13 » сентября 2023 г.

Студент  /М.В. Левин/

Реферат

Пояснительная записка содержит 59 страниц, 32 рисунка, 2 таблицы, 5 использованных источников и 3 приложения.

МИКРОСЕРВИСЫ, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ,
ТУРИСТИЧЕСКОЕ АГЕНТСТВО, REST API, СУБД, MS SQL, DOTNET, C#,
ASP.NET, WINDOWS FORMS.

Объект разработки – клиент-серверное приложение для учета и анализа клиентов турагентства, реализованное современными средствами backend-разработки.

Цель разработки: изучить предметную область и провести её анализ, выделить бизнес-процессы, а также реализовать и протестировать клиентскую и серверную часть приложения.

Результаты разработки: проанализирована предметная область туристического агентства, выделены основные бизнес-процессы, разработана база данных, клиентское и серверное приложения для учета и анализа клиентов.

Разработка проводилась на языке программирования C# 10 в среде разработки Microsoft Visual Studio 2022. Серверная часть реализована с использованием технологии ASP.NET WebAPI, клиентская часть с использованием - Windows Forms. В качестве СУБД была выбрана MSSQL.

					ПГУ 09.03.04 – 07КП201.14 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Левин М.В.	<i>Левин</i>		Конструирование ПС учета и анализа клиентов информационной системы туристической фирмы Пояснительная записка	Лит.	Лист	Листов
Провер.		Иванчуков А.Г.	<i>Иванчуков</i>	16.04.24			3	59
Реценз.						Группа 20ВП1		
Н. Контр.								
Утверд.								

Оглавление

ВВЕДЕНИЕ	6
1. Постановка задачи и анализ требований	7
1.1. Основные понятия и определения	7
1.2. Постановка задачи	7
1.3. Формулировка и анализ требований	8
2. Проектирование	12
2.1. Структура программного обеспечения	12
2.2. Проектирование структур данных	13
2.3. Проектирование алгоритмов	14
2.4. Проектирование пользовательского интерфейса	15
3. Кодирование	21
3.1. Кодирование серверной части приложения	21
3.2. Кодирование клиентской части приложения	24
4. Отладка и тестирование	26
ЗАКЛЮЧЕНИЕ	28
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	29
ПРИЛОЖЕНИЕ А	30
ПРИЛОЖЕНИЕ Б	43
ПРИЛОЖЕНИЕ В	53

ВВЕДЕНИЕ

На данный момент существует множество подходов к разработке приложений. Одним из подходов является разбиение приложения на микросервисы. В данной курсовом проекте будет реализовано приложение для учета и анализа клиентов информационной системы туристической фирмы.

Помимо серверной части приложения также будет разработано приложение, работающее на персональном компьютере, в качестве клиентской части.

В ходе реализации курсового проекта будут проанализированы требования, спроектирована схема базы данных, построены такие диаграммы как диаграмма вариантов использования, диаграмма компонентов, диаграмма последовательности и диаграмма классов. Будут реализованы клиентская и серверная часть приложения, а также проведено функциональное тестирование.

Разработка приложения будет осуществляться на языке программирования C# 10 в среде разработки Microsoft Visual Studio 2022. Серверная часть будет разрабатываться при помощи технологии ASP.NET WEB API, а клиентская часть при помощи – Windows Forms.

1. Постановка задачи и анализ требований

1.1. Основные понятия и определения

JSON Web Token (JWT) - это JSON объект, структура которого определена в открытом стандарте RFC 7519. Обычно он используется для передачи информации об аутентифицированных участниках, а также он может быть дополнительно зашифрован [1].

Микросервис - независимо развертываемый сервис, который взаимодействует с другими микросервисами через один или несколько протоколов связи [1].

Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, получивший распространение в середине 2010-х годов в связи с развитием практик гибкой разработки и DevOps [2].

ORM - технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» [3].

REST - архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Другими словами, REST — это набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать [4].

1.2. Постановка задачи

Требуется разработать программные средства учета и анализа клиентов информационной системы туристической фирмы.

В процессе разработки серверной части необходимо выделить минимум 3 программных модуля, отвечающие за определенный функционал

приложения, и реализовать их в виде отдельных микросервисов. Также необходимо предусмотреть, чтобы было межсервисное взаимодействие.

В процессе разработки клиентской части необходимо создать приложение с графическим интерфейсом пользователя, которое будет осуществлять взаимодействия с серверной частью приложения при помощи технологии REST API.

1.3. Формулировка и анализ требований

К разрабатываемым программным средствам были выделены следующие функциональные требования:

- авторизация клиентов и турагентов;
- регистрация клиентов;
- получение списка клиентов;
- создание и хранение информации о турах;
- обновление информации о туре;
- получение списка туров;
- создание и хранение заявок на бронирование тура;
- получение списка заявок;
- изменение статуса заявки;
- получение статистики по клиентам;
- получение статистики по странам.

В соответствии с выявленными требованиями была разработана диаграмма вариантов использования UML, представленная на рисунке 1

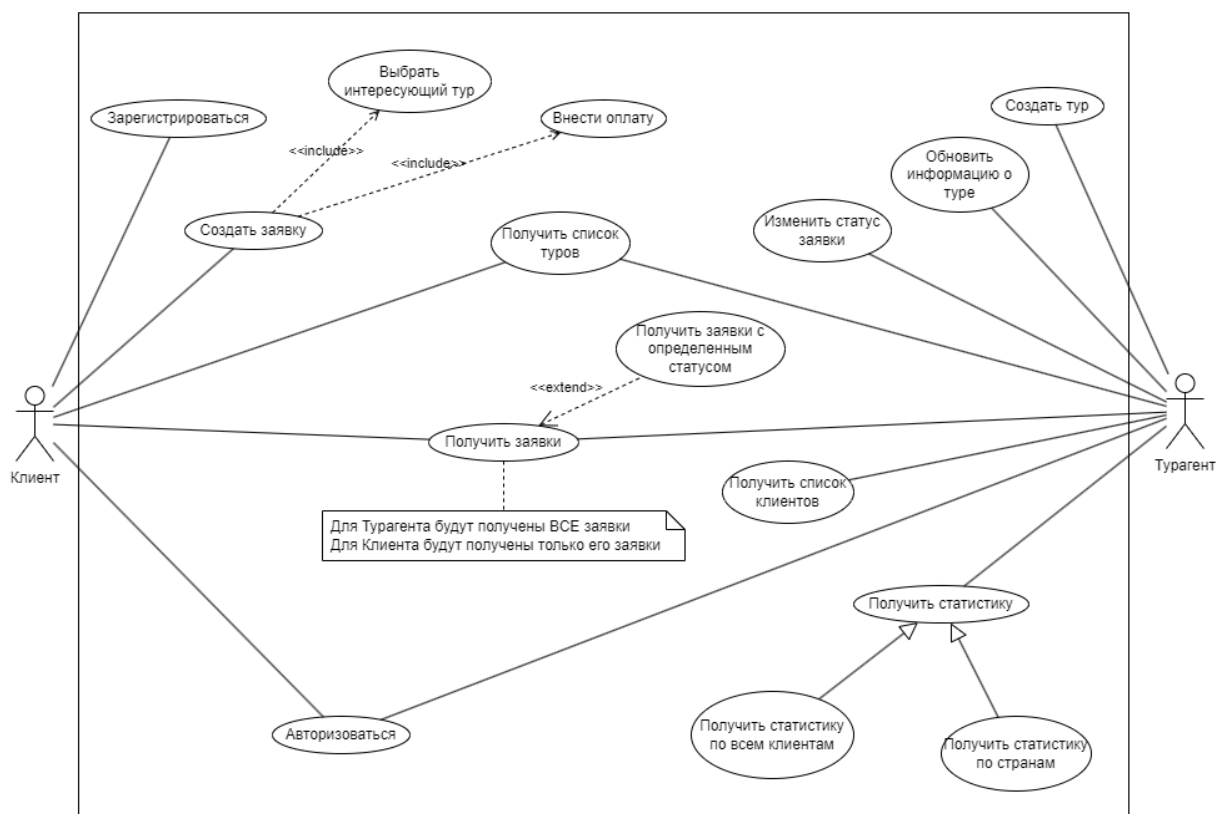


Рисунок 1 – Диаграмма вариантов использования

Сценарий варианта использования «Авторизоваться» представлен на рисунке 2.

Авторизоваться
Краткое описание: система проверяет существование логина и пароля в базе данных и генерирует JWT
Предусловие: Пользователь ввёл данные
Основной поток: <ol style="list-style-type: none"> 1. система проверяет существование логина; 2. система проверяет совпадение пароля; 3. система генерирует и отправляет JWT
Постусловие: Пользователь авторизован

Рисунок 2 - Сценарий варианта использования «Авторизоваться»

Сценарий варианта использования «Получить список туров» представлен на рисунке 3.

Получить список туров
Краткое описание: серверная часть приложения отправляет список туров
Предусловие: Пользователь находится на вкладке с турами
Основной поток: <ol style="list-style-type: none"> 1. пользователь запрашивает список туров; 2. микросервис туров получает все записи из базы данных; 3. микросервис возвращает список туров, конвертированных в JSON объекты
Постусловие: Список туров получен и отображен

Рисунок 3 - Сценарий варианта использования «Получить список туров»

Сценарий варианта использования «Создать заявку» представлен на рисунке 4.

Создать заявку
Краткое описание: Клиент создает заявку на бронирование тура
Предусловие: Пользователь должен быть авторизован и иметь роль «клиент»
Основной поток: <ol style="list-style-type: none"> 1. клиент выбирает интересующий его тур; 2. вносит оплату за тур; 3. отправляет заявку
Постусловие: Заявка на бронирование тура появилась в базе данных со статусом «НЕОБРАБОТАНА»

Рисунок 4 - Сценарий варианта использования «Создать заявку»

Сценарий варианта использования «Получение статистики по клиентам» представлен на рисунке 5.

Получение статистики по клиентам
Краткое описание: формирование статистики по клиентам: количество купленных туров и общая потраченная сумма
Предусловие: Пользователь должен быть авторизован и иметь роль «турагент»
Основной поток: <ol style="list-style-type: none"> 1. микросервис аналитики обращается к микросервису с турами и запрашивает список туров через REST API; 2. микросервис аналитики группирует записи по клиентам и считает для каждого из них количество купленных туров и общую потраченную сумму
Постусловие: Турагент получил статистику по всем клиентам

Рисунок 5 - Сценарий варианта использования «Получение статистики по клиентам»

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования были представлены их спецификации, которые позволяют отобразить последовательность действий пользователя и системы.

2. Проектирование

2.1. Структура программного обеспечения

Разрабатываемое приложение будет состоять из серверной и клиентской частей.

Клиентская часть будет представлять собой приложение, работающее на персональном компьютере на операционной системе Windows 7 и выше.

Серверная часть будет содержать 4 программных модуля:

- микросервис авторизации, работающий с информацией о пользователях;
- микросервис туров и заявок на бронирование туров;
- микросервис аналитики, работающий с информацией об оставленных заявках. Будет получать данные от микросервиса туров;
- API шлюз, выступающий в роли маршрутизатора API-запросов к выше перечисленным микросервисам.

После авторизации с помощью первого микросервиса пользователь получает JWT, который хранит идентификатор пользователя, его роль и токен доступа, позволяющий получить доступ к остальным микросервисам. На рисунке 6 представлена диаграмма компонентов системы.

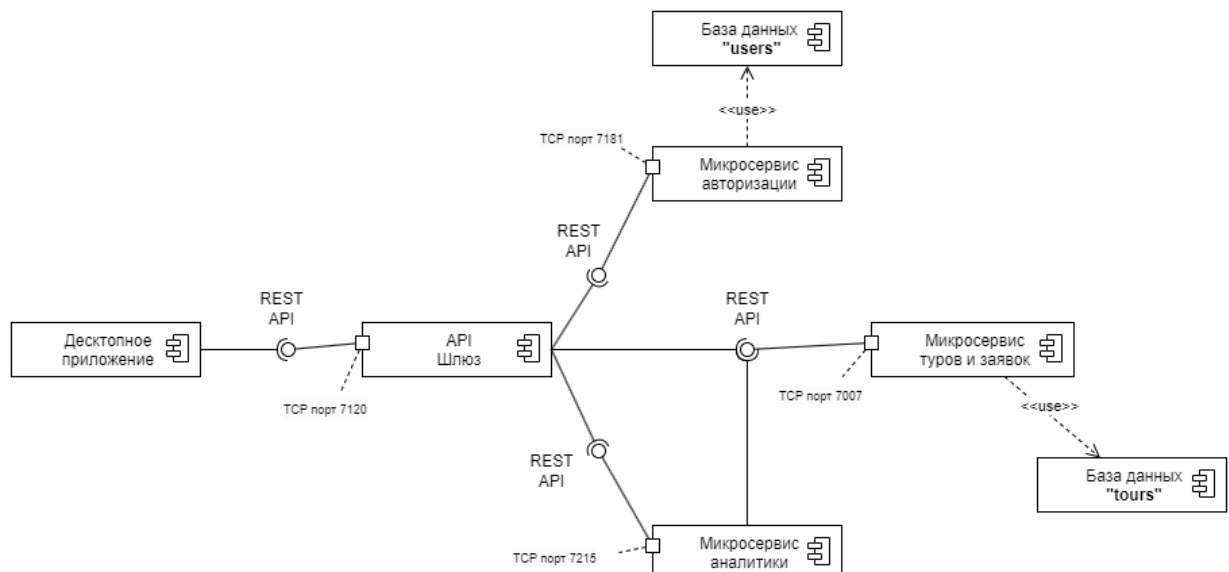


Рисунок 6 – Диаграмма компонентов

Клиентское приложение для получения и обновления данных будет посылать HTTP запросы на API шлюз, который в свою очередь будет пересылать их в соответствующие микросервисы при помощи технологии REST API.

2.2. Проектирование структур данных

Для хранения данных о пользователях в микросервисе авторизации будет использоваться база данных, состоящая из 3 таблиц: «Account», «Client», «TourAgent». Схема базы данных представлена на рисунке 7.

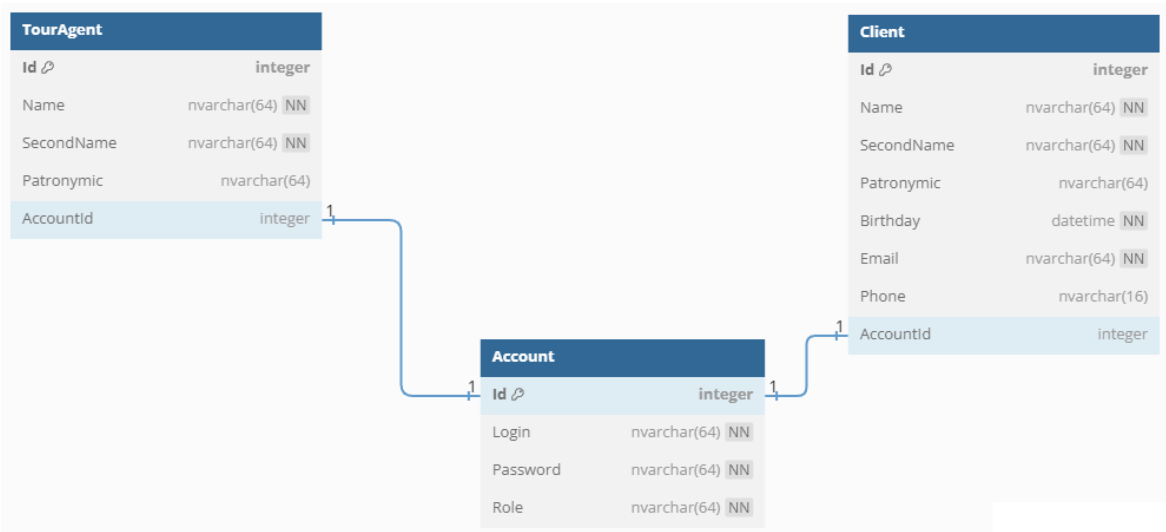


Рисунок 7 – Схема базы данных для микросервиса авторизации

Для хранения данных о турах и заявках на бронирование этих туров в соответствующем микросервисе будет использоваться база данных, состоящая из 2 таблиц: «Tour», «Request». Схема базы данных представлена на рисунке 8.

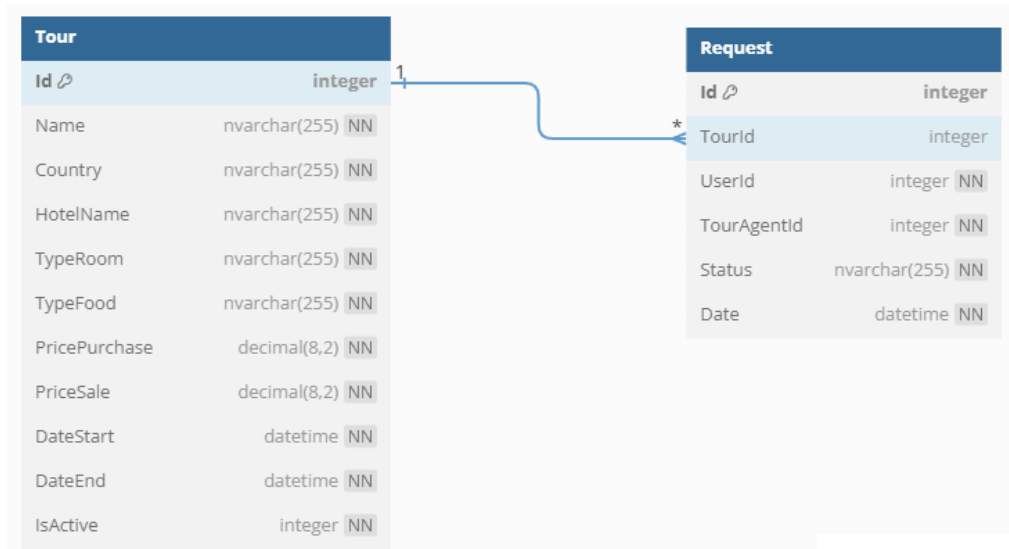


Рисунок 8 – Схема базы данных для микросервиса туров и заявок

Каждый микросервис будет принимать и отдавать данные в формате JSON.

2.3. Проектирование алгоритмов

В разрабатываемом приложении будет выделено 2 роли: клиент и турагент.

Клиент может зарегистрироваться и авторизоваться в системе, просмотреть список туров и оформить заявку на бронирование интересующего тура.

Турагент может авторизоваться в системе, добавлять и обновлять информацию о турах, получать список всех заявок от клиентов, а также просматривать статистику по клиентам.

В процессе формирования статистики по клиентам, происходит межсервисная коммуникация, для более подробного описания этого процесса была создана диаграмма последовательности, представленная на рисунке 9.

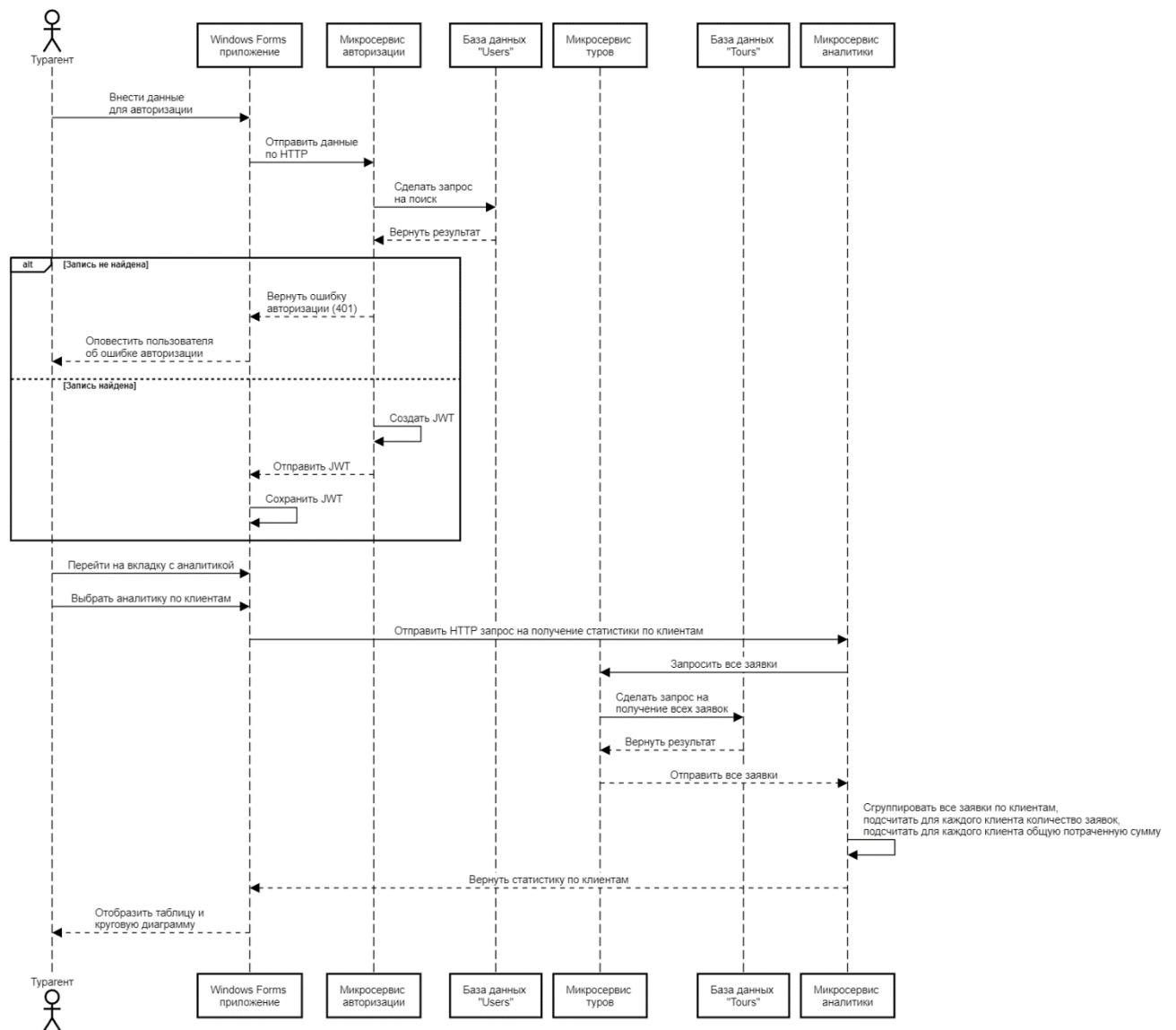


Рисунок 9 – Диаграмма последовательности

В начале турагенту необходимо авторизоваться в системе, после чего выбрать вкладку со статистикой. Далее происходит обращение к микросервису аналитики, который запрашивает все проданные туры с микросервиса туров. В конце происходит обработка полученного списка и возвращение ответа турагенту.

2.4. Проектирование пользовательского интерфейса

Пользовательский интерфейс приложения будет содержать следующие формы:

- форма входа, содержащая две вкладки: авторизация, регистрация;

- основная форма, содержащая четыре вкладки: клиенты, туры, заявки и статистика;
- форма добавления/обновления тура.

Вкладка авторизации на первой форме (рис. 10) содержит следующие элементы:

- 1 – поле для ввода логина пользователя;
- 2 – поле для ввода пароля пользователя;
- 3 – кнопка «Войти».

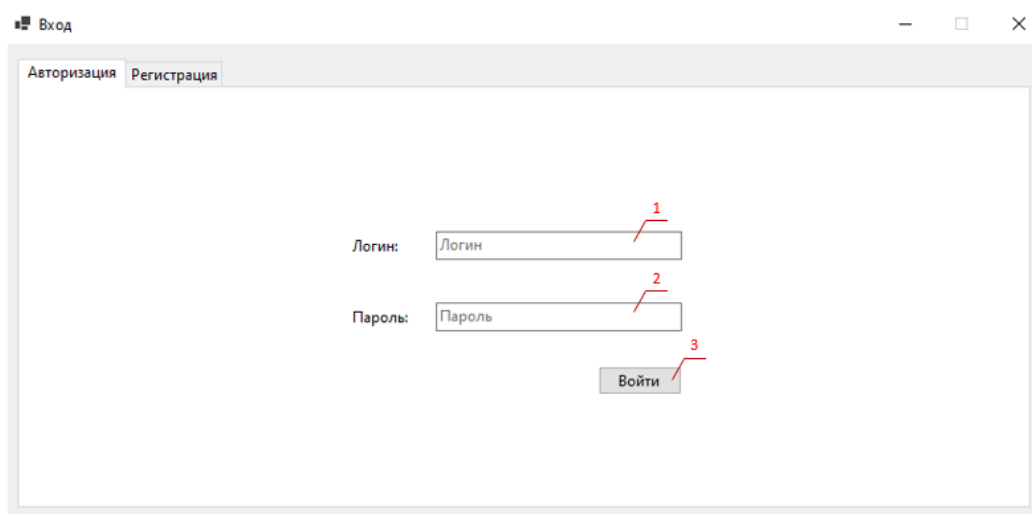
The image shows a screenshot of a web application window titled 'Вход' (Login). Inside the window, there are two tabs: 'Авторизация' (Authorization) and 'Регистрация' (Registration). The 'Авторизация' tab is active. It contains two input fields: 'Логин' (Login) and 'Пароль' (Password). Below these fields is a button labeled 'Войти' (Login). Red numbers 1, 2, and 3 are placed next to the login field, password field, and login button respectively, indicating the elements described in the text.

Рисунок 10 – Вкладка авторизации

Вкладка регистрации на первой форме (рис. 11) содержит следующие элементы:

- 1 – поле для ввода фамилии пользователя;
- 2 – поле для ввода имени пользователя;
- 3 – поле для ввода отчества пользователя;
- 4 – поле для ввода даты рождения пользователя;
- 5 – поле для ввода электронной почты пользователя;
- 6 – поле для ввода номера телефона пользователя;
- 7 – поле для ввода логина пользователя;
- 8 – поле для ввода пароля пользователя;
- 9 – поле для повторного ввода пароля пользователя;
- 10 – кнопка «Зарегистрироваться».

Вход

Авторизация Регистрация

Фамилия:

Имя:

Отчество:

Дата рождения:

Почта:

Номер телефона:

Логин:

Пароль:

Повторите пароль:

Рисунок 11 – Вкладка регистрации

Вкладка с клиентами на основной форме (рис. 12) содержит следующие элементы:

- 1 – поле, отображающее роль пользователя;
- 2 – поле, отображающее ФИО пользователя;
- 3 – таблица, содержащая всех зарегистрированных пользователей;
- 4 – кнопка «Загрузить», которая запрашивает список клиентов.

Учет и анализ клиентов в турагентстве

Роль:

Имя:

Клиенты Туры Заявки Статистика

ID	Фамилия	Имя	Отчество	Дата рождения	Почта	Номер телефона
1	левин	максим	Вячеславович	09.08.2002 0:00:00	testd3@mail.ru	+12345678910
2	Петров	Виктор	Иванович	18.01.2000 0:00:00	victor@mail.ru	+98765432120
10001	Сидоров	Василий	Петрович	14.04.1988 17:59:54	some123@mail.ru	12345678910

Рисунок 12 – Вкладка с клиентами

Вкладка с турами на основной форме (рис. 13) содержит следующие элементы:

- 1 – таблица, содержащая все туры;
- 2 – кнопка «Добавить», которая открывает форму для добавления нового тура;
- 3 – кнопка «Загрузить», которая запрашивает список туров.

Учет и анализ клиентов в турагентстве

Роль: touragent Имя: Иванов И. И.

Клиенты Туры Заявки Статистика

Добавить Загрузить

ID	Название	Страна	Отель	Тип комнаты	Тип питания	Цена покупки	Цена продажи	Начало	Окончание	Активно
6	Спа туры	Франция	Мини отель ...	Дополнител...	полупансион	180000,00	210000,00	05.11.2024 0:0...	20.11.2024 0:0...	1
7	Культурный ...	Турция	Отель Звездн...	одиночный	полный панс...	160000,00	190000,00	01.12.2024 0:0...	15.12.2024 0:0...	1
8	Экстрим туры	Франция	Гостиница П...	двойной	все включено	200000,00	240000,00	10.01.2025 0:0...	25.01.2025 0:0...	0
9	Лыжные туры	Турция	Отель Солне...	двойной	полупансион	230000,00	270000,00	05.02.2025 0:0...	20.02.2025 0:0...	1
10	Круизы	Кипр	Гостиница Ве...	тройной	все включено	280000,00	320000,00	01.03.2025 0:0...	15.03.2025 0:0...	1
11	Отдых на море	Испания	Отель Алые ...	четырехмест...	полный панс...	150000,00	200000,00	15.07.2024 0:0...	30.07.2024 0:0...	0
12	Экскурсионн...	Франция	Гостиница Ц...	Дополнител...	все включено	180000,00	220000,00	10.07.2024 0:0...	25.07.2024 0:0...	0
13	Спортивный ...	Испания	Отель Золот...	одиночный	завтраки	120000,00	150000,00	05.08.2024 0:0...	20.08.2024 0:0...	1
14	Спа туры	Франция	Мини отель ...	тройной	все включено	180000,00	210000,00	05.11.2024 0:0...	20.11.2024 0:0...	1
15	Культурный ...	Турция	Отель Звездн...	четырехмест...	полупансион	160000,00	190000,00	01.12.2024 0:0...	15.12.2024 0:0...	1
16	Экстрим туры	Франция	Гостиница П...	Дополнител...	завтраки	200000,00	240000,00	10.01.2025 0:0...	25.01.2025 0:0...	1
17	Лыжные туры	Турция	Отель Солне...	одиночный	все включено	230000,00	270000,00	05.02.2025 0:0...	20.02.2025 0:0...	1
18	Круизы	Кипр	Гостиница Ве...	двойной	полный панс...	180000,00	220000,00	01.03.2025 0:0...	15.03.2025 0:0...	1
19	Отдых на море	Испания	Отель Алые ...	двойной	все включено	150000,00	200000,00	15.07.2024 0:0...	30.07.2024 0:0...	1
20	Экскурсионн...	Франция	Гостиница Ц...	тройной	завтраки	180000,00	220000,00	10.07.2024 0:0...	25.07.2024 0:0...	0
21	Спортивный ...	Испания	Отель Золот...	четырехмест...	полупансион	120000,00	150000,00	05.08.2024 0:0...	20.08.2024 0:0...	0

Рисунок 13 – Вкладка с турами

Форма добавления/обновления тура (рис. 14) содержит следующие элементы:

- 1 – поле для ввода названия тура;
- 2 – поле для ввода названия страны;
- 3 – поле для ввода названия отеля;
- 4 – поле для ввода типа комнаты;
- 5 – поле для ввода типа питания;
- 6 – поле для ввода цены покупки у туроператора;
- 7 – поле для ввода цены продажи клиентам;
- 8 – поле для ввода даты начала тура;
- 9 – поле для ввода даты окончания тура;
- 10 – поле для указания видимости тура для клиентов;
- 11 – кнопка «Отменить»;
- 12 – кнопка «ОК», которая подтверждает создание нового тура.

Добавление/Обновление тура

1

Название

2

Страна

3

Отель

4

Тип комнаты

5

Тип питания

6

Цена покупки

7

Цена продажи

8

Дата начала

9

Дата окончания

10

☒ Тур активен

11

Отменить

12

OK

Рисунок 14 – Форма добавления/обновления тура

Вкладка с заявками на тур на основной форме (рис. 15) содержит следующие элементы:

- 1 – таблица, содержащая все заявки;
- 2 – поле для выбора статуса заявки;
- 3 – кнопка «Загрузить», которая запрашивает список заявок.

Учет и анализ клиентов в турагентстве

Роль: touragent Имя: Иванов И. И.

Клиенты Туры Заявки Статистика

Статус: 2 3 Загрузить

ID	ID Клиента	ID Тура	ID Турагента	Статус	Дата
1	1	29	1	ОПЛАЧЕНА	05.02.2024 0:00:00
2	1	1	1	ОТМЕНЕНА	05.03.2024 0:00:00
3	1	2	1	ОПЛАЧЕНА	05.05.2024 0:00:00
4	2	20	1	ОПЛАЧЕНА	15.03.2024 0:00:00
5	2	12	1	ОПЛАЧЕНА	05.04.2024 0:00:00
6	10001	11	1	ОПЛАЧЕНА	01.05.2024 0:00:00
7	10001	19	1	ОТМЕНЕНА	01.04.2024 0:00:00
8	1	1	1	ОПЛАЧЕНА	22.03.2024 19:54:17
9	1	21	1	ОПЛАЧЕНА	03.04.2024 15:58:15
10007	10001	3	1	ОПЛАЧЕНА	04.04.2024 15:53:01
10008	10001	4	1	ОПЛАЧЕНА	04.04.2024 15:54:22
10009	10001	8	1	ОПЛАЧЕНА	04.04.2024 17:35:43

Рисунок 15 – Вкладка с заявками на тур

Вкладка со статистикой на основной форме (рис. 16) содержит следующие элементы:

- 1 – таблица, содержащая статистику;
- 2 – кнопка «Статистика по странам», которая запрашивает статистику по всем странам;
- 3 – кнопка «Статистика по клиентам», которая запрашивает статистику по всем клиентам;
- 4 – Круговая диаграмма

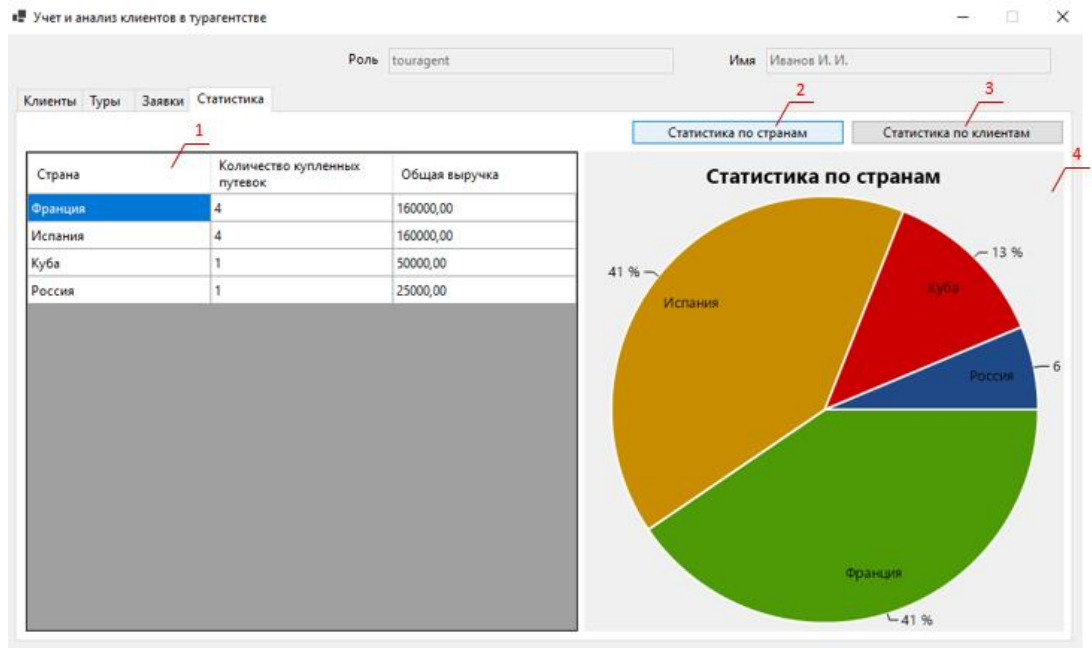


Рисунок 16 – Вкладка со статистикой

Вначале пользователь попадает на форму входа в систему. После авторизации открывается основная форма. При закрытии этой формы снова отобразиться форма входа.

3. Кодирование

3.1. Кодирование серверной части приложения

Реализация серверной части приложения осуществлялась на языке программирования C# при помощи технологии ASP.NET WEB API. В качестве базы данных использовалась MSSQL, а для работы с ней из программного кода использовалась ORM - Entity Framework.

Для каждого микросервиса были созданы специальные классы, называемые котроллерами, предназначенные преимущественно для обработки запросов протокола HTTP.

На рисунках 17 - 19 представлены диаграммы классов для микросервисов авторизации, туров/заявок и аналитики соответственно.

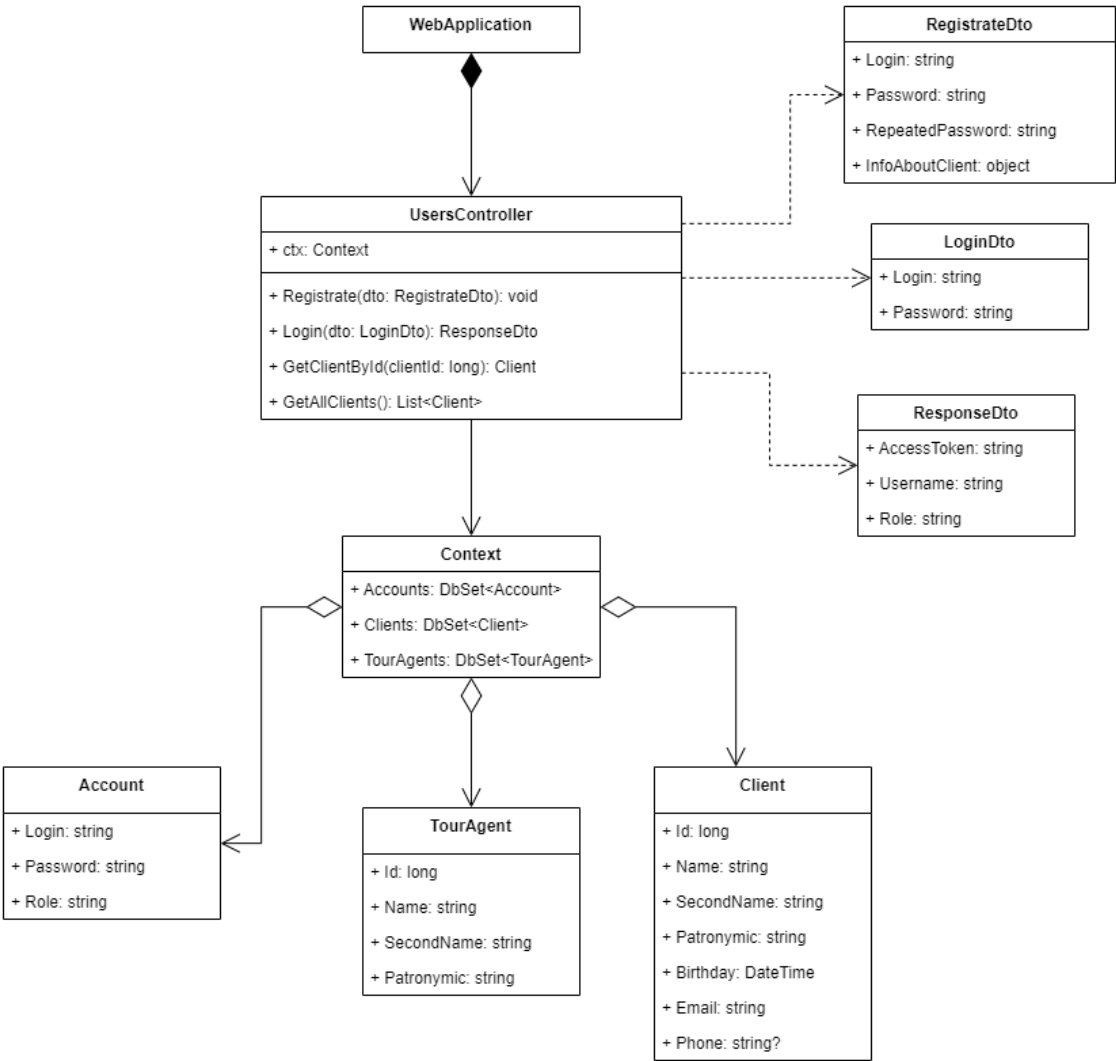


Рисунок 17 – Диаграмма классов для микросервиса авторизации

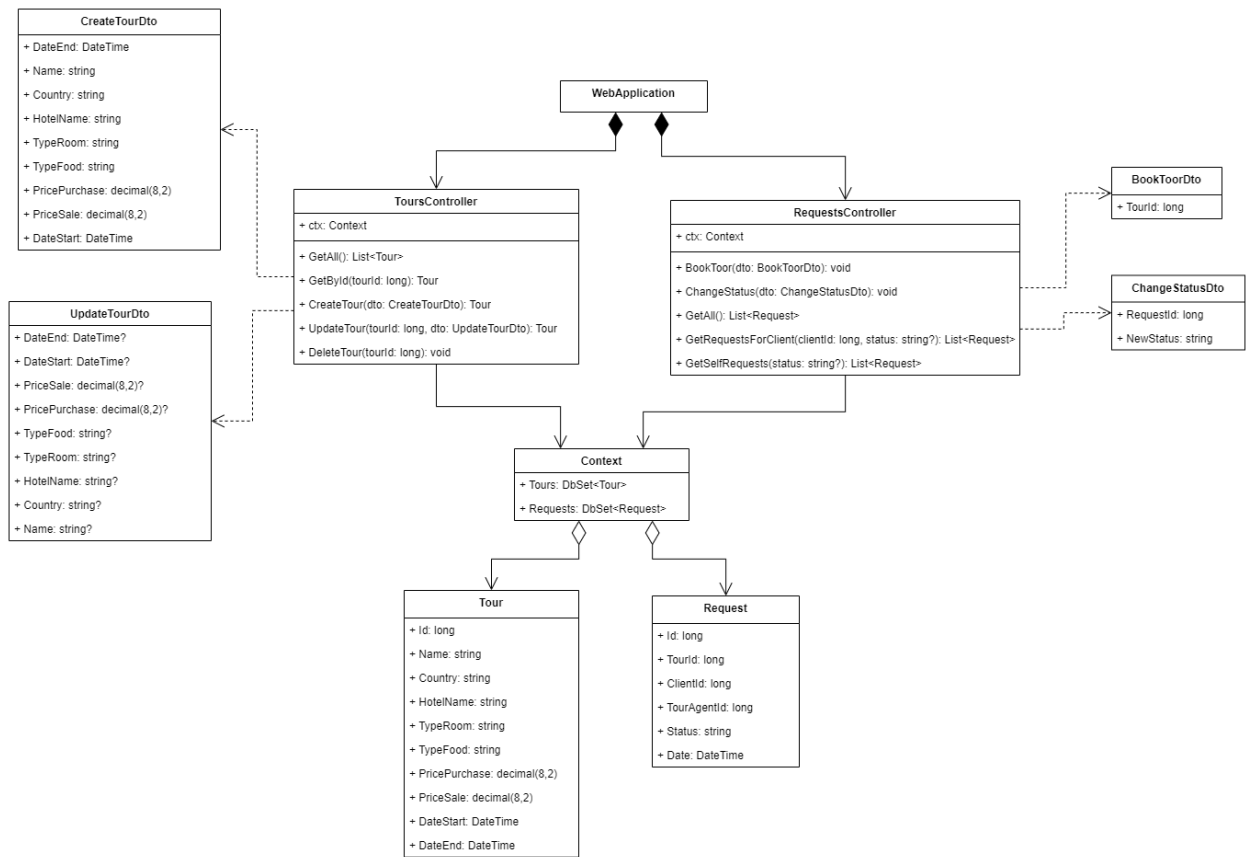


Рисунок 18 – Диаграмма классов для микросервиса туров и заявок

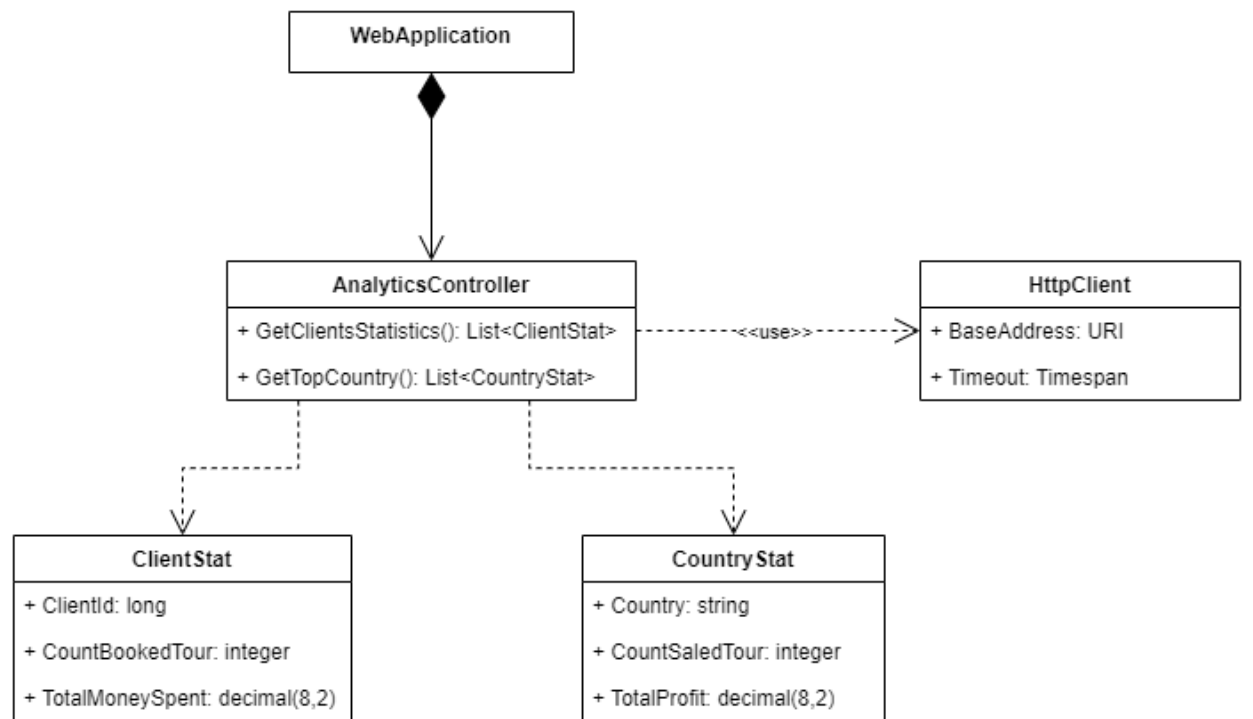


Рисунок 19 – Диаграмма классов для микросервиса аналитики

Также было создано специальное веб-приложение: API Gateway (API шлюз). Оно выполняет функции маршрутизатора API-запросов, которые получает на вход от клиента, обрабатывает и рассылает их разным сервисам, а после отправляет ответ обратно клиенту.

В таблице 1 описаны все конечные точки серверной части приложения, к которым можно отправлять HTTP запросы для выполнения определенных операций. Каждая конечная точка представляет собой конкретный URI (Uniform Resource Identifier), который соответствует определенному ресурсу или функциональности системы.

Таблица 1 – конечные точки серверной части приложения

URI запроса	HTTP метод	Описание	Параметры запроса	Требуемая Роль
api/auth/registration	POST	регистрация клиента	нет	неважно
api/auth/login	POST	авторизация клиента	нет	неважно
api/clients	GET	получить всех клиентов	нет	турагент
api/clients/{client_id}	GET	получить клиента по идентификатору	client_id – идентификатор клиента	турагент
api/clients/me	GET	получить информацию о себе (клиент)	нет	клиент
api/touragents/me	GET	получить информацию о себе (турагент)	нет	турагент
api/tours	GET	получить все туры	нет	неважно
api/tours	POST	добавить новый тур	нет	турагент
api/tours/{tour_id}	GET	получить тур по идентификатору	tour_id – идентификатор тура	неважно
api/tours/{tour_id}	UPDATE	обновить данные о туре	tour_id – идентификатор тура	турагент
api/tours/{tour_id}	DELETE	удалить тур	tour_id – идентификатор тура	турагент
api/requests/me?status={status}	GET	получить все свои заявки	status – статус заявки	клиент

api/requests?status={ status }	GET	получить все заявки	status – статус заявки	турагент
--------------------------------	-----	---------------------	------------------------	----------

Продолжение таблицы 1

api/requests/{ client_id }?status={ status }	GET	получить все заявки клиента по идентификатору	client_id - идентификатор клиента; status - статус заявки	турагент
api/requests	POST	создать заявку на тур	нет	клиент
api/requests/change_status	POST	изменить статус заявки	нет	турагент
api/analytics/client_stats	GET	Получить статистику по количеству купленных туров и общей потраченной сумме для каждого клиента	нет	турагент
api/analytics/top_country	GET	Получить статистику по количеству купленных туров и общей потраченной сумме в каждой из стран	нет	турагент

Код серверной части приложения представлен в приложении А.

3.2. Кодирование клиентской части приложения

Реализация клиентской части приложения осуществлялась на языке программирования С# при помощи технологии Windows Forms.

На рисунке 20 представлена диаграмма классов для приложения с графическим интерфейсом. На ней отображены используемые формы и графические элементы, которые представляют собой интерфейс пользователя.

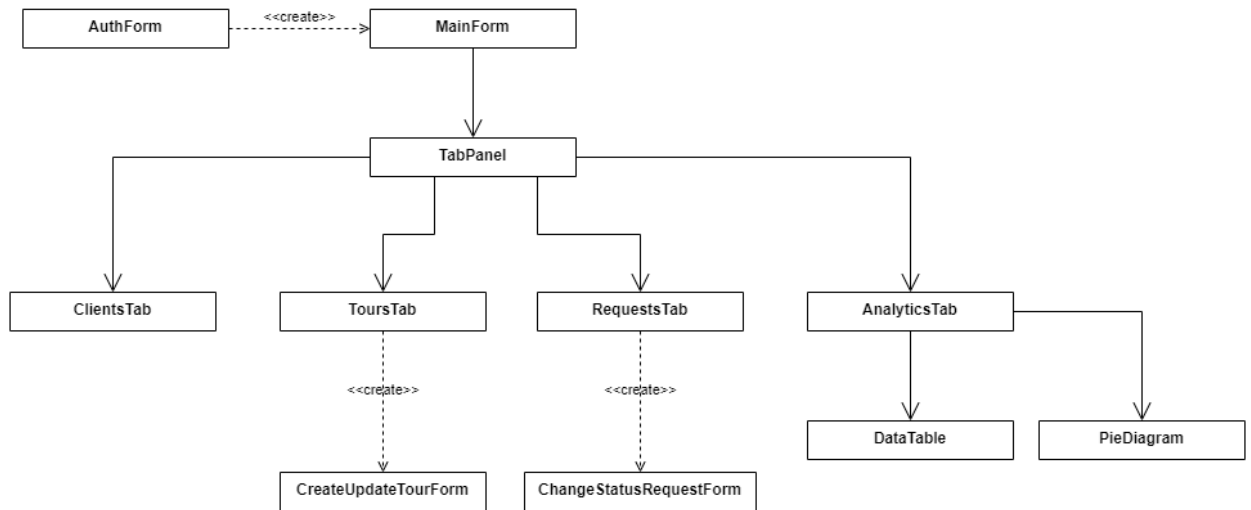


Рисунок 20 – Диаграмма классов для клиентского приложения

Код клиентской части приложения представлен в приложении Б.

Код всего приложения доступен по ссылке
<https://github.com/martin1917/TravelAgency>

4. Отладка и тестирование

Функциональное тестирование - процесс обеспечения качества (QA) в рамках цикла разработки программного обеспечения, необходимый для проверки реализуемости функциональных требований, согласно спецификации тестируемого программного обеспечения. Оно проводится для оценки соответствия системы или компонента заданным функциональным требованиям. Функциональное тестирование проводится по принципу черного ящика, в связи с чем функциональность ПО можно протестировать, не зная принципа его внутренней работы. Это снижает требования к тестировщикам в части знания языков программирования или конкретных аспектов реализации программного обеспечения [5].

При решении поставленной задачи будет проведено функциональное тестирование, так как оно наиболее наглядно отображает корректность выполнения функций приложения, соответствующих функциональным требованиям. Результаты тестирования представлены в таблице 2.

Таблица 2 - Результаты тестирования

Состав теста	Ожидаемый результат	Наблюдаемый результат	Результат теста
Регистрация нового клиента	Должна добавиться запись о новом клиенте в БД	Сервер успешно выполнил регистрацию	Тест пройден (рис. В.1)
Регистрация клиента с указанием занятого логина	Сервер должен вернуть ошибку, запись о новом клиенте не должна добавляться в БД.	Сервер вернул 400 ошибку (BadRequest)	Тест пройден (рис. В.2)
Авторизация клиента	Сервер должен вернуть JWT	Сервер авторизовал клиента и вернул JWT	Тест пройден (рис. В.3)
Клиент запрашивает список всех зарегистрированных клиентов	Сервер должен вернуть ошибку, так как у пользователя роль клиента	Сервер вернул ошибку доступа 403 (Forbidden)	Тест пройден (рис. В.4)

Продолжение таблицы 2

Клиент запрашивает информацию о себе	Сервер должен вернуть информацию	Сервер вернул JSON объект, содержащий нужную информацию	Тест пройден (рис. В.5)
Запросить список заявок, будучи неавторизованным в системе	Должна произойти ошибка авторизации	Сервер вернул ошибку авторизации 401 (Unauthorized)	Тест пройден (рис. В.6)
Турагент запрашивает список заявок	Сервер должен вернуть список всех заявок	Сервер вернул список всех заявок	Тест пройден (рис. В.7)
Турагент запрашивает статистику по клиентам	Сервер должен вернуть статистику по каждому клиенту	Сервер вернул статистику по каждому клиенту	Тест пройден (рис. В.8)
Турагент делает запрос на бронирование тура	Сервер должен вернуть ошибку доступа, так как бронировать туры могут только клиенты	Сервер вернул ошибку доступа 403 (Forbidden)	Тест пройден (рис. В.9)
Клиент делает запрос на бронирование тура	Сервер должен забронировать тур для клиента	Сервер забронировал тур для клиента	Тест пройден (рис. В.10)
Турагент запрашивает все новые заявки для указанного клиента	Сервер должен вернуть все новые заявки для клиента	Сервер вернул все новые заявки для клиента	Тест пройден (рис. В.11)
Турагент берет заявку в обработку	Сервер должен изменить статус у указанной заявки	Сервер изменил статус у указанной заявки	Тест пройден (рис. В.12)

В ходе выполнения тестирования несовпадения ожидаемого и наблюдаемого результата не выявлены. Следовательно, можно сделать вывод, что приложение работает корректно

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта было разработано приложение для учета и анализа клиентов информационной системы туристической фирмы с использованием микросервисной архитектуры.

Были сформулированы и проанализированы требования к разрабатываемому приложению. Создана диаграмма вариантов использования.

На этапе проектирования была создана схема базы данных, построены диаграммы компонентов и последовательности.

На этапе кодирования была выполнена программная реализация серверной и клиентской частей приложения, а также созданы диаграммы классов для основных компонентов системы.

Произведено функциональное тестирование разработанного программного обеспечения. Все тесты пройдены успешно, ошибок не выявлено.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Создание микросервисов. 2-е изд. — СПб.: Питер, 2023. — 624 с.: ил. — (Серия «Бестселлеры O'Reilly»). ISBN 978-5-4461-1145-9.
2. Мартин Фаулер. Микросервисы [Электронный ресурс]. URL: <https://martinfowler.com/articles/microservices.html> (дата обращения 10.04.2024)
3. ORM [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/ORM> (дата обращения: 10.04.2024)
4. REST [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/REST> (дата обращения 10.04.2024)
5. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М.: «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series).

ПРИЛОЖЕНИЕ А

(Исходный код серверной части приложения)

Файл Users/Models/Account.cs

```
namespace Users.Models
{
    public class Account
    {
        public long Id { get; set; }

        public string Login { get; set; }

        public string Password { get; set; }

        public string Role { get; set; }

        public Client? Client { get; set; }
        public TourAgent? TourAgent { get; set; }
    }
}
```

Файл Users/Models/Client.cs

```
namespace Users.Models
{
    public class Client
    {
        public long Id { get; set; }

        public string Name { get; set; }

        public string SecondName { get; set; }

        public string? Patronymic { get; set; }

        public DateTime Birthday { get; set; }

        public string Email { get; set; }

        public string? PhoneNumber { get; set; }

        public long AccountId { get; set; }
        public Account? Account { get; set; }
    }
}
```

Файл Users/Models/TourAgent.cs

```
namespace Users.Models
{
    public class TourAgent
    {
        public long Id { get; set; }

        public string Name { get; set; }

        public string SecondName { get; set; }

        public string? Patronymic { get; set; }

        public long AccountId { get; set; }
        public Account? Account { get; set; }
    }
}
```

Файл Users/Data/Context.cs

```
using Microsoft.EntityFrameworkCore;
using Users.Models;
```

```

namespace Users.Data
{
    public class Context : DbContext
    {
        public Context(DbContextOptions<Context> options) : base(options) { }

        public DbSet<Account> Accounts { get; set; }

        public DbSet<Client> Clients { get; set; }

        public DbSet<TourAgent> TourAgents { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Account>()
                .HasOne(x => x.Client)
                .WithOne(c => c.Account)
                .HasForeignKey<Client>(c => c.AccountId);

            modelBuilder.Entity<Account>()
                .HasOne(x => x.TourAgent)
                .WithOne(t => t.Account)
                .HasForeignKey<TourAgent>(t => t.AccountId);
        }
    }
}

```

Файл Users/Controllers/UserController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Users.Data;
using Users.Dto;
using Users.Models;

namespace Users.Controllers
{
    [Route("api/users")]
    [ApiController]
    public class UsersController : ControllerBase
    {
        private readonly Context ctx;
        private readonly IConfiguration cfg;

        public UsersController(Context ctx, IConfiguration cfg)
        {
            this.ctx = ctx;
            this.cfg = cfg;
        }

        [HttpGet]
        [Authorize(Roles = "touragent")]
        public IActionResult GetAllClient()
        {
            return Ok(ctx.Clients);
        }

        [HttpGet("{clientId}")]
        [Authorize(Roles = "touragent")]
        public IActionResult GetClientById(long clientId)
        {
            return Ok(ctx.Clients.FirstOrDefault(c => c.Id == clientId));
        }

        [HttpGet("me")]
        [Authorize(Roles = "client")]
    }
}

```

```

public async Task<IActionResult> GetSelfInfo()
{
    var claimsIdentity = User.Identity as ClaimsIdentity;
    long id = long.Parse(claimsIdentity!.FindFirst("id")!.Value);
    var client = await ctx.Clients.FirstAsync(c => c.Id == id);
    return Ok(client);
}

[HttpGet("touragents/me")]
[Authorize(Roles = "touragent")]
public async Task<IActionResult> GetSelfInfoTouragent()
{
    var claimsIdentity = User.Identity as ClaimsIdentity;
    long id = long.Parse(claimsIdentity!.FindFirst("id")!.Value);
    var touragent = await ctx.TourAgents.FirstAsync(c => c.Id == id);
    return Ok(touragent);
}

[HttpPost("login")]
public async Task<IActionResult> Login(LoginDto loginDto)
{
    var login = loginDto.Login;
    var password = loginDto.Password;

    var account = await ctx.Accounts.FirstOrDefaultAsync(a =>
a.Login.Equals(login) && a.Password.Equals(password));
    if (account is null)
    {
        return NotFound("Invalid login or password");
    }

    var id = account.Role switch
    {
        "client" => ctx.Clients.First(c => c.AccountId == account.Id).Id,
        "touragent" => ctx.TourAgents.First(c => c.AccountId == account.Id).Id
    };

    var claims = new List<Claim>
    {
        new Claim("name", login),
        new Claim("role", account.Role),
        new Claim("id", $"{id}"),
    };

    var now = DateTime.UtcNow;

    var claimsIdentity = new ClaimsIdentity(claims, "Token");
    var jwt = new JwtSecurityToken(
        issuer: cfg["JWT_ISSUER"],
        audience: "anyone",
        notBefore: now,
        claims: claims,
        expires:
now.Add(TimeSpan.FromHours(double.Parse(cfg["JWT_LIFETIME_IN_HOURS"]))),
        signingCredentials: new SigningCredentials(new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(cfg["JWT_SECRET"])),
SecurityAlgorithms.HmacSha256)
    );

    var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);
    var response = new
    {
        username = login,
        role = account.Role,
        access_token = encodedJwt,
    };

    return Ok(response);
}

```

```

[HttpPost("registration")]
public async Task<IActionResult> Registrare(RegistrationDto registrationDto)
{
    var existAccount = await ctx.Accounts.FirstOrDefaultAsync(a =>
a.Login.Equals(registrationDto.Login));
    if (existAccount is not null)
    {
        return BadRequest("User already exist");
    }

    var newAccount = new Account
    {
        Login = registrationDto.Login,
        Password = registrationDto.Password,
        Role = "client"
    };

    var newClient = new Client
    {
        Name = registrationDto.Name,
        SecondName = registrationDto.SecondName,
        Patronymic = registrationDto.Patronymic,
        Birthday = registrationDto.Birthday,
        Email = registrationDto.Email,
        PhoneNumber = registrationDto.PhoneNumber,
        Account = newAccount,
    };

    await ctx.Accounts.AddAsync(newAccount);
    await ctx.Clients.AddAsync(newClient);
    await ctx.SaveChangesAsync();
    return Ok();
}
}

```

Файл Tours/Models/Tour.cs

```

namespace Tours.Models
{
    public class Tour
    {
        public long Id { get; set; }

        public string Name { get; set; }

        public string Country { get; set; }

        public string HotelName { get; set; }

        public string TypeRoom { get; set; }

        public string TypeFood { get; set; }

        public decimal PricePurchase { get; set; }

        public decimal PriceSale { get; set; }

        public DateTime DateStart { get; set; }

        public DateTime DateEnd { get; set; }

        public int IsActive { get; set; }
    }
}

```

Файл Tours/Models/Request.cs

```

namespace Tours.Models
{
    public class Request

```

```

    {
        public long Id { get; set; }

        public long TourId { get; set; }

        public long UserId { get; set; }

        public long? TourAgentId { get; set; }

        public string Status { get; set; }

        public DateTime Date { get; set; }

        public Tour? Tour { get; set; }
    }
}

```

Файл Tours/Data/Context.cs

```

using Microsoft.EntityFrameworkCore;
using Tours.Models;

namespace Tours.Data
{
    public class Context : DbContext
    {
        public Context(DbContextOptions<Context> options) : base(options) { }

        public DbSet<Tour> Tours { get; set; }

        public DbSet<Request> Requests { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Tour>().Property(x => x.PricePurchase).HasPrecision(8,
2);
            modelBuilder.Entity<Tour>().Property(x => x.PriceSale).HasPrecision(8, 2);
            base.OnModelCreating(modelBuilder);
        }
    }
}

```

Файл Tours/Controllers/TourController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.IdentityModel.Tokens.Jwt;
using Tours.Data;
using Tours.Dto;
using Tours.Models;

namespace Tours.Controllers
{
    [Route("api/tours")]
    [ApiController]
    public class ToursController : ControllerBase
    {
        private readonly Context ctx;

        public ToursController(Context ctx)
        {
            this.ctx = ctx;
        }

        [HttpGet]
        public IActionResult GetAll()
        {
            if (!Request.Headers.Authorization.Any())
            {
                return Ok(ctx.Tours.Where(t => t.IsActive == 1));
            }
        }
    }
}

```

```

var authHeader = Request.Headers.Authorization[0];
if (!string.IsNullOrEmpty(authHeader) && authHeader.StartsWith("Bearer "))
{
    var encodedJwt = authHeader["Bearer ".Length..];
    try
    {
        var jwt = new JwtSecurityTokenHandler().ReadJwtToken(encodedJwt);
        var role = jwt.Claims.First(c => c.Type.Equals("role"));
        if (role.Value.Equals("touragent"))
        {
            return Ok(ctx.Tours);
        }
    }
    catch (Exception e) {}
}

return Ok(ctx.Tours.Where(t => t.IsActive == 1));
}

[HttpGet("{tourId}")]
public IActionResult GetById(long tourId)
{
    return Ok(ctx.Tours.FirstOrDefault(t => t.Id == tourId));
}

[HttpPost]
[Authorize(Roles = "touragent")]
public async Task<IActionResult> CreateTour(CreateTourDto createTourDto)
{
    var tour = new Tour
    {
        Name = createTourDto.Name,
        Country = createTourDto.Country,
        HotelName = createTourDto.HotelName,
        TypeRoom = createTourDto.TypeRoom,
        TypeFood = createTourDto.TypeFood,
        PricePurchase = createTourDto.PricePurchase,
        PriceSale = createTourDto.PriceSale,
        DateStart = createTourDto.DateStart,
        DateEnd = createTourDto.DateEnd,
        IsActive = 1
    };

    await ctx.Tours.AddAsync(tour);
    await ctx.SaveChangesAsync();
    return Ok(tour);
}

[HttpPut("{id}")]
[Authorize(Roles = "touragent")]
public async Task<IActionResult> UpdateTour(long id, UpdateTourDto
updateTourDto)
{
    var tour = await ctx.Tours.FirstOrDefaultAsync(t => t.Id == id);
    if (tour is null)
    {
        return NotFound($"Tour with id = {id} doesn't exist");
    }
    if (!string.IsNullOrEmpty(updateTourDto.Name))
    {
        tour.Name = updateTourDto.Name;
    }
    if (!string.IsNullOrEmpty(updateTourDto.Country))
    {
        tour.Country = updateTourDto.Country;
    }
    if (!string.IsNullOrEmpty(updateTourDto.HotelName))
    {
        tour.HotelName = updateTourDto.HotelName;
    }
}

```

```

    }
    if (!string.IsNullOrEmpty(updateTourDto.TypeRoom))
    {
        tour.TypeRoom = updateTourDto.TypeRoom;
    }
    if (!string.IsNullOrEmpty(updateTourDto.TypeFood))
    {
        tour.TypeFood = updateTourDto.TypeFood;
    }
    if (!string.IsNullOrEmpty(updateTourDto.Name))
    {
        tour.Name = updateTourDto.Name;
    }
    if (updateTourDto.PricePurchase is not null)
    {
        tour.PricePurchase = updateTourDto.PricePurchase.Value;
    }
    if (updateTourDto.PriceSale is not null)
    {
        tour.PriceSale = updateTourDto.PriceSale.Value;
    }
    if (updateTourDto.DateStart is not null)
    {
        tour.DateStart = updateTourDto.DateStart.Value;
    }
    if (updateTourDto.DateEnd is not null)
    {
        tour.DateEnd = updateTourDto.DateEnd.Value;
    }
    if (updateTourDto.IsActive is not null)
    {
        tour.IsActive = updateTourDto.IsActive.Value;
    }

    ctx.Tours.Update(tour);
    await ctx.SaveChangesAsync();
    return Ok(tour);
}

[HttpDelete("{id}")]
[Authorize(Roles = "touragent")]
public async Task<IActionResult> DeleteTour(long id)
{
    var tour = await ctx.Tours.FirstOrDefaultAsync(t => t.Id == id);
    if (tour is null)
    {
        return NotFound($"Tour with id = {id} doesn't exist");
    }

    ctx.Tours.Remove(tour);
    await ctx.SaveChangesAsync();
    return Ok($"Tour with id = {id} has deleted");
}
}

```

Файл Tours/Controllers/ RequestsController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Data;
using System.Security.Claims;
using Tours.Data;
using Tours.Dto;
using Tours.Models;

namespace Tours.Controllers
{
    [Route("api/requests")]
    [ApiController]

```



```

public class RequestsController : ControllerBase
{
    private readonly Context ctx;
    private static readonly string[] states = { "new", "paid", "cancel",
"wait_operator_answer" };

    public RequestsController(Context ctx)
    {
        this.ctx = ctx;
    }

    [HttpPost]
    [Authorize(Roles = "client")]
    public async Task<IActionResult> BookToor(BookTourDto bookTourDto)
    {
        var claimsIdentity = User.Identity as ClaimsIdentity;
        long id = long.Parse(claimsIdentity!.FindFirst("id")!.Value);

        var newRequest = new Request
        {
            TourId = bookTourDto.ToorId,
            UserId = id,
            TourAgentId = null,
            Status = "new",
            Date = DateTime.UtcNow
        };

        await ctx.Requests.AddAsync(newRequest);

        var bookedTour = ctx.Tours.First(t => t.Id == bookTourDto.ToorId);
        bookedTour.IsActive = 0;

        await ctx.SaveChangesAsync();
        return Ok(newRequest);
    }

    [HttpPost("change_status")]
    [Authorize(Roles = "touragent")]
    public async Task<IActionResult> ChangeStatus(ChangeStatusDto changeStatusDto)
    {
        var claimsIdentity = User.Identity as ClaimsIdentity;
        long id = long.Parse(claimsIdentity!.FindFirst("id")!.Value);

        if (!states.Contains(changeStatusDto.NewState))
        {
            return BadRequest($"Status invalid ({changeStatusDto.NewState})");
        }

        if (changeStatusDto.NewState.Equals("wait_operator_answer"))
        {
            return await TakeRequest(changeStatusDto.RequestId, id);
        }
        else if (changeStatusDto.NewState.Equals("paid"))
        {
            return await ConfirmRequest(changeStatusDto.RequestId);
        }
        else if (changeStatusDto.NewState.Equals("cancel"))
        {
            return await CancelRequest(changeStatusDto.RequestId);
        }

        return BadRequest($"Bad status");
    }

    [NonAction]
    private async Task<IActionResult> TakeRequest(long requestId, long
tourAgentId)
    {
        var request = ctx.Requests.FirstOrDefault(r => r.Id == requestId);
        if (request is null)
    }

```

```

    {
        return BadRequest("Request not exist");
    }

    if (!request.Status.Equals("new"))
    {
        return BadRequest("Request already in work");
    }

    request.Status = "wait_operator_answer";
    request.TourAgentId = tourAgentId;

    var tour = ctx.Tours.FirstOrDefault(t => t.Id == request.TourId);
    if (tour is null)
    {
        return BadRequest("Tour not exist for this request");
    }
    tour.IsActive = 0;

    await ctx.SaveChangesAsync();
    return Ok($"Request ({request.Id}) in work");
}

[NonAction]
public async Task<IActionResult> ConfirmRequest(long requestId)
{
    var request = ctx.Requests.FirstOrDefault(r => r.Id == requestId);
    if (request is null)
    {
        return BadRequest("Request not exist");
    }

    if (!request.Status.Equals("wait_operator_answer"))
    {
        return BadRequest("Request already confirmed");
    }

    request.Status = "paid";

    await ctx.SaveChangesAsync();
    return Ok($"Request ({request.Id}) has confirmed ");
}

[NonAction]
public async Task<IActionResult> CancelRequest(long requestId)
{
    var request = ctx.Requests.FirstOrDefault(r => r.Id == requestId);
    if (request is null)
    {
        return BadRequest("Request not exist");
    }

    if (!request.Status.Equals("paid") &&
!request.Status.Equals("wait_operator_answer"))
    {
        return BadRequest("Request has not been paid");
    }

    request.Status = "cancel";

    var tour = ctx.Tours.FirstOrDefault(t => t.Id == request.TourId);
    if (tour is null)
    {
        return BadRequest("Tour not exist for this request");
    }
    tour.IsActive = 1;

    await ctx.SaveChangesAsync();
    return Ok($"Request ({request.Id}) has canceled ");
}

```

```

[HttpGet]
[Authorize(Roles = "touragent")]
public IActionResult GetAllRequests([FromQuery] string? status)
{
    if (string.IsNullOrEmpty(status) || !states.Contains(status))
    {
        return Ok(ctx.Requests
            .Include(r => r.Tour));
    }

    return Ok(ctx.Requests
        .Where(r => r.Status.Equals(status))
        .Include(r => r.Tour));
}

[HttpGet("{clientId}")]
[Authorize(Roles = "touragent")]
public IActionResult GetRequestsForClient(long clientId, [FromQuery] string?
status)
{
    if (string.IsNullOrEmpty(status) || !states.Contains(status))
    {
        return Ok(ctx.Requests
            .Where(r => r.UserId == clientId)
            .Include(r => r.Tour));
    }

    return Ok(ctx.Requests
        .Where(r => r.UserId == clientId)
        .Where(r => r.Status.Equals(status))
        .Include(r => r.Tour));
}

[HttpGet("me")]
[Authorize(Roles = "client")]
public IActionResult GetSelfRequests([FromQuery] string? status)
{
    var claimsIdentity = User.Identity as ClaimsIdentity;
    long id = long.Parse(claimsIdentity!.FindFirst("id")!.Value);

    if (string.IsNullOrEmpty(status) || !states.Contains(status))
    {
        return Ok(ctx.Requests
            .Where(r => r.UserId == id)
            .Include(r => r.Tour));
    }

    return Ok(ctx.Requests
        .Where(r => r.UserId == id)
        .Where(r => r.Status.Equals(status))
        .Include(r => r.Tour));
}
}

```

Файл Analytics/Controllers/ AnalyticsController.cs

```

using Analytics.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Data;
using System.Text.Json;

namespace Analytics.Controllers
{
    [Route("api/analytics")]
    [ApiController]
    public class AnalyticsController : ControllerBase
    {
        private readonly IHttpClientFactory httpClientFactory;
    }
}

```

```

        private static readonly JsonSerializerOptions options = new() {
            PropertyNameCaseInsensitive = true };

        public AnalyticsController(IHttpClientFactory httpClientFactory)
        {
            this.httpClientFactory = httpClientFactory;
        }

        [HttpGet("client_stats")]
        [Authorize(Roles = "touragent")]
        public async Task<IActionResult> GetClientsStatistics()
        {
            var tourHttpClient = httpClientFactory.CreateClient("Requests");
            tourHttpClient.DefaultRequestHeaders.Add("Authorization", new string[] {
                Request.Headers.Authorization });

            List<Request>? requests;
            using (var response = await
                tourHttpClient.GetAsync("requests?status=paid"))
            {
                var stream = await response.Content.ReadAsStreamAsync();
                requests = await
                    JsonSerializer.DeserializeAsync<List<Request>>(stream, options);
            }

            if (requests is null)
            {
                return NotFound();
            }

            var stats = (from req in requests
                group req by req.UserId into g
                select new
                {
                    ClientId = g.Key,
                    Count = g.Count(),
                    TotalMoney = g.Sum(r => r.Tour!.PriceSale)
                }).OrderByDescending(g => g.TotalMoney);

            return Ok(stats);
        }

        [HttpGet("top_country")]
        [Authorize(Roles = "touragent")]
        public async Task<IActionResult> GetTopCountry()
        {
            var tourHttpClient = httpClientFactory.CreateClient("Requests");
            tourHttpClient.DefaultRequestHeaders.Add("Authorization", new string[] {
                Request.Headers.Authorization });

            List<Request>? requests;
            using (var response = await
                tourHttpClient.GetAsync("requests?status=paid"))
            {
                var stream = await response.Content.ReadAsStreamAsync();
                requests = await
                    JsonSerializer.DeserializeAsync<List<Request>>(stream, options);
            }

            if (requests is null)
            {
                return NotFound();
            }

            var stats = (from req in requests
                group req by req.Tour!.Country into g
                select new
                {
                    Country = g.Key,

```

```
        Count = g.Count(),  
        TotalProfit = g.Sum(r => r.Tour!.PriceSale -  
r.Tour!.PricePurchase)  
    }).OrderByDescending(g => g.TotalProfit);  
  
    return Ok(stats);  
}  
}
```

ПРИЛОЖЕНИЕ Б

(Исходный код клиентской части приложения)

Файл WinFormsApp1/AuthForm.cs

```

using System.ComponentModel.DataAnnotations;
using System.Net;
using System.Net.Http.Json;
using WinFormsApp1.Models;

namespace WinFormsApp1
{
    public partial class AuthForm : Form
    {
        public AuthForm()
        {
            InitializeComponent();
        }

        private async void btnLogin_Click(object sender, EventArgs e)
        {
            var credentials = new LoginModel
            {
                Login = tbLogin.Text,
                Password = tbPassword.Text
            };

            var results = new List<ValidationResult>();
            var context = new ValidationContext(credentials);
            if (!Validator.TryValidateObject(credentials, context, results, true))
            {
                var errorMessage = string.Join("\n", results.Select(err => $"•
{err.ErrorMessage};"));
                MessageBox.Show($"Ошибка валидации", "Ошибка валидации",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            var httpClient = HttpClientFactory.Create();

            var loginResponse = await httpClient.PostAsJsonAsync("auth/login",
            credentials);
            if (loginResponse.StatusCode != HttpStatusCode.OK)
            {
                MessageBox.Show($"Ошибка\nКод: {loginResponse.StatusCode}", "Ответ",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            var authResponse = await
            loginResponse.Content.ReadFromJsonAsync<AuthResponse>();

            CurrentUser.Username = authResponse.Username;
            CurrentUser.Role = authResponse.Role;
            CurrentUser.AccessToken = authResponse.AccessToken;

            var mainForm = new MainForm();
            mainForm.FormClosed += (s, args) => Show();
            mainForm.Show();
            Hide();
        }

        private async void btnRegistrat_Click(object sender, EventArgs e)
        {
            var newClient = new RegistrationModel
            {
                SecondName = tbSecondName.Text,
                Name = tbName.Text,
                Patronymic = tbPatronymic.Text,
                Birthday = dtpBirthday.Value,
                Email = tbEmail.Text,
                PhoneNumber = tbPhoneNumber.Text,
                Login = tbLoginNew.Text,
            }
        }
    }
}

```

```

        Password = tbPasswordNew.Text,
        RepeatPassword = tbRepeatPassword.Text
    };

    var results = new List<ValidationResult>();
    var context = new ValidationContext(newClient);
    if (!Validator.TryValidateObject(newClient, context, results, true))
    {
        var errorMessage = string.Join("\n", results.Select(err => $"{err.ErrorMessage}"));
        MessageBox.Show($"{errorMessage}", "Ошибка валидации",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    var httpClient = HttpClientFactory.Create();

    var response = await httpClient.PostAsJsonAsync("auth/registration",
newClient);
    if (response.StatusCode == HttpStatusCode.OK)
    {
        MessageBox.Show($"Создание успешно", "Ответ", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show($"Ошибка\nКод: {response.StatusCode}", "Ответ",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}
}

```

Файл WinFormsApp1/MainForm.cs

```

using OxyPlot;
using OxyPlot.Series;
using System.Net.Http.Json;
using WinFormsApp1.Models;

namespace WinFormsApp1
{
    public partial class MainForm : Form
    {
        public static Dictionary<string, string> StatusToText { get; } = new()
        {
            { "new", "НЕОБРАБОТАНА" },
            { "wait_operator_answer", "ОЖИДАНИЕ" },
            { "paid", "ОПЛАЧЕНА" },
            { "cancel", "ОТМЕНЕНА" },
        };

        public static Dictionary<string, string> TextToStatus { get; } = new()
        {
            { "НЕОБРАБОТАНА", "new" },
            { "ОЖИДАНИЕ", "wait_operator_answer" },
            { "ОПЛАЧЕНА", "paid" },
            { "ОТМЕНЕНА", "cancel" },
        };

        private DataGridViewCell selectedCellTour;

        private DataGridViewCell selectedCellRequest;

        public MainForm()
        {
            InitializeComponent();
            if (CurrentUser.Role!.Equals("client"))
            {

```



```

        tabControll1.TabPages.Remove(tabClients);
        tabControll1.TabPages.Remove(tabAnalytics);

        btnAddTour.Visible = false;
        dataGridViewTours.CellDoubleClick -=
dataGridViewTours_CellDoubleClick!;
        dataGridViewTours.CellMouseClick += dataGridViewTours_CellMouseClick!;

        btnLoadRequests.Click -= btnLoadRequests_Click!;
        btnLoadRequests.Click += btnLoadRequests_ClickClient!;
        dataGridViewRequests.CellMouseClick -=
dataGridViewRequests_CellMouseClick!;
    }
}

private async void MainForm_Load(object sender, EventArgs e)
{
    tbCurrentRole.Text = CurrentUser.Role;

    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);

    if (CurrentUser.Role.Equals("client"))
    {
        var response = await httpClient.GetAsync("clients/me");
        if (!response.IsSuccessStatusCode)
        {
            return;
        }

        var data = await response.Content.ReadFromJsonAsync<ClientModel>();
        tbCurrentUsername.Text = $"{data.SecondName} {data.Name[0]}.
{data.Patronymic?[0]}.";
    }
    else if (CurrentUser.Role.Equals("touragent"))
    {
        var response = await httpClient.GetAsync("touragents/me");
        if (!response.IsSuccessStatusCode)
        {
            return;
        }

        var data = await response.Content.ReadFromJsonAsync<TourAgentModel>();
        tbCurrentUsername.Text = $"{data.SecondName} {data.Name[0]}.
{data.Patronymic?[0]}.";
    }
}

private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
{
    CurrentUser.Username = null;
    CurrentUser.Role = null;
    CurrentUser.AccessToken = null;
}

#region clients tab
private async void btnLoadClients_Click(object sender, EventArgs e)
{
    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);

    var response = await httpClient.GetAsync("clients");
    if (!response.IsSuccessStatusCode)
    {
        return;
    }
}

```

```

        dataGridViewClients.Rows.Clear();

        var clients = await
response.Content.ReadFromJsonAsync<List<ClientModel>>();
        foreach(var client in clients)
        {
            dataGridViewClients.Rows.Add(
                client.Id,
                client.SecondName,
                client.Name,
                client.Patronymic,
                client.Birthday,
                client.Email,
                client.PhoneNumber);
        }
    }
#endregion

#region tours tab
private async void btnLoadTours_Click(object sender, EventArgs e)
{
    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer
{CurrentUser.AccessToken}");

    var response = await httpClient.GetAsync("tours");
    if (!response.IsSuccessStatusCode)
    {
        return;
    }

    dataGridViewTours.Rows.Clear();
    var tours = await response.Content.ReadFromJsonAsync<List<TourModel>>();
    foreach (var tour in tours)
    {
        dataGridViewTours.Rows.Add(
            tour.Id,
            tour.Name,
            tour.Country,
            tour.HotelName,
            tour.TypeRoom,
            tour.TypeFood,
            tour.PricePurchase,
            tour.PriceSale,
            tour.DateStart,
            tour.DateEnd,
            tour.IsActive);
    }
}

private async void btnAddTour_Click(object sender, EventArgs e)
{
    var createTourForm = new CreateUpdateTourForm();
    createTourForm.ShowDialog();

    if (createTourForm.TourModel is null)
    {
        return;
    }

    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer
{CurrentUser.AccessToken}");

    var addResponse = await httpClient.PostAsJsonAsync("tours",
createTourForm.TourModel);
    if (addResponse.IsSuccessStatusCode)

```

```

        {
            var addedTour = await
addResponse.Content.ReadFromJsonAsync<TourModel>();
            dataGridViewTours.Rows.Add(
                addedTour.Id,
                addedTour.Name,
                addedTour.Country,
                addedTour.HotelName,
                addedTour.TypeRoom,
                addedTour.TypeFood,
                addedTour.PricePurchase,
                addedTour.PriceSale,
                addedTour.DateStart,
                addedTour.DateEnd,
                addedTour.IsActive);
        }
    }

    private async void dataGridViewTours_CellDoubleClick(object sender,
DataGridViewCellEventArgs e)
    {
        if (dataGridViewTours.Rows[e.RowIndex].Cells[e.ColumnIndex].Value is null)
        {
            return;
        }

        var row = dataGridViewTours.Rows[e.RowIndex];
        var selectedTour = new TourModel
        {
            Id = long.Parse(row.Cells[0].Value.ToString()),
            Name = row.Cells[1].Value.ToString(),
            Country = row.Cells[2].Value.ToString(),
            HotelName = row.Cells[3].Value.ToString(),
            TypeRoom = row.Cells[4].Value.ToString(),
            TypeFood = row.Cells[5].Value.ToString(),
            PricePurchase = decimal.Parse(row.Cells[6].Value.ToString()),
            PriceSale = decimal.Parse(row.Cells[7].Value.ToString()),
            DateStart = DateTime.Parse(row.Cells[8].Value.ToString()),
            DateEnd = DateTime.Parse(row.Cells[9].Value.ToString()),
            IsActive = int.Parse(row.Cells[10].Value.ToString())
        };

        var createTourForm = new CreateUpdateTourForm(selectedTour);
        createTourForm.ShowDialog();

        if (!createTourForm.ReadyToUpdate)
        {
            return;
        }

        var httpClient = HttpClientFactory.Create();

        httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer
{CurrentUser.AccessToken}");

        var updateResponse = await
httpClient.PutAsJsonAsync($"tours/{selectedTour.Id}", createTourForm.TourModel);
        if (updateResponse.IsSuccessStatusCode)
        {
            var updatedTour = await
updateResponse.Content.ReadFromJsonAsync<TourModel>();
            row.Cells[0].Value = updatedTour.Id;
            row.Cells[1].Value = updatedTour.Name;
            row.Cells[2].Value = updatedTour.Country;
            row.Cells[3].Value = updatedTour.HotelName;
            row.Cells[4].Value = updatedTour.TypeRoom;
            row.Cells[5].Value = updatedTour.TypeFood;
            row.Cells[6].Value = updatedTour.PricePurchase;
            row.Cells[7].Value = updatedTour.PriceSale;
            row.Cells[8].Value = updatedTour.DateStart;

```

```

        row.Cells[9].Value = updatedTour.DateEnd;
        row.Cells[10].Value = updatedTour.IsActive;
    }

    dataGridViewTours.Update();
}

private void dataGridViewTours_CellMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
{
    if (e.Button == MouseButtons.Right)
    {
        var cell = selectedCellTour =
dataGridViewTours.Rows[e.RowIndex].Cells[e.ColumnIndex];
        Rectangle r =
dataGridViewTours.GetCellDisplayRectangle(cell.ColumnIndex, cell.RowIndex, false);
        var p = new Point(r.X + r.Width / 2, r.Y + r.Height / 2);
        contextMenuStripTours.Show(dataGridViewTours, p);
    }
}
#endregion

#region requests tab
private async void btnLoadRequests_Click(object sender, EventArgs e)
{
    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);

    if (!TextToStatus.TryGetValue(comboBoxStatus.Text, out var status))
    {
        status = "";
    }

    var response = await httpClient.GetAsync($"requests?status={status}");
    if (!response.IsSuccessStatusCode)
    {
        return;
    }

    dataGridViewRequests.Rows.Clear();
    var requests = await
response.Content.ReadFromJsonAsync<List<RequestModel>>();
    foreach (var req in requests)
    {
        dataGridViewRequests.Rows.Add(
            req.Id,
            req.UserId,
            req.TourId,
            req.TourAgentId,
            StatusToText[req.Status],
            req.Date
        );
    }
}

private async void btnLoadRequests_ClickClient(object sender, EventArgs e)
{
    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);

    if (!TextToStatus.TryGetValue(comboBoxStatus.Text, out var status))
    {
        status = "";
    }

    var response = await httpClient.GetAsync($"requests/me?status={status}");

```

```

        if (!response.IsSuccessStatusCode)
        {
            return;
        }

        dataGridViewRequests.Rows.Clear();
        var requests = await
response.Content.ReadFromJsonAsync<List<RequestModel>>();
        foreach (var req in requests)
        {
            dataGridViewRequests.Rows.Add(
                req.Id,
                req.UserId,
                req.TourId,
                req.TourAgentId,
                StatusToText[req.Status],
                req.Date
            );
        }
    }

    private async void contextMenuStrip1_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)
    {
        var httpClient = HttpClientFactory.Create();

        httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);

        var row = dataGridViewTours.Rows[selectedCellTour.RowIndex];
        var tourId = long.Parse(row.Cells[0].Value.ToString());
        var response = await httpClient.PostAsJsonAsync("requests", new { ToorId =
tourId });
        if (!response.IsSuccessStatusCode)
        {
            return;
        }

        dataGridViewTours.Rows.Remove(row);

        var addedRequest = await
response.Content.ReadFromJsonAsync<RequestModel>();
        if (dataGridViewRequests.RowCount != 0)
        {
            dataGridViewRequests.Rows.Add(
                addedRequest.Id,
                addedRequest.UserId,
                addedRequest.TourId,
                addedRequest.TourAgentId,
                StatusToText[addedRequest.Status],
                addedRequest.Date);
        }
    }

    private void dataGridViewRequests_CellMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
    {
        if (e.Button == MouseButtons.Right)
        {
            var cell = selectedCellRequest =
dataGridViewRequests.Rows[e.RowIndex].Cells[e.ColumnIndex];
            Rectangle r =
dataGridViewRequests.GetCellDisplayRectangle(cell.ColumnIndex, cell.RowIndex, false);
            var p = new Point(r.X + r.Width / 2, r.Y + r.Height / 2);
            contextMenuStripRequests.Show(dataGridViewRequests, p);
        }
    }

    private void contextMenuStripRequests_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)

```

```

{
    var row = dataGridViewRequests.Rows[selectedCellRequest.RowIndex];
    var requestId = long.Parse(row.Cells[0].Value.ToString());
    var status = row.Cells[4].Value.ToString();

    var changeStatusForm = new ChangeStatusForm(requestId, status);
    changeStatusForm.ShowDialog();

    if (changeStatusForm.ChangingIsSuccess)
    {
        row.Cells[4].Value = changeStatusForm.NewState;
    }
}
#endregion

#region stats
private async void btnStatCountry_Click(object sender, EventArgs e)
{
    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);
    var response = await httpClient.GetAsync("analytics/top_country");
    if (!response.IsSuccessStatusCode)
    {
        return;
    }

    var countriesStats = await
response.Content.ReadFromJsonAsync<List<CountriesStat>>();

    dataGridViewStats.Rows.Clear();
    dataGridViewStats.Columns.Clear();

    dataGridViewStats.Columns.Add("ColumnStatsCountry", "Страна");
    dataGridViewStats.Columns.Add("ColumnStatsCountryCountTours", "Количество
купленных путевок");
    dataGridViewStats.Columns.Add("ColumnStatsTotalProfit", "Общая выручка");

    var model = new PlotModel { Title = "Статистика по странам" };
    var series = new PieSeries { StrokeThickness = 2.0, InsideLabelPosition =
0.8, AngleSpan = 360, StartAngle = 0 };
    foreach (var stat in countriesStats)
    {
        dataGridViewStats.Rows.Add(stat.Country, stat.Count,
stat.TotalProfit);
        series.Slices.Add(new PieSlice($"{stat.Country}",
(double)stat.TotalProfit));
    }
    model.Series.Add(series);
    plotView1.Model = model;
}

private async void btnStatClient_Click(object sender, EventArgs e)
{
    var httpClient = HttpClientFactory.Create();

    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " +
CurrentUser.AccessToken);
    var response = await httpClient.GetAsync("analytics/client_stats");
    if (!response.IsSuccessStatusCode)
    {
        return;
    }

    var clientsStats = await
response.Content.ReadFromJsonAsync<List<ClientsStat>>();

    dataGridViewStats.Rows.Clear();
    dataGridViewStats.Columns.Clear();

```

```

        dataGridViewStats.Columns.Add("ColumnStatsClientID", "ID Клиента");
        dataGridViewStats.Columns.Add("ColumnStatsCountTours", "Количество
купленных путевок");
        dataGridViewStats.Columns.Add("ColumnStatsTotalMoney", "Общая потраченная
сумма");

        var model = new PlotModel { Title = "Статистика по клиентам" };
        var series = new PieSeries { StrokeThickness = 2.0, InsideLabelPosition =
0.8, AngleSpan = 360, StartAngle = 0, };
        foreach (var stat in clientsStats)
        {
            dataGridViewStats.Rows.Add(stat.ClientId, stat.Count,
stat.TotalMoney);
            series.Slices.Add(new PieSlice($"{stat.ClientId}",
(double)stat.TotalMoney));
        }

        model.Series.Add(series);
        plotView1.Model = model;
    }
    #endregion
}

```

Файл WinFormsApp1/CurrentUser.cs

```

namespace WinFormsApp1
{
    public static class CurrentUser
    {
        public static string? Username { get; set; }

        public static string? Role { get; set; }

        public static string? AccessToken { get; set; }
    }
}

```

Файл WinFormsApp1/HttpClientFactory.cs

```

namespace WinFormsApp1
{
    public static class HttpClientFactory
    {
        public static HttpClient Create()
        {
            return new HttpClient
            {
                BaseAddress = new Uri("https://localhost:7120/api/"),
                Timeout = new TimeSpan(0, 0, 30)
            };
        }
    }
}

```

ПРИЛОЖЕНИЕ В
(Скриншоты тестирования)

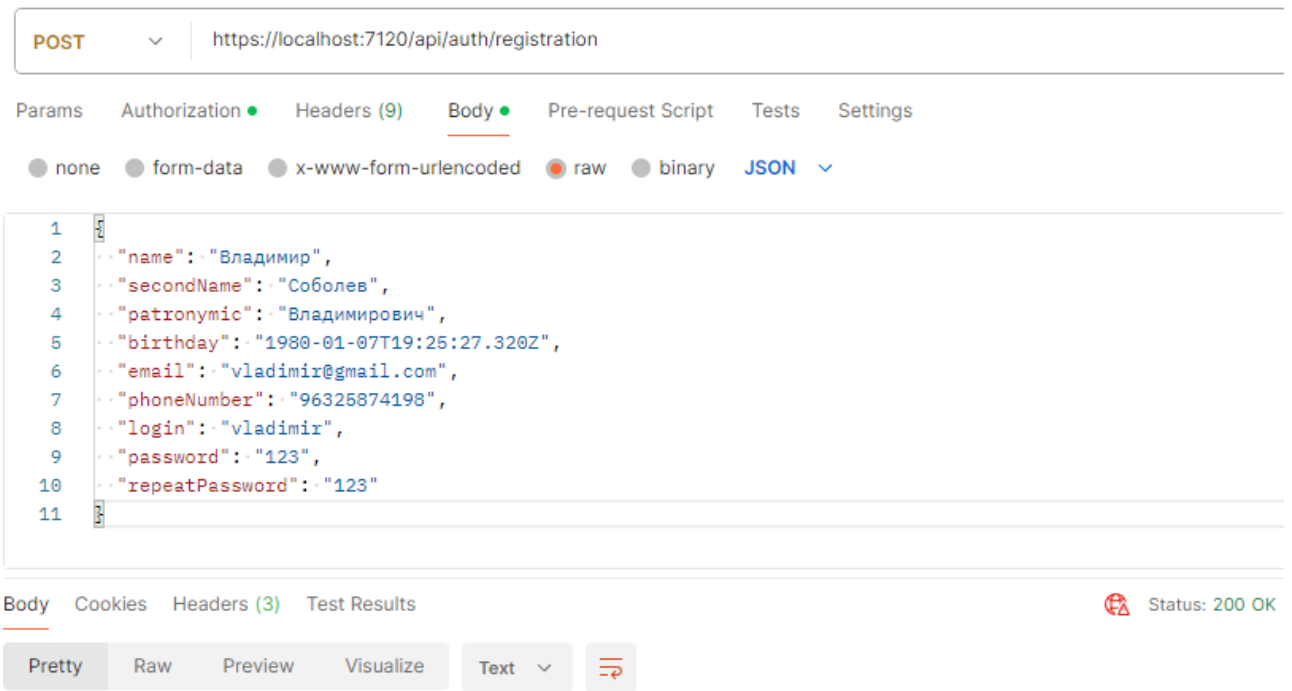


Рисунок В.1 – Успешная регистрация клиента

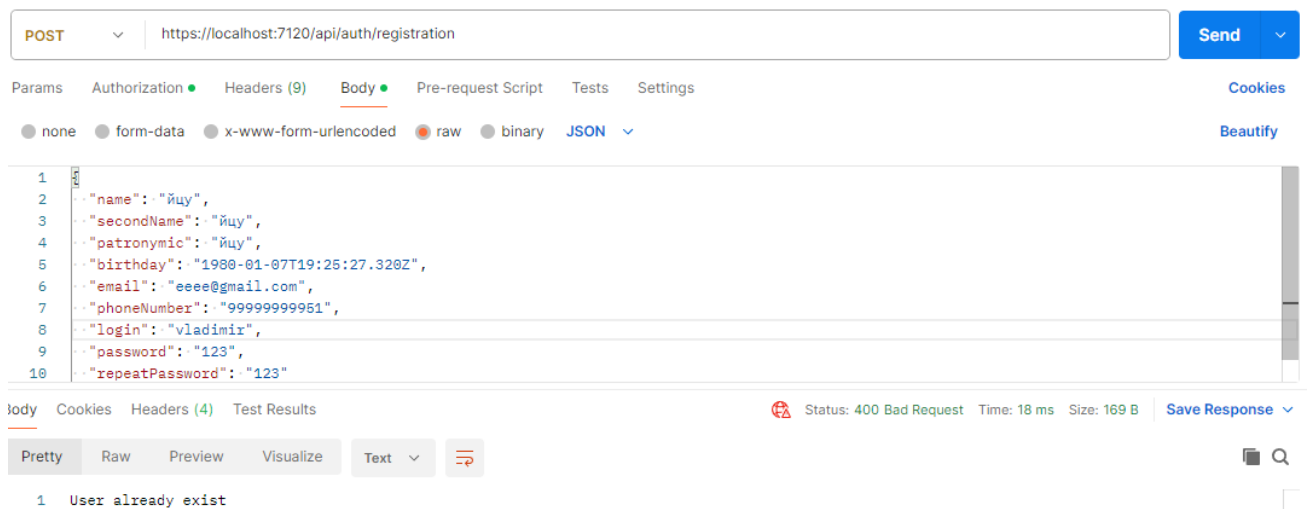


Рисунок В.2 – Ошибка регистрация при указании занятого логина

A screenshot of the Chrome DevTools Network tab. The top bar shows a GET request to https://localhost:7120/api/clients. Below the tabs (Params, Authorization, Headers, Body, Pre-request Script, Tests, Settings), the Headers section is selected, showing 6 hidden headers. A table lists headers with columns Key, Value, and Bulk Edit. One header, 'Authorization', is expanded, showing its value as 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyYyWlIljldmVhZGltZXNjaWJvY2xpZnW50lbnwWQioiOiMDAwMSIsIm5ZIiI6MTcxMjc2MzcwMiwiZmxwIjozeEYODY3MzEyLCJpc3MiOiJNeUFIdGhtZXJkZXIIILCJhdWQiOiJhbnRvbmlufQ.HCPR9LzZC0ZCu-3kF-n2IMutIVt4 ID INK iwaKn3nF1AI'. At the bottom, the status bar indicates 'Status: 403 Forbidden' with response time 'Time: 146 ms' and size 'Size: 99 B'. There are also links for 'Save Response' and view options like 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and a search icon.

GET

https://localhost:7120/api/clients/me

Send

Params

Authorization ●

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Headers

<> 6 hidden

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm9jaWVudmVudGItaXILCjYyb...	
	Key	Value	

Body

Cookies

Headers (4)

Test Results

Status: 200 OK Time: 114 ms Size: 382 B Save Response ✓

Pretty

Raw

Preview

Visualize

JSON ▾

⌵

```
1 {  
2   "id": 20001,  
3   "name": "Владимир",  
4   "secondName": "Соболев",  
5   "patronymic": "Владимирович",  
6   "birthday": "1980-01-07T19:25:27.32",  
7   "email": "vladimir@gmail.com",  
8   "phoneNumber": "96325874198",  
9   "accountId": 20001,  
10  "account": null  
11 }
```

Рисунок В.5 – Успешно получена информация о текущем клиенте

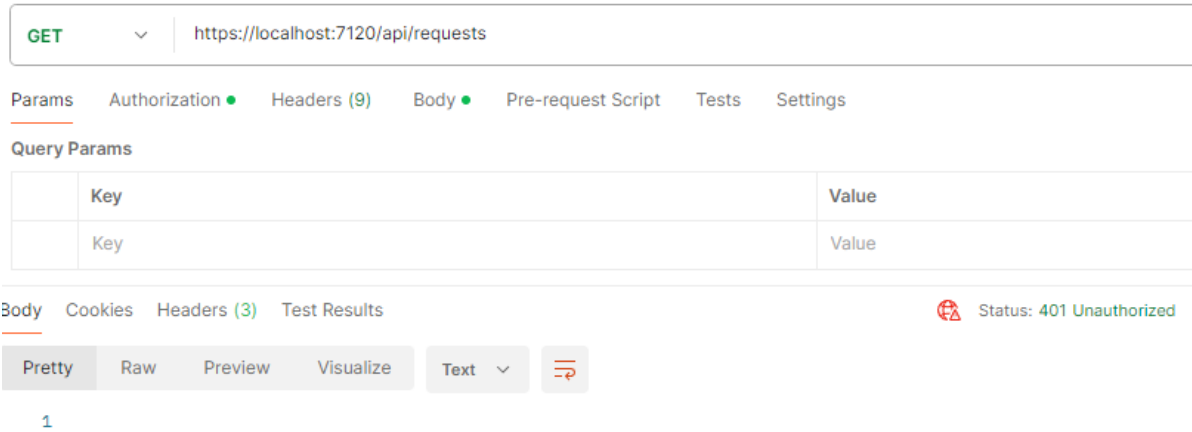


Рисунок В.6 – Ошибка авторизации при получении списка заявок

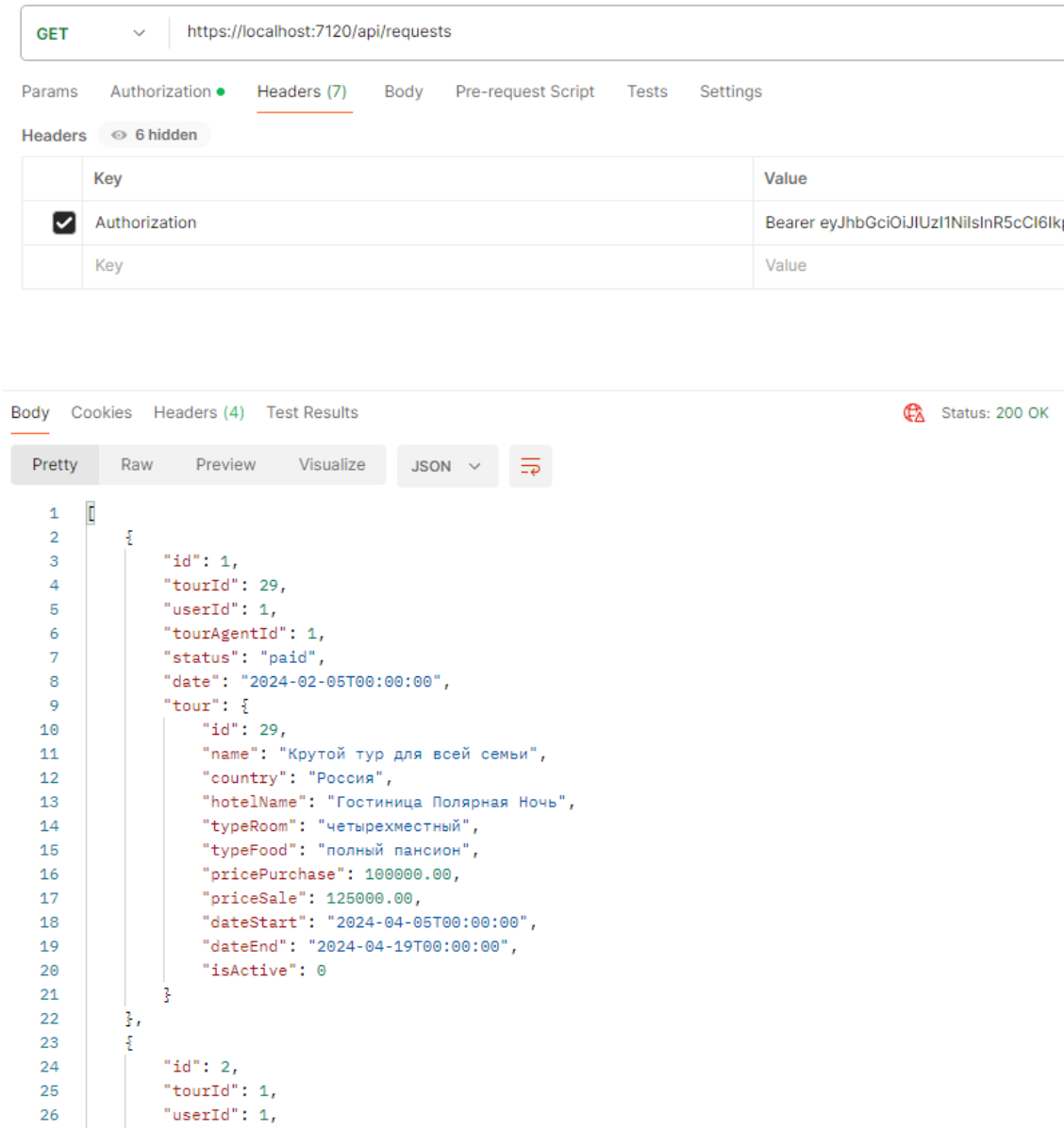


Рисунок В.7 – Успешное получение списка всех заявок

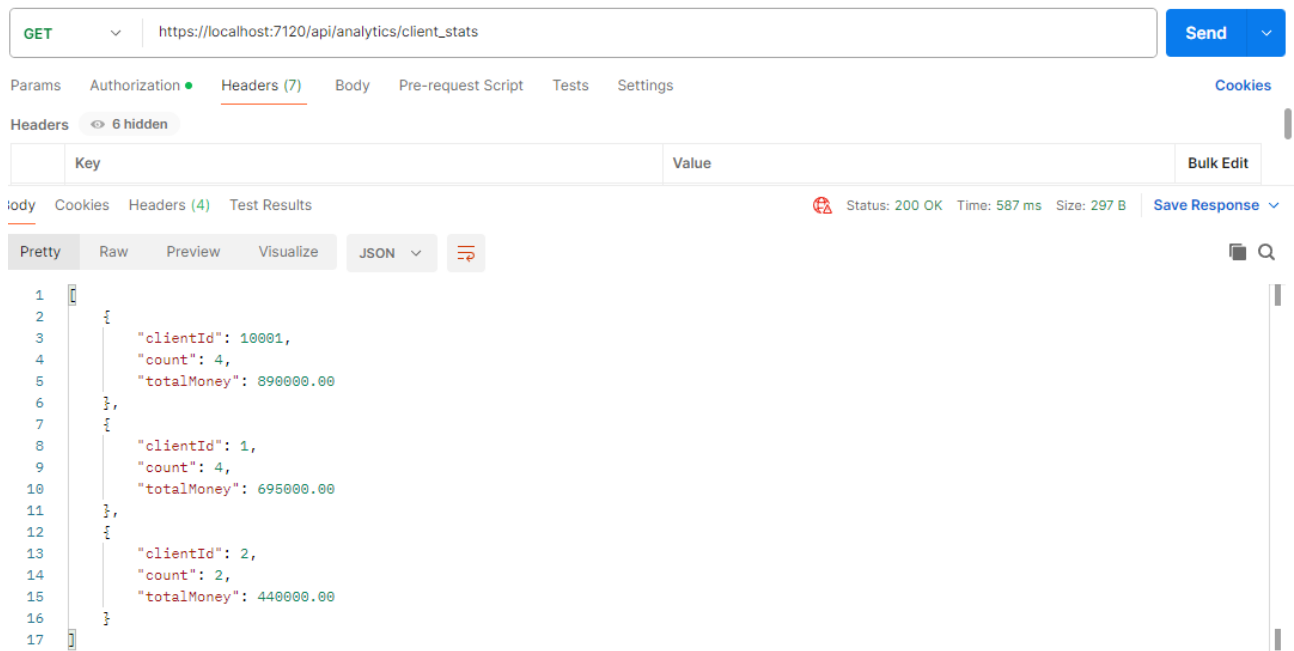


Рисунок В.8 – Успешное получение статистики по клиентам

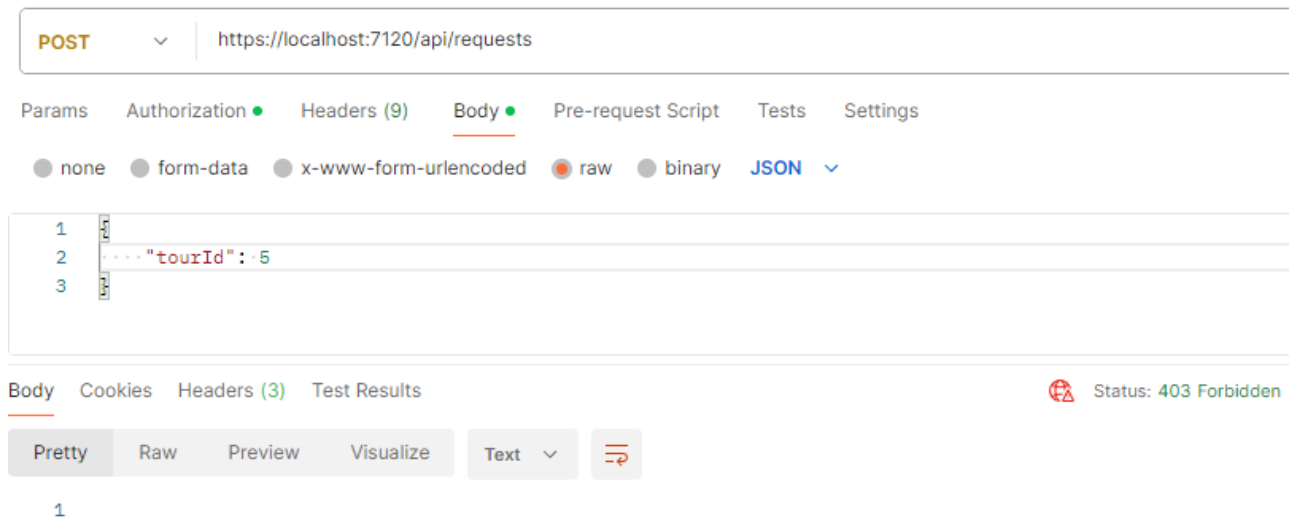


Рисунок В.9 – Возникновение ошибки бронирования тура у турагента

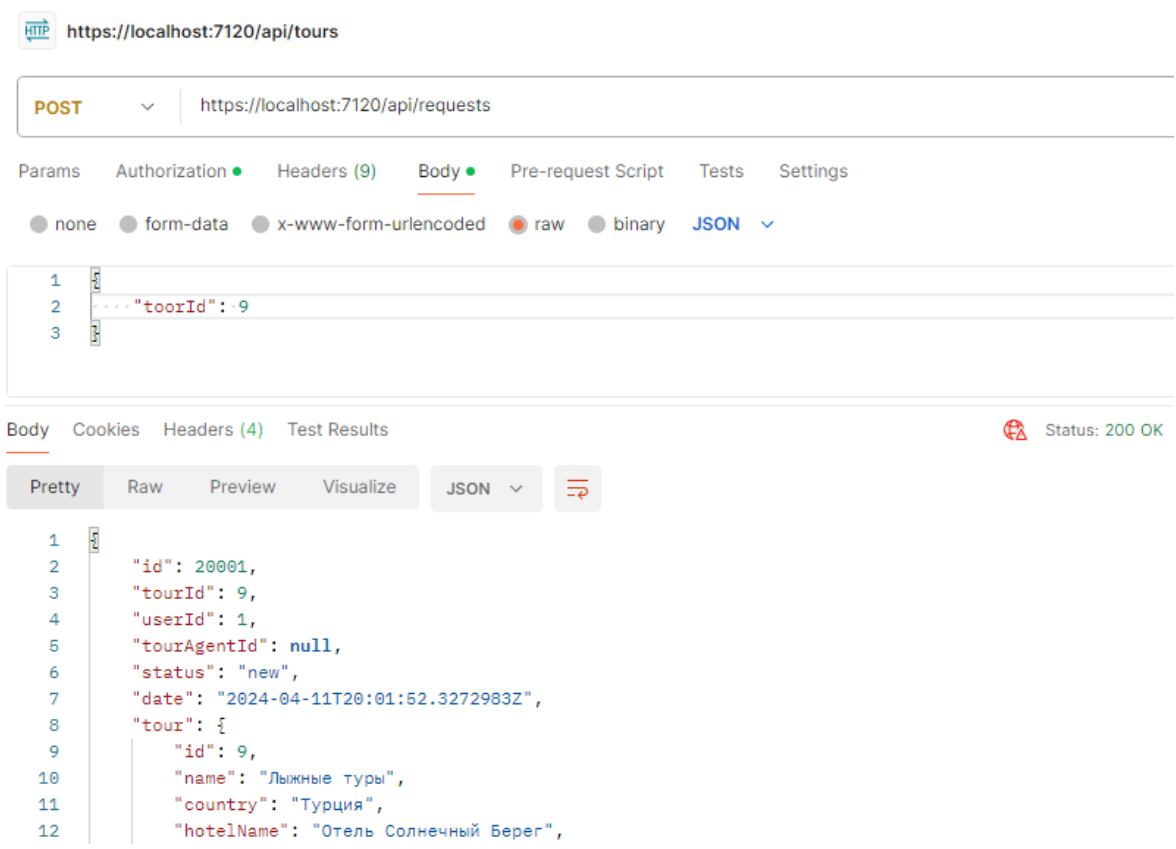


Рисунок В.10 – Успешное бронирование тура клиентом

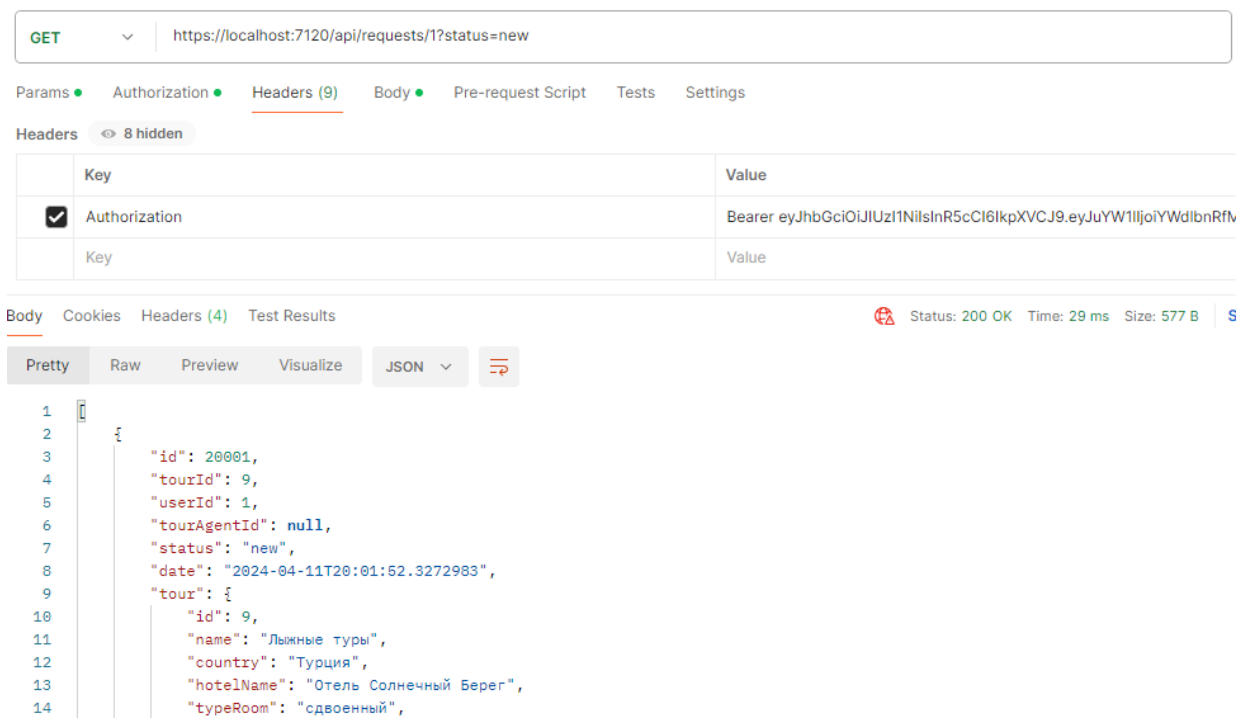


Рисунок В.11 – Успешное получение всех новых заявок для указанного клиента

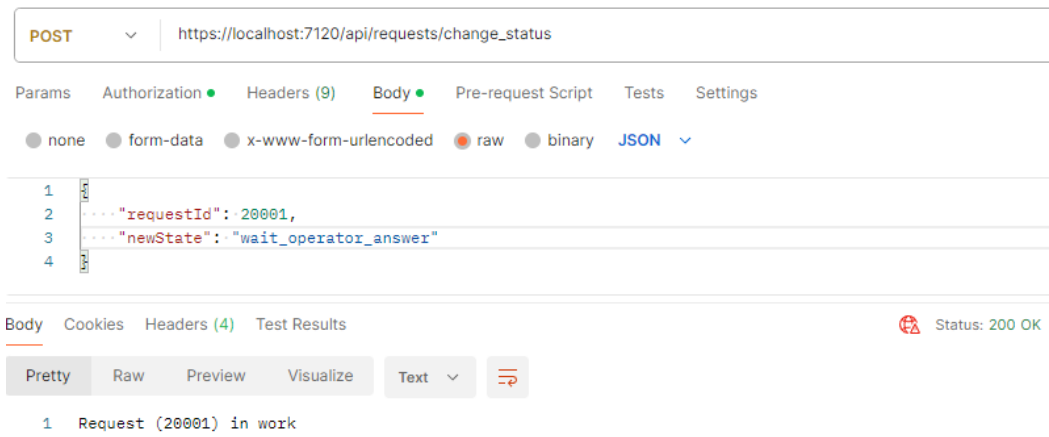


Рисунок В.12 – Успешное изменение статуса у заявки