

**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**

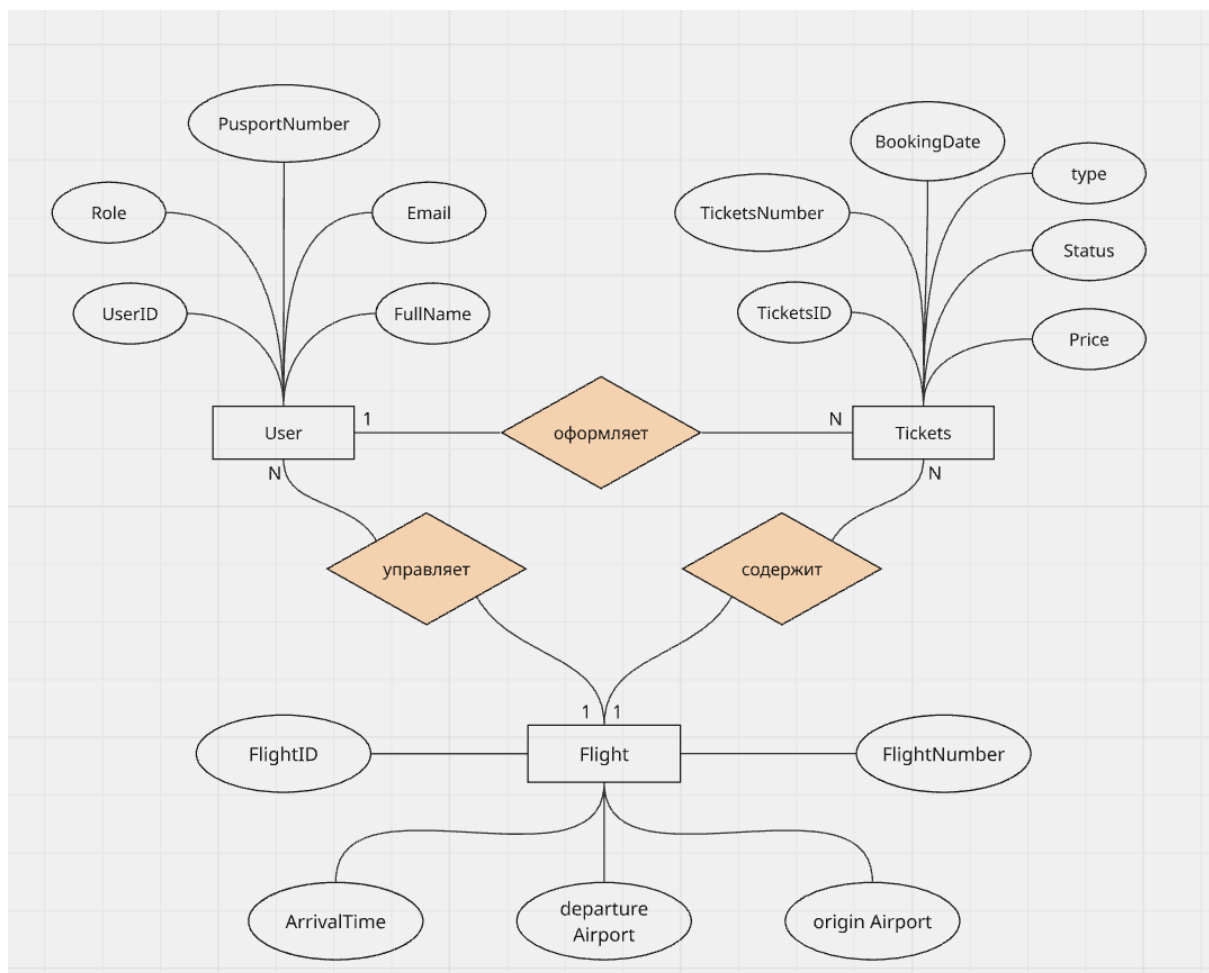
1. Разработать консольную информационную систему для автоматизации процесса продажи авиабилетов.
2. Обеспечить регистрацию рейсов, управление местами и пассажирами, оформление и возврат билетов.
3. Продемонстрировать применение актуальных технологий: Node.js для серверной части и интеграций, C# с ADO.NET 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



er-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)

## 1. Program.cs

### 1.1. Настройка культуры

```
var culture = new CultureInfo("ru-RU");
CultureInfo.CurrentCulture = culture;
CultureInfo.CurrentUICulture = culture;
Thread.CurrentThread.CurrentCulture = culture;
Thread.CurrentThread.CurrentUICulture = culture;
```

настройка культуры (рисунок 9)

Устанавливает русскую локаль (чтобы цены и даты отображались в рублях и привычном формате)

### 1.2. Настройка Dependency Injection и логирования

```
var host = Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging =>
    {
        logging.ClearProviders(); // Отключаем все логирование
        logging.SetMinimumLevel(LogLevel.Warning); // Показываем только предупреждения и ошибки
    })
    .ConfigureServices((context, services) =>
    {
        services.AddDbContext<AirlineDbContext>(options =>
        {
            options.UseSqlite("Data Source=airline.db")
                .EnableSensitiveDataLogging(false)
                .LogTo(Console.WriteLine, LogLevel.Warning); // Логируем только предупреждения
        });

        services.AddScoped<IAirlineService, AirlineService>();
        services.AddScoped<IFlightService, FlightService>();
        services.AddScoped<IPassengerService, PassengerService>();
        services.AddScoped<IBookingService, BookingService>();
    })
    .Build();
```

логирование (рисунок 10)

### Что делает:

- Создаёт DI-контейнер (Host) — стандарт для приложений .NET.
- Настраивает логирование (показывает только предупреждения и ошибки).
- Регистрирует сервисы и базу данных (EF Core, SQLite).
- Подключает слои логики (через интерфейсы).

## 1.3. Создание и инициализация базы

```
using (var scope = host.Services.CreateScope())  
{  
    var context = scope.ServiceProvider.GetRequiredService<AirlineDbContext>();  
    await context.Database.EnsureCreatedAsync();  
    await SeedDataAsync(context);  
}
```

создание и инициализация бд (рисунок 11)

### Что делает:

- Создаёт новую область (scope) для получения зависимостей.
- Проверяет наличие базы данных и создаёт её, если нет.
- Вызывает метод SeedDataAsync — добавляет стартовые данные (авиакомпания и рейсы).



## 1.4. Запуск основного приложения

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

запуск (рисунок 12)

Запускает основной цикл консольного приложения.

## 1.5. SeedDataAsync

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

добавление данных в бд (рисунок 13)

**Добавляет начальные данные в базу:**

- Авиакомпании (Airlines)
- Рейсы (Flights)

Если данные уже есть, ничего не добавляется.

## 2. AirlineTicketApp

Класс управляет логикой консольного интерфейса (взаимодействие с пользователем).

### 2.1. Конструктор

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

конструктор (рисунок 14)

Сохраняет контейнер сервисов для последующего создания областей (scope).

## 2.2. Основной цикл программы

```
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;
            }
        }
    }
}
```

основное меню программы(рисунок 15)

### Что делает:

- Отображает главное меню.
- Ждёт выбор пользователя.
- В зависимости от выбора вызывает соответствующий метод:

Поиск рейсов — SearchFlightsAsync()

Все рейсы — ShowAllFlightsAsync()

Бронирование — BookTicketsAsync()

Пассажиры — ManagePassengersAsync()

Мои бронирования — ShowBookingsAsync()

Отмена — CancelBookingAsync()

Статистика — ShowStatisticsAsync()

Выход — завершает программу

### **3. Основные методы интерфейса**

#### **3.1. SearchFlightsAsync()**

- Спрашивает у пользователя:
  - Город отправления / прибытия
  - Дату вылета
- Ищет подходящие рейсы через IFlightService
- Отображает таблицу с результатами поиска.

#### **3.2. ShowAllFlightsAsync()**

- Показывает таблицу всех рейсов.
- Позволяет:
  - Обновить список
  - Перейти к бронированию билета
  - Вернуться назад

### **3.3. BookTicketsAsync()**

- Выбирает рейс
- Вводит данные пассажира
- Выбирает класс обслуживания
- Рассчитывает цену
- Создаёт билет и бронирование через IBookingService
- Обновляет количество свободных мест в рейсе

### **3.4. ManagePassengersAsync()**

Позволяет:

- Просмотреть всех пассажиров
- Отредактировать данные конкретного пассажира

### **3.5. ShowAllPassengersAsync()**

- Выводит таблицу всех пассажиров с их паспортами, телефонами, датами рождения и т.д.

### **3.6. EditPassengerAsync()**

- Позволяет выбрать пассажира из списка.
- Изменить его данные.
- Сохраняет изменения через IPassengerService.

### 3.7. ShowBookingsAsync()

- Показывает список всех бронирований с билетами, пассажирами и рейсами.
- Можно обновить список или отменить бронирование.

### 3.8. CancelBookingAsync()

- Позволяет ввести номер бронирования.
- Проверяет его статус.
- При подтверждении отменяет и возвращает места в рейсах.

### 3.9. ShowStatisticsAsync()

- (Часть не показана, но очевидно)
- Отображает общую статистику:
  - Кол-во рейсов, пассажиров, бронирований
  - Доходы и загрузку рейсов

## 4. Presentation Layer (UI)

**Класс:** AirlineTicketApp

**Файл:** AirlineTicketApp.cs

Этот слой отвечает за взаимодействие с пользователем (через консольное меню).

Он не содержит бизнес-логики — только вызывает методы сервисов.

**Функции слоя:**

- Отображение интерфейса (через Spectre.Console).
- Ввод данных пользователем.
- Вызов сервисов для обработки действий (поиск рейсов, бронирование и т.д.).
- Отображение результатов и ошибок.

### **Примеры классов и методов:**

- RunAsync() — главный цикл программы, показывает меню.
- SearchFlightsAsync() — поиск рейсов по городу и дате.
- BookTicketsAsync() — создание бронирования.
- ShowBookingsAsync() — просмотр всех бронирований.

### **Используемые библиотеки:**

Spectre.Console — для цветного интерфейса, таблиц и диалогов.

## **5. Business Logic Layer (BLL)**

Слой бизнес-логики отвечает за реализацию всех правил и операций приложения.

Он взаимодействует с базой данных через DbContext, но не напрямую из интерфейса — всё идёт через интерфейсы сервисов.

### **Основная идея:**

UI ничего не знает о базе данных, он вызывает методы вроде FindFlights() или BookTicket(), а уже сервисы выполняют логику и обращаются к БД.

## Примеры логики внутри сервисов

### FlightService

- Проверяет наличие рейсов по маршруту и дате.
- Возвращает список подходящих рейсов.
- Обновляет количество мест при бронировании.

### BookingService

- Проверяет наличие свободных мест.
- Создаёт запись бронирования (Booking).
- Создаёт билеты (Ticket) для каждого пассажира.
- При отмене бронирования — возвращает места на рейс.

### PassengerService

- Добавляет нового пассажира.
- Проверяет корректность паспорта.
- Позволяет редактировать или удалять пассажира.

## 6. Data Access Layer (DAL)

**Класс:** AirlineDbContext

**Файл:** AirlineDbContext.cs

Этот слой отвечает за взаимодействие с базой данных через **Entity Framework Core**.

### Основные функции:

- Определяет таблицы (через DbSet<>).
- Настраивает связи между таблицами (One-to-Many, Many-to-Many).
- Обеспечивает миграции и инициализацию БД.

```
public class AirlineDbContext : DbContext
{
    public AirlineDbContext(DbContextOptions<AirlineDbContext> options) : base(options)
    {
    }

    public DbSet<Airline> Airlines { get; set; }
    public DbSet<Flight> Flights { get; set; }
    public DbSet<Passenger> Passengers { get; set; }
    public DbSet<Ticket> Tickets { get; set; }
    public DbSet<Booking> Bookings { get; set; }
}
```

функции для работы с таблицами (рисунок 16)

**База данных:** SQLite

### Подключение:

```
options.UseSqlite("Data Source=airline.db")
```

подключение к дб (рисунок 17)



## 7. 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" }
};

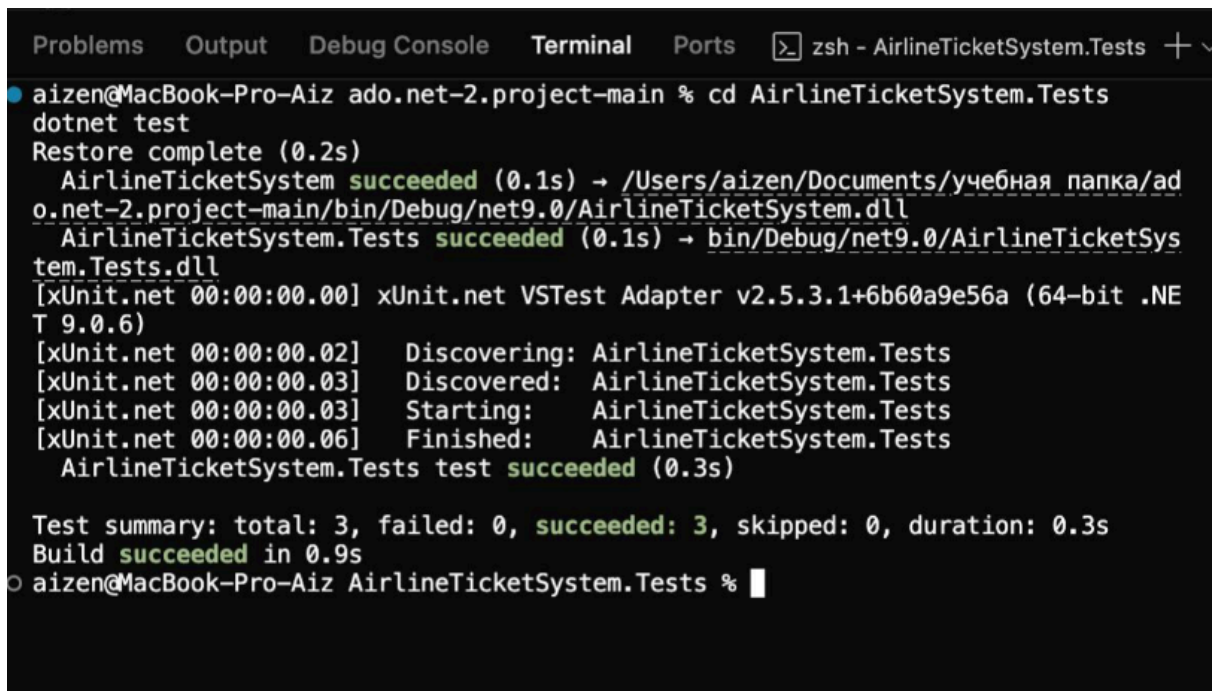
```

блок кода инициализации данных (рисунок 18)

## 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.

## Тестирование

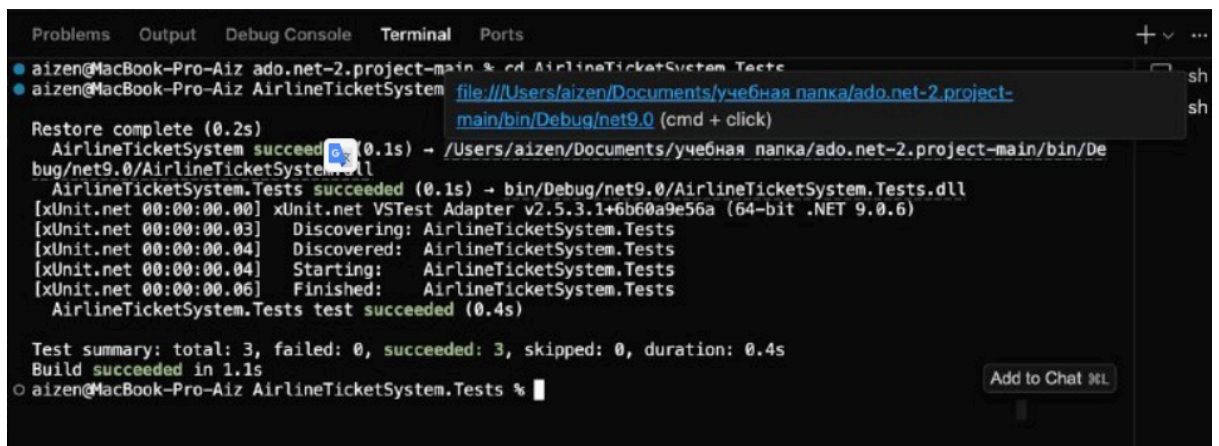


```
Problems Output Debug Console Terminal Ports zsh - AirlineTicketSystem.Tests +
aizen@MacBook-Pro-Aiz ado.net-2.project-main % cd AirlineTicketSystem.Tests
dotnet test
Restore complete (0.2s)
  AirlineTicketSystem succeeded (0.1s) → /Users/aizen/Documents/учебная папка/ado.net-2.project-main/bin/Debug/net9.0/AirlineTicketSystem.dll
  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 19)



```
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 20)