**INSTITUTO INFNET**

ESCOLA SUPERIOR DE TECNOLOGIA

GRADUAÇÃO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS


**AT**

Aluno: Enzo Furtini

28 de jun. de 2025

https://github.com/G-itch/Enzo_Furtini_DR4_AT

# Sistema de Agência de Turismo - ASP.NET Core - AT

## Descrição

Sistema completo de gerenciamento para agência de turismo desenvolvido em ASP.NET Core com Entity Framework Core, Razor Pages e implementação de delegates e eventos.

## Funcionalidades Implementadas

### Parte 1 - Delegates e Events

#### *1. Delegate para Cálculo de Descontos*

- **Página**: `/CalculoDesconto`
- **Funcionalidade**: Delegate personalizado `CalculateDiscountDelegate` que aplica desconto de 10% (ou percentual customizado) no valor informado
- **Implementação**: `DelegatesService.CalculateDiscount()`

**Código do Delegate**:

```
// DelegatesService.cs
public delegate decimal CalculateDiscountDelegate(decimal preco, decimal
percentualDesconto);

public class DelegatesService
{
    private readonly CalculateDiscountDelegate _calculateDiscountDelegate;

    public DelegatesService()
    {
        _calculateDiscountDelegate = CalculateDiscount;
    }

    public decimal CalculateDiscount(decimal preco, decimal
percentualDesconto = 10)
    {
        return preco * (1 - percentualDesconto / 100);
    }

    public decimal AplicarDesconto(decimal preco, decimal percentualDesconto
= 10)
    {
        return _calculateDiscountDelegate(preco, percentualDesconto);
    }
}
```

**Página Razor**:

```
// CalculoDesconto.cshtml.cs
public class CalculoDescontoModel : PageModel
```

```
{
    private readonly DelegatesService _delegatesService;

    [BindProperty]
    [Required(ErrorMessage = "O preço é obrigatório")]
    [Range(0.01, double.MaxValue, ErrorMessage = "O preço deve ser maior que
zero")]
    public decimal Preco { get; set; }

    [BindProperty]
    [Range(0, 100, ErrorMessage = "O percentual deve estar entre 0 e 100")]
    public decimal PercentualDesconto { get; set; } = 10;

    public IActionResult OnPost()
    {
        if (!ModelState.IsValid)
            return Page();

        // Usa o delegate para calcular o desconto
        PrecoComDesconto = _delegatesService.AplicarDesconto(Preco,
PercentualDesconto);
        Economia = Preco - PrecoComDesconto;
        ResultadoCalculado = true;

        return Page();
    }
}
```

*2. Multicast Delegate para Registro de Logs*

- **Funcionalidade**: Delegate `LogDelegate` com três métodos encadeados:
  - `LogToConsole`: Registra no console
  - `LogToFile`: Salva em arquivo `wwwroot/files/system.log`
  - `LogToMemory`: Simula log em memória
- **Implementação**: `DelegatesService.RegistrarOperacao()`

**Código do Multicast Delegate**:

```
// DelegatesService.cs
public delegate void LogDelegate(string mensagem);

public class DelegatesService
{
    private readonly LogDelegate _logDelegate;

    public DelegatesService()
    {
        // Inicializa o multicast delegate para logs
        _logDelegate = LogToConsole;
        _logDelegate += LogToFile;
```

```
        _logDelegate += LogToMemory;
    }

    public void LogToConsole(string mensagem)
    {
        Console.WriteLine($"[CONSOLE] {DateTime.Now:yyyy-MM-dd HH:mm:ss} -
{mensagem}");
    }

    public void LogToFile(string mensagem)
    {
        var logPath = Path.Combine("wwwroot", "files", "system.log");
        var logDir = Path.GetDirectoryName(logPath);

        if (!Directory.Exists(logDir))
            Directory.CreateDirectory(logDir!);

        File.AppendAllText(logPath, $"[FILE] {DateTime.Now:yyyy-MM-dd
HH:mm:ss} - {mensagem}{Environment.NewLine}");
    }

    public void LogToMemory(string mensagem)
    {
        Console.WriteLine($"[MEMORY] {DateTime.Now:yyyy-MM-dd HH:mm:ss} -
{mensagem}");
    }

    public void RegistrarOperacao(string operacao)
    {
        _logDelegate($"Operação realizada: {operacao}");
    }
}
```

*3. Func com Expressão Lambda*

- **Página**: `/CalculoValorTotal`
- **Funcionalidade**: `Func<int, int, decimal>` que calcula valor total baseado em número de diárias e valor da diária
- **Implementação**: `DelegatesService.CalculateTotalValue`

**Código do Func com Lambda**:

```
// DelegatesService.cs
public Func<int, int, decimal> CalculateTotalValue = (numeroDiarias,
valorDiaria) =>
    numeroDiarias * valorDiaria;
```

**Página Razor**:

```
// CalculoValorTotal.cshtml.cs
public class CalculoValorTotalModel : PageModel
{
    private readonly DelegatesService _delegatesService;

    [BindProperty]
    [Required(ErrorMessage = "O número de diárias é obrigatório")]
    [Range(1, 30, ErrorMessage = "O número de diárias deve estar entre 1 e
30")]
    public int NumeroDiarias { get; set; }

    [BindProperty]
    [Required(ErrorMessage = "O valor da diária é obrigatório")]
    [Range(0.01, double.MaxValue, ErrorMessage = "O valor da diária deve ser
maior que zero")]
    public decimal ValorDiaria { get; set; }

    public IActionResult OnPost()
    {
        if (!ModelState.IsValid)
            return Page();

        // Usa o Func com expressão lambda para calcular o valor total
        ValorTotal = _delegatesService.CalculateTotalValue(NumeroDiarias,
(int)ValorDiaria);
        ResultadoCalculado = true;

        return Page();
    }
}
```

*4. Evento de Alerta para Limite de Capacidade*

- **Funcionalidade**: Evento `CapacityReached` na classe `PacoteTuristico` que dispara quando a capacidade máxima é atingida
- **Implementação**: `PacoteTuristico.OnCapacityReached()` e `DelegatesService.OnCapacityReached()`

**Código do Evento**:

```
// PacoteTuristico.cs
public class PacoteTuristico
{
    // Propriedades...

    // Evento para quando a capacidade for atingida
    public event EventHandler<CapacityReachedEventArgs>? CapacityReached;

    // Método para disparar o evento
    protected virtual void OnCapacityReached(int reservasAtuais)
```

```
    {
        CapacityReached?.Invoke(this, new CapacityReachedEventArgs
        {
            PacoteId = Id,
            Titulo = Titulo,
            CapacidadeMaxima = CapacidadeMaxima,
            ReservasAtuais = reservasAtuais
        });
    }

    // Método para verificar se a capacidade foi atingida
    public void VerificarCapacidade()
    {
        if (Reservas.Count >= CapacidadeMaxima)
        {
            OnCapacityReached(Reservas.Count);
        }
    }
}

public class CapacityReachedEventArgs : EventArgs
{
    public int PacoteId { get; set; }
    public string Titulo { get; set; } = string.Empty;
    public int CapacidadeMaxima { get; set; }
    public int ReservasAtuais { get; set; }
}
```

**Integração com Delegate**:

```
// DelegatesService.cs
public void OnCapacityReached(object? sender, CapacityReachedEventArgs e)
{
    var mensagem = $"ALERTA: Capacidade máxima atingida para o pacote
'{e.Titulo}' (ID: {e.PacoteId}). " +
                   $"Capacidade: {e.CapacidadeMaxima}, Reservas atuais:
{e.ReservasAtuais}";

    _logDelegate(mensagem);
}
```

## Parte 2 - ASP.NET Razor Pages

### *5. Cadastro de Entidade com Validação*

- **Página**: `/CreateCidade`
- **Funcionalidade**: Formulário com validação usando Data Annotations
- **Validações**: Nome obrigatório com mínimo de 3 caracteres, relacionamento com país

**Código do Modelo com Validações**:

```
// CidadeDestino.cs
public class CidadeDestino
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O nome da cidade é obrigatório")]
    [StringLength(100, MinimumLength = 3, ErrorMessage = "O nome da cidade
deve ter entre 3 e 100 caracteres")]
    public string Nome { get; set; } = string.Empty;

    [StringLength(500)]
    public string? Descricao { get; set; }

    public int PaisDestinoId { get; set; }
    public virtual PaisDestino PaisDestino { get; set; } = null!;
}
```

**Código da Página Razor**:

```
// CreateCidade.cshtml.cs
public class CreateCidadeModel : PageModel
{
    private readonly EnzoFurtiniDR4ATContext _context;
    private readonly DelegatesService _delegatesService;

    [BindProperty]
    public CidadeDestino Cidade { get; set; } = new();

    public SelectList PaisesOptions { get; set; } = null!;

    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            await LoadPaisesOptions();
            return Page();
        }

        _context.CidadesDestino.Add(Cidade);
        await _context.SaveChangesAsync();

        // Registra a operação usando o delegate
        _delegatesService.RegistrarOperacao($"Nova cidade cadastrada:
{Cidade.Nome}");

        return RedirectToPage("/Cidades/Index");
    }
}
```

**Página Razor**:

```
<!-- CreateCidade.cshtml -->
@page @model Enzo_Furtini_DR4_AT.Pages.CreateCidadeModel

<form method="post">
  <div class="mb-3">
    <label asp-for="Cidade.Nome" class="form-label">Nome da Cidade *</label>
    <input
      asp-for="Cidade.Nome"
      class="form-control"
      placeholder="Digite o nome da cidade"
    />
    <span asp-validation-for="Cidade.Nome" class="text-danger"></span>
    <div class="form-text">O nome deve ter pelo menos 3 caracteres.</div>
  </div>

  <div class="mb-3">
    <label asp-for="Cidade.PaisDestinoId" class="form-label">País *</label>
    <select
      asp-for="Cidade.PaisDestinoId"
      class="form-select"
      asp-items="Model.PaisesOptions"
    >
      <option value="">Selecione um país</option>
    </select>
    <span asp-validation-for="Cidade.PaisDestinoId"
class="text-danger"></span>
  </div>

  <button type="submit" class="btn btn-primary">Cadastrar Cidade</button>
</form>
```

*6. Cadastro de Entidade com Objeto Complexo*

- **Página**: `/CreateCidade`
- **Funcionalidade**: Cadastro de cidade com campos complexos (nome, descrição, país)
- **Validações**: `[Required]`, `[StringLength]`, `[MinLength]`

**Código do Objeto Complexo**:

```
// CidadeDestino.cs - Objeto Complexo
public class CidadeDestino
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O nome da cidade é obrigatório")]
    [StringLength(100, MinimumLength = 3, ErrorMessage = "O nome da cidade
deve ter entre 3 e 100 caracteres")]
    public string Nome { get; set; } = string.Empty;
```

```
    [StringLength(500)]
    public string? Descricao { get; set; }

    [Required(ErrorMessage = "O país é obrigatório")]
    public int PaisDestinoId { get; set; }
    public virtual PaisDestino PaisDestino { get; set; } = null!;
}
```

*7. Detalhamento via Roteamento na URL*

- **Página**: `/CidadeDetails/{id:int}`
- **Funcionalidade**: Exibe detalhes da cidade recebendo ID via rota
- **Implementação**: Roteamento com parâmetros `@page "{id:int}"`

**Código da Página com Roteamento**:

```
// CidadeDetails.cshtml.cs
public class CidadeDetailsModel : PageModel
{
    private readonly EnzoFurtiniDR4ATContext _context;

    public CidadeDetailsModel(EnzoFurtiniDR4ATContext context)
    {
        _context = context;
    }

    public CidadeDestino? Cidade { get; set; }

    public async Task<IActionResult> OnGetAsync(int id)
    {
        Cidade = await _context.CidadesDestino
            .Include(c => c.PaisDestino)
            .Include(c => c.PacotesTuristicos)
            .FirstOrDefaultAsync(c => c.Id == id && !c.IsDeleted);

        if (Cidade == null)
        {
            return NotFound();
        }

        return Page();
    }
}
```

**Página Razor com Roteamento**:

```
<!-- CidadeDetails.cshtml -->
@page "{id:int}" @model Enzo_Furtini_DR4_AT.Pages.CidadeDetailsModel

<div class="card">
```

```
  <div class="card-header">
    <h2>@Model.Cidade.Nome</h2>
  </div>
  <div class="card-body">
    <div class="row mb-3">
      <div class="col-md-4">
        <strong>Nome da Cidade:</strong>
      </div>
      <div class="col-md-8">@Model.Cidade.Nome</div>
    </div>

    <div class="row mb-3">
      <div class="col-md-4">
        <strong>País:</strong>
      </div>
      <div class="col-md-8">@Model.Cidade.PaisDestino?.Nome</div>
    </div>
  </div>
</div>
```

*8. Sistema de Notas com Leitura e Escrita de Arquivos*

- **Página**: `/ViewNotes`
- **Funcionalidade**:
  - Criar notas e salvar como arquivos .txt em `wwwroot/files/`
  - Listar arquivos disponíveis
  - Visualizar conteúdo dos arquivos
- **Implementação**: `System.IO` para manipulação de arquivos

**Código do Sistema de Notas**:

```
// ViewNotes.cshtml.cs
public class ViewNotesModel : PageModel
{
    private readonly DelegatesService _delegatesService;

    [BindProperty]
    [Required(ErrorMessage = "O título é obrigatório")]
    [StringLength(100, MinimumLength = 3, ErrorMessage = "O título deve ter
entre 3 e 100 caracteres")]
    public string Titulo { get; set; } = string.Empty;

    [BindProperty]
    [Required(ErrorMessage = "O conteúdo é obrigatório")]
    [StringLength(2000, MinimumLength = 10, ErrorMessage = "O conteúdo deve
ter entre 10 e 2000 caracteres")]
    public string Conteudo { get; set; } = string.Empty;

    public List<ArquivoNota> ArquivosNotas { get; set; } = new();
    public string? ConteudoSelecionado { get; set; }
```

```
    public string? ArquivoSelecionado { get; set; }

    public void OnGet(string? arquivo)
    {
        CarregarArquivosNotas();

        if (!string.IsNullOrEmpty(arquivo))
        {
            VisualizarArquivo(arquivo);
        }
    }

    public IActionResult OnPost()
    {
        if (!ModelState.IsValid)
        {
            CarregarArquivosNotas();
            return Page();
        }

        SalvarNota();
        CarregarArquivosNotas();

        return Page();
    }

    private void SalvarNota()
    {
        try
        {
            var diretorio = Path.Combine("wwwroot", "files");
            if (!Directory.Exists(diretorio))
            {
                Directory.CreateDirectory(diretorio);
            }

            var nomeArquivo = $"{Titulo.Replace(" ", "_").Replace("/",
"_").Replace("\\", "_")}_{DateTime.Now:yyyyMMdd_HHmmss}.txt";
            var caminhoArquivo = Path.Combine(diretorio, nomeArquivo);

            var conteudoCompleto = $"Título: {Titulo}\nData:
{DateTime.Now:dd/MM/yyyy HH:mm:ss}\n\n{Conteudo}";
            System.IO.File.WriteAllText(caminhoArquivo, conteudoCompleto);

            _delegatesService.RegistrarOperacao($"Nova nota criada:
{Titulo}");
        }
        catch (Exception ex)
        {
```

```
            Mensagem = $"Erro ao salvar a nota: {ex.Message}";
        }
    }

    private void CarregarArquivosNotas()
    {
        try
        {
            var diretorio = Path.Combine("wwwroot", "files");
            if (Directory.Exists(diretorio))
            {
                var arquivos = Directory.GetFiles(diretorio, "*.txt")
                    .Select(f => new ArquivoNota
                    {
                        Nome = Path.GetFileName(f),
                        Caminho = f,
                        DataCriacao = System.IO.File.GetCreationTime(f)
                    })
                    .OrderByDescending(a => a.DataCriacao)
                    .ToList();

                ArquivosNotas = arquivos;
            }
        }
        catch (Exception ex)
        {
            Mensagem = $"Erro ao carregar arquivos: {ex.Message}";
        }
    }

    private void VisualizarArquivo(string nomeArquivo)
    {
        try
        {
            var caminhoArquivo = Path.Combine("wwwroot", "files",
nomeArquivo);
            if (System.IO.File.Exists(caminhoArquivo))
            {
                ConteudoSelecionado =
System.IO.File.ReadAllText(caminhoArquivo);
                ArquivoSelecionado = nomeArquivo;
            }
        }
        catch (Exception ex)
        {
            Mensagem = $"Erro ao visualizar arquivo: {ex.Message}";
        }
    }
}
```

```
public class ArquivoNota
{
    public string Nome { get; set; } = string.Empty;
    public string Caminho { get; set; } = string.Empty;
    public DateTime DataCriacao { get; set; }
}
```

## Parte 3 - Entity Framework Core

### *9. Criação de DbContext*

- **Classe**: `EnzoFurtiniDR4ATContext`
- **Configuração**: DbSet para todas as entidades do sistema
- **Conexão**: SQL Server LocalDB configurada em `appsettings.json`

**Código do DbContext**:

```
// EnzoFurtiniDR4ATContext.cs
public class EnzoFurtiniDR4ATContext : DbContext
{
    public EnzoFurtiniDR4ATContext(DbContextOptions<EnzoFurtiniDR4ATContext>
options)
        : base(options)
    {
    }

    public DbSet<Cliente> Clientes { get; set; }
    public DbSet<PaisDestino> PaisesDestino { get; set; }
    public DbSet<CidadeDestino> CidadesDestino { get; set; }
    public DbSet<PacoteTuristico> PacotesTuristicos { get; set; }
    public DbSet<Reserva> Reservas { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Configuração do Cliente
        modelBuilder.Entity<Cliente>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.Property(e => e.Nome).IsRequired().HasMaxLength(100);
            entity.Property(e => e.Email).IsRequired().HasMaxLength(100);
            entity.Property(e => e.Telefone).IsRequired().HasMaxLength(20);
            entity.Property(e => e.CPF).IsRequired().HasMaxLength(14);
            entity.Property(e =>
e.DataCadastro).HasDefaultValueSql("GETDATE()");

            // Índice único para CPF
            entity.HasIndex(e => e.CPF).IsUnique();
```

```
            entity.HasIndex(e => e.Email).IsUnique();
        });

        // Configuração do PaisDestino
        modelBuilder.Entity<PaisDestino>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.Property(e => e.Nome).IsRequired().HasMaxLength(100);
            entity.Property(e => e.Codigo).HasMaxLength(3);
        });

        // Configuração do CidadeDestino
        modelBuilder.Entity<CidadeDestino>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.Property(e => e.Nome).IsRequired().HasMaxLength(100);
            entity.Property(e => e.Descricao).HasMaxLength(500);

            // Relacionamento com PaisDestino
            entity.HasOne(e => e.PaisDestino)
                  .WithMany(p => p.Cidades)
                  .HasForeignKey(e => e.PaisDestinoId)
                  .OnDelete(DeleteBehavior.Restrict);
        });

        // Configuração do PacoteTuristico
        modelBuilder.Entity<PacoteTuristico>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.Property(e => e.Titulo).IsRequired().HasMaxLength(200);
            entity.Property(e =>
e.Descricao).IsRequired().HasMaxLength(1000);
            entity.Property(e => e.Preco).HasColumnType("decimal(18,2)");

            // Relacionamento muitos-para-muitos com CidadeDestino
            entity.HasMany(e => e.Destinos)
                  .WithMany(c => c.PacotesTuristicos)
                  .UsingEntity(j => j.ToTable("PacoteTuristicoDestinos"));
        });

        // Configuração do Reserva
        modelBuilder.Entity<Reserva>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.Property(e =>
e.ValorTotal).HasColumnType("decimal(18,2)");
            entity.Property(e => e.Observacoes).HasMaxLength(500);

            // Relacionamento com Cliente
```

```
            entity.HasOne(e => e.Cliente)
                  .WithMany(c => c.Reservas)
                  .HasForeignKey(e => e.ClienteId)
                  .OnDelete(DeleteBehavior.Restrict);

            // Relacionamento com PacoteTuristico
            entity.HasOne(e => e.PacoteTuristico)
                  .WithMany(p => p.Reservas)
                  .HasForeignKey(e => e.PacoteTuristicoId)
                  .OnDelete(DeleteBehavior.Restrict);

            // Índice composto para evitar reservas duplicadas
            entity.HasIndex(e => new { e.ClienteId, e.PacoteTuristicoId,
e.DataReserva })
                  .IsUnique();
        });

        // Configuração de exclusão lógica global
        modelBuilder.Entity<Cliente>().HasQueryFilter(e => !e.IsDeleted);
        modelBuilder.Entity<PaisDestino>().HasQueryFilter(e => !e.IsDeleted);
        modelBuilder.Entity<CidadeDestino>().HasQueryFilter(e =>
!e.IsDeleted);
        modelBuilder.Entity<PacoteTuristico>().HasQueryFilter(e =>
!e.IsDeleted);
        modelBuilder.Entity<Reserva>().HasQueryFilter(e => !e.IsDeleted);
    }
}
```

**Registro no Program.cs**:

```
// Program.cs
builder.Services.AddDbContext<EnzoFurtiniDR4ATContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnec
tion")));
```

**String de conexão**:

```
// appsettings.json
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=EnzoFurtiniDR4AT;Trusted_Connection=
true;MultipleActiveResultSets=true"
  }
}
```

*10. Modelagem e Relacionamento entre Entidades*

- **Entidades**: Cliente, PaisDestino, CidadeDestino, PacoteTuristico, Reserva
- **Relacionamentos**:

  - Um-para-muitos: País → Cidades, Cliente → Reservas, Pacote → Reservas
  - Muitos-para-muitos: Pacotes ↔ Cidades
- **Configuração**: Fluent API e Data Annotations

**Código das Entidades**:

```
// Cliente.cs
public class Cliente
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O nome é obrigatório")]
    [StringLength(100, MinimumLength = 3, ErrorMessage = "O nome deve ter
entre 3 e 100 caracteres")]
    public string Nome { get; set; } = string.Empty;

    [Required(ErrorMessage = "O email é obrigatório")]
    [EmailAddress(ErrorMessage = "Email inválido")]
    public string Email { get; set; } = string.Empty;

    [Required(ErrorMessage = "O telefone é obrigatório")]
    [Phone(ErrorMessage = "Telefone inválido")]
    public string Telefone { get; set; } = string.Empty;

    [Required(ErrorMessage = "O CPF é obrigatório")]
    [StringLength(14, MinimumLength = 11, ErrorMessage = "CPF inválido")]
    public string CPF { get; set; } = string.Empty;

    public DateTime DataCadastro { get; set; } = DateTime.Now;
    public bool IsDeleted { get; set; } = false;
    public DateTime? DeletedAt { get; set; }

    // Propriedade de navegação
    public virtual ICollection<Reserva> Reservas { get; set; } = new
List<Reserva>();
}

// PaisDestino.cs
public class PaisDestino
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O nome do país é obrigatório")]
    [StringLength(100, MinimumLength = 3, ErrorMessage = "O nome do país deve
ter entre 3 e 100 caracteres")]
    public string Nome { get; set; } = string.Empty;

    [StringLength(3, MinimumLength = 2, ErrorMessage = "O código do país deve
ter entre 2 e 3 caracteres")]
```

```
    public string Codigo { get; set; } = string.Empty;

    public bool IsDeleted { get; set; } = false;
    public DateTime? DeletedAt { get; set; }

    // Propriedade de navegação
    public virtual ICollection<CidadeDestino> Cidades { get; set; } = new
List<CidadeDestino>();
}

// CidadeDestino.cs
public class CidadeDestino
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O nome da cidade é obrigatório")]
    [StringLength(100, MinimumLength = 3, ErrorMessage = "O nome da cidade
deve ter entre 3 e 100 caracteres")]
    public string Nome { get; set; } = string.Empty;

    [StringLength(500)]
    public string? Descricao { get; set; }

    public int PaisDestinoId { get; set; }
    public virtual PaisDestino PaisDestino { get; set; } = null!;

    public bool IsDeleted { get; set; } = false;
    public DateTime? DeletedAt { get; set; }

    // Propriedade de navegação para pacotes turísticos
    public virtual ICollection<PacoteTuristico> PacotesTuristicos { get; set;
} = new List<PacoteTuristico>();
}

// PacoteTuristico.cs
public class PacoteTuristico
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O título é obrigatório")]
    [StringLength(200, MinimumLength = 5, ErrorMessage = "O título deve ter
entre 5 e 200 caracteres")]
    public string Titulo { get; set; } = string.Empty;

    [Required(ErrorMessage = "A descrição é obrigatória")]
    [StringLength(1000, MinimumLength = 10, ErrorMessage = "A descrição deve
ter entre 10 e 1000 caracteres")]
    public string Descricao { get; set; } = string.Empty;
```

```
    [Required(ErrorMessage = "A data de início é obrigatória")]
    [DataType(DataType.Date)]
    public DateTime DataInicio { get; set; }

    [Required(ErrorMessage = "A data de fim é obrigatória")]
    [DataType(DataType.Date)]
    public DateTime DataFim { get; set; }

    [Required(ErrorMessage = "A capacidade máxima é obrigatória")]
    [Range(1, 100, ErrorMessage = "A capacidade deve estar entre 1 e 100")]
    public int CapacidadeMaxima { get; set; }

    [Required(ErrorMessage = "O preço é obrigatório")]
    [Range(0.01, double.MaxValue, ErrorMessage = "O preço deve ser maior que
zero")]
    public decimal Preco { get; set; }

    public bool IsDeleted { get; set; } = false;
    public DateTime? DeletedAt { get; set; }

    // Propriedades de navegação
    public virtual ICollection<CidadeDestino> Destinos { get; set; } = new
List<CidadeDestino>();
    public virtual ICollection<Reserva> Reservas { get; set; } = new
List<Reserva>();

    // Evento para quando a capacidade for atingida
    public event EventHandler<CapacityReachedEventArgs>? CapacityReached;
}

// Reserva.cs
public class Reserva
{
    public int Id { get; set; }

    public int ClienteId { get; set; }
    public virtual Cliente Cliente { get; set; } = null!;

    public int PacoteTuristicoId { get; set; }
    public virtual PacoteTuristico PacoteTuristico { get; set; } = null!;

    [Required(ErrorMessage = "A data da reserva é obrigatória")]
    [DataType(DataType.Date)]
    public DateTime DataReserva { get; set; } = DateTime.Now;

    [Required(ErrorMessage = "O número de participantes é obrigatório")]
    [Range(1, 10, ErrorMessage = "O número de participantes deve estar entre
1 e 10")]
    public int NumeroParticipantes { get; set; }
```

```
    [Required(ErrorMessage = "O valor total é obrigatório")]
    [Range(0.01, double.MaxValue, ErrorMessage = "O valor total deve ser
maior que zero")]
    public decimal ValorTotal { get; set; }

    [StringLength(500)]
    public string? Observacoes { get; set; }

    public bool IsDeleted { get; set; } = false;
    public DateTime? DeletedAt { get; set; }
}
```

## Parte 4 - Scaffolding + Autenticação

### *11. Operações CRUD com Scaffolding e Personalização*

- **Funcionalidade**: Páginas CRUD geradas para todas as entidades
- **Personalização**: Exibição de dados relacionados e formatação adequada

**Código de Exemplo para Página Index Personalizada**:

```
// Cidades/Index.cshtml.cs
public class CidadesIndexModel : PageModel
{
    private readonly EnzoFurtiniDR4ATContext _context;

    public CidadesIndexModel(EnzoFurtiniDR4ATContext context)
    {
        _context = context;
    }

    public IList<CidadeDestino> CidadeDestino { get; set; } = default!;

    public async Task OnGetAsync()
    {
        // Carrega dados relacionados (País)
        CidadeDestino = await _context.CidadesDestino
            .Include(c => c.PaisDestino)
            .Where(c => !c.IsDeleted)
            .ToListAsync();
    }
}
```

**Página Razor Personalizada**:

```
<!-- Cidades/Index.cshtml -->
@page @model Enzo_Furtini_DR4_AT.Pages.Cidades.IndexModel

<h1>Cidades de Destino</h1>
```

```
<p>
  <a asp-page="Create" class="btn btn-primary">Nova Cidade</a>
</p>

<table class="table">
  <thead>
    <tr>
      <th>Nome</th>
      <th>País</th>
      <th>Descrição</th>
      <th>Ações</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in Model.CidadeDestino) {
    <tr>
      <td>@item.Nome</td>
      <td>@item.PaisDestino?.Nome</td>
      <td>@item.Descricao</td>
      <td>
        <a
          asp-page="./Details"
          asp-route-id="@item.Id"
          class="btn btn-info btn-sm"
          >Detalhes</a
        >
        <a
          asp-page="./Edit"
          asp-route-id="@item.Id"
          class="btn btn-warning btn-sm"
          >Editar</a
        >
        <a
          asp-page="./Delete"
          asp-route-id="@item.Id"
          class="btn btn-danger btn-sm"
          >Excluir</a
        >
      </td>
    </tr>
    }
  </tbody>
</table>
```

### *12. Exclusão Lógica e Autenticação*

- **Exclusão Lógica**: Campo `IsDeleted` e `DeletedAt` em todas as entidades
- **Autenticação**: Sistema simples baseado em cookies
- **Credenciais**: admin/admin123
- **Proteção**: Atributo `[Authorize]` em páginas sensíveis

**Código de Exclusão Lógica**:

```
// Exemplo de método de exclusão lógica
public async Task<IActionResult> OnPostDeleteAsync(int id)
{
    var cliente = await _context.Clientes.FindAsync(id);

    if (cliente != null)
    {
        // Exclusão lógica
        cliente.IsDeleted = true;
        cliente.DeletedAt = DateTime.Now;

        await _context.SaveChangesAsync();
        _delegatesService.RegistrarOperacao($"Cliente excluído logicamente:
{cliente.Nome}");
    }

    return RedirectToPage("./Index");
}
```

**Código do Sistema de Autenticação**:

```
// Account/Login.cshtml.cs
public class LoginModel : PageModel
{
    [BindProperty]
    [Required(ErrorMessage = "O usuário é obrigatório")]
    public string Username { get; set; } = string.Empty;

    [BindProperty]
    [Required(ErrorMessage = "A senha é obrigatória")]
    public string Password { get; set; } = string.Empty;

    [BindProperty]
    public bool RememberMe { get; set; }

    public string? ErrorMessage { get; set; }

    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        // Autenticação simples baseada em credenciais hardcoded
        if (IsValidUser(Username, Password))
        {
            var claims = new List<Claim>
```

```
            {
                new Claim(ClaimTypes.Name, Username),
                new Claim(ClaimTypes.Role, "Admin")
            };

            var claimsIdentity = new ClaimsIdentity(claims, "Cookies");
            var claimsPrincipal = new ClaimsPrincipal(claimsIdentity);

            var authProperties = new AuthenticationProperties
            {
                IsPersistent = RememberMe,
                ExpiresUtc = RememberMe ? DateTimeOffset.UtcNow.AddDays(7) :
null
            };

            await HttpContext.SignInAsync("Cookies", claimsPrincipal,
authProperties);

            return RedirectToPage("/Index");
        }

        ErrorMessage = "Usuário ou senha inválidos.";
        return Page();
    }

    private bool IsValidUser(string username, string password)
    {
        // Credenciais hardcoded para demonstração
        return username == "admin" && password == "admin123";
    }
}
```

**Configuração no Program.cs**:

```
// Program.cs
// Configuração da autenticação
builder.Services.AddAuthentication("Cookies")
    .AddCookie(options =>
    {
        options.LoginPath = "/Account/Login";
        options.LogoutPath = "/Account/Logout";
        options.AccessDeniedPath = "/Account/AccessDenied";
    });

builder.Services.AddAuthorization();

// ... outras configurações
```

```
app.UseAuthentication();
app.UseAuthorization();
```

**Proteção de Páginas**:

```
// Exemplo de página protegida
[Authorize]
public class AdminModel : PageModel
{
    public void OnGet()
    {
        // Apenas usuários autenticados podem acessar
    }
}
```