

**Академия TOP
РПО-1**

Проект “Система продажи авиабилетов”

Тема работы:

«Разработка консольной системы продажи авиабилетов
на технологиях Node.js, C#, ADO.NET, Entity Framework Core»

Выполнили:

Васильев Данил, Зайцев Архип, Белый Данил

Группа: РПО-1

Санкт-Петербург,
2025

Введение

В современных условиях цифровизации важно создавать доступные и эффективные инструменты продажи авиабилетов, обеспечивающие быструю работу с данными, безопасность и удобство для пользователей. Консольные приложения особенно актуальны для внутренних корпоративных систем и учебных задач, где приоритетом являются функциональность, надежность и простота поддержки.

Данный проект направлен на создание консольной автоматизированной системы управления процессом продажи авиабилетов на основе технологий Node.js и C#, с хранением данных в реляционной базе и использованием современных подходов доступа к данным (ADO.NET, Entity Framework Core)

Цели работы

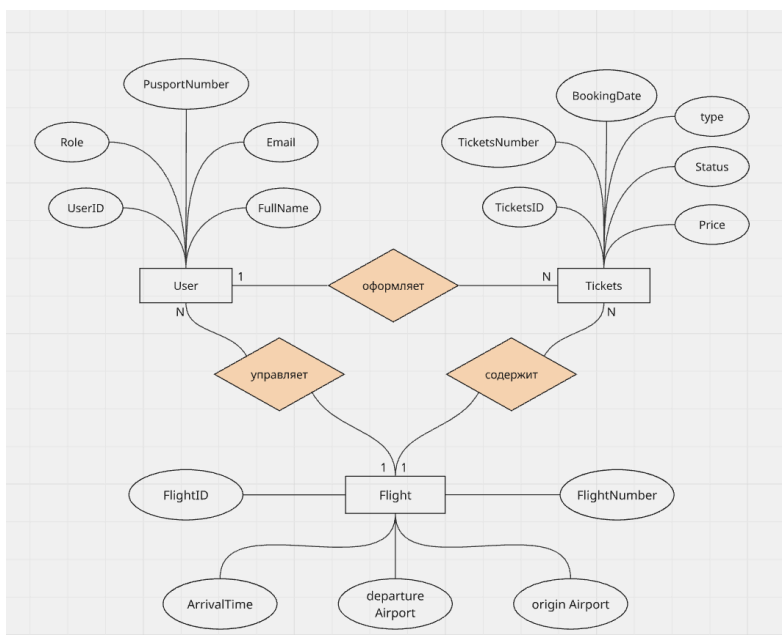
1. Разработать консольную информационную систему для автоматизации процесса продажи авиабилетов.
2. Обеспечить регистрацию рейсов, управление местами и пассажирами, оформление и возврат билетов.
3. Продемонстрировать применение актуальных технологий: Node.js для серверной части и интеграций, C# с ADO.NET и Entity Framework для бизнес-логики и работы с базой данных
4. Реализовать надежную структуру хранения информации о клиентах, рейсах, билетах; обеспечить поиск, фильтрацию и отчёты.

План работы

- ☐ Анализ предметной области
- ☐ Вывод и готовый документ (пропорционально)
- ☐ Исследование процесса авиаперевозок, требований к системам бронирования и продажи авиабилетов
- ☐ Проектирование архитектуры приложения

- ☐ Создание структуры базовых сущностей: Рейсы, Пассажиры, Билеты, Места и Пользователи (Администратор, Кассир, Клиент).
- ☐ Разработка структуры базы данных
- ☐ Создание таблиц для хранения информации о рейсах, пассажирах, билетах, расписании и свободных местах.
- ☐ Реализация бизнес-логики
- ☐ Интеграция с Node.js
- ☐ Обеспечение взаимодействия с внешними API или модулями (при необходимости).
- ☐ Реализация доступа к базе данных
- ☐ Применение ADO.NET для низкоуровневого взаимодействия
- ☐ Использование Entity Framework Core для работы с бизнес-объектами
- ☐ Тестирование и отладка приложения

ER-Диаграмма



ер-диаграмма системы продажи авиабилетов (рисунок 1)

Требования 3НФ

- Все значения в таблицах — атомарные (1НФ).
- Все неключевые атрибуты зависят только от полного первичного ключа (2НФ).
- Нет транзитивных зависимостей: все неключи зависят только от ключа и ни от чего другого (3НФ).

Нормализация 1НФ

Tickets_info

Имя столбца	Тип данных	Разрешить ...
userId	int	<input checked="" type="checkbox"/>
fullName	nvarchar(100)	<input checked="" type="checkbox"/>
email	nvarchar(100)	<input checked="" type="checkbox"/>
passportNumber	nvarchar(50)	<input checked="" type="checkbox"/>
role	nvarchar(50)	<input checked="" type="checkbox"/>
ticketId	int	<input checked="" type="checkbox"/>
ticketNumber	nvarchar(50)	<input checked="" type="checkbox"/>
bookingDate	date	<input checked="" type="checkbox"/>
type	nvarchar(50)	<input checked="" type="checkbox"/>
status	nvarchar(50)	<input checked="" type="checkbox"/>
price	decimal(10, 2)	<input checked="" type="checkbox"/>
flightId	int	<input checked="" type="checkbox"/>
flightNumber	nvarchar(50)	<input checked="" type="checkbox"/>
departureAirport	nvarchar(100)	<input checked="" type="checkbox"/>
arrivalAirport	nvarchar(100)	<input checked="" type="checkbox"/>
departureTime	datetime	<input checked="" type="checkbox"/>
arrivalTime	datetime	<input checked="" type="checkbox"/>

демонстрирует исходную структуру до разделения на атомарные значения (рисунок 2)

Нормализация 2НФ

User

Имя столбца	Тип данных	Разрешить ...
userId	int	<input type="checkbox"/>
fullName	nvarchar(100)	<input checked="" type="checkbox"/>
email	nvarchar(100)	<input checked="" type="checkbox"/>
passportNumber	nvarchar(50)	<input checked="" type="checkbox"/>
role	nvarchar(50)	<input checked="" type="checkbox"/>

таблицы, разделённые по сущностям: пользователи, рейсы и билеты, отражающие зависимость атрибутов только от полного ключа (рисунок 3)

Flights

Имя столбца	Тип данных	Разрешить ...
flightId	int	<input type="checkbox"/>
flightNumber	nvarchar(50)	<input checked="" type="checkbox"/>
departureAirport	nvarchar(100)	<input checked="" type="checkbox"/>
arrivalAirport	nvarchar(100)	<input checked="" type="checkbox"/>
departureTime	datetime	<input checked="" type="checkbox"/>
arrivalTime	datetime	<input checked="" type="checkbox"/>

таблицы, разделённые по сущностям: пользователи, рейсы и билеты, отражающие зависимость атрибутов только от полного ключа (рисунок 4)

Tickets

Имя столбца	Тип данных	Разрешить ...
ticketId	int	<input type="checkbox"/>
ticketNumber	nvarchar(50)	<input checked="" type="checkbox"/>
bookingDate	date	<input checked="" type="checkbox"/>
type	nvarchar(50)	<input checked="" type="checkbox"/>
status	nvarchar(50)	<input checked="" type="checkbox"/>
price	decimal(10, 2)	<input checked="" type="checkbox"/>
userId	int	<input checked="" type="checkbox"/>
flightId	int	<input checked="" type="checkbox"/>

таблицы, разделённые по сущностям: пользователи, рейсы и билеты, отражающие зависимость атрибутов только от полного ключа (рисунок 5)

Нормализация 3НФ

Airports

Имя столбца	Тип данных	Разрешить ...
airportId	int	<input type="checkbox"/>
airportName	nvarchar(100)	<input checked="" type="checkbox"/>

окончательная форма нормализованных таблиц, где устранены транзитивные зависимости между полями (рисунок 6)

Flights

Имя столбца	Тип данных	Разрешить ...
flightId	int	<input type="checkbox"/>
flightNumber	nvarchar(50)	<input checked="" type="checkbox"/>
departureAirportId	int	<input checked="" type="checkbox"/>
arrivalAirportId	int	<input checked="" type="checkbox"/>
departureTime	datetime	<input checked="" type="checkbox"/>
arrivalTime	datetime	<input checked="" type="checkbox"/>

окончательная форма нормализованных таблиц, где устранены транзитивные зависимости между полями (рисунок 7)

User

Имя столбца	Тип данных	Разрешить ...
userId	int	<input type="checkbox"/>
fullName	nvarchar(100)	<input checked="" type="checkbox"/>
email	nvarchar(100)	<input checked="" type="checkbox"/>
passportNumber	nvarchar(50)	<input checked="" type="checkbox"/>
role	nvarchar(50)	<input checked="" type="checkbox"/>

окончательная форма нормализованных таблиц, где устранены транзитивные зависимости между полями (рисунок 8)

Схема связей

Связь	Тип	Описание
User - Tickets	1:M	один пользователь может покупать несколько билетов
Flight - Tickets	1:M	один рейс может быть связан с несколькими билетами

Схема связей (таблица 1)

Data Layer (Models / Entities)

Этот слой определяет сущности (модели), которые соответствуют таблицам базы данных.

Сущность	Атрибуты	Связи
Airline	Id, Name	1 → Много Flight
Flight	Id, FlightNumber, Origin, Destination, DepartureTime, ArrivalTime, Price, AvailableSeats	Многие → 1 Airline
Passenger	Id, FullName, Passport, Phone, BirthDate	1 → Много Ticket
Booking	Id, BookingDate, Status	1 → Много Ticket
Ticket	Id, FlightId, PassengerId, BookingId, SeatClass, Price	Связь между Flight, Passenger, Booking

8. Инициализация данных (SeedDataAsync)

При первом запуске создаются:

- 3 авиакомпании
- 6 рейсов

```
if (await context.Airlines.AnyAsync())
    return;

var airlines = new List<Airline>
{
    new() { Name = "Аэрофлот", Code = "SU", Description = "Российская авиакомпания" },
    new() { Name = "S7 Airlines", Code = "S7", Description = "Сибирские авиалинии" },
    new() { Name = "Уральские авиалинии", Code = "U6", Description = "Уральские авиалинии" },
    new() { Name = "Победа", Code = "DP", Description = "Бюджетная авиакомпания" }
};
```

блок кода инициализации данных (рисунок 9)

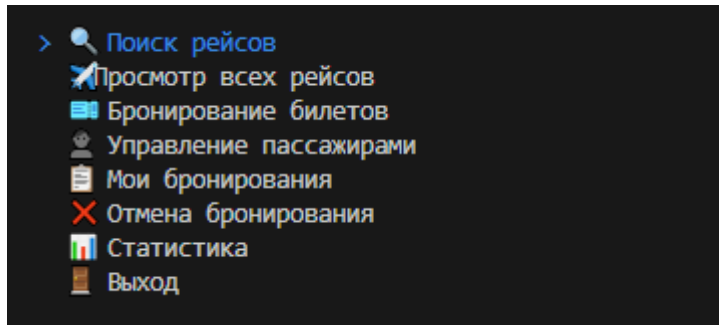
9. Пример жизненного цикла работы программы

1. Пользователь запускает программу.
2. Program создаёт БД и заполняет начальными данными.
3. Отображается главное меню (через Spectre.Console).
4. Пользователь выбирает “ Поиск рейсов”.
5. AirlineTicketApp вызывает метод IFlightService.FindFlights().
6. FlightService обращается к AirlineDbContext и получает список рейсов.
7. Результат показывается пользователю в виде таблицы.
8. Пользователь бронирует билет → создаётся Booking и Ticket.

9. Изменяется количество свободных мест в Flight.

10. Все данные сохраняются в SQLite.

Интерфейс



главное меню программы(рисунок 10)

№	Рейс	Авиакомпания	Откуда	Куда	Время отправления	Время прибытия	Свободных мест	Цена
1	SU123	Аэрофлот	Москва	Санкт-Петербург	02.10.2025 20:16	02.10.2025 22:16	136	5 000,00 Р
2	S7456	S7 Airlines	Москва	Екатеринбург	04.10.2025 00:16	04.10.2025 04:16	120	8 000,00 Р
3	U6789	Уральские авиалинии	Санкт-Петербург	Сочи	04.10.2025 18:16	04.10.2025 22:16	180	12 000,00 Р

Что дальше?

- Обновить список
- Забронировать билет
- Назад

все рейсы(рисунок 11)

Данные пассажира:

Имя (или 'назад' для отмены): 213

Фамилия (или 'назад' для отмены): йу

Номер паспорта (или 'назад' для отмены): 12321313

Email (или 'назад' для отмены): йуццййу

Телефон (или 'назад' для отмены): 123213123123

Дата рождения (dd.MM.yyyy) или 'назад' для отмены: 22.11.2005

✓ Новый пассажир добавлен в систему.

Подтвердить бронирование за 24 000,00 Р? [y/n] (y): н

Please select one of the available options

Подтвердить бронирование за 24 000,00 Р? [y/n] (y): у

✓ Бронирование успешно создано!Номер бронирования: BK202510221205443813Номер билета: TK202510221205441187

Нажмите любую клавишу для продолжения...

■ Бронирование билетов

Данные пассажира:

Имя (или 'назад' для отмены): йцу
 Фамилия (или 'назад' для отмены): йцу
 Номер паспорта (или 'назад' для отмены): 1232131
 Email (или 'назад' для отмены): йцуЙ
 Телефон (или 'назад' для отмены): 1231313
 Дата рождения (dd.MM.yyyy) или 'назад' для отмены: 22.11.2005
☒ Новый пассажир добавлен в систему.

Подтвердить бронирование за 36 000,00 Р? [y/n] (y): y

☒ Бронирование успешно создано!Номер бронирования: BK202510221210078093Номер билета: TK202510221210072761
 Нажмите любую клавишу для продолжения...

бронирование билета(рисунок 12)

Управление пассажирами

Все пассажиры

№	Имя	Фамилия	Паспорт	Email	Телефон	Дата рождения
1	Иван	Петров	1234567890	ivan.petrov@example.com	+7-999-123-45-67	15.05.1990
2	213	йу	12321313	йуццйцйу	123213123123	22.11.2005
3	йцу	йцу	1232131	йцуЙ	1231313	22.11.2005

Нажмите любую клавишу для продолжения...

управление пассажирами(рисунок 13)

Бронирование #2

Номер бронирования: VK202510011018219347
Статус: Подтверждено
Дата бронирования: 01.10.2025 10:18
Общая сумма: 10 000,00 ₽


Билет	Пассажир	Рейс	Класс	Цена
-------	----------	------	-------	------


Бронирование #1


Номер бронирования: VK202510011016412252
Статус: Подтверждено
Дата бронирования: 01.10.2025 10:16
Общая сумма: 10 000,00 ₽

Билет	Пассажир	Рейс	Класс	Цена
-------	----------	------	-------	------

Что дальше?

>  Обновить список

 Отменить бронирование

 Назад

мои бронировки(рисунок 14)

Статистика системы

Показатель	Значение
Всего рейсов	3
Всего пассажиров	3
Всего бронирований	16
- Активных	16
- Отменённых	0
Общая выручка	200 000,00 Р
Всего мест	450
- Занято	16
- Свободно	434
Заполняемость	3,56%

Что дальше?

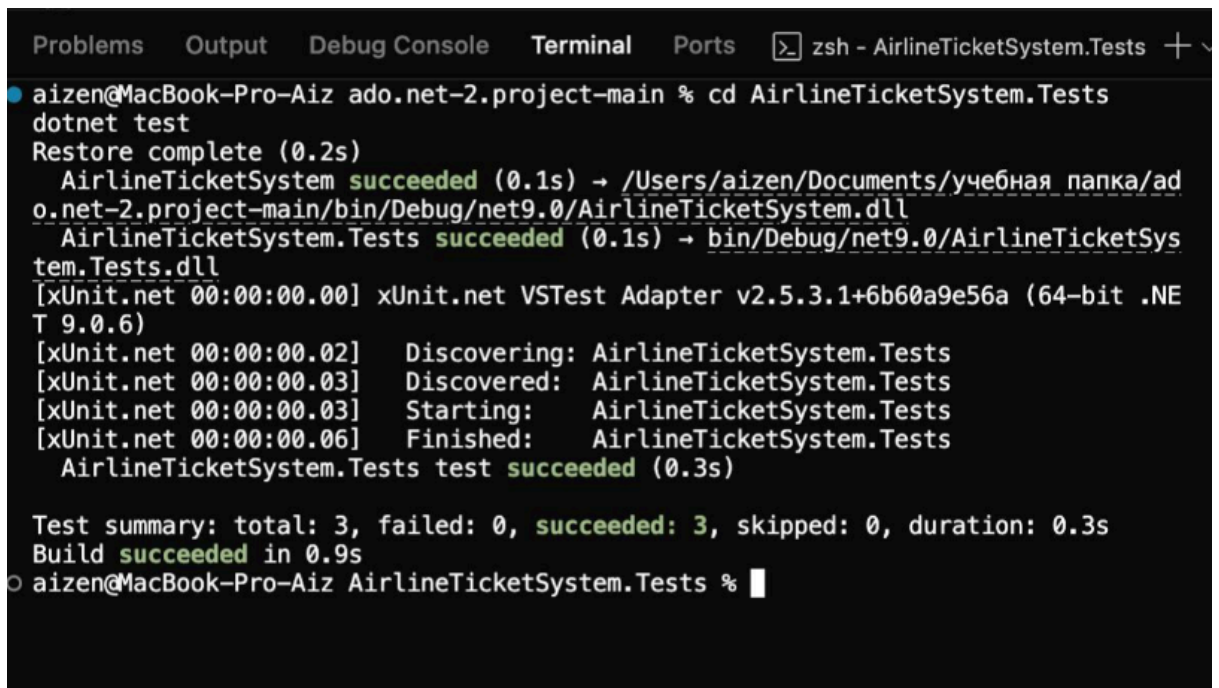
>

Обновить статистику

Назад

статистика(рисунок 15)

Тестирование

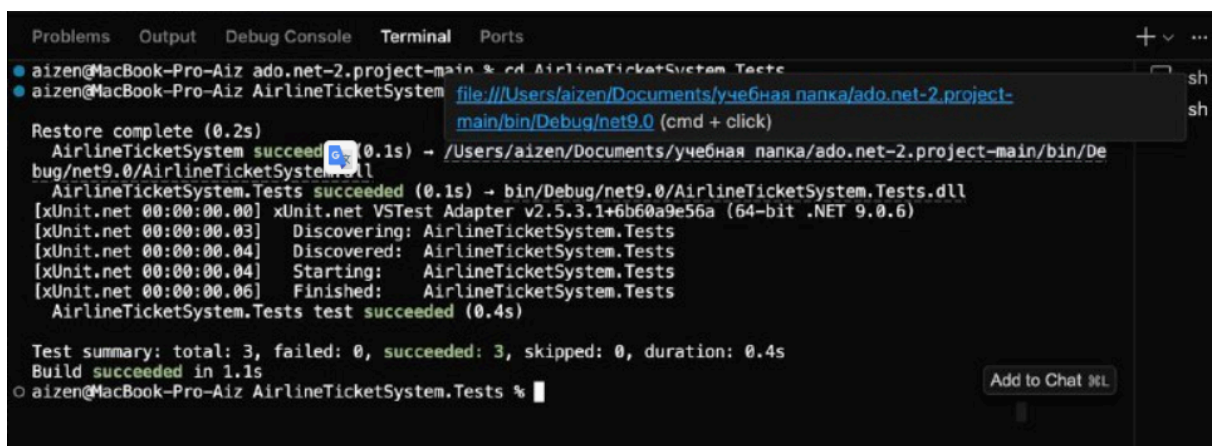


```
Problems Output Debug Console Terminal Ports zsh - AirlineTicketSystem.Tests +
aizen@MacBook-Pro-Aiz ado.net-2.project-main % cd AirlineTicketSystem.Tests
dotnet test
Restore complete (0.2s)
  AirlineTicketSystem succeeded (0.1s) → /Users/aizen/Documents/учебная папка/ado.net-2.project-main/bin/Debug/net9.0/AirlineTicketSystem.dll
  AirlineTicketSystem.Tests succeeded (0.1s) → bin/Debug/net9.0/AirlineTicketSystem.Tests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.5.3.1+6b60a9e56a (64-bit .NET 9.0.6)
[xUnit.net 00:00:00.02]   Discovering: AirlineTicketSystem.Tests
[xUnit.net 00:00:00.03]   Discovered:   AirlineTicketSystem.Tests
[xUnit.net 00:00:00.03]   Starting:    AirlineTicketSystem.Tests
[xUnit.net 00:00:00.06]   Finished:     AirlineTicketSystem.Tests
  AirlineTicketSystem.Tests test succeeded (0.3s)

Test summary: total: 3, failed: 0, succeeded: 3, skipped: 0, duration: 0.3s
Build succeeded in 0.9s
aizen@MacBook-Pro-Aiz AirlineTicketSystem.Tests %
```

стандарт тест (запуск cd AirlineTicketSystem.Tests

dotnet test)(рисунок 16)



```
Problems Output Debug Console Terminal Ports
aizen@MacBook-Pro-Aiz ado.net-2.project-main % cd AirlineTicketSystem.Tests
aizen@MacBook-Pro-Aiz AirlineTicketSystem file:///Users/aizen/Documents/учебная папка/ado.net-2.project-main/bin/Debug/net9.0 (cmd + click)
Restore complete (0.2s)
  AirlineTicketSystem succeeded (0.1s) → /Users/aizen/Documents/учебная папка/ado.net-2.project-main/bin/Debug/net9.0/AirlineTicketSystem.dll
  AirlineTicketSystem.Tests succeeded (0.1s) → bin/Debug/net9.0/AirlineTicketSystem.Tests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.5.3.1+6b60a9e56a (64-bit .NET 9.0.6)
[xUnit.net 00:00:00.03]   Discovering: AirlineTicketSystem.Tests
[xUnit.net 00:00:00.04]   Discovered:   AirlineTicketSystem.Tests
[xUnit.net 00:00:00.04]   Starting:    AirlineTicketSystem.Tests
[xUnit.net 00:00:00.06]   Finished:     AirlineTicketSystem.Tests
  AirlineTicketSystem.Tests test succeeded (0.4s)

Test summary: total: 3, failed: 0, succeeded: 3, skipped: 0, duration: 0.4s
Build succeeded in 1.1s
aizen@MacBook-Pro-Aiz AirlineTicketSystem.Tests %
```

подробный вывод (рисунок 17)

// Запуск приложения

```
var app = new AirlineTicketApp(host.Services);

await app.RunAsync();

}

private static async Task SeedDataAsync(AirlineDbContext
context)
{
    if (await context.Airlines.AnyAsync())
        return;

    var airlines = new List<Airline>
    {
        new() { Name = "Аэрофлот", Code = "SU",
Description = "Российская авиакомпания" },
        new() { Name = "S7 Airlines", Code = "S7",
Description = "Сибирские авиалинии" },
        new() { Name = "Уральские авиалинии", Code =
"U6", Description = "Уральские авиалинии" },
        new() { Name = "Победа", Code = "DP", Description
= "Бюджетная авиакомпания" }
    };
};
```

```

context.Airlines.AddRange(airlines);

await context.SaveChangesAsync();


var flights = new List<Flight>

{

    new()

    {

        FlightNumber = "SU123",

        DepartureCity = "Москва",

        ArrivalCity = "Санкт-Петербург",

        DepartureTime =
DateTime.Now.AddDays(1).AddHours(10),

        ArrivalTime =
DateTime.Now.AddDays(1).AddHours(12),

        TotalSeats = 150,

        AvailableSeats = 150,

        BasePrice = 5000,

        AirlineId = 1

    },

    new()

    {

        FlightNumber = "S7456",

```

```

        DepartureCity = "Москва",

        ArrivalCity = "Екатеринбург",

        DepartureTime =
DateTime.Now.AddDays(2).AddHours(14),

        ArrivalTime =
DateTime.Now.AddDays(2).AddHours(18),

        TotalSeats = 120,

        AvailableSeats = 120,

        BasePrice = 8000,

        AirlineId = 2

    },

    new()

    {

        FlightNumber = "U6789",

        DepartureCity = "Санкт-Петербург",

        ArrivalCity = "Сочи",

        DepartureTime =
DateTime.Now.AddDays(3).AddHours(8),

        ArrivalTime =
DateTime.Now.AddDays(3).AddHours(12),

        TotalSeats = 180,

        AvailableSeats = 180,

        BasePrice = 12000,

        AirlineId = 3

    }

```



```

        };

        context.Flights.AddRange(flights);

        await context.SaveChangesAsync();

    }

}

public class AirlineTicketApp
{
    private readonly IServiceProvider _serviceProvider;

    public AirlineTicketApp(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    public async Task RunAsync()
    {
        AnsiConsole.Write(new
        FigletText("Авиабилеты").Color(Color.Blue));
    }
}

```

```
        AnsiConsole.Write(new Markup("[bold green]Система  
продажи авиабилетов[/]").Centered()));
```

```
        AnsiConsole.WriteLine();
```

```
        while (true)
```

```
        {
```

```
            var choice = AnsiConsole.Prompt(  
                new SelectionPrompt<string>()
```

```
                .Title("[bold blue]Выберите  
действие: [/]")
```

```
                .AddChoices(new[]
```

```
                {
```

```
                    "🔍 Поиск рейсов",
```

```
                    "✈️ Просмотр всех рейсов",
```

```
                    "📅 Бронирование билетов",
```

```
                    "👤 Управление пассажирами",
```

```
                    "📋 Мои бронирования",
```

```
                    "❌ Отмена бронирования",
```

```
                    "📊 Статистика",
```

```
                    "🚪 Выход"
```

```
                }));
```

```
try
{
    switch (choice)
    {
        case "🔍 Поиск рейсов":
            await SearchFlightsAsync();
            break;

        case "✈️ Просмотр всех рейсов":
            await ShowAllFlightsAsync();
            break;

        case "🎫 Бронирование билетов":
            await BookTicketsAsync();
            break;

        case "👤 Управление пассажирами":
            await ManagePassengersAsync();
            break;

        case "📋 Мои бронирования":
            await ShowBookingsAsync();
            break;

        case "❌ Отмена бронирования":
            await CancelBookingAsync();
            break;

        case "📊 Статистика":
```

```

        await ShowStatisticsAsync();

        break;

        case "🔴 Выход":

            AnsiConsole.Write(new Markup("[bold
red]До свидания![/]"));

            return;

        }

    }

    catch (Exception ex)

    {

        AnsiConsole.WriteException(ex);

    }

    AnsiConsole.WriteLine();

    AnsiConsole.Write(new Markup("[dim]Нажмите любую
клавишу для продолжения...[/]"));

    Console.ReadKey();

    AnsiConsole.Clear();

}

}

```

```

private async Task SearchFlightsAsync()

```

```

{

    while (true)

    {

        AnsiConsole.Write(new Markup("[bold blue]🔍 Поиск
рейсов[/]"));

        AnsiConsole.WriteLine();


        var departureCity = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Город
отправления (или 'назад' для выхода):[/]")

                .AllowEmpty());


        if (string.IsNullOrEmpty(departureCity) ||
departureCity.ToLower() == "назад")

        {

            return;

        }


        var arrivalCity = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Город прибытия
(или 'назад' для выхода):[/]")

                .AllowEmpty());

```

```

        if (string.IsNullOrWhiteSpace(arrivalCity) ||
arrivalCity.ToLower() == "назад")

        {

            return;

        }


        var departureDateStr =
AnsiConsole.Ask<string>("[green]Дата отправления (dd.ММ.yyyy) или
Enter для любой:[/]", "");

        DateTime? departureDate = null;

        if (!string.IsNullOrEmpty(departureDateStr))

        {

            if (DateTime.TryParseExact(departureDateStr,
"dd.ММ.yyyy", null, System.Globalization.DateTimeStyles.None, out
var parsedDate))

            {

                departureDate = parsedDate;

            }

        }

```

```

        using var scope = _serviceProvider.CreateScope();

        var flightService =
scope.ServiceProvider.GetRequiredService<IFlightService>();

        var flights = await
flightService.SearchFlightsAsync(departureCity, arrivalCity,
departureDate);

        if (!flights.Any())

        {

            AnsiConsole.Write(new Markup("[red]Рейсы не
найденны.[/]"));

        }

        else

        {

            var table = new Table();

            table.AddColumn("№");

            table.AddColumn("Рейс");

            table.AddColumn("Авиакомпания");

            table.AddColumn("Откуда");

            table.AddColumn("Куда");

            table.AddColumn("Время отправления");

```

```

        table.AddColumn("Время прибытия");

        table.AddColumn("Свободных мест");

        table.AddColumn("Цена");


        for (int i = 0; i < flights.Count; i++)
        {
            var flight = flights[i];

            table.AddRow(

                (i + 1).ToString(),

                flight.FlightNumber,

                flight.Airline.Name,

                flight.DepartureCity,

                flight.ArrivalCity,

                flight.DepartureTime.ToString("dd.MM.yyyy HH:mm"),

                flight.ArrivalTime.ToString("dd.MM.yyyy HH:mm"),

                flight.AvailableSeats.ToString(),

                $"{flight.BasePrice:C}"

            );
        }

```



```

        AnsiConsole.Write(table);
    }

    AnsiConsole.WriteLine();

    var continueSearch =
        AnsiConsole.Confirm("[yellow]Выполнить новый поиск?[/]", false);

    if (!continueSearch)
    {
        return;
    }

    AnsiConsole.Clear();
}

}

private async Task ShowAllFlightsAsync()
{
    while (true)
    {
        AnsiConsole.Write(new Markup("[bold blue]✈️ Все рейсы[/]"));

        AnsiConsole.WriteLine();
    }
}

```

```

        using var scope = _serviceProvider.CreateScope();

        var flightService =
scope.ServiceProvider.GetRequiredService<IFlightService>();

        var flights = await
flightService.GetAllFlightsAsync();

        if (!flights.Any())

        {

            AnsiConsole.Write(new Markup("[red]Рейсы не
найденны.[/]"));

            AnsiConsole.WriteLine();

            AnsiConsole.Write(new Markup("[dim]Нажмите
любую клавишу для возврата...[/]"));

            Console.ReadKey();

            return;

        }

        var table = new Table();

        table.Border = TableBorder.Rounded;

        table.AddColumn("№");

```

```

        table.AddColumn("Рейс");

        table.AddColumn("Авиакомпания");

        table.AddColumn("Откуда");

        table.AddColumn("Куда");

        table.AddColumn("Время отправления");

        table.AddColumn("Время прибытия");

        table.AddColumn("Свободных мест");

        table.AddColumn("Цена");


        for (int i = 0; i < flights.Count; i++)
        {

            var flight = flights[i];

            var seatsColor = flight.AvailableSeats > 50 ?

"green" :

                                flight.AvailableSeats > 10 ?

"yellow" : "red";


            table.AddRow(

                (i + 1).ToString(),

                flight.FlightNumber,

                flight.Airline.Name,

                flight.DepartureCity,

                flight.ArrivalCity,

```

```

                                flight.DepartureTime.ToString("dd.MM.yyyy
HH:mm"),
                                flight.ArrivalTime.ToString("dd.MM.yyyy
HH:mm"),

                                $"[{seatsColor}]{flight.AvailableSeats}[/]",

                                $"{flight.BasePrice:C}"

                                );
    }

```

```

AnsiConsole.Write(table);

AnsiConsole.WriteLine();

```

```

var action = AnsiConsole.Prompt(

    new SelectionPrompt<string>()

        .Title("[green]Что дальше?[/]")

        .AddChoices(new[]

        {

            "🔄 Обновить список",

            "🛒 Забронировать билет",

            "⬅️ BACK Назад"

        })

    ));

```

```

        if (action == "⬅️ Назад")
        {
            return;
        }

        else if (action == "🎫 Забронировать билет")
        {
            AnsiConsole.Clear();

            await BookTicketsAsync();

            return;
        }

        AnsiConsole.Clear();
    }
}

private async Task BookTicketsAsync()
{
    AnsiConsole.Write(new Markup("[bold blue] 🎫  
Бронирование билетов[/]"));

    AnsiConsole.WriteLine();
}

```

```

        using var scope = _serviceProvider.CreateScope();

        var flightService =
scope.ServiceProvider.GetRequiredService<IFlightService>();

        var passengerService =
scope.ServiceProvider.GetRequiredService<IPassengerService>();

        var bookingService =
scope.ServiceProvider.GetRequiredService<IBookingService>();

```

// Выбор рейса

```

        var flights = await
flightService.GetAllFlightsAsync();

        if (!flights.Any())

        {

            AnsiConsole.Write(new Markup("[red]Нет доступных
рейсов.[/]"));

            return;

        }

        var flightChoices = flights.Select(f =>
$"{f.FlightNumber} - {f.DepartureCity} → {f.ArrivalCity}
({f.DepartureTime:dd.MM.yyyy HH:mm})").ToList();

        flightChoices.Add("⬅️ BACK Назад");

        var selectedFlightStr = AnsiConsole.Prompt(

            new SelectionPrompt<string>()

                .Title("[green]Выберите рейс:[/]")

```

```

        .AddChoices(flightChoices));

    if (selectedFlightStr == "⬅️ Назад")
    {
        return;
    }

    var selectedFlight =
    flights[flightChoices.IndexOf(selectedFlightStr)];

// Проверка доступности мест

    if (selectedFlight.AvailableSeats <= 0)
    {
        AnsiConsole.Write(new Markup("[red]На выбранном
рейсе нет свободных мест.[/]"));

        return;
    }

// Ввод данных пассажира

    AnsiConsole.WriteLine();

    AnsiConsole.Write(new Markup("[bold]Данные
пассажира:[/]"));

    AnsiConsole.WriteLine();

```

```

        var firstName = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Имя (или 'назад'
для отмены): [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(firstName) ||
firstName.ToLower() == "назад")

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено. [/]"));

            return;

        }

        var lastName = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Фамилия (или
'назад' для отмены): [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(lastName) ||
lastName.ToLower() == "назад")

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено. [/]"));

            return;

        }

```



```

        var passportNumber = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Номер паспорта
(или 'назад' для отмены): [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(passportNumber) ||
passportNumber.ToLower() == "назад")

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено. [/]"));

            return;

        }

        var email = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Email (или 'назад'
для отмены): [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(email) ||
email.ToLower() == "назад")

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено. [/]"));

            return;

        }

```

```

        var phoneNumber = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Телефон (или
'назад' для отмены): [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(phoneNumber) ||
phoneNumber.ToLower() == "назад")

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено. [/]"));

            return;

        }

        var dateOfBirthStr = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Дата рождения
(dd.мм.yyyy) или 'назад' для отмены: [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(dateOfBirthStr) ||
dateOfBirthStr.ToLower() == "назад")

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено. [/]"));

            return;

        }

```

```

        if (!DateTime.TryParseExact(dateOfBirthStr,
            "dd.MM.yyyy", null, System.Globalization.DateTimeStyles.None, out
            var dateOfBirth))
        {
            AnsiConsole.Write(new Markup("[red]Неверный
            формат даты. Бронирование отменено.[/]"));

            return;
        }

```

// Поиск или создание пассажира

```

        var passenger = await
        passengerService.GetPassengerByPassportAsync(passportNumber);

        if (passenger == null)
        {
            passenger = new Passenger
            {
                FirstName = firstName,
                LastName = lastName,
                PassportNumber = passportNumber,
                Email = email,
                PhoneNumber = phoneNumber,
                DateOfBirth = dateOfBirth
            };

            passenger = await
            passengerService.CreatePassengerAsync(passenger);

```

```

        AnsiConsole.Write(new Markup("[green]✓ Новый
пассажир добавлен в систему.[/]"));

        AnsiConsole.WriteLine();

    }

    else

    {

        AnsiConsole.Write(new Markup("[green]✓ Пассажир
найден в системе.[/]"));

        AnsiConsole.WriteLine();

    }

```

// Выбор класса

```

var classChoices = new List<string>

{

    "Economy",

    "Business",

    "First",

    "⬅️ BACK Назад"

};

var selectedClass = AnsiConsole.Prompt(

    new SelectionPrompt<string>()

        .Title("[green]Выберите класс:[/]")

        .AddChoices(classChoices));

```

```

        if (selectedClass == "🔙 Назад")
        {
            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено.[/]"));

            return;
        }

        var ticketClass =
Enum.Parse<TicketClass>(selectedClass);

// Расчет цены

        var priceMultiplier = ticketClass switch
        {

            TicketClass.Economy => 1.0m,

            TicketClass.Business => 2.0m,

            TicketClass.First => 3.0m,

            _ => 1.0m

        };

        var ticketPrice = selectedFlight.BasePrice *
priceMultiplier;

// Подтверждение бронирования

```

```

        AnsiConsole.WriteLine();

        var confirm =
AnsiConsole.Confirm($"[green]Подтвердить бронирование за
{ticketPrice:C}?[/]");

        if (!confirm)

        {

            AnsiConsole.Write(new
Markup("[yellow]Бронирование отменено.[/]"));

            return;

        }

```

// Создание билета и бронирования

```

var ticket = new Ticket

{

    FlightId = selectedFlight.Id,

    PassengerId = passenger.Id,

    Price = ticketPrice,

    Class = ticketClass,

    Status = TicketStatus.Active,

    BookingDate = DateTime.Now

};

var booking = await
bookingService.CreateBookingAsync(new List<Ticket> { ticket });

```

// Обновление доступных мест

```
        await
flightService.UpdateAvailableSeatsAsync(selectedFlight.Id, 1);

        AnsiConsole.WriteLine();

        AnsiConsole.Write(new Markup($"[bold green]✅
Бронирование успешно создано![/]"));

        AnsiConsole.Write(new Markup($"[green]Номер
бронирования: {booking.BookingNumber}[/]"));

        AnsiConsole.Write(new Markup($"[green]Номер билета:
{ticket.TicketNumber}[/]"));

    }

    private async Task ManagePassengersAsync()

    {

        while (true)

        {

            AnsiConsole.Write(new Markup("[bold blue]👤
Управление пассажирами[/]"));

            AnsiConsole.WriteLine();

            using var scope = _serviceProvider.CreateScope();

            var passengerService =
scope.ServiceProvider.GetRequiredService<IPassengerService>();

            var action = AnsiConsole.Prompt(
```

```

new SelectionPrompt<string>()

    .Title("[green]Выберите действие:[/]")

    .AddChoices(new[]

        {

            "📋 Просмотр всех пассажиров",

            "✎ Редактирование пассажира",

            "⬅️ BACK Назад"

        }

    ));

if (action == "⬅️ BACK Назад")

{

    return;

}

if (action == "📋 Просмотр всех пассажиров")

{

    await ShowAllPassengersAsync();

}

else if (action == "✎ Редактирование пассажира")

{

    await EditPassengerAsync();

}

```



```

        AnsiConsole.WriteLine();

        AnsiConsole.Write(new Markup("[dim]Нажмите любую
клавишу для продолжения...[/]"));

        Console.ReadKey();

        AnsiConsole.Clear();

    }

}

private async Task ShowAllPassengersAsync()

{

    AnsiConsole.Write(new Markup("[bold blue]📋 Все
пассажиры[/]"));

    AnsiConsole.WriteLine();

    using var scope = _serviceProvider.CreateScope();

    var passengerService =
scope.ServiceProvider.GetRequiredService<IPassengerService>();

    var passengers = await
passengerService.GetAllPassengersAsync();

    if (!passengers.Any())

    {

        AnsiConsole.Write(new Markup("[red]Пассажиры не
найжены.[/]"));

```

```

        return;
    }

    var table = new Table();

    table.AddColumn("№");

    table.AddColumn("Имя");

    table.AddColumn("Фамилия");

    table.AddColumn("Паспорт");

    table.AddColumn("Email");

    table.AddColumn("Телефон");

    table.AddColumn("Дата рождения");

    for (int i = 0; i < passengers.Count; i++)
    {
        var passenger = passengers[i];

        table.AddRow(

            (i + 1).ToString(),

            passenger.FirstName,

            passenger.LastName,

            passenger.PassportNumber,

            passenger.Email ?? "",

            passenger.PhoneNumber ?? "",

            passenger.DateOfBirth.ToString("dd.MM.yyyy")
        );
    }
}

```

```

        );
    }

    AnsiConsole.Write(table);
}

private async Task EditPassengerAsync()
{
    AnsiConsole.Write(new Markup("[bold blue]✎  

Редактирование пассажира[/]"));

    AnsiConsole.WriteLine();

    using var scope = _serviceProvider.CreateScope();

    var passengerService =
scope.ServiceProvider.GetRequiredService<IPassengerService>();

    var passengers = await
passengerService.GetAllPassengersAsync();

    if (!passengers.Any())
    {
        AnsiConsole.Write(new Markup("[red]Пассажиры не  

найденны.[/]"));

        return;
    }
}

```

```
// Добавляем опцию "Назад" в список  
выбора
```

```
var passengerChoices = passengers.Select(p =>  
    $"{p.FirstName} {p.LastName} -  
{p.PassportNumber}").ToList();  
  
passengerChoices.Add("⬅️ Назад");  
  
var selectedPassengerStr = AnsiConsole.Prompt(  
    new SelectionPrompt<string>()  
        .Title("[green]Выберите пассажира для  
редактирования: [/]")  
        .AddChoices(passengerChoices));  
  
if (selectedPassengerStr == "⬅️ Назад")  
{  
    return;  
}  
  
var selectedPassenger =  
passengers[passengerChoices.IndexOf(selectedPassengerStr)];  
  
// Показываем текущие данные  
  
var panel = new Panel(  

```

```

        $"{bold}Имя:[/] {selectedPassenger.FirstName}\n"
+
        $"{bold}Фамилия:[/]
{selectedPassenger.LastName}\n" +
        $"{bold}Паспорт:[/]
{selectedPassenger.PassportNumber}\n" +
        $"{bold}Email:[/] {selectedPassenger.Email ?? "не
указан"} [dim](не редактируется)[/]\n" +
        $"{bold}Телефон:[/]
{selectedPassenger.PhoneNumber ?? "не указан"}\n" +
        $"{bold}Дата рождения:[/]
{selectedPassenger.DateOfBirth:dd.MM.yyyy}")

    {
        Header = new PanelHeader($"{bold blue}Текущие
данные пассажира[/]")

    };

    AnsiConsole.Write(panel);

    AnsiConsole.WriteLine();

// Запрашиваем новые данные

    AnsiConsole.Write(new Markup($"{bold}Введите новые
данные (нажмите Enter, чтобы оставить текущее значение):[/]"));

    AnsiConsole.WriteLine();

```

```

        var firstName =
AnsiConsole.Ask<string>("[green]Имя: [/]",
selectedPassenger.FirstName);

        var lastName =
AnsiConsole.Ask<string>("[green]Фамилия: [/]",
selectedPassenger.LastName);

        var passportNumber =
AnsiConsole.Ask<string>("[green]Номер паспорта: [/]",
selectedPassenger.PassportNumber);

        var phoneNumber =
AnsiConsole.Ask<string>("[green]Телефон: [/]",
selectedPassenger.PhoneNumber ?? "");

        var dateOfBirthStr = AnsiConsole.Ask<string>(
            "[green]Дата рождения (dd.MM.yyyy): [/]",
            selectedPassenger.DateOfBirth.ToString("dd.MM.yyyy"));

        DateTime dateOfBirth = selectedPassenger.DateOfBirth;

        if (!string.IsNullOrEmpty(dateOfBirthStr) &&
dateOfBirthStr !=
selectedPassenger.DateOfBirth.ToString("dd.MM.yyyy"))
        {
            if (!DateTime.TryParseExact(dateOfBirthStr,
"dd.MM.yyyy", null, System.Globalization.DateTimeStyles.None, out
dateOfBirth))
            {
                AnsiConsole.Write(new Markup("[red]Неверный
формат даты. Изменения не применены. [/]"));
            }
        }
    }
}

```

```

        return;
    }
}

// Подтверждение изменений

var confirm = AnsiConsole.Confirm("[yellow]Сохранить
изменения?[/]");

if (!confirm)

{

    AnsiConsole.Write(new Markup("[yellow]Изменения
отменены.[/]"));

    return;

}

```

```

// Обновляем данные

selectedPassenger.FirstName = firstName;

selectedPassenger.LastName = lastName;

selectedPassenger.PassportNumber = passportNumber;

selectedPassenger.PhoneNumber =
string.IsNullOrWhiteSpace(phoneNumber) ? null : phoneNumber;

selectedPassenger.DateOfBirth = dateOfBirth;


var success = await
passengerService.UpdatePassengerAsync(selectedPassenger);

```

```

        if (success)
        {
            AnsiConsole.Write(new Markup("[bold green]✅
Данные пассажира успешно обновлены![/]"));
        }
        else
        {
            AnsiConsole.Write(new Markup("[red]Ошибка при
обновлении данных пассажира.[/]"));
        }
    }

    private async Task ShowBookingsAsync()
    {
        while (true)
        {
            AnsiConsole.Write(new Markup("[bold blue]📋 Мои
бронирования[/]"));

            AnsiConsole.WriteLine();

            using var scope = _serviceProvider.CreateScope();

            var bookingService =
scope.ServiceProvider.GetRequiredService<IBookingService>();

```



```

        var bookings = await
bookingService.GetAllBookingsAsync();

        if (!bookings.Any())

        {

            AnsiConsole.Write(new
Markup("[red]Бронирования не найдены.[/]"));

            AnsiConsole.WriteLine();

            AnsiConsole.Write(new Markup("[dim]Нажмите
любую клавишу для возврата...[/]"));

            Console.ReadKey();

            return;

        }

        foreach (var booking in bookings)

        {

            var statusColor = booking.Status ==
BookingStatus.Confirmed ? "green" :

                                booking.Status ==
BookingStatus.Cancelled ? "red" : "blue";

            var panel = new Panel($"[bold]Номер
бронирования:[/] {booking.BookingNumber}\n" +

                                $"[bold]Статус:[/]
{GetStatusText(booking.Status)}\n" +

                                $"[bold]Дата
бронирования:[/] {booking.BookingDate:dd.MM.yyyy HH:mm}\n" +

```

```

                                $"[bold]Общая сумма:[/]
{booking.TotalAmount:C}")

                                {

                                    Header = new PanelHeader($"[bold
{statusColor}]Бронирование #{booking.Id}[/]",

                                    Border = BoxBorder.Rounded

                                };

                                AnsiConsole.Write(panel);

                                var table = new Table();

                                table.Border = TableBorder.Rounded;

                                table.AddColumn("Билет");

                                table.AddColumn("Пассажир");

                                table.AddColumn("Рейс");

                                table.AddColumn("Класс");

                                table.AddColumn("Цена");

                                foreach (var ticket in booking.Tickets)

                                {

                                    table.AddRow(

                                        ticket.TicketNumber,

                                        $" {ticket.Passenger.FirstName}
{ticket.Passenger.LastName}",

```

```

                $"{ticket.Flight.FlightNumber}
({ticket.Flight.DepartureCity} → {ticket.Flight.ArrivalCity})",

                ticket.Class.ToString(),

                $"{ticket.Price:C}"

            );
        }

```

```

        AnsiConsole.Write(table);

        AnsiConsole.WriteLine();
    }

```

```

var action = AnsiConsole.Prompt(

    new SelectionPrompt<string>()

        .Title("[green]Что дальше?[/]")

        .AddChoices(new[]

            {

                "🔄 Обновить список",

                "❌ Отменить бронирование",

                "⬅️ BACK Назад"

            }

        ));

```

```

if (action == "⬅️ BACK Назад")

{

    return;
}

```

```

    }

    else if (action == "✗ Отменить бронирование")
    {

        AnsiConsole.Clear();

        await CancelBookingAsync();

        return;

    }

    AnsiConsole.Clear();

}

}

private async Task CancelBookingAsync()
{

    while (true)
    {

        AnsiConsole.Write(new Markup("[bold blue]✗  
Отмена бронирования[/]")));

        AnsiConsole.WriteLine();

        using var scope = _serviceProvider.CreateScope();

        var bookingService =
scope.ServiceProvider.GetRequiredService<IBookingService>();

```

```

        var bookingNumber = AnsiConsole.Prompt(

            new TextPrompt<string>("[green]Введите номер
бронирования (или 'назад' для выхода): [/]")

                .AllowEmpty());

        if (string.IsNullOrEmpty(bookingNumber) ||
bookingNumber.ToLower() == "назад")

        {

            return;

        }

        var booking = await
bookingService.GetBookingByNumberAsync(bookingNumber);

        if (booking == null)

        {

            AnsiConsole.Write(new
Markup("[red]Бронирование не найдено. [/]"));

            AnsiConsole.WriteLine();

            var retry =
AnsiConsole.Confirm("[yellow]Попробовать снова? [/]");

            if (!retry)

            {

                return;

            }

            AnsiConsole.Clear();

```

```

        continue;
    }

    if (booking.Status == BookingStatus.Cancelled)
    {
        AnsiConsole.Write(new
Markup("[red]Бронирование уже отменено.[/]"));

        AnsiConsole.WriteLine();

        var retry =
AnsiConsole.Confirm("[yellow]Попробовать с другим номером?[/]");

        if (!retry)
        {
            return;
        }

        AnsiConsole.Clear();

        continue;
    }

```

// Показываем детали бронирования

```

var panel = new Panel(

    $"[bold]Номер бронирования:[/]"
{booking.BookingNumber}\n" +

    $"[bold]Статус:[/]"
{GetStatusText(booking.Status)}\n" +

```

```

        $"[bold]Дата бронирования:[/]
{booking.BookingDate:dd.MM.yyyy HH:mm}\n" +

        $"[bold]Общая сумма:[/]
{booking.TotalAmount:C}\n" +

        $"[bold]Количество билетов:[/]
{booking.Tickets.Count}")

    {

        Header = new PanelHeader($"[bold blue]Детали
бронирования[/]")

    };

    AnsiConsole.Write(panel);

    AnsiConsole.WriteLine();

    var confirm = AnsiConsole.Confirm($"[red]Вы
уверены, что хотите отменить это бронирование?[/]");

    if (!confirm)

    {

        AnsiConsole.Write(new Markup($"[yellow]Отмена
бронирования отменена.[/]"));

        return;

    }

    var success = await
bookingService.CancelBookingAsync(booking.Id);

    if (success)

```

```

{

    // Восстанавливаем места на рейсах

    var flightService =
scope.ServiceProvider.GetRequiredService<IFlightService>();

    foreach (var ticket in booking.Tickets)

    {

        var flight = await
flightService.GetFlightByIdAsync(ticket.FlightId);

        if (flight != null)

        {

            flight.AvailableSeats++;

            var context =
scope.ServiceProvider.GetRequiredService<AirlineDbContext>();

            await context.SaveChangesAsync();

        }

    }

    AnsiConsole.Write(new Markup("[bold green]✅
Бронирование успешно отменено![/]"));

}

else

{

    AnsiConsole.Write(new Markup("[red]Ошибка при
отмене бронирования.[/]"));

}

```



```

        AnsiConsole.WriteLine();

        var another =
AnsiConsole.Confirm("[yellow]Отменить еще одно бронирование?[/]",
false);

        if (!another)

        {

            return;

        }

        AnsiConsole.Clear();

    }

}

private async Task ShowStatisticsAsync()

{

    while (true)

    {

        AnsiConsole.Write(new Markup("[bold blue]📊
Статистика системы[/]"));

        AnsiConsole.WriteLine();

        using var scope = _serviceProvider.CreateScope();

        var context =
scope.ServiceProvider.GetRequiredService<AirlineDbContext>();

```

```

        var totalFlights = await
context.Flights.CountAsync();

        var totalPassengers = await
context.Passengers.CountAsync();

        var totalBookings = await
context.Bookings.CountAsync();

        var activeBookings = await
context.Bookings.CountAsync(b => b.Status ==
BookingStatus.Confirmed);

        var cancelledBookings = await
context.Bookings.CountAsync(b => b.Status ==
BookingStatus.Cancelled);

        var totalRevenue = await context.Bookings

            .Where(b => b.Status !=
BookingStatus.Cancelled)

            .SumAsync(b => b.TotalAmount);

        var totalSeats = await context.Flights.SumAsync(f
=> f.TotalSeats);

        var availableSeats = await
context.Flights.SumAsync(f => f.AvailableSeats);

        var occupiedSeats = totalSeats - availableSeats;

        var occupancyRate = totalSeats > 0 ?
(double)occupiedSeats / totalSeats * 100 : 0;


        var stats = new Table();

        stats.Border = TableBorder.Rounded;

        stats.AddColumn(new
TableColumn("Показатель").Centered());

```

```

        stats.AddColumn(new
TableColumn("Значение").Centered());

        stats.AddRow("[bold]Всего рейсов[/]",
$" [green]{totalFlights}[/]");

        stats.AddRow("[bold]Всего пассажиров[/]",
$" [green]{totalPassengers}[/]");

        stats.AddRow("[bold]Всего бронирований[/]",
$" [green]{totalBookings}[/]");

        stats.AddRow("    - Активных",
$" [green]{activeBookings}[/]");

        stats.AddRow("    - Отменённых",
$" [red]{cancelledBookings}[/]");

        stats.AddRow("[bold]Общая выручка[/]",
$" [blue]{totalRevenue:C}[/]");

        stats.AddRow("[bold]Всего мест[/]",
$" [yellow]{totalSeats}[/]");

        stats.AddRow("    - Занято",
$" [red]{occupiedSeats}[/]");

        stats.AddRow("    - Свободно",
$" [green]{availableSeats}[/]");

        stats.AddRow("[bold]Заполняемость[/]",
$" [cyan]{occupancyRate:F2}%[/]");

        AnsiConsole.Write(stats);

        AnsiConsole.WriteLine();

```

// Топ популярных направлений

```

        var topRoutes = await context.Tickets

            .Include(t => t.Flight)

            .Where(t => t.Status !=
TicketStatus.Cancelled)

            .GroupBy(t => new { t.Flight.DepartureCity,
t.Flight.ArrivalCity })

            .Select(g => new

                {

                    Route = $"{g.Key.DepartureCity} →
{g.Key.ArrivalCity}",

                    Count = g.Count(),

                    Revenue = g.Sum(t => t.Price)

                })

            .OrderByDescending(r => r.Count)

            .Take(5)

            .ToListAsync();

        if (topRoutes.Any())

        {

            AnsiConsole.Write(new Markup("[bold blue] 🔥
Топ-5 популярных направлений: [/]" ));

            AnsiConsole.WriteLine();

            var routesTable = new Table();

            routesTable.Border = TableBorder.Rounded;

```

```

routesTable.AddColumn("Место");

routesTable.AddColumn("Направление");

routesTable.AddColumn("Билетов продано");

routesTable.AddColumn("Выручка");


for (int i = 0; i < topRoutes.Count; i++)

{

    var medal = i == 0 ? "🏆" : i == 1 ? "🥈"
: i == 2 ? "🥉" : $"{i + 1}.";

    routesTable.AddRow(

        medal,

        topRoutes[i].Route,

        topRoutes[i].Count.ToString(),

        $"{topRoutes[i].Revenue:C}"

    );

}


AnsiConsole.Write(routesTable);

AnsiConsole.WriteLine();

}


var action = AnsiConsole.Prompt(

    new SelectionPrompt<string>()

        .Title("[green]Что дальше?[/]")

```

```

        .AddChoices(new[]
        {
            "🔄 Обновить статистику",
            "⬅️ Назад"
        }));

    if (action == "⬅️ Назад")
    {
        return;
    }

    AnsiConsole.Clear();
}

private string GetStatusText(BookingStatus status)
{
    return status switch
    {
        BookingStatus.Confirmed =>
            "[green]Подтверждено[/]",
        BookingStatus.Cancelled => "[red]Отменено[/]",
        BookingStatus.Completed => "[blue]Завершено[/]",
        _ => status.ToString()
    };
}

```

```
};  
  
}  
  
}  
  
}
```