.\RealtyHub.ApiService\.config\dotnet-tools.json


```
{
  "version": 1,
  "isRoot": true,
  "tools": {
    "dotnet-ef": {
      "version": "9.0.1",
      "commands": [
        "dotnet-ef"
      ],
      "rollForward": false
    }
  }
}
```

.\RealtyHub.ApiService\Common\Api\AppExtension.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.FileProviders;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Utilities.FakeEntities;

namespace RealtyHub.ApiService.Common.Api;

/// <summary>
/// Classe estática que agrupa métodos de extensão para configurar
/// e inicializar o ambiente da aplicação.
/// </summary>
public static class AppExtension
{
    /// <summary>
    /// Configura rotas de criação de dados de teste em ambiente
    /// de desenvolvimento (clientes e propriedades).
    /// </summary>
    /// <remarks>
    /// Este método é utilizado para popular o banco de dados
    /// com dados de teste durante o desenvolvimento.
    /// </remarks>
    /// <param name="app">Instância do aplicativo.</param>
    public static void ConfigureDevEnvironment(this WebApplication app)
    {
        app.MapPost("v1/customers/createmany", async (
                AppDbContext context,
                [FromQuery] int individualQuantity = 0,
                [FromQuery] int bussinessQuantity = 0) =>
        {
            if (individualQuantity > 0)
            {
                var individualCustomers = CustomerFake
                    .GetFakeIndividualCustomers(individualQuantity);
                await context.Customers.AddRangeAsync(individualCustomers);
            }

            if (bussinessQuantity > 0)
            {
                var businessCustomers = CustomerFake
                    .GetFakeBusinessCustomers(bussinessQuantity);
                await context.Customers.AddRangeAsync(businessCustomers);

            }

            await context.SaveChangesAsync();

            return Results.Created();
        });

        app.MapPost("v1/properties/createmany", async (
            AppDbContext context,
            [FromQuery] int quantity = 0,
            [FromQuery] int customerId = 0,
            [FromQuery] int condominiumId = 0) =>
        {
            if (quantity > 0)
            {
                var properties = PropertyFake
                    .GetFakeProperties(quantity, customerId, condominiumId);
                await context.Properties.AddRangeAsync(properties);
            }

            await context.SaveChangesAsync();

            return Results.Created();
        });
    }

    /// <summary>
    /// Habilita a autenticação e autorização na aplicação.
    /// </summary>
```

```
    /// <remarks>
    /// Este método habilita a autenticação e autorização
    /// para a aplicação, permitindo o uso de cookies
    /// e tokens JWT.
    /// </remarks>
    /// <param name="app">Instância do aplicativo.</param>
    public static void UseSecurity(this WebApplication app)
    {
        app.UseAuthentication();
        app.UseAuthorization();
    }

    /// <summary>
    /// Configura a exibição de arquivos estáticos, definindo
    /// os caminhos físicos e lógicos para cada pasta.
    /// </summary>
    /// <remarks>
    /// Este método configura os diretórios para armazenar
    /// fotos, contratos, templates de contratos e relatórios.
    /// </remarks>
    /// <param name="app">Instância do aplicativo.</param>
    public static void UseStaticFiles(this WebApplication app)
    {
        app.UseStaticFiles(new StaticFileOptions
        {
            FileProvider = new PhysicalFileProvider(Configuration.PhotosPath),
            RequestPath = "/photos"
        });

        app.UseStaticFiles(new StaticFileOptions
        {
            FileProvider = new PhysicalFileProvider(Configuration.ContractsPath),
            RequestPath = "/contracts"
        });

        app.UseStaticFiles(new StaticFileOptions
        {
            FileProvider = new PhysicalFileProvider(Configuration.ContractTemplatesPath),
            RequestPath = "/contracts-templates"
        });

        app.UseStaticFiles(new StaticFileOptions
        {
            FileProvider = new PhysicalFileProvider(Configuration.ReportsPath),
            RequestPath = "/reports"
        });
    }

    /// <summary>
    /// Aplica automaticamente as migrações do banco de dados
    /// ao iniciar a aplicação.
    /// </summary>
    /// <remarks>
    /// Este método garante que o banco de dados esteja atualizado
    /// com as últimas migrações antes de iniciar a aplicação.
    /// </remarks>
    /// <param name="app">Instância do aplicativo.</param>
    public static void ApplyMigrations(this WebApplication app)
    {
        using var scope = app.Services.CreateScope();
        var dbContext = scope.ServiceProvider.GetRequiredService<AppDbContext>();
        dbContext.Database.Migrate();
    }
}
```

.\RealtyHub.ApiService\Common\Api\BuilderExtension.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.ApiService.Handlers;
using RealtyHub.ApiService.Models;
using RealtyHub.ApiService.Services;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Services;

namespace RealtyHub.ApiService.Common.Api;

/// <summary>
/// Classe estática que agrupa métodos de extensão para configurar
/// documentações, diretórios, autenticação, autorização e serviços
/// da aplicação.
/// </summary>
public static class BuilderExtension
{
    /// <summary>
    /// Configura diretórios necessários para a aplicação.
    /// </summary>
    /// <remarks>
    /// Este método cria diretórios para armazenar contratos,
    /// fotos, templates de contratos, relatórios e logos.
    /// </remarks>
    /// <param name="builder">Instância do construtor da aplicação.</param>
    public static void AddDirectories(this WebApplicationBuilder builder)
    {
        var basePath = builder.Environment.ContentRootPath;

        Configuration.ContractsPath = Path.Combine(basePath, "Sources", "Contracts");
        if (!Directory.Exists(Configuration.ContractsPath))
            Directory.CreateDirectory(Configuration.ContractsPath);

        Configuration.PhotosPath = Path.Combine(basePath, "Sources", "Photos");
        if (!Directory.Exists(Configuration.PhotosPath))
            Directory.CreateDirectory(Configuration.PhotosPath);

        Configuration.ContractTemplatesPath = Path.Combine(basePath, "Sources",
"ContractTemplates");
        if (!Directory.Exists(Configuration.ContractTemplatesPath))
            Directory.CreateDirectory(Configuration.ContractTemplatesPath);

        Configuration.ReportsPath = Path.Combine(basePath, "Sources", "Reports");
        if (!Directory.Exists(Configuration.ReportsPath))
            Directory.CreateDirectory(Configuration.ReportsPath);

        Configuration.LogosPath = Path.Combine(basePath, "Sources", "Logos");
        if (!Directory.Exists(Configuration.LogosPath))
            Directory.CreateDirectory(Configuration.LogosPath);
    }

    /// <summary>
    /// Configura as variáveis de ambiente e caminhos de conexão.
    /// </summary>
    /// <remarks>
    /// Este método lê as variáveis de ambiente e configura
    /// as strings de conexão necessárias para a aplicação.
    /// </remarks>
    /// <param name="builder">Instância do construtor da aplicação.</param>
    public static void AddConfiguration(this WebApplicationBuilder builder)
    {
        Core.Configuration.ConnectionString =
            builder
                .Configuration
                .GetConnectionString("DefaultConnection")
            ?? string.Empty;

        Core.Configuration.BackendUrl = builder.Configuration
            .GetValue<string>("BackendUrl") ?? string.Empty;

        Core.Configuration.FrontendUrl = builder.Configuration
```

```
        .GetValue<string>("FrontendUrl") ?? string.Empty;

    Configuration.EmailSettings.EmailPassword =
        builder.Configuration.GetValue<string>("EmailPassword") ?? string.Empty;
}

/// <summary>
/// Configura e ativa a documentação da API com Swagger.
/// </summary>
/// <remarks>
/// Este método adiciona o Swagger à aplicação, permitindo
/// visualizar a documentação da API e testar os endpoints.
/// </remarks>
/// <param name="builder">Instância do construtor da aplicação.</param>
public static void AddDocumentation(this WebApplicationBuilder builder)
{
    builder.Services.AddEndpointsApiExplorer();
    builder.Services.AddSwaggerGen(options =>
    {
        options.OrderActionsBy((apiDesc) => apiDesc.GroupName);
        options.CustomSchemaIds(n => n.FullName);
    });
}

/// <summary>
/// Adiciona e configura a autenticação e autorização da aplicação.
/// </summary>
/// <remarks>
/// Este método configura a autenticação com cookies e JWT,
/// permitindo o uso de autenticação baseada em token.
/// </remarks>
/// <param name="builder">Instância do construtor da aplicação.</param>
public static void AddSecurity(this WebApplicationBuilder builder)
{
    builder.Services
        .AddAuthentication(IdentityConstants.ApplicationScheme)
        .AddIdentityCookies();

    builder.Services.AddAuthorization();
}

/// <summary>
/// Configura os contextos de dados, habilitando o Entity Framework e Identity.
/// </summary>
/// <remarks>
/// Este método adiciona o DbContext da aplicação e configura
/// o Identity para gerenciar usuários e roles.
/// </remarks>
/// <param name="builder">Instância do construtor da aplicação.</param>
public static void AddDataContexts(this WebApplicationBuilder builder)
{
    builder
        .Services
        .AddDbContext<AppDbContext>(
            x =>
            {
                x.UseNpgsql(Core.Configuration.ConnectionString)
                    .EnableSensitiveDataLogging()
                    .EnableDetailedErrors();
            });

    builder.Services
        .AddIdentityCore<User>(options =>
        {
            options.SignIn.RequireConfirmedAccount = false;
            options.SignIn.RequireConfirmedEmail = true;
            options.SignIn.RequireConfirmedPhoneNumber = false;
            options.User.RequireUniqueEmail = true;
            options.Password.RequireLowercase = false;
            options.Password.RequireUppercase = false;
            options.Password.RequireDigit = false;
            options.Password.RequireNonAlphanumeric = false;
            options.Password.RequiredUniqueChars = 1;
        })
        .AddRoles<IdentityRole<long>>()
        .AddEntityFrameworkStores<AppDbContext>()
```

```
            .AddApiEndpoints();
    }

    /// <summary>
    /// Configura o CORS para permitir chamadas de origens específicas.
    /// </summary>
    /// <remarks>
    /// Este método adiciona uma política de CORS que permite
    /// chamadas de origens específicas, como o frontend e o backend.
    /// </remarks>
    /// /// <param name="builder">Instância do construtor da aplicação.</param>
    public static void AddCrossOrigin(this WebApplicationBuilder builder)
    {
        builder.Services.AddCors(
            options => options.AddPolicy(
                Configuration.CorsPolicyName,
                policy => policy
                    .WithOrigins([
                        Core.Configuration.BackendUrl,
                        Core.Configuration.FrontendUrl
                    ])
                    .AllowAnyMethod()
                    .AllowAnyHeader()
                    .AllowCredentials()
            ));
    }

    /// <summary>
    /// Registra serviços adicionais necessários para a aplicação.
    /// </summary>
    /// <remarks>
    /// Este método adiciona serviços como manipuladores, serviços de email e templates
de contratos.
    /// </remarks>
    /// <param name="builder">Instância do construtor da aplicação.</param>
    public static void AddServices(this WebApplicationBuilder builder)
    {
        builder.Services.AddProblemDetails();
        builder.Services.AddTransient<ICustomerHandler, CustomerHandler>();
        builder.Services.AddTransient<IPropertyHandler, PropertyHandler>();
        builder.Services.AddTransient<ICondominiumHandler, CondominiumHandler>();
        builder.Services.AddTransient<IViewingHandler, ViewingHandler>();
        builder.Services.AddTransient<IOfferHandler, OfferHandler>();
        builder.Services.AddTransient<IContractHandler, ContractHandler>();
        builder.Services.AddTransient<IEmailService, EmailService>();
        builder.Services.AddTransient<IPropertyPhotosHandler, PropertyPhotosHandler>();
        builder.Services.AddTransient<IContractTemplateHandler,
ContractTemplatesHandler>();
    }
}
```

.\RealtyHub.ApiService\Common\Api\IEndpoint.cs

```
namespace RealtyHub.ApiService.Common.Api;

/// <summary>
/// Interface que define um endpoint para o aplicativo ASP.NET Core.
/// </summary>
public interface IEndpoint
{
    /// <summary>
    /// Método responsável por mapear os endpoints do aplicativo.
    /// </summary>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    /// <remarks>
    /// Este método deve ser implementado para definir os endpoints
    /// específicos do aplicativo, incluindo métodos HTTP, rotas,
    /// parâmetros e respostas.
    /// </remarks>
    static abstract void Map(IEndpointRouteBuilder app);
}
```

.\RealtyHub.ApiService\Configuration.cs

```
namespace RealtyHub.ApiService;

/// <summary>
/// Representa as configurações globais da aplicação.
/// </summary>
/// <remarks>
/// Esta classe contém configurações importantes, como a política de CORS,
/// caminhos para diretórios e configurações de e-mail utilizados pelo serviço.
/// </remarks>
public class Configuration
{
    /// <summary>
    /// Nome da política de CORS utilizada pela aplicação.
    /// </summary>
    public const string CorsPolicyName = "realtyhub";

    /// <summary>
    /// Obtém ou define o caminho para os arquivos de contratos.
    /// </summary>
    public static string ContractsPath { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o caminho para os arquivos de fotos.
    /// </summary>
    public static string PhotosPath { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o caminho para os templates de contrato.
    /// </summary>
    public static string ContractTemplatesPath { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o caminho para os relatórios.
    /// </summary>
    public static string ReportsPath { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o caminho para os logos.
    /// </summary>
    public static string LogosPath { get; set; } = string.Empty;

    /// <summary>
    /// Configurações de e-mail utilizadas pela aplicação.
    /// </summary>
    public static EmailConfiguration EmailSettings { get; set; } = new();

    /// <summary>
    /// Representa as configurações de e-mail.
    /// </summary>
    public class EmailConfiguration
    {
        /// <summary>
        /// Email de origem utilizado para envio das mensagens.
        /// </summary>
        /// <value>O email de origem utilizado para envio das mensagens.</value>
        public string EmailFrom { get; set; } = "realtyhub.br@gmail.com";

        /// <summary>
        /// Senha do email de origem.
        /// </summary>
        /// <value>A senha do email de origem.</value>
        public string EmailPassword { get; set; } = string.Empty;
    }
}
```

.\RealtyHub.ApiService\Data\AppDbContext.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Models;
using RealtyHub.Core.Models;
using System.Reflection;

namespace RealtyHub.ApiService.Data;

/// <summary>
/// Representa o contexto de dados da aplicação, integrando o ASP.NET Identity com as
entidades do sistema.
/// </summary>
public class AppDbContext :
    IdentityDbContext
    <
        User,
        IdentityRole<long>,
        long,
        IdentityUserClaim<long>,
        IdentityUserRole<long>,
        IdentityUserLogin<long>,
        IdentityRoleClaim<long>,
        IdentityUserToken<long>
    >
{
    /// <summary>
    /// Inicializa uma nova instância de <c><see cref="AppDbContext"/></c> utilizando as
opções configuradas para o contexto.
    /// </summary>
    /// <param name="options">As opções configuradas para o contexto.</param>
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {
    }

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see
cref="Customer"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Customer}"/></c> representando os
clientes.</value>
    public DbSet<Customer> Customers { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see
cref="Property"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Property}"/></c> representando os
imóveis.</value>
    public DbSet<Property> Properties { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see cref="Viewing"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Viewing}"/></c> representando as
visitas.</value>
    public DbSet<Viewing> Viewing { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see cref="Offer"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Offer}"/></c> representando as
propostas.</value>
    public DbSet<Offer> Offers { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see cref="Payment"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Payment}"/></c> representando os
pagamentos.</value>
    public DbSet<Payment> Payments { get; set; } = null!;
```

```
    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see
cref="Contract"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Contract}"/></c> representando os
contratos.</value>
    public DbSet<Contract> Contracts { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see
cref="PropertyPhoto"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{PropertyPhoto}"/></c> representando
as fotos de propriedades.</value>
    public DbSet<PropertyPhoto> PropertyPhotos { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see
cref="ContractTemplate"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{ContractTemplate}"/></c>
representando os templates de contrato.</value>
    public DbSet<ContractTemplate> ContractTemplates { get; set; } = null!;

    /// <summary>
    /// Obtém ou define o conjunto de dados para a entidade <c><see
cref="Condominium"/></c>.
    /// </summary>
    /// <value>Uma instância de <c><see cref="DbSet{Condominium}"/></c> representando os
condomínios.</value>
    public DbSet<Condominium> Condominiums { get; set; } = null!;

    /// <summary>
    /// Configura o modelo para o contexto, aplicando as configurações de todas as
entidades contidas no assembly.
    /// </summary>
    /// <param name="builder">O construtor do modelo utilizado para configurar as
entidades.</param>
    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\CondominiumMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Condominium"/></c> para o modelo de
dados.
/// </summary>
public class CondominiumMapping : IEntityTypeConfiguration<Condominium>
{
    /// <summary>
    /// Configura as propriedades e os relacionamentos da entidade <c><see
cref="Condominium"/></c>.
    /// </summary>
    /// <param name="builder">O construtor para configurar a entidade <c><see
cref="Condominium"/></c>.</param>
    public void Configure(EntityTypeBuilder<Condominium> builder)
    {
        builder.ToTable("Condominium");

        builder.HasKey(x => x.Id);

        builder.Property(x => x.Name)
            .IsRequired()
            .HasMaxLength(120);

        builder.Property(x => x.Units)
            .IsRequired();

        builder.Property(x => x.Floors)
            .IsRequired();

        builder.Property(x => x.HasElevator)
            .IsRequired();

        builder.Property(x => x.HasSwimmingPool)
            .IsRequired();

        builder.Property(x => x.HasPartyRoom)
            .IsRequired();

        builder.Property(x => x.HasPlayground)
            .IsRequired();

        builder.Property(x => x.HasFitnessRoom)
            .IsRequired();

        builder.Property(x => x.CondominiumValue)
            .IsRequired();

        builder.Property(x => x.IsActive)
            .IsRequired();

        builder.Property(p => p.CreatedAt)
            .HasDefaultValueSql("NOW()")
            .IsRequired();

        builder.Property(p => p.UpdatedAt)
            .HasDefaultValueSql("NOW()")
            .IsRequired();

        builder.OwnsOne(a => a.Address, address =>
        {
            address.Property(a => a.Street)
                .IsRequired()
                .HasColumnName("Street")
                .HasMaxLength(80);

            address.Property(a => a.Neighborhood)
                .IsRequired()
```

```
                .HasColumnName("Neighborhood")
                .HasMaxLength(80);

            address.Property(a => a.City)
                .IsRequired()
                .HasColumnName("City")
                .HasMaxLength(80);

            address.Property(a => a.Number)
                .IsRequired()
                .HasColumnName("Number");

            address.Property(a => a.State)
                .IsRequired()
                .HasColumnName("State")
                .HasMaxLength(80);

            address.Property(a => a.Country)
                .IsRequired()
                .HasColumnName("Country")
                .HasMaxLength(80);

            address.Property(a => a.ZipCode)
                .IsRequired()
                .HasColumnName("ZipCode")
                .HasMaxLength(20);

            address.Property(a => a.Complement)
                .HasColumnName("Complement")
                .HasMaxLength(80);
        });
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\ContractMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Contract"/></c> para o modelo de
dados.
/// </summary>
public class ContractMapping : IEntityTypeConfiguration<Contract>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="Contract"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="Contract"/></c>.</param>
    public void Configure(EntityTypeBuilder<Contract> builder)
    {
        builder.ToTable("Contract");

        builder.HasKey(c => c.Id);

        builder.Property(c => c.IssueDate);

        builder.Property(c => c.SignatureDate);

        builder.Property(c => c.EffectiveDate);

        builder.Property(c => c.TermEndDate);

        builder.Property(c => c.FileId)
            .IsRequired();

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(c => c.IsActive)
            .IsRequired();

        builder.Property(c => c.CreatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.Property(c => c.UpdatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.HasOne(c => c.Offer)
            .WithOne()
            .HasForeignKey<Contract>(c => c.OfferId);

        builder.HasOne(c => c.Seller)
            .WithMany()
            .HasForeignKey(c => c.SellerId);

        builder.HasOne(c => c.Buyer)
            .WithMany()
            .HasForeignKey(c => c.BuyerId);

        builder.Ignore(c => c.FilePath);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\ContractTemplatesMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="ContractTemplate"/></c> para o
modelo de dados.
/// </summary>
public class ContractTemplatesMapping : IEntityTypeConfiguration<ContractTemplate>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="ContractTemplate"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="ContractTemplate"/></c>.</param>
    public void Configure(EntityTypeBuilder<ContractTemplate> builder)
    {
        builder.ToTable("ContractTemplate");

        builder.HasKey(cc => cc.Id);

        builder.Property(cc => cc.Extension)
            .IsRequired();

        builder.Property(cc => cc.Name)
            .IsRequired();

        builder.Property(cc => cc.Type)
            .IsRequired();

        builder.Property(cc => cc.ShowInPage)
            .IsRequired();

        builder.Ignore(cc => cc.Path);

        builder.HasData(new ContractTemplate
        {
            Id = "a2c16556-5098-4496-ae7a-1f9b6d0e8fcf",
            Extension = ".docx",
            Type = EContractModelType.PJPJ,
            Name = "Modelo de Contrato - PJPJ",
            ShowInPage = false
        });

        builder.HasData(new ContractTemplate
        {
            Id = "f7581a63-f4f0-4881-b6ed-6a4100b4182e",
            Extension = ".docx",
            Type = EContractModelType.PFPF,
            Name = "Modelo de Contrato - PFPF",
            ShowInPage = false
        });

        builder.HasData(new ContractTemplate
        {
            Id = "2f4c556d-6850-4b3d-afe9-d7c2bd282718",
            Extension = ".docx",
            Type = EContractModelType.PFPJ,
            Name = "Modelo de Contrato - PFPJ",
            ShowInPage = false
        });

        builder.HasData(new ContractTemplate
        {
            Id = "fd7ed50d-8f42-4288-8877-3cb8095370e7",
            Extension = ".docx",
            Type = EContractModelType.PJPF,
            Name = "Modelo de Contrato - PJPF",
```

```
            ShowInPage = false
        });

        builder.HasData(new ContractTemplate
        {
            Id = "2824aec3-3219-4d81-a97a-c3b80ca72844",
            Extension = ".pdf",
            Type = EContractModelType.None,
            Name = "Modelo Padrão",
            ShowInPage = true
        });
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\CustomerMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Customer"/></c> para o modelo de
dados.
/// </summary>
public class CustomerMapping : IEntityTypeConfiguration<Customer>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="Customer"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="Customer"/></c>.</param>
    public void Configure(EntityTypeBuilder<Customer> builder)
    {
        builder.ToTable("Customer");

        builder.HasKey(c => c.Id);

        builder.Property(c => c.Name)
            .IsRequired()
            .HasMaxLength(80);

        builder.Property(c => c.Email)
            .IsRequired()
            .HasMaxLength(50);

        builder.Property(c => c.Phone)
            .IsRequired()
            .HasMaxLength(30);

        builder.Property(c => c.PersonType)
            .IsRequired();

        builder.Property(c => c.DocumentNumber)
            .IsRequired()
            .HasMaxLength(20);

        builder.Property(c => c.Rg)
            .HasMaxLength(20);

        builder.Property(c => c.RgIssueDate);

        builder.Property(c => c.IssuingAuthority)
            .HasMaxLength(80);

        builder.Property(c => c.Nationality)
            .HasMaxLength(80)
            .IsRequired();

        builder.Property(c => c.MaritalStatus)
            .IsRequired();

        builder.Property(c => c.Occupation)
            .HasMaxLength(80)
            .IsRequired();

        builder.Property(c => c.BusinessName)
            .HasMaxLength(80);

        builder.Property(c => c.IsActive)
            .HasDefaultValue(true);

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(c => c.IsActive)
```

```
            .HasDefaultValue(true);

        builder.Property(c => c.CreatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.Property(c => c.UpdatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.OwnsOne(a => a.Address, address =>
        {
            address.Property(a => a.Street)
                .IsRequired()
                .HasColumnName("Street")
                .HasMaxLength(80);

            address.Property(a => a.Neighborhood)
                .IsRequired()
                .HasColumnName("Neighborhood")
                .HasMaxLength(80);

            address.Property(a => a.City)
                .IsRequired()
                .HasColumnName("City")
                .HasMaxLength(80);

            address.Property(a => a.Number)
                .IsRequired()
                .HasColumnName("Number");

            address.Property(a => a.State)
                .IsRequired()
                .HasColumnName("State")
                .HasMaxLength(80);

            address.Property(a => a.Country)
                .IsRequired()
                .HasColumnName("Country")
                .HasMaxLength(80);

            address.Property(a => a.ZipCode)
                .IsRequired()
                .HasColumnName("ZipCode")
                .HasMaxLength(20);

            address.Property(a => a.Complement)
                .HasColumnName("Complement")
                .HasMaxLength(80);
        });
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityRoleClaimMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="IdentityRoleClaim{TKey}"/></c> para
o modelo de dados.
/// </summary>
public class IdentityRoleClaimMapping : IEntityTypeConfiguration<IdentityRoleClaim<long>>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see
cref="IdentityRoleClaim{TKey}"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see
cref="IdentityRoleClaim{TKey}"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<IdentityRoleClaim<long>> builder)
    {
        builder.ToTable("IdentityRoleClaim");

        builder.HasKey(u => u.Id);

        builder.Property(u => u.ClaimType).HasMaxLength(255);
        builder.Property(u => u.ClaimValue).HasMaxLength(255);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityRoleMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="IdentityRole{TKey}"/></c> para o
modelo de dados.
/// </summary>
public class IdentityRoleMapping : IEntityTypeConfiguration<IdentityRole<long>>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see cref="IdentityRole{TKey}"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see
cref="IdentityRole{TKey}"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<IdentityRole<long>> builder)
    {
        builder.ToTable("IdentityRole");

        builder.HasKey(u => u.Id);

        builder.HasIndex(u => u.NormalizedName).IsUnique();

        builder.Property(u => u.ConcurrencyStamp).IsConcurrencyToken();
        builder.Property(u => u.Name).HasMaxLength(256);
        builder.Property(u => u.NormalizedName).HasMaxLength(256);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityUserClaimMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="IdentityUserClaim{TKey}"/></c> para
o modelo de dados.
/// </summary>
public class IdentityUserClaimMapping : IEntityTypeConfiguration<IdentityUserClaim<long>>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see
cref="IdentityUserClaim{TKey}"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see
cref="IdentityUserClaim{TKey}"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<IdentityUserClaim<long>> builder)
    {
        builder.ToTable("IdentityClaim");

        builder.HasKey(u => u.Id);

        builder.Property(u => u.ClaimType).HasMaxLength(255);
        builder.Property(u => u.ClaimValue).HasMaxLength(255);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityUserLoginMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="IdentityUserLogin{TKey}"/></c> para
o modelo de dados.
/// </summary>
public class IdentityUserLoginMapping : IEntityTypeConfiguration<IdentityUserLogin<long>>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see
cref="IdentityUserLogin{TKey}"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see
cref="IdentityUserLogin{TKey}"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<IdentityUserLogin<long>> builder)
    {
        builder.ToTable("IdentityUserLogin");

        builder.HasKey(u => new { u.LoginProvider, u.ProviderKey });
        builder.Property(u => u.LoginProvider).HasMaxLength(128);
        builder.Property(u => u.ProviderKey).HasMaxLength(128);
        builder.Property(u => u.ProviderDisplayName).HasMaxLength(255);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityUserMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.ApiService.Models;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="User"/></c> para o modelo de dados.
/// </summary>
public class IdentityUserMapping : IEntityTypeConfiguration<User>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see cref="User"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see cref="User"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.ToTable("IdentityUser");

        builder.HasKey(u => u.Id);

        builder.HasIndex(u => u.NormalizedUserName).IsUnique();
        builder.HasIndex(u => u.NormalizedEmail).IsUnique();

        builder.Property(u => u.Creci);
        builder.Property(u => u.GivenName).HasMaxLength(100);
        builder.Property(u => u.Email).HasMaxLength(180);
        builder.Property(u => u.NormalizedEmail).HasMaxLength(180);
        builder.Property(u => u.UserName).HasMaxLength(160);
        builder.Property(u => u.NormalizedUserName).HasMaxLength(180);
        builder.Property(u => u.PhoneNumber).HasMaxLength(20);
        builder.Property(u => u.ConcurrencyStamp).IsConcurrencyToken();

        builder.HasMany<IdentityUserClaim<long>>()
            .WithOne()
            .HasForeignKey(u => u.UserId)
            .IsRequired();

        builder.HasMany<IdentityUserLogin<long>>()
            .WithOne()
            .HasForeignKey(u => u.UserId)
            .IsRequired();

        builder.HasMany<IdentityUserToken<long>>()
            .WithOne()
            .HasForeignKey(u => u.UserId)
            .IsRequired();

        builder.HasMany<IdentityUserRole<long>>()
            .WithOne()
            .HasForeignKey(u => u.UserId)
            .IsRequired();
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityUserRoleMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="IdentityUserRole{TKey}"/></c> para o
modelo de dados.
/// </summary>
public class IdentityUserRoleMapping : IEntityTypeConfiguration<IdentityUserRole<long>>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see
cref="IdentityUserRole{TKey}"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see
cref="IdentityUserRole{TKey}"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<IdentityUserRole<long>> builder)
    {
        builder.ToTable("IdentityUserRole");

        builder.HasKey(u => new { u.UserId, u.RoleId });
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\Identity\IdentityUserTokenMapping.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace RealtyHub.ApiService.Data.Mappings.Identity;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="IdentityUserToken{TKey}"/></c> para
o modelo de dados.
/// </summary>
public class IdentityUserTokenMapping : IEntityTypeConfiguration<IdentityUserToken<long>>
{
    /// <summary>
    /// Configura as propriedades da entidade <c><see
cref="IdentityUserToken{TKey}"/></c>.
    /// </summary>
    /// <param name="builder">
    /// O construtor utilizado para configurar a entidade <c><see
cref="IdentityUserToken{TKey}"/></c>.
    /// </param>
    public void Configure(EntityTypeBuilder<IdentityUserToken<long>> builder)
    {
        builder.ToTable("IdentityUserToken");

        builder.HasKey(u => new { u.UserId, u.LoginProvider, u.Name });
        builder.Property(u => u.LoginProvider).HasMaxLength(120);
        builder.Property(u => u.Name).HasMaxLength(180);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\OfferMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Offer"/></c> para o modelo de dados.
/// </summary>
public class OfferMapping : IEntityTypeConfiguration<Offer>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="Offer"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="Offer"/></c>.</param>
    public void Configure(EntityTypeBuilder<Offer> builder)
    {
        builder.ToTable("Offer");

        builder.HasKey(x => x.Id);

        builder.Property(x => x.SubmissionDate)
            .IsRequired();

        builder.Property(x => x.Amount)
            .IsRequired();

        builder.Property(x => x.OfferStatus)
            .IsRequired();

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(x => x.CreatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.Property(x => x.UpdatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.HasOne(x => x.Property)
            .WithMany()
            .HasForeignKey(x => x.PropertyId);

        builder.HasOne(x => x.Buyer)
            .WithMany()
            .HasForeignKey(x => x.BuyerId);

        builder.HasOne(x => x.Contract)
            .WithOne(x => x.Offer)
            .HasForeignKey<Contract>(x => x.OfferId);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\PaymentMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Payment"/></c> para o modelo de
dados.
/// </summary>
public class PaymentMapping : IEntityTypeConfiguration<Payment>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="Payment"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="Payment"/></c>.</param>
    public void Configure(EntityTypeBuilder<Payment> builder)
    {
        builder.ToTable("Payment");

        builder.HasKey(x => x.Id);

        builder.Property(x => x.Amount)
            .IsRequired();

        builder.Property(x => x.PaymentType)
            .IsRequired();

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(x => x.IsActive)
            .HasDefaultValue(true);

        builder.Property(x => x.CreatedAt)
            .HasDefaultValueSql("NOW()");

        builder.Property(x => x.UpdatedAt)
            .HasDefaultValueSql("NOW()");

        builder.HasOne(p => p.Offer)
            .WithMany(o => o.Payments)
            .HasForeignKey(p => p.OfferId);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\PropertyMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Property"/></c> para o modelo de
dados.
/// </summary>
public class PropertyMapping : IEntityTypeConfiguration<Property>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="Property"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="Property"/></c>.</param>
    public void Configure(EntityTypeBuilder<Property> builder)
    {
        builder.ToTable("Property");

        builder.HasKey(p => p.Id);

        builder.Property(p => p.Title)
            .IsRequired()
            .HasMaxLength(120);

        builder.Property(p => p.Description)
            .IsRequired();

        builder.Property(p => p.Price)
            .IsRequired();

        builder.Property(p => p.PropertyType)
            .IsRequired();

        builder.Property(p => p.Bedroom)
            .IsRequired();

        builder.Property(p => p.Bathroom)
            .IsRequired();

        builder.Property(p => p.Garage)
            .IsRequired();

        builder.Property(p => p.Area)
            .IsRequired();

        builder.Property(p => p.TransactionsDetails);

        builder.Property(p => p.IsNew)
            .IsRequired();

        builder.Property(p => p.RegistryNumber)
            .IsRequired();

        builder.Property(p => p.RegistryRecord)
            .IsRequired();

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(p => p.IsActive)
            .IsRequired();

        builder.Property(p => p.ShowInHome)
            .IsRequired();

        builder.HasOne(p => p.Seller)
            .WithMany(c => c.Properties)
            .HasForeignKey(p => p.SellerId);
```

```
        builder.HasOne(p => p.Condominium)
            .WithMany(c => c.Properties)
            .HasForeignKey(p => p.CondominiumId);

        builder.Property(p => p.CreatedAt)
            .HasDefaultValueSql("NOW()")
            .IsRequired();

        builder.Property(p => p.UpdatedAt)
            .HasDefaultValueSql("NOW()")
            .IsRequired();

        builder.OwnsOne(a => a.Address, address =>
        {
            address.Property(a => a.Street)
                .IsRequired()
                .HasColumnName("Street")
                .HasMaxLength(80);

            address.Property(a => a.Neighborhood)
                .IsRequired()
                .HasColumnName("Neighborhood")
                .HasMaxLength(80);

            address.Property(a => a.Number)
                .IsRequired()
                .HasColumnName("Number");

            address.Property(a => a.City)
                .IsRequired()
                .HasColumnName("City")
                .HasMaxLength(80);

            address.Property(a => a.State)
                .IsRequired()
                .HasColumnName("State")
                .HasMaxLength(80);

            address.Property(a => a.Country)
                .IsRequired()
                .HasColumnName("Country")
                .HasMaxLength(80);

            address.Property(a => a.ZipCode)
                .IsRequired()
                .HasColumnName("ZipCode")
                .HasMaxLength(20);

            address.Property(a => a.Complement)
                .HasColumnName("Complement")
                .HasMaxLength(80);
        });

        builder.HasMany(p => p.PropertyPhotos)
            .WithOne(pi => pi.Property)
            .HasForeignKey(pi => pi.PropertyId);
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\PropertyPhotosMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="PropertyPhoto"/></c> para o modelo
de dados.
/// </summary>
public class PropertyPhotosMapping : IEntityTypeConfiguration<PropertyPhoto>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="PropertyPhoto"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="PropertyPhoto"/></c>.</param>
    public void Configure(EntityTypeBuilder<PropertyPhoto> builder)
    {
        builder.ToTable("PropertyPhotos");

        builder.HasKey(pi => pi.Id);

        builder.Property(pi => pi.Extension)
            .IsRequired();

        builder.Property(pi => pi.IsThumbnail)
            .IsRequired();

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(pi => pi.IsActive)
            .IsRequired();

        builder.Property(pi => pi.PropertyId)
            .IsRequired();

        builder.Property(pi => pi.CreatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");

        builder.Property(pi => pi.UpdatedAt)
            .IsRequired()
            .HasDefaultValueSql("NOW()");
    }
}
```

.\RealtyHub.ApiService\Data\Mappings\ViewingMapping.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Data.Mappings;

/// <summary>
/// Configura o mapeamento da entidade <c><see cref="Viewing"/></c> para o modelo de
dados.
/// </summary>
public class ViewingMapping : IEntityTypeConfiguration<Viewing>
{
    /// <summary>
    /// Configura as propriedades e relacionamentos da entidade <c><see
cref="Viewing"/></c>.
    /// </summary>
    /// <param name="builder">O construtor utilizado para configurar a entidade <c><see
cref="Viewing"/></c>.</param>
    public void Configure(EntityTypeBuilder<Viewing> builder)
    {
        builder.ToTable("Viewing");

        builder.HasKey(v => v.Id);

        builder.Property(v => v.ViewingDate)
            .IsRequired();

        builder.Property(v => v.ViewingStatus)
            .IsRequired();

        builder.HasOne(v => v.Buyer)
            .WithMany()
            .HasForeignKey(v => v.BuyerId);

        builder.HasOne(v => v.Property)
            .WithMany()
            .HasForeignKey(v => v.PropertyId);

        builder.Property(c => c.UserId)
            .IsRequired();

        builder.Property(v => v.CreatedAt)
            .HasDefaultValueSql("NOW()")
            .IsRequired();

        builder.Property(v => v.UpdatedAt)
            .HasDefaultValueSql("NOW()")
            .IsRequired();
    }
}
```

.\RealtyHub.ApiService\Endpoints\Condominiums\CreateCondominiumEndpoint.cs

```
using System.Security.Claims;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Condominiums;

/// <summary>
/// Endpoint responsável por criar novos condomínios.
/// </summary>
/// <remarks>
/// Implementa <see cref="IEndpoint"/> para mapear a rota de criação de condomínios.
/// </remarks>
public class CreateCondominiumEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para criar um condomínio.
    /// </summary>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/", HandlerAsync)
            .WithName("Condominiums: Create")
            .WithSummary("Cria um novo condomínio")
            .WithDescription("Cria um novo condomínio")
            .WithOrder(1)
            .Produces<Response<Condominium?>>(StatusCodes.Status201Created)
            .Produces<Response<Condominium?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para criar um condomínio.
    /// </summary>
    /// <remarks>
    /// O método atribui o ID do usuário autenticado a requisição,
    /// e invoca o handler para criar o condomínio.
    /// </remarks>
    /// <param name="user">Instância de <c><see cref="ClaimsPrincipal"/></c> contendo os
dados do usuário autenticado.</param>
    /// <param name="handler">Instância de <c><see cref="ICondominiumHandler"/></c> que
executa a criação de condomínios.</param>
    /// <param name="request">Objeto <c><see cref="Condominium"/></c> contendo os dados
do novo condomínio.</param>
    /// <returns>
    /// Um <c><see cref="IResult"/></c> representando o resultado da operação:
    /// <para>- HTTP 201 Created com o recurso criado, se a criação for
bem-sucedida;</para>
    /// <para>- HTTP 400 BadRequest com os detalhes, em caso de erro.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICondominiumHandler handler,
        Condominium request)
    {
        request.UserId = user.Identity?.Name ?? string.Empty;
        Response<Condominium?> result = await handler.CreateAsync(request);

        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Condominiums\DeleteCondominiumEndpoint.cs

```
using System.Security.Claims;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Condominiums;

/// <summary>
/// Endpoint responsável por deletar um condomínio.
/// </summary>
/// <remarks>
/// Implementa <c><see cref="IEndpoint"/></c> para mapear a rota de exclusão de
condomínios.
/// </remarks>
public class DeleteCondominiumEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para a exclusão de condomínios.
    /// </summary>
    /// <remarks>
    /// Este método registra a rota HTTP DELETE que aceita um parâmetro de identificação
do condomínio,
    /// configurando os códigos de resposta HTTP esperados.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app) =>
        app.MapDelete("/{id:long}", HandlerAsync)
            .WithName("Condominiums: Delete")
            .WithSummary("Deleta um condomínio")
            .WithDescription("Deleta um condomínio")
            .WithOrder(4)
            .Produces(StatusCodes.Status204NoContent)
            .Produces<Response<Condominium?>>(StatusCodes.Status404NotFound);

    /// <summary>
    /// Manipulador da rota para a deleção de um condomínio.
    /// </summary>
    /// <remarks>
    /// O método atribui o ID do usuário autenticado à requisição,
    /// e invoca o handler para deletar o condomínio.
    /// </remarks>
    /// <param name="user">Objeto <c><see cref="ClaimsPrincipal"/></c> que contém os
dados do usuário autenticado.</param>
    /// <param name="handler">Instância de <c><see cref="ICondominiumHandler"/></c> que
executa a exclusão do condomínio.</param>
    /// <param name="id">ID do condomínio a ser deletado.</param>
    /// <returns>
    /// Um objeto <c><see cref="IResult"/></c> representando o resultado da operação:
    /// <para>- HTTP 204 No Content, se a exclusão for bem-sucedida;</para>
    /// <para>- HTTP 404 Not Found, se o condomínio não for encontrado.</para>
    /// <para>- HTTP 400 BadRequest com os detalhes, em caso de erro.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICondominiumHandler handler,
        long id)
    {
        var request = new DeleteCondominiumRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.DeleteAsync(request);

        return result.IsSuccess
            ? Results.NoContent()
            : Results.NotFound(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Condominiums\GetAllCondominiumsEndpoint.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Condominiums;

/// <summary>
/// Endpoint responsável por obter todos os condomínios.
/// </summary>
/// <remarks>
/// Implementa <c><see cref="IEndpoint"/></c> para mapear a rota de listagem de
condomínios.
/// </remarks>
public class GetAllCondominiumsEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para obter todos os condomínios.
    /// </summary>
    /// <remarks>
    /// Registra a rota responsável por retornar a lista de condomínios com paginação e
opção de filtro.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app) =>
        app.MapGet("/", HandlerAsync)
            .WithName("Condominiums: Get All")
            .WithSummary("Obtém todos os condomínios")
            .WithDescription("Obtém todos os condomínios")
            .WithOrder(5)
            .Produces<PagedResponse<List<Condominium>>>()
            .Produces<PagedResponse<List<Condominium>>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para obter todos os condomínios.
    /// </summary>
    /// <remarks>
    /// Aplica paginação, possibilita filtrar pelo campo desejado e retorna apenas
condomínios ativos.
    /// </remarks>
    /// <param name="user">Objeto <c><see cref="ClaimsPrincipal"/></c> que contém os
dados do usuário autenticado.</param>
    /// <param name="handler">Instância de <c><see cref="ICondominiumHandler"/></c> que
executa a busca de condomínios.</param>
    /// <param name="filterBy">Campo utilizado para filtrar os condomínios.</param>
    /// <param name="pageNumber">Número da página solicitada.</param>
    /// <param name="pageSize">Quantidade de itens por página.</param>
    /// <param name="searchTerm">Termo utilizado na busca de condomínios.</param>
    /// <returns>
    /// Um objeto <c><see cref="IResult"/></c> representando o resultado da operação:
    /// <para>- HTTP 200 OK com a lista paginada, se a busca for bem-sucedida;</para>
    /// <para>- HTTP 400 BadRequest com os detalhes, em caso de erro.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICondominiumHandler handler,
        [FromQuery] string filterBy = "",
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize,
        [FromQuery] string searchTerm = "")
    {
        var request = new GetAllCondominiumsRequest
        {
            PageNumber = pageNumber,
            PageSize = pageSize,
            SearchTerm = searchTerm,
            FilterBy = filterBy
        };
```

```
        var result = await handler.GetAllAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Condominiums\GetCondominiumByIdEndpoint.cs

```
using System.Security.Claims;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Condominiums;

/// <summary>
/// Endpoint responsável por recuperar um condomínio específico pelo seu ID.
/// </summary>
/// <remarks>
/// Implementa <c><see cref="IEndpoint"/></c> para mapear a rota de obtenção de um
condomínio.
/// </remarks>
public class GetCondominiumByIdEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar um condomínio pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Este método registra a rota HTTP GET que aceita um parâmetro de identificação do
condomínio,
    /// configurando os códigos de resposta HTTP esperados.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app) =>
        app.MapGet("/{id:long}", HandlerAsync)
            .WithName("Condominiums: Get by id")
            .WithSummary("Recupera um condomínio")
            .WithDescription("Recupera um condomínio baseado no seu ID")
            .WithOrder(3)
            .Produces<Response<Condominium?>>()
            .Produces<Response<Condominium?>>(StatusCodes.Status404NotFound);

    /// <summary>
    /// Manipulador da rota para a obtenção de um condomínio.
    /// </summary>
    /// <remarks>
    /// O método atribui o ID do usuário autenticado à requisição,
    /// e invoca o handler para buscar o condomínio.
    /// </remarks>
    /// <param name="user">Objeto <c><see cref="ClaimsPrincipal"/></c> que contém os
dados do usuário autenticado.</param>
    /// <param name="handler">Instância de <c><see cref="ICondominiumHandler"/></c> que
executa a busca do condomínio.</param>
    /// <param name="id">ID do condomínio a ser recuperado.</param>
    /// <returns>
    /// Um objeto <c><see cref="IResult"/></c> representando o resultado da operação:
    /// <para>- HTTP 200 OK com os detalhes do condomínio, se encontrado;</para>
    /// <para>- HTTP 404 Not Found, se o condomínio não for encontrado.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICondominiumHandler handler,
        long id)
    {
        var request = new GetCondominiumByIdRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.GetByIdAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.NotFound(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Condominiums\UpdateCondominiumEndpoint.cs

```
using System.Security.Claims;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Condominiums;

/// <summary>
/// Endpoint responsável por atualizar os dados de um condomínio existente.
/// </summary>
/// <remarks>
/// Implementa <c><see cref="IEndpoint"/></c> para mapear a rota de atualização de
condomínios.
/// </remarks>
public class UpdateCondominiumEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para atualizar um condomínio.
    /// </summary>
    /// <remarks>
    /// Este método registra a rota HTTP PUT que aceita um parâmetro de identificação do
condomínio
    /// e os dados atualizados, configurando os códigos de resposta HTTP esperados.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app) =>
        app.MapPut("/{id:long}", HandlerAsync)
            .WithName("Condominiums: Update")
            .WithSummary("Atualiza um condomínio")
            .WithDescription("Atualiza os dados de um condomínio existente")
            .WithOrder(2)
            .Produces<Response<Condominium?>>()
            .Produces<Response<Condominium?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota para a atualização de um condomínio.
    /// </summary>
    /// <remarks>
    /// Este método extrai o ID do condomínio e os dados atualizados da requisição,
    /// atribui o ID do usuário autenticado à requisição e invoca o handler para realizar
a atualização.
    /// </remarks>
    /// <param name="user">Objeto <c><see cref="ClaimsPrincipal"/></c> que contém os
dados do usuário autenticado.</param>
    /// <param name="handler">Instância de <c><see cref="ICondominiumHandler"/></c> que
executa a atualização do condomínio.</param>
    /// <param name="request">Objeto <c><see cref="Condominium"/></c> contendo os dados
atualizados do condomínio.</param>
    /// <param name="id">ID do condomínio a ser atualizado.</param>
    /// <returns>
    /// Um objeto <c><see cref="IResult"/></c> representando o resultado da operação:
    /// <para>- HTTP 200 OK com os dados atualizados do condomínio, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICondominiumHandler handler,
        Condominium request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;

        var result = await handler.UpdateAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\ContractsTemplates\GetAllContractTemplatesEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.ContractsTemplates;

/// <summary>
/// Endpoint responsável por recuperar todos os modelos de contrato.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
modelos de contrato.
/// </remarks>
public class GetAllContractTemplatesEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todos os modelos de contrato.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista de todos os modelos de contrato
disponíveis.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/", HandlerAsync)
            .WithName("ContractTemplates: Get All ContractTemplates")
            .WithSummary("Recupera todos os modelos de contrato")
            .WithDescription("Recupera todos os modelos de contrato")
            .WithOrder(4)
            .Produces<Response<ContractTemplate?>>()
            .Produces<Response<ContractTemplate?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que recebe a requisição para recuperar todos os modelos de
contrato.
    /// </summary>
    /// <remarks>
    /// Este método chama o handler para buscar todos os modelos de contrato disponíveis
e retorna o resultado.
    /// </remarks>
    /// <param name="handler">Instância de <see cref="IContractTemplateHandler"/>
responsável pelas operações relacionadas a modelos de contrato.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista de modelos de contrato, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    public static async Task<IResult> HandlerAsync(
        IContractTemplateHandler handler)
    {
        var result = await handler.GetAllAsync();

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Contracts\CreateContractEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Contracts;

/// <summary>
/// Endpoint responsável por criar novos contratos.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de criação de
contratos.
/// </remarks>
public class CreateContractEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para criar um contrato.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe o objeto <see cref="Contract"/> e invoca o
manipulador para criar o contrato.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/", HandlerAsync)
            .WithName("Contracts: Create")
            .WithSummary("Cria um novo contrato")
            .WithDescription("Cria um novo contrato")
            .WithOrder(1)
            .Produces<Response<Contract?>>(StatusCodes.Status201Created)
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para criar um contrato.
    /// </summary>
    /// <remarks>
    /// Este método extrai as informações do contrato do corpo da requisição, associa o
ID do usuário autenticado,
    /// e chama o handler para criar o novo contrato. Retorna Created com os dados do
contrato criado ou BadRequest em caso de falha.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IContractHandler"/> responsável
pelas operações relacionadas a contratos.</param>
    /// <param name="request">Objeto <see cref="Contract"/> contendo os dados para
criação do contrato.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 201 Created com os dados do contrato criado, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IContractHandler handler,
        Contract request)
    {
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.CreateAsync(request);

        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Contracts\DeleteContractEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Contracts;

/// <summary>
/// Endpoint responsável por deletar um contrato.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de exclusão lógica
de contratos.
/// </remarks>
public class DeleteContractEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para deletar um contrato.
    /// </summary>
    /// <remarks>
    /// Registra a rota DELETE que espera um parâmetro numérico (ID) na URL e chama o
manipulador para executar a operação.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapDelete("/{id:long}", HandlerAsync)
            .WithName("Contracts: Delete")
            .WithSummary("Deleta um contrato")
            .WithDescription("Deleta um contrato")
            .WithOrder(3)
            .Produces<Response<Contract?>>(StatusCodes.Status204NoContent)
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para deletar um contrato.
    /// </summary>
    /// <remarks>
    /// Este método extrai o ID do contrato da rota, associa o ID do usuário autenticado
à requisição,
    /// e chama o handler para realizar a exclusão lógica do contrato. Retorna NoContent
se a operação for bem-sucedida
    /// ou BadRequest em caso de falha.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IContractHandler"/> responsável
pelas operações relacionadas a contratos.</param>
    /// <param name="id">ID do contrato a ser deletado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 204 No Content, se a exclusão for bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IContractHandler handler,
        long id)
    {
        var request = new DeleteContractRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.DeleteAsync(request);

        return result.IsSuccess
            ? Results.NoContent()
            : Results.BadRequest(result);
    }
```

```
}
```

.\RealtyHub.ApiService\Endpoints\Contracts\GetAllContractsEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Contracts;

/// <summary>
/// Endpoint responsável por recuperar todos os contratos.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> e mapeia a rota de listagem de
contratos.
/// </remarks>
public class GetAllContractsEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todos os contratos.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de contratos associados ao
usuário autenticado.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/", HandlerAsync)
            .WithName("Contracts: Get All")
            .WithSummary("Recupera todos os contratos")
            .WithDescription("Recupera todos os contratos")
            .WithOrder(5)
            .Produces<PagedResponse<List<Contract>?>>()
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para recuperar todos os contratos.
    /// </summary>
    /// <remarks>
    /// Este método aplica paginação e retorna apenas os contratos associados ao usuário
autenticado.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IContractHandler"/> responsável
pelas operações relacionadas a contratos.</param>
    /// <param name="pageNumber">Número da página solicitada.</param>
    /// <param name="pageSize">Quantidade de itens por página.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando o resultado da operação:
    /// <para>- HTTP 200 OK com a lista paginada de contratos, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IContractHandler handler,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize)
    {
        var request = new GetAllContractsRequest
        {
            UserId = user.Identity?.Name ?? string.Empty,
            PageNumber = pageNumber,
            PageSize = pageSize
        };

        var result = await handler.GetAllAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
```

```
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Contracts\GetContractByIdEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Contracts;

/// <summary>
/// Endpoint responsável por recuperar um contrato específico pelo seu ID.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção de um
contrato pelo ID.
/// </remarks>
public class GetContractByIdEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar um contrato pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que espera um parâmetro numérico (ID) e chama o manipulador
para retornar o contrato correspondente.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/{id:long}", HandlerAsync)
            .WithName("Contracts: Get by Id")
            .WithSummary("Recupera um contrato")
            .WithDescription("Recupera um contrato baseado no seu ID")
            .WithOrder(4)
            .Produces<Response<Contract?>>()
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para recuperar um contrato pelo seu
ID.
    /// </summary>
    /// <remarks>
    /// Este método constrói uma requisição com o ID extraído da rota e o usuário
autenticado,
    /// chama o handler para obter o contrato e retorna uma resposta adequada com base no
resultado.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IContractHandler"/> responsável
pelas operações relacionadas a contratos.</param>
    /// <param name="id">ID do contrato a ser recuperado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes do contrato, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IContractHandler handler,
        long id)
    {
        var request = new GetContractByIdRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.GetByIdAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
```

```
    }
}
```

.\RealtyHub.ApiService\Endpoints\Contracts\UpdateContractEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Contracts;

/// <summary>
/// Endpoint responsável por atualizar os dados de um contrato existente.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de atualização de
contratos.
/// </remarks>
public class UpdateContractEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para atualizar um contrato.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que espera um parâmetro numérico (ID) e os dados atualizados
do contrato,
    /// chamando o manipulador para executar a operação.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}", HandlerAsync)
            .WithName("Contracts: Update")
            .WithSummary("Atualiza um contrato")
            .WithDescription("Atualiza um contrato")
            .WithOrder(2)
            .Produces<Response<Contract?>>()
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para atualizar um contrato.
    /// </summary>
    /// <remarks>
    /// Este método extrai o ID do contrato e os dados atualizados da requisição,
    /// associa o ID do usuário autenticado à requisição e chama o handler para realizar
a atualização.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IContractHandler"/> responsável
pelas operações relacionadas a contratos.</param>
    /// <param name="request">Objeto <see cref="Contract"/> contendo os dados atualizados
do contrato.</param>
    /// <param name="id">ID do contrato a ser atualizado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os dados atualizados do contrato, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IContractHandler handler,
        Contract request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.UpdateAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Customers\CreateCustomerEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Customers;

/// <summary>
/// Endpoint responsável por criar novos clientes.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de criação de
clientes.
/// </remarks>
public class CreateCustomerEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para criar um cliente.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe os dados do cliente e chama o manipulador para
criar o cliente.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/", HandlerAsync)
            .WithName("Customers: Create")
            .WithSummary("Cria um novo cliente")
            .WithDescription("Cria um novo cliente")
            .WithOrder(1)
            .Produces<Response<Customer?>>(StatusCodes.Status201Created)
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para criar um cliente.
    /// </summary>
    /// <remarks>
    /// Este método extrai as informações do cliente do corpo da requisição, associa o ID
do usuário autenticado,
    /// e chama o handler para criar o novo cliente. Retorna Created com os dados do
cliente criado ou BadRequest em caso de falha.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="ICustomerHandler"/> responsável
pelas operações relacionadas a clientes.</param>
    /// <param name="request">Objeto <see cref="Customer"/> contendo os dados para
criação do cliente.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 201 Created com os dados do cliente criado, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICustomerHandler handler,
        Customer request)
    {
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.CreateAsync(request);

        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Customers\DeleteCustomerEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Customers;

/// <summary>
/// Endpoint responsável por deletar um cliente.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de exclusão lógica
de clientes.
/// </remarks>
public class DeleteCustomerEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para deletar um cliente.
    /// </summary>
    /// <remarks>
    /// Registra a rota DELETE que espera um parâmetro numérico (ID) na URL e chama o
manipulador para executar a operação.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapDelete("/{id:long}", HandlerAsync)
            .WithName("Customers: Delete")
            .WithSummary("Deleta um cliente")
            .WithDescription("Deleta um cliente")
            .WithOrder(3)
            .Produces<Response<Customer?>>(StatusCodes.Status204NoContent)
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para deletar um cliente.
    /// </summary>
    /// <remarks>
    /// Este método extrai o ID do cliente da rota, associa o ID do usuário autenticado à
requisição,
    /// e chama o handler para realizar a exclusão lógica do cliente. Retorna NoContent
se a operação for bem-sucedida
    /// ou BadRequest em caso de falha.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="ICustomerHandler"/> responsável
pelas operações relacionadas a clientes.</param>
    /// <param name="id">ID do cliente a ser deletado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 204 No Content, se a exclusão for bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICustomerHandler handler,
        long id)
    {
        var request = new DeleteCustomerRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.DeleteAsync(request);

        return result.IsSuccess
            ? Results.NoContent()
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Customers\GetAllCustomersEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Customers;

/// <summary>
/// Endpoint responsável por recuperar todos os clientes.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
clientes.
/// </remarks>
public class GetAllCustomersEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todos os clientes.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de clientes associados ao
usuário autenticado.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/", HandlerAsync)
            .WithName("Customers: Get All")
            .WithSummary("Recupera todos os clientes")
            .WithDescription("Recupera todos os clientes")
            .WithOrder(5)
            .Produces<PagedResponse<List<Customer>?>>()
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para recuperar todos os clientes.
    /// </summary>
    /// <remarks>
    /// Este método aplica paginação e permite buscar clientes com base em um termo de
pesquisa.
    /// Retorna apenas os clientes associados ao usuário autenticado.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="ICustomerHandler"/> responsável
pelas operações relacionadas a clientes.</param>
    /// <param name="pageNumber">Número da página solicitada.</param>
    /// <param name="pageSize">Quantidade de itens por página.</param>
    /// <param name="searchTerm">Termo utilizado para buscar clientes.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de clientes, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICustomerHandler handler,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize,
        [FromQuery] string searchTerm = "")
    {
        var request = new GetAllCustomersRequest
        {
            UserId = user.Identity?.Name ?? string.Empty,
            PageNumber = pageNumber,
            PageSize = pageSize,
            SearchTerm = searchTerm
        };
        var result = await handler.GetAllAsync(request);
```

```
        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Customers\GetCustomerByIdEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Customers;

/// <summary>
/// Endpoint responsável por recuperar um cliente específico pelo seu ID.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção de um
cliente pelo ID.
/// </remarks>
public class GetCustomerByIdEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar um cliente pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que espera um parâmetro numérico (ID) e chama o manipulador
para retornar o cliente correspondente.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/{id:long}", HandlerAsync)
            .WithName("Customers: Get by Id")
            .WithSummary("Recupera um cliente")
            .WithDescription("Recupera um cliente")
            .WithOrder(4)
            .Produces<Response<Customer?>>()
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para recuperar um cliente pelo seu
ID.
    /// </summary>
    /// <remarks>
    /// Este método constrói uma requisição com o ID extraído da rota e o usuário
autenticado,
    /// chama o handler para obter o cliente e retorna uma resposta adequada com base no
resultado.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="ICustomerHandler"/> responsável
pelas operações relacionadas a clientes.</param>
    /// <param name="id">ID do cliente a ser recuperado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes do cliente, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICustomerHandler handler,
        long id)
    {
        var request = new GetCustomerByIdRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.GetByIdAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
```

```
}
```

.\RealtyHub.ApiService\Endpoints\Customers\UpdateCustomerEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Customers;

/// <summary>
/// Endpoint responsável por atualizar os dados de um cliente existente.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de atualização de
clientes.
/// </remarks>
public class UpdateCustomerEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para atualizar um cliente.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que espera um parâmetro numérico (ID) e os dados atualizados
do cliente,
    /// chamando o manipulador para executar a operação.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}", HandlerAsync)
            .WithName("Customers: Update")
            .WithSummary("Atualiza um cliente")
            .WithDescription("Atualiza um cliente")
            .WithOrder(2)
            .Produces<Response<Customer?>>()
            .Produces(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que recebe a requisição para atualizar um cliente.
    /// </summary>
    /// <remarks>
    /// Este método extrai o ID do cliente e os dados atualizados da requisição,
    /// associa o ID do usuário autenticado à requisição e chama o handler para realizar
a atualização.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="ICustomerHandler"/> responsável
pelas operações relacionadas a clientes.</param>
    /// <param name="request">Objeto <see cref="Customer"/> contendo os dados atualizados
do cliente.</param>
    /// <param name="id">ID do cliente a ser atualizado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os dados atualizados do cliente, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        ICustomerHandler handler,
        Customer request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.UpdateAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Emails\SendContractEmailEndpoint.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Requests.Emails;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;

namespace RealtyHub.ApiService.Endpoints.Emails;

/// <summary>
/// Endpoint responsável por enviar um email contendo um contrato como anexo.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de envio de emails
com contratos.
/// </remarks>
public class SendContractEmailEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para enviar um email com o contrato.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe os dados do email e do contrato e chama o
manipulador para enviar o email.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapPost("/contract", HandlerAsync)
            .WithName("Emails: SendContract")
            .WithSummary("Envia um email com o contrato")
            .WithDescription("Envia um email com o contrato")
            .WithOrder(1)
            .Produces(StatusCodes.Status200OK)
            .Produces<Response<string>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que recebe a requisição para enviar um email com o contrato.
    /// </summary>
    /// <remarks>
    /// Este método busca o contrato no banco de dados, verifica sua existência, gera o
caminho do anexo
    /// e chama o serviço de email para enviar o contrato. Retorna uma resposta
apropriada com base no resultado.
    /// </remarks>
    /// <param name="emailService">Instância de <see cref="IEmailService"/> responsável
pelo envio de emails.</param>
    /// <param name="request">Objeto <see cref="AttachmentMessage"/> contendo os dados do
email e do contrato.</param>
    /// <param name="context">Instância de <see cref="AppDbContext"/> para acesso ao
banco de dados.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK, se o email for enviado com sucesso;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição ou o contrato não for
encontrado.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        IEmailService emailService,
        AttachmentMessage request,
        AppDbContext context)
    {
        var contract = await context
            .Contracts
            .FirstOrDefaultAsync(c => c.Id == request.ContractId);

        if (contract is null)
            return Results.BadRequest(new Response<string>(null, 404, "Contrato não
encontrado"));

        var attachmentPath = Path.Combine(Configuration.ContractsPath,
```

```
$"{contract.FileId}.pdf");

        request.AttachmentPath = attachmentPath;

        var result = await emailService.SendContractAsync(request);

        return result.IsSuccess
            ? Results.Ok()
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Endpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Endpoints.Condominiums;
using RealtyHub.ApiService.Endpoints.Contracts;
using RealtyHub.ApiService.Endpoints.ContractsTemplates;
using RealtyHub.ApiService.Endpoints.Customers;
using RealtyHub.ApiService.Endpoints.Emails;
using RealtyHub.ApiService.Endpoints.Identity;
using RealtyHub.ApiService.Endpoints.Offers;
using RealtyHub.ApiService.Endpoints.Properties;
using RealtyHub.ApiService.Endpoints.Reports;
using RealtyHub.ApiService.Endpoints.Viewings;
using RealtyHub.ApiService.Models;

namespace RealtyHub.ApiService.Endpoints;

/// <summary>
/// Classe responsável por mapear todos os endpoints da aplicação.
/// </summary>
/// <remarks>
/// Esta classe organiza e registra os endpoints agrupados por suas respectivas áreas
funcionais,
/// como clientes, imóveis, condomínios, contratos, entre outros.
/// </remarks>
public static class Endpoint
{
    /// <summary>
    /// Mapeia todos os endpoints da aplicação.
    /// </summary>
    /// <param name="app">A instância do aplicativo <see cref="WebApplication"/>.</param>
    public static void MapEndpoints(this WebApplication app)
    {
        var endpoints = app.MapGroup("");

        // Health Check
        endpoints.MapGroup("/")
            .WithTags("Health Check")
            .MapGet("/", () => new { message = "Ok" });

        // Identity Endpoints
        endpoints.MapGroup("v1/identity")
            .WithTags("Identity")
            .MapIdentityApi<User>();

        endpoints.MapGroup("v1/identity")
            .WithTags("Identity")
            .MapEndpoint<LogoutEndpoint>()
            .MapEndpoint<RegisterUserEndpoint>()
            .MapEndpoint<ManageInfoEndpoint>()
            .MapEndpoint<ConfirmEmailEndpoint>()
            .MapEndpoint<ForgotPasswordEndpoint>()
            .MapEndpoint<ResetPasswordEndpoint>();

        // Customer Endpoints
        endpoints.MapGroup("v1/customers")
            .WithTags("Customers")
            .RequireAuthorization()
            .MapEndpoint<CreateCustomerEndpoint>()
            .MapEndpoint<UpdateCustomerEndpoint>()
            .MapEndpoint<DeleteCustomerEndpoint>()
            .MapEndpoint<GetCustomerByIdEndpoint>()
            .MapEndpoint<GetAllCustomersEndpoint>();

        // Property Endpoints
        endpoints.MapGroup("v1/properties")
            .WithTags("Properties")
            .RequireAuthorization()
            .MapEndpoint<CreatePropertyEndpoint>()
            .MapEndpoint<UpdatePropertyEndpoint>()
            .MapEndpoint<DeletePropertyEndpoint>()
            .MapEndpoint<CreatePropertyPhotosEndpoint>()
            .MapEndpoint<DeletePropertyPhotoEndpoint>()
            .MapEndpoint<GetAllViewingsByPropertyEndpoint>()
```

```
        .MapEndpoint<UpdatePropertyPhotosEndpoint>();

    endpoints.MapGroup("v1/properties")
        .WithTags("Properties")
        .MapEndpoint<GetPropertyByIdEndpoint>()
        .MapEndpoint<GetAllPropertyPhotosByPropertyEndpoint>()
        .MapEndpoint<GetAllPropertiesEndpoint>();

    // Condominium Endpoints
    endpoints.MapGroup("v1/condominiums")
        .WithTags("Condominiums")
        .RequireAuthorization()
        .MapEndpoint<CreateCondominiumEndpoint>()
        .MapEndpoint<UpdateCondominiumEndpoint>()
        .MapEndpoint<DeleteCondominiumEndpoint>()
        .MapEndpoint<GetCondominiumByIdEndpoint>()
        .MapEndpoint<GetAllCondominiumsEndpoint>();

    // Viewing Endpoints
    endpoints.MapGroup("v1/viewings")
        .WithTags("Viewings")
        .RequireAuthorization()
        .MapEndpoint<ScheduleViewingEndpoint>()
        .MapEndpoint<RescheduleViewingEndpoint>()
        .MapEndpoint<CancelViewingEndpoint>()
        .MapEndpoint<GetViewingByIdEndpoint>()
        .MapEndpoint<GetAllViewingsEndpoint>()
        .MapEndpoint<DoneViewingEndpoint>();

    // Offer Endpoints
    endpoints.MapGroup("v1/offers")
        .WithTags("Offers")
        .RequireAuthorization()
        .MapEndpoint<UpdateOfferEndpoint>()
        .MapEndpoint<AcceptOfferEndpoint>()
        .MapEndpoint<RejectOfferEndpoint>()
        .MapEndpoint<GetOfferByIdEndpoint>()
        .MapEndpoint<GetAllOffersByPropertyEndpoint>()
        .MapEndpoint<GetAllOffersByCustomerEndpoint>()
        .MapEndpoint<GetAllOffersEndpoint>()
        .MapEndpoint<GetOfferAcceptedEndpoint>();

    endpoints.MapGroup("v1/offers")
        .WithTags("Offers")
        .MapEndpoint<CreateOfferEndpoint>();

    // Contract Endpoints
    endpoints.MapGroup("v1/contracts")
        .WithTags("Contracts")
        .RequireAuthorization()
        .MapEndpoint<CreateContractEndpoint>()
        .MapEndpoint<UpdateContractEndpoint>()
        .MapEndpoint<DeleteContractEndpoint>()
        .MapEndpoint<GetContractByIdEndpoint>()
        .MapEndpoint<GetAllContractsEndpoint>();

    // Contract Templates Endpoints
    endpoints.MapGroup("v1/contracts-templates")
        .WithTags("Contract Templates")
        .RequireAuthorization()
        .MapEndpoint<GetAllContractTemplatesEndpoint>();

    // Report Endpoints
    endpoints.MapGroup("v1/reports")
        .WithTags("Reports")
        .RequireAuthorization()
        .MapEndpoint<OfferReportEndpoint>()
        .MapEndpoint<PropertyReportEndpoint>();

    // Email Endpoints
    endpoints.MapGroup("v1/emails")
        .WithTags("Emails")
        .RequireAuthorization()
        .MapEndpoint<SendContractEmailEndpoint>();
}
```

```
    /// <summary>
    /// Método auxiliar para mapear um endpoint genérico.
    /// </summary>
    /// <typeparam name="TEndpoint">O tipo do endpoint que implementa <see
cref="IEndpoint"/>.</typeparam>
    /// <param name="app">O construtor de rotas do aplicativo.</param>
    /// <returns>O construtor de rotas atualizado.</returns>
    private static IEndpointRouteBuilder MapEndpoint<TEndpoint>(this
IEndpointRouteBuilder app)
        where TEndpoint : IEndpoint
    {
        TEndpoint.Map(app);
        return app;
    }
}
```

.\RealtyHub.ApiService\Endpoints\Identity\ConfirmEmailEndpoint.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.WebUtilities;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;
using System.Text;

namespace RealtyHub.ApiService.Endpoints.Identity;

/// <summary>
/// Endpoint responsável por confirmar o email de um usuário.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de confirmação de
email.
/// </remarks>
public class ConfirmEmailEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para confirmar o email de um usuário.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que recebe o ID do usuário e o token de confirmação como
parâmetros de consulta
    /// e chama o manipulador para confirmar o email.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("confirm-email", Handler)
            .Produces<Response<string>>()
            .Produces<Response<string>>(StatusCodes.Status404NotFound)
            .Produces<Response<string>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que recebe a requisição para confirmar o email de um usuário.
    /// </summary>
    /// <remarks>
    /// Este método busca o usuário pelo ID, verifica se o email já foi confirmado e,
caso contrário,
    /// decodifica o token de confirmação e chama o serviço de confirmação de email.
    /// </remarks>
    /// <param name="userManager">Gerenciador de usuários <see cref="UserManager{User}"/>
para realizar operações relacionadas ao usuário.</param>
    /// <param name="claimsPrincipal">Objeto <see cref="ClaimsPrincipal"/> contendo os
dados do usuário autenticado.</param>
    /// <param name="userId">ID do usuário a ser confirmado <see cref="string"/>.</param>
    /// <param name="token">Token de confirmação codificado <see cref="string"/>.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK, se o email for confirmado com sucesso;</para>
    /// <para>- HTTP 400 Bad Request, se o email já estiver confirmado ou o token for
inválido;</para>
    /// <para>- HTTP 404 Not Found, se o usuário não for encontrado.</para>
    /// </returns>
    private static async Task<IResult> Handler(
        UserManager<User> userManager,
        ClaimsPrincipal claimsPrincipal,
        [FromQuery] string userId,
        [FromQuery] string token)
    {
        var user = await userManager.FindByIdAsync(userId);
        if (user is null)
            return Results.NotFound(new Response<string>(null, 400,
                "Usuário não encontrado!"));

        var emailConfirmed = await userManager.IsEmailConfirmedAsync(user);
        if (emailConfirmed)
```

```
            return Results.BadRequest(new Response<string>(null, 400,
                "Email já confirmado!"));

        var decodedBytes = WebEncoders.Base64UrlDecode(token);
        var decodedToken = Encoding.UTF8.GetString(decodedBytes);
        var result = await userManager.ConfirmEmailAsync(user, decodedToken);

        return result.Succeeded
            ? Results.Ok(new Response<string>(null, 200,
                "Email confirmado com sucesso!"))
            : Results.BadRequest(result.Errors);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Identity\ForgotPasswordEndpoint.cs

```
using System.Text;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.WebUtilities;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Models;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Core.Requests.Emails;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;

namespace RealtyHub.ApiService.Endpoints.Identity;

/// <summary>
/// Endpoint responsável por iniciar o processo de recuperação de senha de um usuário.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de solicitação de
recuperação de senha.
/// </remarks>
public class ForgotPasswordEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para solicitar a recuperação de senha.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe o email do usuário e envia um link de redefinição
de senha.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapPost("/forgot-password", HandlerAsync)
            .Produces<Response<string>>()
            .Produces<Response<string>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que recebe a requisição para recuperação de senha.
    /// </summary>
    /// <remarks>
    /// Este método busca o usuário pelo email fornecido, gera um token de redefinição de
senha,
    /// cria um link de redefinição e envia o link por email.
    /// </remarks>
    /// <param name="userManager">Gerenciador de usuários <see cref="UserManager{User}"/>
para realizar operações relacionadas ao usuário.</param>
    /// <param name="emailService">Serviço de email <see cref="IEmailService"/>
responsável por enviar o link de redefinição de senha.</param>
    /// <param name="request">Objeto <see cref="ForgotPasswordRequest"/> contendo o email
do usuário.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK, se o link de redefinição de senha for enviado com
sucesso;</para>
    /// <para>- HTTP 400 Bad Request, se o usuário não for encontrado.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        UserManager<User> userManager,
        IEmailService emailService,
        ForgotPasswordRequest request)
    {
        var user = await userManager.FindByEmailAsync(request.Email);
        if (user is null)
            return Results.BadRequest(
                new Response<string>(null, 404, "Usuário não encontrado"));

        var token = await userManager.GeneratePasswordResetTokenAsync(user);
        var encodedToken = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(token));
        var resetLink = $"{Core.Configuration.FrontendUrl}" +
                        $"/recuperar-senha?" +
                        $"userId={user.Id}" +
```

```
                    $"&token={encodedToken}";
        var message = new ResetPasswordMessage
        {
            EmailTo = user.Email!,
            ResetPasswordLink = resetLink
        };
        await emailService.SendResetPasswordLinkAsync(message);
        return Results.Ok(new Response<string>(null, message: "Verifique seu email!"));
    }
}
```

.\RealtyHub.ApiService\Endpoints\Identity\LogoutEndpoint.cs

```
using Microsoft.AspNetCore.Identity;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Models;

namespace RealtyHub.ApiService.Endpoints.Identity;

/// <summary>
/// Endpoint responsável por realizar o logout de um usuário autenticado.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de logout.
/// </remarks>
public class LogoutEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para realizar o logout.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que realiza o logout do usuário autenticado.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/logout", HandlerAsync)
            .RequireAuthorization();

    /// <summary>
    /// Manipulador da rota que realiza o logout do usuário autenticado.
    /// </summary>
    /// <remarks>
    /// Este método utiliza o <see cref="SignInManager{User}"/> para realizar o logout do
usuário atual.
    /// </remarks>
    /// <param name="signInManager">Gerenciador de autenticação <see
cref="SignInManager{User}"/>.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK, se o logout for realizado com sucesso.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(SignInManager<User> signInManager)
    {
        await signInManager.SignOutAsync();
        return Results.Ok();
    }
}
```

.\RealtyHub.ApiService\Endpoints\Identity\ManageInfoEndpoint.cs

```
using Microsoft.AspNetCore.Identity;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Identity;

/// <summary>
/// Endpoint responsável por gerenciar as informações do usuário autenticado.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção das
informações do usuário autenticado.
/// </remarks>
public class ManageInfoEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para obter as informações do usuário autenticado.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna as informações do usuário autenticado, como nome,
email, CRECI e claims.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/manageinfo", HandlerAsync)
            .Produces<UserResponse>()
            .Produces<Response<string>>(StatusCodes.Status401Unauthorized)
            .RequireAuthorization();
    }

    /// <summary>
    /// Manipulador da rota que retorna as informações do usuário autenticado.
    /// </summary>
    /// <remarks>
    /// Este método utiliza o <see cref="UserManager{User}"/> para buscar o usuário
autenticado e retorna suas informações.
    /// </remarks>
    /// <param name="userManager">Gerenciador de usuários <see cref="UserManager{User}"/>
para realizar operações relacionadas ao usuário.</param>
    /// <param name="claimsPrincipal">Objeto <see cref="ClaimsPrincipal"/> contendo os
dados do usuário autenticado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com as informações do usuário, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 401 Unauthorized, se o usuário não estiver autenticado.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        UserManager<User> userManager,
        ClaimsPrincipal claimsPrincipal)
    {
        var user = await userManager.GetUserAsync(claimsPrincipal);

        return user is null
            ? Results.Unauthorized()
            : Results.Ok(new UserResponse
            {
                GivenName = user.GivenName,
                Creci = user.Creci,
                Email = user.Email ?? string.Empty,
                UserName = user.UserName ?? string.Empty,
                Claims = claimsPrincipal.Claims.ToDictionary(c => c.Type, c => c.Value)
            });
    }
}
```

.\RealtyHub.ApiService\Endpoints\Identity\RegisterUserEndpoint.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.WebUtilities;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Models;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Core.Requests.Emails;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;
using System.Security.Claims;
using System.Text;

namespace RealtyHub.ApiService.Endpoints.Identity;

/// <summary>
/// Endpoint responsável por registrar um novo usuário.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de registro de
usuários.
/// </remarks>
public class RegisterUserEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para registrar um novo usuário.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe os dados do usuário e cria uma nova conta.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapPost("/registeruser", HandlerAsync)
            .Produces<Response<string>>()
            .Produces<Response<string>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que registra um novo usuário.
    /// </summary>
    /// <remarks>
    /// Este método cria um novo usuário com base nos dados fornecidos, adiciona claims
ao usuário,
    /// gera um token de confirmação de email e envia o link de confirmação para o email
do usuário.
    /// </remarks>
    /// <param name="request">Objeto <see cref="RegisterRequest"/> contendo os dados do
usuário a ser registrado.</param>
    /// <param name="userManager">Gerenciador de usuários <see cref="UserManager{User}"/>
para realizar operações relacionadas ao usuário.</param>
    /// <param name="emailService">Serviço de email <see cref="IEmailService"/>
responsável por enviar o link de confirmação de email.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK, se o usuário for registrado com sucesso e o email de
confirmação enviado;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na criação do usuário ou no envio
do email.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        RegisterRequest request,
        UserManager<User> userManager,
        IEmailService emailService)
    {
        var user = new User
        {
            UserName = request.Email,
            Email = request.Email,
            GivenName = request.GivenName,
            Creci = request.Creci
        };
```

```
        var createResult = await userManager.CreateAsync(user, request.Password);

        if (!createResult.Succeeded)
            return Results.BadRequest(createResult.Errors);

        var claims = new List<Claim>
        {
            new("Creci", user.Creci),
            new(ClaimTypes.GivenName, user.GivenName)
        };

        var addClaimsResult = await userManager.AddClaimsAsync(user, claims);
        if (!addClaimsResult.Succeeded)
            return Results.BadRequest(addClaimsResult.Errors);

        var token = await userManager.GenerateEmailConfirmationTokenAsync(user);
        var encodedToken = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(token));
        var confirmationLink = $"{Core.Configuration.FrontendUrl}" +
                               $"/confirmar-email?" +
                               $"userId={user.Id}" +
                               $"&token={encodedToken}";

        var emailMessage = new ConfirmEmailMessage
        {
            EmailTo = user.Email,
            ConfirmationLink = confirmationLink
        };

        var emailResult = await emailService.SendConfirmationLinkAsync(emailMessage);

        return emailResult.IsSuccess
            ? Results.Ok(new Response<string>(null,
                message: "Usuário registrado com sucesso! Verifique seu e-mail!"))
            : Results.BadRequest(emailResult.Message);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Identity\ResetPasswordEndpoint.cs

```
using System.Text;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.WebUtilities;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Models;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;

namespace RealtyHub.ApiService.Endpoints.Identity;

/// <summary>
/// Endpoint responsável por redefinir a senha de um usuário.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de redefinição de
senha.
/// </remarks>
public class ResetPasswordEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para redefinir a senha de um usuário.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe o ID do usuário, o token de redefinição e a nova
senha.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapPost("/reset-password", HandlerAsync)
            .Produces<Response<string>>()
            .Produces<Response<string>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que realiza a redefinição de senha de um usuário.
    /// </summary>
    /// <remarks>
    /// Este método busca o usuário pelo ID, decodifica o token de redefinição de senha,
    /// e redefine a senha do usuário com base nos dados fornecidos.
    /// </remarks>
    /// <param name="userManager">Gerenciador de usuários <see cref="UserManager{User}"/>
para realizar operações relacionadas ao usuário.</param>
    /// <param name="emailService">Serviço de email <see cref="IEmailService"/> para
envio de notificações, se necessário.</param>
    /// <param name="request">Objeto <see cref="ResetPasswordRequest"/> contendo os dados
da nova senha.</param>
    /// <param name="userId">ID do usuário <see cref="string"/> cujo senha será
redefinida.</param>
    /// <param name="token">Token de redefinição de senha codificado <see
cref="string"/>.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK, se a senha for redefinida com sucesso;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na redefinição de senha ou o
usuário não for encontrado.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        UserManager<User> userManager,
        IEmailService emailService,
        ResetPasswordRequest request,
        [FromQuery] string userId,
        [FromQuery] string token)
    {
        var user = await userManager.FindByIdAsync(userId);
        if (user is null)
            return Results.BadRequest(
                new Response<string>(null, 404, "Usuário não encontrado"));
```

```
        var decodedBytes = WebEncoders.Base64UrlDecode(token);
        var decodedToken = Encoding.UTF8.GetString(decodedBytes);

        var resetResult = await userManager
            .ResetPasswordAsync(user, decodedToken, request.PasswordResetModel.Password);

        return resetResult.Succeeded
            ? Results.Ok(new Response<string>(null, message: "Senha redefinida com
sucesso"))
            : Results.BadRequest(resetResult.Errors);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\AcceptOfferEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por aceitar uma proposta.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de aceitação de
propostas.
/// </remarks>
public class AcceptOfferEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para aceitar uma proposta.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que aceita uma proposta com base no ID fornecido.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}/accept", HandlerAsync)
            .WithName("Offers: Accept")
            .WithSummary("Aceita uma proposta")
            .WithDescription("Aceita uma proposta")
            .WithOrder(7)
            .Produces<Response<Offer?>>()
            .Produces<Response<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que aceita uma proposta.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para aceitar uma proposta com base no ID
fornecido e no usuário autenticado,
    /// e chama o handler para processar a aceitação da proposta.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="id">ID da proposta a ser aceita.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da proposta aceita, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        long id)
    {
        var request = new AcceptOfferRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.AcceptAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\CreateOfferEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por criar uma nova proposta.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de criação de
propostas.
/// </remarks>
public class CreateOfferEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para criar uma nova proposta.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe os dados da proposta e cria uma nova proposta.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/", HandlerAsync)
            .WithName("Offers: Create")
            .WithSummary("Cria uma proposta")
            .WithDescription("Cria uma proposta")
            .WithOrder(1)
            .Produces<Response<Offer?>>(StatusCodes.Status201Created)
            .Produces<Response<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que cria uma nova proposta.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados da proposta, associa o ID do usuário autenticado à
proposta,
    /// e chama o handler para criar a proposta.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="request">Objeto <see cref="Offer"/> contendo os dados da proposta a
ser criada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 201 Created com os detalhes da proposta criada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        Offer request)
    {
        request.UserId = user.Identity?.Name ?? string.Empty;

        var result = await handler.CreateAsync(request);

        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\GetAllOffersByCustomerEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por recuperar todas as propostas feitas por um cliente.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
propostas feitas por um cliente.
/// </remarks>
public class GetAllOffersByCustomerEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todas as propostas feitas por um cliente.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de propostas feitas por um
cliente específico.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/customer/{id:long}", HandlerAsync)
            .WithName("Offers: Get All by Customer")
            .WithSummary("Recupera todas as propostas feitas por um cliente")
            .WithDescription("Recupera todas as propostas feitas por um cliente")
            .WithOrder(8)
            .Produces<PagedResponse<Offer?>>()
            .Produces<PagedResponse<Offer?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que retorna todas as propostas feitas por um cliente.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar as propostas feitas por um cliente
específico,
    /// aplicando filtros de data e paginação, e chama o handler para processar a
requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="id">ID do cliente cujas propostas serão recuperadas.</param>
    /// <param name="startDate">Data inicial para filtrar as propostas <see
cref="string"/> (opcional).</param>
    /// <param name="endDate">Data final para filtrar as propostas <see cref="string"/>
(opcional).</param>
    /// <param name="pageNumber">Número da página solicitada <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageNumber"/>).</param>
    /// <param name="pageSize">Quantidade de itens por página <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageSize"/>).</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de propostas, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        long id,
        [FromQuery] string? startDate,
```

```
        [FromQuery] string? endDate,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize)
    {
        var request = new GetAllOffersByCustomerRequest
        {
            CustomerId = id,
            UserId = user.Identity?.Name ?? string.Empty,
            PageNumber = pageNumber,
            PageSize = pageSize,
            StartDate = startDate,
            EndDate = endDate
        };
        var result = await handler.GetAllOffersByCustomerAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\GetAllOffersByPropertyEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por recuperar todas as propostas de um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
propostas de um imóvel.
/// </remarks>
public class GetAllOffersByPropertyEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todas as propostas de um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de propostas associadas a um
imóvel específico.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/property/{id:long}", HandlerAsync)
            .WithName("Offers: Get All by Property")
            .WithSummary("Recupera todas as propostas de um imóvel")
            .WithDescription("Recupera todas as propostas de um imóvel")
            .WithOrder(6)
            .Produces<PagedResponse<Offer?>>()
            .Produces<PagedResponse<Offer?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que retorna todas as propostas de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar as propostas associadas a um imóvel
específico,
    /// aplicando filtros de data e paginação, e chama o handler para processar a
requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="id">ID do imóvel cujas propostas serão recuperadas.</param>
    /// <param name="startDate">Data inicial para filtrar as propostas <see
cref="string"/> (opcional).</param>
    /// <param name="endDate">Data final para filtrar as propostas <see cref="string"/>
(opcional).</param>
    /// <param name="pageNumber">Número da página solicitada <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageNumber"/>).</param>
    /// <param name="pageSize">Quantidade de itens por página <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageSize"/>).</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de propostas, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        long id,
        [FromQuery] string? startDate,
```

```
        [FromQuery] string? endDate,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize)
    {
        var request = new GetAllOffersByPropertyRequest
        {
            PropertyId = id,
            UserId = user.Identity?.Name ?? string.Empty,
            PageNumber = pageNumber,
            PageSize = pageSize,
            StartDate = startDate,
            EndDate = endDate
        };
        var result = await handler.GetAllOffersByPropertyAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\GetAllOffersEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por recuperar todas as propostas.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
todas as propostas.
/// </remarks>
public class GetAllOffersEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todas as propostas.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de todas as propostas.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/", HandlerAsync)
            .WithName("Offers: Get All")
            .WithSummary("Recupera todas as propostas")
            .WithDescription("Recupera todas as propostas")
            .WithOrder(5)
            .Produces<PagedResponse<Offer?>>()
            .Produces<PagedResponse<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna todas as propostas.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar todas as propostas, aplicando filtros
de data e paginação,
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="startDate">Data inicial para filtrar as propostas <see
cref="string"/> (opcional).</param>
    /// <param name="endDate">Data final para filtrar as propostas <see cref="string"/>
(opcional).</param>
    /// <param name="pageNumber">Número da página solicitada <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageNumber"/>).</param>
    /// <param name="pageSize">Quantidade de itens por página <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageSize"/>).</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de propostas, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        [FromQuery] string? startDate,
        [FromQuery] string? endDate,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize)
    {
        var request = new GetAllOffersRequest
        {
```

```
            UserId = user.Identity?.Name ?? string.Empty,
            PageNumber = pageNumber,
            PageSize = pageSize,
            StartDate = startDate,
            EndDate = endDate
        };
        var result = await handler.GetAllAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\GetOfferAcceptedEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por recuperar a proposta aceita de um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção da
proposta aceita de um imóvel.
/// </remarks>
public class GetOfferAcceptedEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar a proposta aceita de um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna a proposta aceita associada a um imóvel
específico.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/property/{id:long}/accepted", HandlerAsync)
            .WithName("Offers: Get Accepted")
            .WithSummary("Recupera a proposta aceita")
            .WithDescription("Recupera a proposta aceita")
            .WithOrder(4)
            .Produces<Response<Offer?>>()
            .Produces<Response<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna a proposta aceita de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar a proposta aceita associada a um
imóvel específico
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="id">ID do imóvel cuja proposta aceita será recuperada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da proposta aceita, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        long id)
    {
        var request = new GetOfferAcceptedByProperty
        {
            PropertyId = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.GetAcceptedByProperty(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\GetOfferByIdEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por recuperar uma proposta específica pelo seu ID.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção de uma
proposta pelo ID.
/// </remarks>
public class GetOfferByIdEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar uma proposta pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna os detalhes de uma proposta específica com base
no ID fornecido.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/{id:long}", HandlerAsync)
            .WithName("Offers: Get By Id")
            .WithSummary("Recupera uma proposta")
            .WithDescription("Recupera uma proposta")
            .WithOrder(4)
            .Produces<Response<Offer?>>()
            .Produces<Response<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna os detalhes de uma proposta pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar uma proposta específica com base no
ID fornecido
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="id">ID da proposta a ser recuperada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da proposta, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        long id)
    {
        var request = new GetOfferByIdRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.GetByIdAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\RejectOfferEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por rejeitar uma proposta.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de rejeição de
propostas.
/// </remarks>
public class RejectOfferEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para rejeitar uma proposta.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que rejeita uma proposta com base no ID fornecido.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}/reject", HandlerAsync)
            .WithName("Offers: Reject")
            .WithSummary("Rejeita uma proposta")
            .WithDescription("Rejeita uma proposta")
            .WithOrder(3)
            .Produces<Response<Offer?>>()
            .Produces<Response<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que rejeita uma proposta.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para rejeitar uma proposta com base no ID
fornecido e no usuário autenticado,
    /// e chama o handler para processar a rejeição da proposta.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="id">ID da proposta a ser rejeitada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da proposta rejeitada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        long id)
    {
        var request = new RejectOfferRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.RejectAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Offers\UpdateOfferEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Offers;

/// <summary>
/// Endpoint responsável por atualizar uma proposta existente.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de atualização de
propostas.
/// </remarks>
public class UpdateOfferEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para atualizar uma proposta.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que recebe os dados atualizados da proposta e o ID da
proposta a ser modificada.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}", HandlerAsync)
            .WithName("Offers: Update")
            .WithSummary("Atualiza uma proposta")
            .WithDescription("Atualiza uma proposta")
            .WithOrder(2)
            .Produces<Response<Offer?>>()
            .Produces<Response<Offer?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que atualiza uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados atualizados da proposta, associa o ID do usuário
autenticado à proposta,
    /// e chama o handler para realizar a atualização.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IOfferHandler"/> responsável pelas
operações relacionadas a propostas.</param>
    /// <param name="request">Objeto <see cref="Offer"/> contendo os dados atualizados da
proposta.</param>
    /// <param name="id">ID da proposta a ser atualizada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da proposta atualizada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IOfferHandler handler,
        Offer request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;

        var result = await handler.UpdateAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\CreatePropertyEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por criar um novo imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de criação de
imóveis.
/// </remarks>
public class CreatePropertyEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para criar um novo imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe os dados do imóvel e cria um novo registro.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/", HandlerAsync)
            .WithName("Properties: Create")
            .WithSummary("Cria um novo imóvel")
            .WithDescription("Cria um novo imóvel")
            .WithOrder(1)
            .Produces<Response<Property?>>(StatusCodes.Status201Created)
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que cria um novo imóvel.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados do imóvel, associa o ID do usuário autenticado ao
imóvel,
    /// e chama o handler para criar o registro do imóvel.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IPropertyHandler"/> responsável
pelas operações relacionadas a imóveis.</param>
    /// <param name="request">Objeto <see cref="Property"/> contendo os dados do imóvel a
ser criado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 201 Created com os detalhes do imóvel criado, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IPropertyHandler handler,
        Property request)
    {
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.CreateAsync(request);

        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\CreatePropertyPhotosEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por adicionar fotos a um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de adição de fotos
a imóveis.
/// </remarks>
public class CreatePropertyPhotosEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para adicionar fotos a um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe as fotos e o ID do imóvel para associá-las.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapPost("/{id:long}/photos", HandlerAsync)
            .WithName("Properties: Add Photos")
            .WithSummary("Adiciona fotos a um imóvel")
            .WithDescription("Adiciona fotos a um imóvel")
            .WithOrder(6)
            .Produces<Response<PropertyPhoto?>>(StatusCodes.Status201Created)
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que adiciona fotos a um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método recebe as fotos enviadas na requisição, associa o ID do imóvel e o ID
do usuário autenticado,
    /// e chama o handler para processar a adição das fotos.
    /// </remarks>
    /// <param name="httpRequest">Objeto <see cref="HttpRequest"/> contendo os dados da
requisição HTTP.</param>
    /// <param name="id">ID do imóvel ao qual as fotos serão associadas.</param>
    /// <param name="handler">Instância de <see cref="IPropertyPhotosHandler"/>
responsável pelas operações relacionadas a fotos de imóveis.</param>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 201 Created com os detalhes da foto adicionada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        HttpRequest httpRequest,
        long id,
        IPropertyPhotosHandler handler,
        ClaimsPrincipal user)
    {
        var request = new CreatePropertyPhotosRequest
        {
            HttpRequest = httpRequest,
            PropertyId = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.CreateAsync(request);
```

```
        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\DeletePropertyEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por deletar um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de exclusão de
imóveis.
/// </remarks>
public class DeletePropertyEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para deletar um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota DELETE que recebe o ID do imóvel a ser excluído.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapDelete("/{id:long}", HandlerAsync)
            .WithName("Properties: Delete")
            .WithSummary("Deleta um imóvel")
            .WithDescription("Deleta um imóvel")
            .WithOrder(3)
            .Produces(StatusCodes.Status204NoContent)
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que deleta um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para excluir um imóvel com base no ID fornecido e
no usuário autenticado,
    /// e chama o handler para processar a exclusão.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IPropertyHandler"/> responsável
pelas operações relacionadas a imóveis.</param>
    /// <param name="id">ID do imóvel a ser excluído.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 204 No Content, se a exclusão for bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IPropertyHandler handler,
        long id)
    {
        var request = new DeletePropertyRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.DeleteAsync(request);

        return result.IsSuccess
            ? Results.NoContent()
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\DeletePropertyPhotoEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por excluir uma foto de um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de exclusão de
fotos de imóveis.
/// </remarks>
public class DeletePropertyPhotoEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para excluir uma foto de um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota DELETE que recebe o ID do imóvel e o ID da foto a ser excluída.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapDelete("/{id:long}/photos/{photoId}", HandlerAsync)
            .WithName("Properties: Delete Photos")
            .WithSummary("Exclui uma foto de um imóvel")
            .WithDescription("Exclui uma foto de um imóvel")
            .WithOrder(7)
            .Produces<Response<PropertyPhoto?>>(StatusCodes.Status204NoContent)
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que exclui uma foto de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para excluir uma foto de um imóvel com base no ID
do imóvel,
    /// no ID da foto e no usuário autenticado, e chama o handler para processar a
exclusão.
    /// </remarks>
    /// <param name="id">ID do imóvel ao qual a foto está associada.</param>
    /// <param name="handler">Instância de <see cref="IPropertyPhotosHandler"/>
responsável pelas operações relacionadas a fotos de imóveis.</param>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="photoId">ID da foto a ser excluída.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 204 No Content, se a exclusão for bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        long id,
        IPropertyPhotosHandler handler,
        ClaimsPrincipal user,
        string photoId)
    {
        var request = new DeletePropertyPhotoRequest
        {
            Id = photoId,
            PropertyId = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.DeleteAsync(request);
```

```
        return result.IsSuccess
            ? Results.NoContent()
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\GetAllPropertiesEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por recuperar todos os imóveis.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
todos os imóveis.
/// </remarks>
public class GetAllPropertiesEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todos os imóveis.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de todos os imóveis, com
suporte a filtros e pesquisa.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/", HandlerAsync)
            .WithName("Properties: Get All")
            .WithSummary("Recupera todos os imóveis")
            .WithDescription("Recupera todos os imóveis")
            .WithOrder(5)
            .Produces<PagedResponse<List<Property>?>>()
            .Produces<PagedResponse<Property?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna todos os imóveis.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar todos os imóveis, aplicando filtros,
paginação e termos de pesquisa,
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IPropertyHandler"/> responsável
pelas operações relacionadas a imóveis.</param>
    /// <param name="filterBy">Filtro para os imóveis <see cref="string"/>
(opcional).</param>
    /// <param name="pageNumber">Número da página solicitada <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageNumber"/>).</param>
    /// <param name="pageSize">Quantidade de itens por página <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageSize"/>).</param>
    /// <param name="searchTerm">Termo de pesquisa para buscar imóveis <see
cref="string"/> (opcional).</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de imóveis, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IPropertyHandler handler,
        [FromQuery] string filterBy = "",
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize,
        [FromQuery] string searchTerm = "")
    {
        var request = new GetAllPropertiesRequest
```

```
        {
            PageNumber = pageNumber,
            PageSize = pageSize,
            SearchTerm = searchTerm,
            FilterBy = filterBy
        };

        var result = await handler.GetAllAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\GetAllPropertyPhotosByPropertyEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por recuperar todas as fotos de um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
fotos de um imóvel.
/// </remarks>
public class GetAllPropertyPhotosByPropertyEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todas as fotos de um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna todas as fotos associadas a um imóvel específico.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/{id:long}/photos", HandlerAsync)
            .WithName("Properties: Get Photos By Property")
            .WithSummary("Recupera todas as fotos de um imóvel")
            .WithDescription("Recupera todas as fotos de um imóvel")
            .WithOrder(8)
            .Produces<Response<List<PropertyPhoto>?>>()
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que retorna todas as fotos de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar todas as fotos associadas a um imóvel
específico
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="id">ID do imóvel cujas fotos serão recuperadas.</param>
    /// <param name="handler">Instância de <see cref="IPropertyPhotosHandler"/>
responsável pelas operações relacionadas a fotos de imóveis.</param>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista de fotos do imóvel, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        long id,
        IPropertyPhotosHandler handler,
        ClaimsPrincipal user)
    {
        var request = new GetAllPropertyPhotosByPropertyRequest
        {
            PropertyId = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.GetAllByPropertyAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
```

```
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\GetAllViewingsByPropertyEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por listar todas as visitas de um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
visitas de um imóvel.
/// </remarks>
public class GetAllViewingsByPropertyEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para listar todas as visitas de um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de visitas associadas a um
imóvel específico.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/{id:long}/viewings", HandlerAsync)
            .WithName("Properties: Get All Viewings")
            .WithSummary("Lista todas as visitas de um imóvel")
            .WithDescription("Lista todas as visitas de um imóvel")
            .WithOrder(6)
            .Produces<PagedResponse<List<Viewing>?>>()
            .Produces<PagedResponse<Property?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna todas as visitas de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar todas as visitas associadas a um
imóvel específico,
    /// aplicando filtros de data e paginação, e chama o handler para processar a
requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IPropertyHandler"/> responsável
pelas operações relacionadas a imóveis.</param>
    /// <param name="id">ID do imóvel cujas visitas serão recuperadas.</param>
    /// <param name="startDate">Data inicial para filtrar as visitas <see cref="string"/>
(opcional).</param>
    /// <param name="endDate">Data final para filtrar as visitas <see cref="string"/>
(opcional).</param>
    /// <param name="pageNumber">Número da página solicitada <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageNumber"/>).</param>
    /// <param name="pageSize">Quantidade de itens por página <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageSize"/>).</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de visitas, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IPropertyHandler handler,
        long id,
        [FromQuery] string? startDate,
        [FromQuery] string? endDate,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
```

```
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize)
    {
        var request = new GetAllViewingsByPropertyRequest
        {
            UserId = user.Identity?.Name ?? string.Empty,
            PropertyId = id,
            PageNumber = pageNumber,
            PageSize = pageSize,
            StartDate = startDate,
            EndDate = endDate
        };

        var result = await handler.GetAllViewingsAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\GetPropertyByIdEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por recuperar um imóvel específico pelo seu ID.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção de um
imóvel pelo ID.
/// </remarks>
public class GetPropertyByIdEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar um imóvel pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna os detalhes de um imóvel específico com base no
ID fornecido.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/{id:long}", HandlerAsync)
            .WithName("Properties: Get by Id")
            .WithSummary("Recupera um imóvel")
            .WithDescription("Recupera um imóvel")
            .WithOrder(4)
            .Produces<Response<Property?>>()
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna os detalhes de um imóvel pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar um imóvel específico com base no ID
fornecido
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IPropertyHandler"/> responsável
pelas operações relacionadas a imóveis.</param>
    /// <param name="id">ID do imóvel a ser recuperado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes do imóvel, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IPropertyHandler handler,
        long id)
    {
        var request = new GetPropertyByIdRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.GetByIdAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
```

```
}
```

.\RealtyHub.ApiService\Endpoints\Properties\UpdatePropertyEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por atualizar um imóvel existente.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de atualização de
imóveis.
/// </remarks>
public class UpdatePropertyEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para atualizar um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que recebe os dados atualizados do imóvel e o ID do imóvel a
ser modificado.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}", HandlerAsync)
            .WithName("Properties: Update")
            .WithSummary("Atualiza um imóvel")
            .WithDescription("Atualiza um imóvel")
            .WithOrder(2)
            .Produces<Response<Property?>>()
            .Produces<Response<Property?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que atualiza um imóvel existente.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados atualizados do imóvel, associa o ID do usuário
autenticado ao imóvel,
    /// e chama o handler para realizar a atualização.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IPropertyHandler"/> responsável
pelas operações relacionadas a imóveis.</param>
    /// <param name="request">Objeto <see cref="Property"/> contendo os dados atualizados
do imóvel.</param>
    /// <param name="id">ID do imóvel a ser atualizado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes do imóvel atualizado, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IPropertyHandler handler,
        Property request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;

        var result = await handler.UpdateAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Properties\UpdatePropertyPhotosEndpoint.cs

```
using System.Security.Claims;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Properties;

/// <summary>
/// Endpoint responsável por atualizar as fotos de um imóvel.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de atualização de
fotos de imóveis.
/// </remarks>
public class UpdatePropertyPhotosEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para atualizar as fotos de um imóvel.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que recebe os dados atualizados das fotos e o ID do imóvel ao
qual elas pertencem.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapPut("/{id:long}/photos", HandlerAsync)
            .WithName("Properties: Update Photos")
            .WithSummary("Atualiza as fotos de um imóvel")
            .WithDescription("Atualiza as fotos de um imóvel")
            .WithOrder(7)
            .Produces<Response<List<PropertyPhoto>?>>()
            .Produces<Response<List<PropertyPhoto>?>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que atualiza as fotos de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados atualizados das fotos, associa o ID do imóvel e o ID
do usuário autenticado,
    /// e chama o handler para realizar a atualização.
    /// </remarks>
    /// <param name="id">ID do imóvel ao qual as fotos pertencem.</param>
    /// <param name="handler">Instância de <see cref="IPropertyPhotosHandler"/>
responsável pelas operações relacionadas a fotos de imóveis.</param>
    /// <param name="request">Objeto <see cref="UpdatePropertyPhotosRequest"/> contendo
os dados atualizados das fotos.</param>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes das fotos atualizadas, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        long id,
        IPropertyPhotosHandler handler,
        UpdatePropertyPhotosRequest request,
        ClaimsPrincipal user)
    {
        request.PropertyId = id;
        request.UserId = user.Identity?.Name ?? string.Empty;

        var result = await handler.UpdateAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
```

```
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Reports\OfferReportEndpoint.cs

```
using System.Security.Claims;
using Microsoft.EntityFrameworkCore;
using QuestPDF.Fluent;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Data;
using RealtyHub.ApiService.Services.Reports;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Reports;

/// <summary>
/// Endpoint responsável por gerar relatórios de propostas a imóveis.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de geração de
relatórios de propostas.
/// </remarks>
public class OfferReportEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para gerar relatórios de propostas a imóveis.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que gera um relatório em PDF contendo informações sobre
propostas a imóveis.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/offer", HandleAsync)
            .WithName("Reports: Offer")
            .WithSummary("Relatório de Propostas a Imóveis")
            .WithDescription("Gera um relatório de propostas a imóveis")
            .WithOrder(2)
            .Produces<Response<Report>>()
            .Produces<Response<Report>>(StatusCodes.Status401Unauthorized)
            .Produces<Response<Report>>(StatusCodes.Status500InternalServerError)
            .Produces<Response<Report>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que gera o relatório de propostas a imóveis.
    /// </summary>
    /// <remarks>
    /// Este método busca os dados de propostas no banco de dados, gera um relatório em
PDF utilizando o serviço de relatórios,
    /// salva o arquivo gerado no servidor e retorna a URL para download do relatório.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="dbContext">Contexto do banco de dados <see cref="AppDbContext"/>
para buscar as propostas.</param>
    /// <param name="httpContext">Contexto HTTP <see cref="HttpContext"/> para construir
a URL do relatório.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a URL do relatório, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição;</para>
    /// <para>- HTTP 401 Unauthorized, se o usuário não estiver autenticado;</para>
    /// <para>- HTTP 500 Internal Server Error, se ocorrer um erro inesperado.</para>
    /// </returns>
    private static async Task<IResult> HandleAsync(
        ClaimsPrincipal user,
        AppDbContext dbContext,
        HttpContext httpContext)
    {
        var offersData = await dbContext.Offers
            .Include(o => o.Property)
            .ThenInclude(p => p!.Seller)
```

```
            .Include(o => o.Buyer)
            .ToListAsync();

        var report = new OfferReportService(offersData);
        var pdfBytes = report.GeneratePdf();

        var fileName = Path.Combine($"Relatorio Propostas
{DateTime.Now:yyyyMMddHHmmss}.pdf");
        var filePath = Path.Combine(Configuration.ReportsPath, fileName);
        await File.WriteAllBytesAsync(filePath, pdfBytes);

        var reportUrl =
$"{httpContext.Request.Scheme}://{httpContext.Request.Host}/reports/{fileName}";

        var reportData = new Report { Url = reportUrl };

        return Results.Ok(new Response<Report>(reportData));
    }
}
```

.\RealtyHub.ApiService\Endpoints\Reports\PropertyReportEndpoint.cs

```
using System.Security.Claims;
using Microsoft.EntityFrameworkCore;
using QuestPDF.Fluent;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Data;
using RealtyHub.ApiService.Services.Reports;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Endpoints.Reports;

/// <summary>
/// Endpoint responsável por gerar relatórios de imóveis.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de geração de
relatórios de imóveis.
/// </remarks>
public class PropertyReportEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para gerar relatórios de imóveis.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que gera um relatório em PDF contendo informações sobre
imóveis.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
    {
        app.MapGet("/property", HandleAsync)
            .WithName("Reports: Property")
            .WithSummary("Relatório de Imóveis")
            .WithDescription("Gera um relatório de imóveis")
            .WithOrder(1)
            .Produces<Response<Report>>()
            .Produces<Response<Report>>(StatusCodes.Status401Unauthorized)
            .Produces<Response<Report>>(StatusCodes.Status500InternalServerError)
            .Produces<Response<Report>>(StatusCodes.Status400BadRequest);
    }

    /// <summary>
    /// Manipulador da rota que gera o relatório de imóveis.
    /// </summary>
    /// <remarks>
    /// Este método busca os dados de imóveis no banco de dados, gera um relatório em PDF
utilizando o serviço de relatórios,
    /// salva o arquivo gerado no servidor e retorna a URL para download do relatório.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="dbContext">Contexto do banco de dados <see cref="AppDbContext"/>
para buscar os imóveis.</param>
    /// <param name="environment">Ambiente de hospedagem <see cref="IHostEnvironment"/>
para acessar configurações do servidor.</param>
    /// <param name="httpContext">Contexto HTTP <see cref="HttpContext"/> para construir
a URL do relatório.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a URL do relatório, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição;</para>
    /// <para>- HTTP 401 Unauthorized, se o usuário não estiver autenticado;</para>
    /// <para>- HTTP 500 Internal Server Error, se ocorrer um erro inesperado.</para>
    /// </returns>
    private static async Task<IResult> HandleAsync(
        ClaimsPrincipal user,
        AppDbContext dbContext,
        IHostEnvironment environment,
        HttpContext httpContext)
    {
```

```
        var propertiesData = await dbContext.Properties
            .ToListAsync();

        var report = new PropertyReportService(propertiesData);
        var pdfBytes = report.GeneratePdf();

        var fileName = Path.Combine($"Relatorio Imoveis
{DateTime.Now:yyyyMMddHHmmss}.pdf");
        var filePath = Path.Combine(Configuration.ReportsPath, fileName);
        await File.WriteAllBytesAsync(filePath, pdfBytes);

        var reportUrl =
$"{httpContext.Request.Scheme}://{httpContext.Request.Host}/reports/{fileName}";
        var reportData = new Report { Url = reportUrl };

        return Results.Ok(new Response<Report>(reportData));
    }
}
```

.\RealtyHub.ApiService\Endpoints\Viewings\CancelViewingEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Viewings;

/// <summary>
/// Endpoint responsável por cancelar uma visita.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de cancelamento de
visitas.
/// </remarks>
public class CancelViewingEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para cancelar uma visita.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que recebe o ID da visita a ser cancelada.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}/cancel", HandlerAsync)
            .WithName("Viewings: Cancel")
            .WithSummary("Cancela uma visita")
            .WithDescription("Cancela uma visita")
            .WithOrder(3)
            .Produces<Response<Viewing?>>()
            .Produces<Response<Viewing?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que cancela uma visita.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para cancelar uma visita com base no ID fornecido
e no usuário autenticado,
    /// e chama o handler para processar o cancelamento.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IViewingHandler"/> responsável
pelas operações relacionadas a visitas.</param>
    /// <param name="id">ID da visita a ser cancelada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da visita cancelada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IViewingHandler handler,
        long id)
    {
        var request = new CancelViewingRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };

        var result = await handler.CancelAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Viewings\DoneViewingEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Viewings;

/// <summary>
/// Endpoint responsável por finalizar uma visita.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de finalização de
visitas.
/// </remarks>
public class DoneViewingEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para finalizar uma visita.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que recebe o ID da visita a ser finalizada.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}/done", HandlerAsync)
            .WithName("Viewings: Done")
            .WithSummary("Finaliza uma visita")
            .WithDescription("Finaliza uma visita")
            .WithOrder(6)
            .Produces<Response<Viewing?>>()
            .Produces<Response<Viewing?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que finaliza uma visita.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para finalizar uma visita com base no ID
fornecido e no usuário autenticado,
    /// e chama o handler para processar a finalização.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IViewingHandler"/> responsável
pelas operações relacionadas a visitas.</param>
    /// <param name="request">Objeto <see cref="DoneViewingRequest"/> contendo os dados
da visita a ser finalizada.</param>
    /// <param name="id">ID da visita a ser finalizada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da visita finalizada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IViewingHandler handler,
        DoneViewingRequest request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;

        var result = await handler.DoneAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Viewings\GetAllViewingsEndpoint.cs

```
using Microsoft.AspNetCore.Mvc;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Viewings;

/// <summary>
/// Endpoint responsável por recuperar todas as visitas.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de listagem de
todas as visitas.
/// </remarks>
public class GetAllViewingsEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar todas as visitas.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna uma lista paginada de todas as visitas, com
suporte a filtros de data.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/", HandlerAsync)
            .WithName("Viewings: Get All")
            .WithSummary("Recupera todas as visitas")
            .WithDescription("Recupera todas as visitas")
            .WithOrder(5)
            .Produces<PagedResponse<List<Viewing>?>>()
            .Produces<PagedResponse<List<Viewing?>>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna todas as visitas.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar todas as visitas, aplicando filtros
de data e paginação,
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IViewingHandler"/> responsável
pelas operações relacionadas a visitas.</param>
    /// <param name="startDate">Data inicial para filtrar as visitas <see cref="string"/>
(opcional).</param>
    /// <param name="endDate">Data final para filtrar as visitas <see cref="string"/>
(opcional).</param>
    /// <param name="pageNumber">Número da página solicitada <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageNumber"/>).</param>
    /// <param name="pageSize">Quantidade de itens por página <see cref="int"/> (padrão:
<see cref="Core.Configuration.DefaultPageSize"/>).</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com a lista paginada de visitas, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IViewingHandler handler,
        [FromQuery] string? startDate,
        [FromQuery] string? endDate,
        [FromQuery] int pageNumber = Core.Configuration.DefaultPageNumber,
        [FromQuery] int pageSize = Core.Configuration.DefaultPageSize)
    {
        var request = new GetAllViewingsRequest
```

```
        {
            UserId = user.Identity?.Name ?? string.Empty,
            PageNumber = pageNumber,
            PageSize = pageSize,
            StartDate = startDate,
            EndDate = endDate
        };

        var result = await handler.GetAllAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Viewings\GetViewingByIdEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Viewings;

/// <summary>
/// Endpoint responsável por recuperar uma visita específica pelo seu ID.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de obtenção de uma
visita pelo ID.
/// </remarks>
public class GetViewingByIdEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para recuperar uma visita pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Registra a rota GET que retorna os detalhes de uma visita específica com base no
ID fornecido.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapGet("/{id:long}", HandlerAsync)
            .WithName("Viewings: Get by Id")
            .WithSummary("Recupera uma visita")
            .WithDescription("Recupera uma visita")
            .WithOrder(4)
            .Produces<Response<Viewing?>>()
            .Produces<Response<Viewing?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que retorna os detalhes de uma visita pelo seu ID.
    /// </summary>
    /// <remarks>
    /// Este método cria uma requisição para buscar uma visita específica com base no ID
fornecido
    /// e chama o handler para processar a requisição.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IViewingHandler"/> responsável
pelas operações relacionadas a visitas.</param>
    /// <param name="id">ID da visita a ser recuperada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da visita, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IViewingHandler handler,
        long id)
    {
        var request = new GetViewingByIdRequest
        {
            Id = id,
            UserId = user.Identity?.Name ?? string.Empty
        };
        var result = await handler.GetByIdAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Viewings\RescheduleViewingEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Viewings;

/// <summary>
/// Endpoint responsável por reagendar uma visita.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de reagendamento de
visitas.
/// </remarks>
public class RescheduleViewingEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para reagendar uma visita.
    /// </summary>
    /// <remarks>
    /// Registra a rota PUT que recebe os dados atualizados da visita e o ID da visita a
ser reagendada.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPut("/{id:long}/reschedule", HandlerAsync)
            .WithName("Viewings: Reschedule")
            .WithSummary("Reagenda uma visita")
            .WithDescription("Reagenda uma visita")
            .WithOrder(2)
            .Produces<Response<Viewing?>>()
            .Produces<Response<Viewing?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que reagenda uma visita.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados atualizados da visita, associa o ID do usuário
autenticado e o ID da visita,
    /// e chama o handler para realizar o reagendamento.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IViewingHandler"/> responsável
pelas operações relacionadas a visitas.</param>
    /// <param name="request">Objeto <see cref="Viewing"/> contendo os dados atualizados
da visita.</param>
    /// <param name="id">ID da visita a ser reagendada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 200 OK com os detalhes da visita reagendada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IViewingHandler handler,
        Viewing request,
        long id)
    {
        request.Id = id;
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.RescheduleAsync(request);

        return result.IsSuccess
            ? Results.Ok(result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Endpoints\Viewings\ScheduleViewingEndpoint.cs

```
using RealtyHub.ApiService.Common.Api;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Security.Claims;

namespace RealtyHub.ApiService.Endpoints.Viewings;

/// <summary>
/// Endpoint responsável por agendar uma nova visita.
/// </summary>
/// <remarks>
/// Implementa a interface <see cref="IEndpoint"/> para mapear a rota de agendamento de
visitas.
/// </remarks>
public class ScheduleViewingEndpoint : IEndpoint
{
    /// <summary>
    /// Mapeia o endpoint para agendar uma nova visita.
    /// </summary>
    /// <remarks>
    /// Registra a rota POST que recebe os dados da visita e cria um novo agendamento.
    /// </remarks>
    /// <param name="app">O construtor de rotas do aplicativo <see
cref="IEndpointRouteBuilder"/>.</param>
    public static void Map(IEndpointRouteBuilder app)
        => app.MapPost("/", HandlerAsync)
            .WithName("Viewings: Schedule")
            .WithSummary("Agenda uma nova visita")
            .WithDescription("Agenda uma nova visita")
            .WithOrder(1)
            .Produces<Response<Viewing?>>(StatusCodes.Status201Created)
            .Produces<Response<Viewing?>>(StatusCodes.Status400BadRequest);

    /// <summary>
    /// Manipulador da rota que agenda uma nova visita.
    /// </summary>
    /// <remarks>
    /// Este método recebe os dados da visita, associa o ID do usuário autenticado à
visita,
    /// e chama o handler para criar o agendamento.
    /// </remarks>
    /// <param name="user">Objeto <see cref="ClaimsPrincipal"/> contendo os dados do
usuário autenticado.</param>
    /// <param name="handler">Instância de <see cref="IViewingHandler"/> responsável
pelas operações relacionadas a visitas.</param>
    /// <param name="request">Objeto <see cref="Viewing"/> contendo os dados da visita a
ser agendada.</param>
    /// <returns>
    /// Um objeto <see cref="IResult"/> representando a resposta HTTP:
    /// <para>- HTTP 201 Created com os detalhes da visita agendada, se a operação for
bem-sucedida;</para>
    /// <para>- HTTP 400 Bad Request, se houver erros na requisição.</para>
    /// </returns>
    private static async Task<IResult> HandlerAsync(
        ClaimsPrincipal user,
        IViewingHandler handler,
        Viewing request)
    {
        request.UserId = user.Identity?.Name ?? string.Empty;
        var result = await handler.ScheduleAsync(request);

        return result.IsSuccess
            ? Results.Created($"/{result.Data?.Id}", result)
            : Results.BadRequest(result);
    }
}
```

.\RealtyHub.ApiService\Handlers\CondominiumHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Extensions;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas a condomínios.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="ICondominiumHandler"/> e fornece
/// métodos para criar, atualizar, deletar e buscar condomínios no banco de dados.
/// </remarks>
public class CondominiumHandler : ICondominiumHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com condomínios.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nos condomínios.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="CondominiumHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nos condomínios.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com
condomínios.</param>
    public CondominiumHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Cria um novo condomínio no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método adiciona um novo condomínio à base de dados e salva as alterações.
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para criação do
condomínio.</param>
    /// <returns>Retorna uma resposta com o condomínio criado e o código de
status.</returns>
    public async Task<Response<Condominium?>> CreateAsync(Condominium request)
    {
        try
        {
            var condominium = new Condominium
            {
                Name = request.Name,
                Address = request.Address,
                Units = request.Units,
                Floors = request.Floors,
                HasElevator = request.HasElevator,
                HasSwimmingPool = request.HasSwimmingPool,
                HasPartyRoom = request.HasPartyRoom,
                HasPlayground = request.HasPlayground,
                HasFitnessRoom = request.HasFitnessRoom,
                CondominiumValue = request.CondominiumValue,
                IsActive = true
            };

            await _context.Condominiums.AddAsync(condominium);
            await _context.SaveChangesAsync();
```

```
            return new Response<Condominium?>(condominium, 201);
        }
        catch
        {
            return new Response<Condominium?>(null, 500, "Não foi possível criar o
condomínio");
        }
    }

    /// <summary>
    /// Atualiza um condomínio existente.
    /// </summary>
    /// <remarks>
    /// Este método atualiza as informações de um condomínio existente no banco de dados.
    /// </remarks>
    /// <param name="request">Objeto contendo as novas informações do condomínio.</param>
    /// <returns>Retorna a resposta com o condomínio atualizado ou um erro se não for
encontrado.</returns>
    public async Task<Response<Condominium?>> UpdateAsync(Condominium request)
    {
        try
        {
            var condominuim = await _context
                .Condominiums
                .FirstOrDefaultAsync(c => c.Id == request.Id && c.IsActive);

            if (condominuim is null)
                return new Response<Condominium?>(null, 404, "Condomínio não
encontrado");

            condominuim.Name = request.Name;
            condominuim.Address = request.Address;
            condominuim.Units = request.Units;
            condominuim.Floors = request.Floors;
            condominuim.HasElevator = request.HasElevator;
            condominuim.HasSwimmingPool = request.HasSwimmingPool;
            condominuim.HasPartyRoom = request.HasPartyRoom;
            condominuim.HasPlayground = request.HasPlayground;
            condominuim.HasFitnessRoom = request.HasFitnessRoom;
            condominuim.CondominiumValue = request.CondominiumValue;
            condominuim.UpdatedAt = DateTime.UtcNow;

            _context.Condominiums.Update(condominuim);
            await _context.SaveChangesAsync();

            return new Response<Condominium?>(condominuim);
        }
        catch
        {
            return new Response<Condominium?>(null, 500, "Não foi possível atualizar o
condomínio");
        }
    }

    /// <summary>
    /// Realiza a exclusão lógica de um condomínio, marcando-o como inativo.
    /// </summary>
    /// <remarks>
    /// Este método altera o status de um condomínio para inativo no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do condomínio a ser
excluído.</param>
    /// <returns>Retorna a resposta com o condomínio atualizado ou um erro se não for
encontrado.</returns>
    public async Task<Response<Condominium?>> DeleteAsync(DeleteCondominiumRequest
request)
    {
        try
        {
            var condominuim = await _context
                .Condominiums
                .FirstOrDefaultAsync(c => c.Id == request.Id && c.IsActive);

            if (condominuim is null)
                return new Response<Condominium?>(null, 404, "Condomínio não
```

```
encontrado");

            condominuim.IsActive = false;
            condominuim.UpdatedAt = DateTime.UtcNow;

            _context.Condominiums.Update(condominuim);
            await _context.SaveChangesAsync();

            return new Response<Condominium?>(condominuim);
        }
        catch
        {
            return new Response<Condominium?>(null, 500, "Não foi possível deletar o
condomínio");
        }
    }

    /// <summary>
    /// Obtém um condomínio específico pelo ID.
    /// </summary>
    /// <remarks>
    /// Este método busca um condomínio no banco de dados com base no ID fornecido.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do condomínio desejado.</param>
    /// <returns>Retorna o objeto do condomínio ou um erro caso não seja
encontrado.</returns>
    public async Task<Response<Condominium?>> GetByIdAsync(GetCondominiumByIdRequest
request)
    {
        try
        {
            var condominuim = await _context
                .Condominiums
                .AsNoTracking()
                .FirstOrDefaultAsync(c => c.Id == request.Id && c.IsActive);

            return condominuim is null
                ? new Response<Condominium?>(null, 404, "Condomínio não encontrado")
                : new Response<Condominium?>(condominuim);
        }
        catch
        {
            return new Response<Condominium?>(null, 500, "Não foi possível buscar o
condomínio");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todos os condomínios ativos, com opção de filtro
por busca.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todos os condomínios ativos no banco de dados e aplica
    /// filtros de busca, se fornecidos.  </para>
    /// <para> A resposta é paginada com base nos parâmetros
    /// de paginação fornecidos na requisição e retorna uma lista de condomínios.</para>
    /// <para>Caso o filtro não seja fornecido, retorna todos os condomínios
ativos.</para>
    /// </remarks>
    /// <param name="request">Requisição que contém parâmetros de paginação e
filtro.</param>
    /// <returns>Retorna uma resposta paginada com os condomínios ativos com base no
filtro ou um erro em caso de falha.</returns>
    public async Task<PagedResponse<List<Condominium>?>>
GetAllAsync(GetAllCondominiumsRequest request)
    {
        try
        {
            var query = _context
                .Condominiums
                .AsNoTracking()
                .Where(c => c.IsActive);

            if (!string.IsNullOrEmpty(request.FilterBy))
                query = query.FilterByProperty(request.SearchTerm, request.FilterBy);
```

```
            var count = await query.CountAsync();

            var condominiums = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            return new PagedResponse<List<Condominium>?>(condominiums, count,
request.PageNumber, request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Condominium>?>(null, 500, "Não foi possível
buscar os condomínios");
        }
    }
}
```

.\RealtyHub.ApiService\Handlers\ContractHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.ApiService.Services;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas a contratos.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="IContractHandler"/> e fornece
/// métodos para criar, atualizar, deletar e buscar contratos no banco de dados.
/// </remarks>
public class ContractHandler : IContractHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com contratos.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nos contratos.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ContractHandler"/>.
    /// </summary>
    /// <param name="context">Contexto do banco de dados para interação com
contratos.</param>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nos contratos.
    /// </remarks>
    public ContractHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Cria um novo contrato no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método cria um novo contrato baseado em uma proposta aceita,
    /// associando comprador, vendedor e dados adicionais.
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para criação do
contrato.</param>
    /// <returns>Retorna uma resposta com o contrato criado e o código de
status.</returns>
    public async Task<Response<Contract?>> CreateAsync(Contract request)
    {
        try
        {
            var offer = await _context
                .Offers
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                .ThenInclude(o => o!.Seller)
                .Include(o => o.Payments)
                .FirstOrDefaultAsync(o => o.Id == request.OfferId);

            if (offer is null)
                return new Response<Contract?>(null, 404, "Proposta não encontrada");

            if (offer.OfferStatus != EOfferStatus.Accepted)
                return new Response<Contract?>(null, 400, "A proposta precisa estar
aceita para criar um contrato");
```

```
            _context.Attach(offer);

            var contract = new Contract
            {
                IssueDate = request.IssueDate,
                SignatureDate = request.SignatureDate,
                EffectiveDate = request.EffectiveDate,
                TermEndDate = request.TermEndDate,
                FileId = Guid.NewGuid().ToString(),
                Buyer = offer.Buyer,
                BuyerId = offer.Buyer!.Id,
                SellerId = offer.Property!.SellerId,
                Seller = offer.Property.Seller,
                Offer = offer,
                OfferId = offer.Id,
                IsActive = true,
                UserId = request.UserId
            };

            await CreateOrUpdateContract(contract, false);

            await _context.Contracts.AddAsync(contract);
            await _context.SaveChangesAsync();

            return new Response<Contract?>(contract, 201, "Contrato criado com sucesso");
        }
        catch
        {
            return new Response<Contract?>(null, 500, "Não foi possível criar o
contrato");
        }
    }

    /// <summary>
    /// Atualiza um contrato existente no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método busca o contrato e ajusta as informações conforme a requisição
fornecida.
    /// </remarks>
    /// <param name="request">Objeto contendo as novas informações do contrato.</param>
    /// <returns>Retorna a resposta com o contrato atualizado ou um erro se não for
encontrado.</returns>
    public async Task<Response<Contract?>> UpdateAsync(Contract request)
    {
        try
        {
            var contract = await _context
                .Contracts
                .Include(o => o.Seller)
                .Include(o => o.Buyer)
                .Include(o => o.Offer)
                .ThenInclude(o => o!.Property)
                .FirstOrDefaultAsync(c => c.Id == request.Id
                                          && c.UserId == request.UserId
                                          && c.IsActive);

            if (contract is null)
                return new Response<Contract?>(null, 404, "Contrato não encontrado");

            var offer = await _context
                .Offers
                .Include(offer => offer.Property)
                .ThenInclude(property => property!.Seller)
                .Include(o => o.Payments)
                .FirstOrDefaultAsync(o => o.Id == request.OfferId);

            if (offer is null)
                return new Response<Contract?>(null, 404, "Proposta não encontrada");

            contract.IssueDate = request.IssueDate;
            contract.SignatureDate = request.SignatureDate;
            contract.EffectiveDate = request.EffectiveDate;
            contract.TermEndDate = request.TermEndDate;
            contract.Buyer = request.Buyer;
            contract.BuyerId = request.BuyerId;
```

```
            contract.SellerId = offer.Property!.SellerId;
            contract.Seller = offer.Property.Seller;
            contract.Offer = offer;
            contract.OfferId = request.OfferId;
            contract.FileId = request.FileId;
            contract.IsActive = request.IsActive;
            contract.UpdatedAt = DateTime.UtcNow;

            await CreateOrUpdateContract(contract, true);

            await _context.SaveChangesAsync();

            return new Response<Contract?>(contract, 200, "Contrato atualizado com
sucesso");
        }
        catch
        {
            return new Response<Contract?>(null, 500, "Não foi possível atualizar o
contrato");
        }
    }

    /// <summary>
    /// Realiza a exclusão lógica de um contrato, marcando-o como inativo.
    /// </summary>
    /// <remarks>
    /// Não remove o registro do banco, apenas altera seu status de ativo para inativo.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do contrato a ser
excluído.</param>
    /// <returns>Retorna a resposta com o status da exclusão ou um erro se não for
encontrado.</returns>
    public async Task<Response<Contract?>> DeleteAsync(DeleteContractRequest request)
    {
        try
        {
            var contract = await _context
                .Contracts
                .FirstOrDefaultAsync(c => c.Id == request.Id
                                          && c.UserId == request.UserId
                                          && c.IsActive);

            if (contract is null)
                return new Response<Contract?>(null, 404, "Contrato não encontrado");

            contract.IsActive = false;
            contract.UpdatedAt = DateTime.UtcNow;
            await _context.SaveChangesAsync();

            return new Response<Contract?>(null, 204, "Contrato excluído com sucesso");
        }
        catch
        {
            return new Response<Contract?>(null, 500, "Não foi possível excluir o
contrato");
        }
    }

    /// <summary>
    /// Obtém um contrato específico pelo ID.
    /// </summary>
    /// <remarks>
    /// Este método localiza o contrato correspondente ao ID fornecido
    /// e retorna o objeto completo, se encontrado.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do contrato desejado.</param>
    /// <returns>Retorna o objeto do contrato ou um erro caso não seja
encontrado.</returns>
    public async Task<Response<Contract?>> GetByIdAsync(GetContractByIdRequest request)
    {
        try
        {
            var contract = await _context
                .Contracts
                .AsNoTracking()
                .Include(c => c.Offer)
```

```
                .ThenInclude(o => o!.Property)
                .Include(c => c.Buyer)
                .Include(c => c.Seller)
                .FirstOrDefaultAsync(c => c.Id == request.Id
                                          && c.UserId == request.UserId
                                          && c.IsActive);

            if (contract is null)
                return new Response<Contract?>(null, 404, "Contrato não encontrado");

            contract.EffectiveDate = contract.EffectiveDate?.ToLocalTime();
            contract.IssueDate = contract.IssueDate?.ToLocalTime();
            contract.SignatureDate = contract.SignatureDate?.ToLocalTime();
            contract.TermEndDate = contract.TermEndDate?.ToLocalTime();

            return new Response<Contract?>(contract);
        }
        catch
        {
            return new Response<Contract?>(null, 500, "Não foi possível encontrar o
contrato");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todos os contratos ativos, com opção de filtro por
data.
    /// </summary>
    /// <remarks>
    /// <para>Este método aplica paginação, filtra por data de emissão e
    /// retorna apenas contratos marcados como ativos. </para>
    /// <para> Caso o filtro não seja fornecido, retorna todos os contratos
ativos.</para>
    /// </remarks>
    /// <param name="request">Requisição que contém parâmetros de paginação e
filtro.</param>
    /// <returns>Retorna uma resposta paginada com os contratos ativos ou um erro em caso
de falha.</returns>
    public async Task<PagedResponse<List<Contract>?>> GetAllAsync(GetAllContractsRequest
request)
    {
        try
        {
            var query = _context
                .Contracts
                .AsNoTracking()
                .Include(c => c.Offer)
                .ThenInclude(o => o!.Property)
                .Include(c => c.Buyer)
                .Include(c => c.Seller)
                .Where(c => c.UserId == request.UserId && c.IsActive);

            if (request.StartDate is not null && request.EndDate is not null)
            {
                var startDate = DateTime.Parse(request.StartDate).ToUniversalTime();
                var endDate = DateTime.Parse(request.EndDate).ToUniversalTime();

                query = query.Where(v => v.IssueDate >= startDate
                                         && v.IssueDate <= endDate);
            }

            var contracts = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            foreach (var contract in contracts)
            {
                contract.EffectiveDate = contract.EffectiveDate?.ToLocalTime();
                contract.IssueDate = contract.IssueDate?.ToLocalTime();
                contract.SignatureDate = contract.SignatureDate?.ToLocalTime();
                contract.TermEndDate = contract.TermEndDate?.ToLocalTime();
            }

            var count = await query.CountAsync();
```

```
            return new PagedResponse<List<Contract>?>(contracts, count,
request.PageNumber, request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Contract>?>(null, 500, "Não foi possível
retornar os contratos");
        }
    }

    /// <summary>
    /// Gera ou atualiza o arquivo de contrato com base nas informações fornecidas e no
modelo apropriado.
    /// </summary>
    /// <remarks>
    /// Este método carrega o modelo correto, atualiza o documento de contrato e salva em
disco.
    /// </remarks>
    /// <param name="contract">Objeto do contrato a ser gerado ou atualizado.</param>
    /// <param name="update">Indica se o contrato está sendo atualizado.</param>
    private async Task CreateOrUpdateContract(Contract contract, bool update)
    {
        var contractModelType = contract.Buyer?.PersonType switch
        {
            EPersonType.Business when contract.Seller?.PersonType == EPersonType.Business
                => EContractModelType.PJPJ,
            EPersonType.Individual when contract.Seller?.PersonType ==
EPersonType.Individual
                => EContractModelType.PFPF,
            EPersonType.Business when contract.Seller?.PersonType ==
EPersonType.Individual
                => EContractModelType.PFPJ,
            _ => EContractModelType.PJPF
        };

        var contractModel = await _context
            .ContractTemplates
            .FirstOrDefaultAsync(cm => cm.Type == contractModelType);

        var docxContractGenerator = new ContractGenerator();
        var pathContactFile = Path.Combine(Configuration.ContractsPath,
$"{contract.Id}.docx");
        if (update)
            File.Delete(pathContactFile);
        docxContractGenerator.GenerateContract(contract, contractModel);
    }
}
```

.\RealtyHub.ApiService\Handlers\ContractTemplatesHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas aos modelos de contrato.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="IContractTemplateHandler"/> e fornece
/// métodos para buscar modelos de contrato no banco de dados.
/// /// </remarks>
public class ContractTemplatesHandler : IContractTemplateHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com modelos de contrato.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nos modelos de contratos.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ContractTemplatesHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nos modelos de contrato.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com os modelos de
contrato.</param>
    public ContractTemplatesHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Obtém todos os modelos de contrato disponíveis no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método busca todos os modelos de contrato e retorna uma lista com os
resultados.
    /// </remarks>
    /// <returns>Retorna uma lista de modelos de contrato ou um erro em caso de
falha.</returns>
    public async Task<Response<List<ContractTemplate>?>> GetAllAsync()
    {
        try
        {
            var contracts = await _context
                .ContractTemplates
                .AsNoTracking()
                .ToListAsync();

            return new Response<List<ContractTemplate>?>(contracts);
        }
        catch
        {
            return new Response<List<ContractTemplate>?>(null, 500, "Não foi possível
buscar os modelos de contrato");
        }
    }
}
```

.\RealtyHub.ApiService\Handlers\CustomerHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas a clientes.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="ICustomerHandler"/> e fornece
/// métodos para criar, atualizar, deletar e buscar clientes no banco de dados.
/// </remarks>
public class CustomerHandler : ICustomerHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com clientes.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nos clientes.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="CustomerHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nos clientes.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com
clientes.</param>
    public CustomerHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Cria um novo cliente no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método adiciona um novo cliente à base de dados e salva as alterações.
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para criação do
cliente.</param>
    /// <returns>Retorna uma resposta com o cliente criado ou um erro se o cliente já
existir.</returns>
    public async Task<Response<Customer?>> CreateAsync(Customer request)
    {
        try
        {
            var isCustomerExists = await _context
                .Customers
                .AnyAsync(c => (c.DocumentNumber == request.DocumentNumber
                               || c.Email == request.Email)
                               && c.UserId == request.UserId);

            if (isCustomerExists)
                return new Response<Customer?>(null, 400, "Cliente já cadastrado");

            var customer = new Customer
            {
                Name = request.Name,
                Email = request.Email,
                Phone = request.Phone,
                DocumentNumber = request.DocumentNumber,
                Nationality = request.Nationality,
                MaritalStatus = request.MaritalStatus,
                Occupation = request.Occupation,
```

```
                PersonType = request.PersonType,
                CustomerType = request.CustomerType,
                Address = request.Address,
                Rg = request.Rg,
                IssuingAuthority = request.IssuingAuthority,
                RgIssueDate = request.RgIssueDate,
                BusinessName = request.BusinessName,
                UserId = request.UserId,
                IsActive = true
            };
            await _context.Customers.AddAsync(customer);
            await _context.SaveChangesAsync();

            return new Response<Customer?>(customer, 201, "Cliente criado com sucesso");
        }
        catch
        {
            return new Response<Customer?>(null, 500, "Não foi possível criar o
cliente");
        }
    }

    /// <summary>
    /// Atualiza um cliente existente no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método atualiza as informações de um cliente existente no banco de dados.
    /// </remarks>
    /// <param name="request">Objeto contendo as novas informações do cliente.</param>
    /// <returns>Retorna a resposta com o cliente atualizado ou um erro se não for
encontrado.</returns>
    public async Task<Response<Customer?>> UpdateAsync(Customer request)
    {
        try
        {
            var customer = await _context
                .Customers
                .FirstOrDefaultAsync(c => c.Id == request.Id
                                          && (string.IsNullOrEmpty(c.UserId) || c.UserId
== request.UserId)
                                          && c.IsActive);

            if (customer is null)
                return new Response<Customer?>(null, 404, "Cliente não encontrado");

            customer.Name = request.Name;
            customer.Email = request.Email;
            customer.Phone = request.Phone;
            customer.DocumentNumber = request.DocumentNumber;
            customer.Nationality = request.Nationality;
            customer.MaritalStatus = request.MaritalStatus;
            customer.Occupation = request.Occupation;
            customer.IssuingAuthority = request.IssuingAuthority;
            customer.RgIssueDate = request.RgIssueDate;
            customer.PersonType = request.PersonType;
            customer.CustomerType = request.CustomerType;
            customer.Address = request.Address;
            customer.Rg = request.Rg;
            customer.BusinessName = request.BusinessName;
            customer.UserId = request.UserId;
            customer.UpdatedAt = DateTime.UtcNow;

            _context.Customers.Update(customer);
            await _context.SaveChangesAsync();

            return new Response<Customer?>(customer);
        }
        catch
        {
            return new Response<Customer?>(null, 500, "Não foi possível atualizar o
cliente");
        }
    }

    /// <summary>
    /// Realiza a exclusão lógica de um cliente, marcando-o como inativo.
```

```
    /// </summary>
    /// <remarks>
    /// Este método altera o status de um cliente para inativo no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do cliente a ser
excluído.</param>
    /// <returns>Retorna a resposta com o status da exclusão ou um erro se não for
encontrado.</returns>
    public async Task<Response<Customer?>> DeleteAsync(DeleteCustomerRequest request)
    {
        try
        {
            var customer = await _context
                .Customers
                .FirstOrDefaultAsync(c => c.Id == request.Id
                                          && (string.IsNullOrEmpty(c.UserId) || c.UserId
== request.UserId)
                                          && c.IsActive);

            if (customer is null)
                return new Response<Customer?>(null, 404, "Cliente não encontrado");

            customer.IsActive = false;
            customer.UpdatedAt = DateTime.UtcNow;

            await _context.SaveChangesAsync();

            return new Response<Customer?>(null, 204, "Cliente excluído com sucesso");
        }
        catch
        {
            return new Response<Customer?>(null, 500, "Não foi possível excluir o
cliente");
        }
    }

    /// <summary>
    /// Obtém um cliente específico pelo ID.
    /// </summary>
    /// <remarks>
    /// Este método busca um cliente no banco de dados com base no ID fornecido.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do cliente desejado.</param>
    /// <returns>Retorna o objeto do cliente ou um erro caso não seja
encontrado.</returns>
    public async Task<Response<Customer?>> GetByIdAsync(GetCustomerByIdRequest request)
    {
        try
        {
            var customer = await _context
                .Customers
                .Include(c => c.Properties)
                .AsNoTracking()
                .FirstOrDefaultAsync(c => c.Id == request.Id
                                          && (c.UserId == request.UserId ||
string.IsNullOrEmpty(c.UserId))
                                          && c.IsActive);

            return customer is null
                ? new Response<Customer?>(null, 404, "Cliente não encontrado")
                : new Response<Customer?>(customer);
        }
        catch
        {
            return new Response<Customer?>(null, 500, "Não foi possível retornar o
cliente");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todos os clientes ativos, com opção de filtro por
busca.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todos os clientes ativos no banco de dados e aplica
    /// filtros de busca e paginação. </para>
```

```
    /// <para> Caso nenhum filtro seja fornecido, retorna todos os clientes ativos.
</para>
    /// </remarks>
    /// <param name="request">Requisição que contém parâmetros de paginação e
filtro.</param>
    /// <returns>Retorna uma resposta paginada com os clientes ativos ou um erro em caso
de falha.</returns>
    public async Task<PagedResponse<List<Customer>?>> GetAllAsync(GetAllCustomersRequest
request)
    {
        try
        {
            var query = _context
                .Customers
                .Include(c => c.Properties)
                .AsNoTracking()
                .Where(c => (c.UserId == request.UserId ||
string.IsNullOrEmpty(c.UserId))
                            && c.IsActive);

            if (!string.IsNullOrEmpty(request.SearchTerm))
            {
                var lowerSearchTerm = request.SearchTerm.ToLower();
                query = query.Where(c => c.Name.ToLower().Contains(lowerSearchTerm) ||
                                         c.Email.ToLower().Contains(lowerSearchTerm) ||

c.DocumentNumber.ToLower().Contains(lowerSearchTerm));
            }

            query = query.OrderBy(c => c.Name);

            var customers = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            var count = await query.CountAsync();

            return new PagedResponse<List<Customer>?>(customers, count,
request.PageNumber, request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Customer>?>(null, 500, "Não foi possível
consultar os clientes");
        }
    }
}
```

.\RealtyHub.ApiService\Handlers\OfferHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas a propostas.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="IOfferHandler"/> e fornece
/// métodos para criar, atualizar, rejeitar, aceitar e buscar propostas no banco de
dados.
/// </remarks>
public class OfferHandler : IOfferHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com propostas.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nas propostas.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="OfferHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nas propostas.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com
propostas.</param>
    public OfferHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Cria uma nova proposta no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método adiciona uma nova proposta à base de dados e salva as alterações.
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para criação da
proposta.</param>
    /// <returns>Retorna uma resposta com a proposta criada ou um erro.</returns>
    public async Task<Response<Offer?>> CreateAsync(Offer request)
    {
        Property? property;
        try
        {
            property = await _context
                .Properties
                .AsNoTracking()
                .FirstOrDefaultAsync(p => p.Id == request.PropertyId
                                       && p.IsActive);

            if (property is null)
                return new Response<Offer?>(null, 404, "Imóvel não encontrado");

            _context.Attach(property);
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Falha ao obter imóvel");
        }
```

```
        try
        {
            var payments = request.Payments.Select(paymentRequest => new Payment
            {
                Amount = paymentRequest.Amount,
                PaymentType = paymentRequest.PaymentType,
                Installments = paymentRequest.Installments,
                InstallmentsCount = paymentRequest.InstallmentsCount,
                UserId = request.UserId,
                IsActive = true
            }).ToList();

            var total = payments.Sum(p => p.Amount);
            if (total < request.Amount)
                return new Response<Offer?>(null, 400,
                    "O valor total dos pagamentos não corresponde ao valor da proposta");

            request.Buyer!.IsActive = true;
            var offer = new Offer
            {
                SubmissionDate = request.SubmissionDate,
                Amount = request.Amount,
                OfferStatus = request.OfferStatus,
                BuyerId = request.BuyerId,
                Buyer = request.Buyer,
                Payments = payments,
                PropertyId = request.PropertyId,
                Property = property,
                UserId = request.UserId
            };

            await _context.Offers.AddAsync(offer);
            await _context.SaveChangesAsync();

            return new Response<Offer?>(offer, 201, "Proposta criada com sucesso");
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Não foi possível criar a proposta");
        }
    }

    /// <summary>
    /// Atualiza uma proposta existente no banco de dados.
    /// </summary>
    /// <remarks>
    /// <para>Este método atualiza as informações de uma proposta existente no banco de
dados.</para>
    /// <para>O método verifica se a proposta existe e se o status da proposta permite a
atualização.</para>
    /// <para>Além disso, ele valida se o valor total dos pagamentos corresponde ao valor
da proposta.</para>
    /// <para>Se a proposta for encontrada e o status permitir, as informações da
proposta são atualizadas.</para>
    /// </remarks>
    /// <param name="request">Objeto contendo as novas informações da proposta.</param>
    /// <returns>Retorna a resposta com a proposta atualizada ou um erro se não for
encontrada.</returns>
    public async Task<Response<Offer?>> UpdateAsync(Offer request)
    {
        try
        {
            var offer = await _context
                .Offers
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                .Include(o => o.Payments)
                .Include(o => o.Contract)
                .FirstOrDefaultAsync(o => o.Id == request.Id);

            if (offer is null)
                return new Response<Offer?>(null, 404,
                    "Proposta não encontrada");

            if (offer.OfferStatus is EOfferStatus.Accepted or EOfferStatus.Rejected)
                return new Response<Offer?>(null, 400,
```

```
                "Não é possível atualizar uma proposta aceita ou rejeitada");

            var total = request.Payments.Sum(p => p.Amount);
            if (total < request.Amount)
                return new Response<Offer?>(null, 400,
                    "O valor total dos pagamentos não corresponde ao valor da proposta");

            var payments = await _context
                .Payments
                .Where(p => p.OfferId == request.Id
                            && p.UserId == request.UserId)
                .ToListAsync();

            offer.SubmissionDate = request.SubmissionDate;
            offer.Amount = request.Amount;
            offer.PropertyId = request.PropertyId;
            offer.BuyerId = request.BuyerId;
            offer.OfferStatus = request.OfferStatus;
            offer.UpdatedAt = DateTime.UtcNow;
            foreach (var payment in payments)
            {
                var updatePaymentRequest = request.Payments
                    .FirstOrDefault(p => p.Id == payment.Id);

                if (updatePaymentRequest is not null)
                {
                    payment.Amount = updatePaymentRequest.Amount;
                    payment.PaymentType = updatePaymentRequest.PaymentType;
                    payment.Installments = updatePaymentRequest.Installments;
                    payment.InstallmentsCount = updatePaymentRequest.InstallmentsCount;
                    payment.UpdatedAt = DateTime.UtcNow;
                    payment.IsActive = true;
                }
                else
                {
                    payment.IsActive = false;
                    payment.UpdatedAt = DateTime.UtcNow;
                }
            }

            foreach (var payment in request.Payments)
            {
                var isExist = payments.Any(p => p.Id == payment.Id);
                if (isExist) continue;

                var newPayment = new Payment
                {
                    Amount = payment.Amount,
                    PaymentType = payment.PaymentType,
                    Installments = payment.Installments,
                    InstallmentsCount = payment.InstallmentsCount,
                    IsActive = true,
                    OfferId = offer.Id,
                    Offer = offer,
                    UserId = request.UserId
                };

                _context.Attach(newPayment);
                offer.Payments.Add(newPayment);
            }

            _context.Offers.Update(offer);
            await _context.SaveChangesAsync();

            return new Response<Offer?>(offer, 200, "Proposta atualizada com sucesso");
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Não foi possível atualizar a
proposta");
        }
    }

    /// <summary>
    /// Rejeita uma proposta existente.
    /// </summary>
```

```
    /// <remarks>
    /// Este método rejeita uma proposta existente no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da proposta a ser
rejeitada.</param>
    /// <returns>Retorna a resposta com a proposta rejeitada ou um erro se não for
encontrada.</returns>
    public async Task<Response<Offer?>> RejectAsync(RejectOfferRequest request)
    {
        try
        {
            var offer = await _context
                .Offers
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                .Include(o => o.Payments)
                .Include(o => o.Contract)
                .FirstOrDefaultAsync(o => o.Id == request.Id);

            if (offer is null)
                return new Response<Offer?>(null, 404,
                    "Proposta não encontrada");

            switch (offer.OfferStatus)
            {
                case EOfferStatus.Accepted:
                    return new Response<Offer?>(null, 400,
                        "Não é possível rejeitar uma proposta aceita");
                case EOfferStatus.Rejected:
                    return new Response<Offer?>(null, 400,
                        "Proposta já está recusada");
            }

            offer.OfferStatus = EOfferStatus.Rejected;
            offer.UpdatedAt = DateTime.UtcNow;

            _context.Offers.Update(offer);
            await _context.SaveChangesAsync();

            return new Response<Offer?>(offer, 200, "Proposta rejeitada com sucesso");
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Não foi possível rejeitar a
proposta");
        }
    }

    /// <summary>
    /// Aceita uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Este método aceita uma proposta existente no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da proposta a ser aceita.</param>
    /// <returns>Retorna a resposta com a proposta aceita ou um erro se não for
encontrada.</returns>
    public async Task<Response<Offer?>> AcceptAsync(AcceptOfferRequest request)
    {
        try
        {
            var offer = await _context
                .Offers
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                .Include(o => o.Payments)
                .Include(o => o.Contract)
                .FirstOrDefaultAsync(o => o.Id == request.Id);

            if (offer is null)
                return new Response<Offer?>(null, 404,
                    "Proposta não encontrada");

            switch (offer.OfferStatus)
            {
                case EOfferStatus.Accepted:
```

```
                    return new Response<Offer?>(null, 400,
                        "Proposta já está aceita");
                case EOfferStatus.Rejected:
                    return new Response<Offer?>(null, 400,
                        "Não é possível aceitar uma proposta recusada");
            }

            offer.OfferStatus = EOfferStatus.Accepted;
            offer.UpdatedAt = DateTime.UtcNow;

            _context.Offers.Update(offer);
            await _context.SaveChangesAsync();

            return new Response<Offer?>(offer, 200, "Proposta aceita com sucesso");
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Não foi possível aceitar a
proposta");
        }
    }

    /// <summary>
    /// Obtém uma proposta específica pelo ID.
    /// </summary>
    /// <remarks>
    /// Este método busca uma proposta no banco de dados com base no ID fornecido.
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da proposta desejada.</param>
    /// <returns>Retorna a proposta ou um erro caso não seja encontrada.</returns>
    public async Task<Response<Offer?>> GetByIdAsync(GetOfferByIdRequest request)
    {
        try
        {
            var offer = await _context
                .Offers
                .AsNoTracking()
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                    .ThenInclude(p => p!.Seller)
                .Include(o => o.Payments)
                .Include(o => o.Contract)
                .FirstOrDefaultAsync(o => o.Id == request.Id);

            if (offer is null)
                return new Response<Offer?>(null, 404, "Proposta não encontrada");

            offer.SubmissionDate = offer.SubmissionDate?.ToLocalTime();

            return new Response<Offer?>(offer);
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Não foi possível buscar a proposta");
        }
    }

    /// <summary>
    /// Obtém a proposta aceita para um imóvel específico.
    /// </summary>
    /// <remarks>
    /// Este método busca a proposta aceita para um imóvel específico no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição contendo o ID do imóvel.</param>
    /// <returns>Retorna a proposta aceita ou um erro caso não seja encontrada.</returns>
    public async Task<Response<Offer?>> GetAcceptedByProperty(GetOfferAcceptedByProperty
request)
    {
        try
        {
            var offer = await _context
                .Offers
                .AsNoTracking()
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                    .ThenInclude(p => p!.Seller)
```

```
                .Include(o => o.Contract)
                .Include(o => o.Payments)
                .FirstOrDefaultAsync(o => o.PropertyId == request.PropertyId
                                          && o.OfferStatus == EOfferStatus.Accepted);

            if (offer is null)
                return new Response<Offer?>(null, 404, "Proposta aceita não encontrada");

            offer.SubmissionDate = offer.SubmissionDate?.ToLocalTime();

            return new Response<Offer?>(offer, 200, "Proposta aceita encontrada");
        }
        catch
        {
            return new Response<Offer?>(null, 500, "Não foi possível buscar a proposta
aceita");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todas as propostas de um imóvel específico.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todas as propostas de um imóvel específico no banco de
dados.</para>
    /// <para>Ele aplica filtros de data e paginação, retornando uma lista de
propostas.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo parâmetros de paginação e filtro por
imóvel.</param>
    /// <returns>Retorna uma resposta paginada com as propostas ou um erro em caso de
falha.</returns>
    public async Task<PagedResponse<List<Offer>?>> GetAllOffersByPropertyAsync(
        GetAllOffersByPropertyRequest request)
    {
        try
        {
            var property = await _context
                .Properties
                .AsNoTracking()
                .FirstOrDefaultAsync(p => p.Id == request.PropertyId
                                          && p.IsActive);

            if (property is null)
                return new PagedResponse<List<Offer>?>(null, 404,
                    "Imóvel não encontrado");

            var query = _context
                .Offers
                .AsNoTracking()
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                    .ThenInclude(p => p!.Seller)
                .Include(o => o.Payments)
                .Include(o => o.Contract)
                .Where(o => o.PropertyId == request.PropertyId);

            if (request.StartDate is not null && request.EndDate is not null)
            {
                var startDate = DateTime.Parse(request.StartDate).ToUniversalTime();
                var endDate = DateTime.Parse(request.EndDate).ToUniversalTime();

                query = query.Where(o => o.SubmissionDate >= startDate
                                         && o.SubmissionDate <= endDate);
            }

            query = query.OrderBy(o => o.SubmissionDate);

            var offers = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            foreach (var offer in offers)
            {
                offer.SubmissionDate = offer.SubmissionDate?.ToLocalTime();
```

```
            }

            var count = await query.CountAsync();

            return new PagedResponse<List<Offer>?>(offers, count, request.PageNumber,
request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Offer>?>(null, 500, "Não foi possível buscar as
propostas");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todas as propostas de um cliente específico.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todas as propostas de um cliente específico no banco de
dados.</para>
    /// <para>Ele aplica filtros de data e paginação, retornando uma lista de
propostas.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo parâmetros de paginação e filtro por
cliente.</param>
    /// <returns>Retorna uma resposta paginada com as propostas ou um erro em caso de
falha.</returns>
    public async Task<PagedResponse<List<Offer>?>> GetAllOffersByCustomerAsync(
       GetAllOffersByCustomerRequest request)
    {
        try
        {
            var customer = await _context
                .Customers
                .AsNoTracking()
                .FirstOrDefaultAsync(p => p.Id == request.CustomerId
                                          && p.IsActive);

            if (customer is null)
                return new PagedResponse<List<Offer>?>(null, 404,
                    "Cliente não encontrado");

            var query = _context
                .Offers
                .AsNoTracking()
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                    .ThenInclude(p => p!.Seller)
                .Include(o => o.Payments)
                .Include(o => o.Contract)
                .Where(o => o.BuyerId == request.CustomerId
                            && o.UserId == request.UserId);

            if (request.StartDate is not null && request.EndDate is not null)
            {
                var startDate = DateTime.Parse(request.StartDate).ToUniversalTime();
                var endDate = DateTime.Parse(request.EndDate).ToUniversalTime();

                query = query.Where(o => o.SubmissionDate >= startDate
                                         && o.SubmissionDate <= endDate);
            }

            query = query.OrderBy(o => o.SubmissionDate);

            var offers = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            foreach (var offer in offers)
            {
                offer.SubmissionDate = offer.SubmissionDate?.ToLocalTime();
            }

            var count = await query.CountAsync();
```

```
            return new PagedResponse<List<Offer>?>(offers, count, request.PageNumber,
request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Offer>?>(null, 500, "Não foi possível buscar as
propostas");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todas as propostas no sistema.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todas as propostas no banco de dados.</para>
    /// <para>Ele aplica filtros de data e paginação, retornando uma lista de
propostas.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo parâmetros de paginação e
filtro.</param>
    /// <returns>Retorna uma resposta paginada com as propostas ou um erro em caso de
falha.</returns>
    public async Task<PagedResponse<List<Offer>?>> GetAllAsync(GetAllOffersRequest
request)
    {
        try
        {
            IQueryable<Offer> query = _context
                .Offers
                .AsNoTracking()
                .Include(o => o.Buyer)
                .Include(o => o.Property)
                .ThenInclude(p => p!.Seller)
                .Include(o => o.Payments)
                .Include(o => o.Contract);

            if (request.StartDate is not null && request.EndDate is not null)
            {
                var startDate = DateTime.Parse(request.StartDate).ToUniversalTime();
                var endDate = DateTime.Parse(request.EndDate).ToUniversalTime();

                query = query.Where(o => o.SubmissionDate >= startDate
                                         && o.SubmissionDate <= endDate);
            }

            var offers = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            foreach (var offer in offers)
            {
                offer.SubmissionDate = offer.SubmissionDate?.ToLocalTime();
            }

            var count = await query.CountAsync();

            return new PagedResponse<List<Offer>?>(offers, count, request.PageNumber,
request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Offer>?>(null, 500, "Não foi possível buscar as
propostas");
        }
    }
}
```

.\RealtyHub.ApiService\Handlers\PropertyHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Extensions;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas a imóveis.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="IPropertyHandler"/> e fornece
/// métodos para criar, atualizar, deletar e buscar imóveis no banco de dados.
/// </remarks>
public class PropertyHandler : IPropertyHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com imóveis.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nos imóveis.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="PropertyHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nos imóveis.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com
imóveis.</param>
    public PropertyHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Cria um novo imóvel no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método adiciona um novo imóvel à base de dados e salva as alterações.
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para criação do
imóvel.</param>
    /// <returns>Retorna uma resposta com o imóvel criado e o código de status.</returns>
    public async Task<Response<Property?>> CreateAsync(Property request)
    {
        try
        {
            var property = new Property
            {
                Title = request.Title,
                Description = request.Description,
                Price = request.Price,
                Address = request.Address,
                PropertyType = request.PropertyType,
                Bedroom = request.Bedroom,
                Bathroom = request.Bathroom,
                Area = request.Area,
                Garage = request.Garage,
                IsNew = request.IsNew,
                RegistryNumber = request.RegistryNumber,
                RegistryRecord = request.RegistryRecord,
                TransactionsDetails = request.TransactionsDetails,
                UserId = request.UserId,
                CondominiumId = request.CondominiumId,
                ShowInHome = request.ShowInHome,
```

```
                SellerId = request.SellerId,
                IsActive = true
            };

            await _context.Properties.AddAsync(property);
            await _context.SaveChangesAsync();

            return new Response<Property?>(property, 201, "Imóvel criado com sucesso");
        }
        catch
        {
            return new Response<Property?>(null, 500, "Não foi possível criar o imóvel");
        }
    }

    /// <summary>
    /// Atualiza um imóvel existente no banco de dados.
    /// </summary>
    /// <remarks>
    /// Este método atualiza as informações de um imóvel existente e salva as alterações.
    /// </remarks>
    /// <param name="request">Objeto contendo as novas informações do imóvel.</param>
    /// <returns>Retorna a resposta com o imóvel atualizado ou um erro se não for
encontrado.</returns>
    public async Task<Response<Property?>> UpdateAsync(Property request)
    {
        try
        {
            var property = await _context
                .Properties
                .FirstOrDefaultAsync(p => p.Id == request.Id && p.IsActive);

            if (property is null)
                return new Response<Property?>(null, 404, "Imóvel não encontrado");

            property.Title = request.Title;
            property.Description = request.Description;
            property.Price = request.Price;
            property.Address = request.Address;
            property.PropertyType = request.PropertyType;
            property.Bedroom = request.Bedroom;
            property.Bathroom = request.Bathroom;
            property.Area = request.Area;
            property.Garage = request.Garage;
            property.IsNew = request.IsNew;
            property.RegistryNumber = request.RegistryNumber;
            property.RegistryRecord = request.RegistryRecord;
            property.TransactionsDetails = request.TransactionsDetails;
            property.UserId = request.UserId;
            property.ShowInHome = request.ShowInHome;
            property.SellerId = request.SellerId;
            property.CondominiumId = request.CondominiumId;
            property.UpdatedAt = DateTime.UtcNow;

            _context.Properties.Update(property);
            await _context.SaveChangesAsync();

            return new Response<Property?>(property, 200, "Imóvel atualizado com
sucesso");
        }
        catch
        {
            return new Response<Property?>(null, 500, "Não foi possível atualizar o
imóvel");
        }
    }

    /// <summary>
    /// Realiza a exclusão lógica de um imóvel, marcando-o como inativo.
    /// </summary>
    /// <remarks>
    /// Este método altera o status de um imóvel para inativo no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do imóvel a ser
excluído.</param>
    /// <returns>Retorna a resposta com o status da exclusão ou um erro se não for
```

```
encontrado.</returns>
    public async Task<Response<Property?>> DeleteAsync(DeletePropertyRequest request)
    {
        try
        {
            var property = await _context
                .Properties
                .FirstOrDefaultAsync(p => p.Id == request.Id
                                       && p.UserId == request.UserId
                                       && p.IsActive);

            if (property is null)
                return new Response<Property?>(null, 404, "Imóvel não encontrado");

            property.IsActive = false;
            property.UpdatedAt = DateTime.UtcNow;

            await _context.SaveChangesAsync();

            return new Response<Property?>(null, 204, "Imóvel deletado com sucesso");
        }
        catch
        {
            return new Response<Property?>(null, 500, "Não foi possível deletar o
imóvel");
        }
    }

    /// <summary>
    /// Obtém um imóvel específico pelo ID.
    /// </summary>
    /// <remarks>
    /// Este método busca um imóvel no banco de dados com base no ID fornecido.
    /// </remarks>
    /// <param name="request">Requisição que contém o ID do imóvel desejado.</param>
    /// <returns>Retorna o objeto do imóvel ou um erro caso não seja
encontrado.</returns>
    public async Task<Response<Property?>> GetByIdAsync(GetPropertyByIdRequest request)
    {
        try
        {
            var query = _context
                .Properties
                .AsNoTracking()
                .Include(p => p.Condominium)
                .Include(p => p.Seller)
                .Include(p => p.PropertyPhotos.Where(photo => photo.IsActive))
                .Where(p => p.Id == request.Id && p.IsActive);

            var property = await query.FirstOrDefaultAsync();

            return property is null
                ? new Response<Property?>(null, 404, "Imóvel não encontrado")
                : new Response<Property?>(property);
        }
        catch
        {
            return new Response<Property?>(null, 500, "Não foi possível retornar o
imóvel");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todos os imóveis ativos, com opção de filtro por
busca.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todos os imóveis ativos no banco de dados e aplica
    /// filtros de paginação e busca, se fornecidos.</para>
    /// <para> Caso o filtro não seja fornecido, retorna todos os imóveis ativos.</para>
    /// </remarks>
    /// <param name="request">Requisição que contém parâmetros de paginação e
filtro.</param>
    /// <returns>Retorna uma resposta paginada com os imóveis ativos ou um erro em caso
de falha.</returns>
    public async Task<PagedResponse<List<Property>?>> GetAllAsync(GetAllPropertiesRequest
```

```
request)
    {
        try
        {
            var query = _context
                .Properties
                .AsNoTracking()
                .Include(p => p.Condominium)
                .Include(p => p.Seller)
                .Include(p => p.PropertyPhotos.Where(photos => photos.IsActive))
                .Where(p => p.IsActive);

            if (!string.IsNullOrEmpty(request.UserId))
                query = query.Where(v => v.UserId == request.UserId);

            if (!string.IsNullOrEmpty(request.FilterBy))
                query = query.FilterByProperty(request.SearchTerm, request.FilterBy);

            query = query.OrderBy(p => p.Title);

            var properties = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            var count = await query.CountAsync();

            return new PagedResponse<List<Property>?>(properties, count,
request.PageNumber, request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Property>?>(null, 500, "Não foi possível
retornar os imóveis");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todas as visitas de um imóvel específico.
    /// </summary>
    /// <remarks>
    /// <para> Este método busca todas as visitas de um imóvel específico no banco de
dados e aplica
    /// filtros de paginação e busca, se fornecidos.</para>
    /// <para> Caso o filtro não seja fornecido, retorna todas as visitas do
imóvel.</para>
    /// </remarks>
    /// <param name="request">Requisição que contém parâmetros de paginação e filtro por
data.</param>
    /// <returns>Retorna uma resposta paginada com as visitas ou um erro em caso de
falha.</returns>
    public async Task<PagedResponse<List<Viewing>?>>
GetAllViewingsAsync(GetAllViewingsByPropertyRequest request)
    {
        try
        {
            var query = _context
                .Viewing
                .AsNoTracking()
                .Include(v => v.Buyer)
                .Include(v => v.Property)
                .Where(v => v.PropertyId == request.PropertyId);

            if (request.StartDate is not null && request.EndDate is not null)
            {
                var startDate = DateTime.Parse(request.StartDate).ToUniversalTime();
                var endDate = DateTime.Parse(request.EndDate).ToUniversalTime();

                query = query.Where(v => v.ViewingDate >= startDate
                                    && v.ViewingDate <= endDate);
            }

            query = query.OrderBy(v => v.ViewingDate);

            var viewings = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
```

```
                .Take(request.PageSize)
                .ToListAsync();

            var count = await query.CountAsync();

            return new PagedResponse<List<Viewing>?>(viewings, count, request.PageNumber,
request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Viewing>?>(null, 500, "Não foi possível
retornar as visitas");
        }
    }
}
```

.\RealtyHub.ApiService\Handlers\PropertyPhotosHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas às fotos de imóveis.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="IPropertyPhotosHandler"/> e fornece
/// métodos para criar, atualizar, deletar e buscar fotos de imóveis no banco de dados.
/// </remarks>
public class PropertyPhotosHandler : IPropertyPhotosHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com fotos dos imóveis.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nas fotos dos imóveis.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="PropertyPhotosHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nas fotos dos imóveis.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com as fotos de
imóveis.</param>
    public PropertyPhotosHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Cria novas fotos para um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método adiciona novas fotos à base de dados e salva as alterações.
    /// </remarks>
    /// <param name="request">Requisição contendo as informações e os arquivos para
criação das fotos.</param>
    /// <returns>Retorna uma resposta indicando o sucesso ou falha da operação.</returns>
    public async Task<Response<PropertyPhoto?>> CreateAsync(CreatePropertyPhotosRequest
request)
    {
        try
        {
            if (request.HttpRequest is null)
                return new Response<PropertyPhoto?>(null, 400, "Requisição inválida");

            var property = await _context
                .Properties
                .AsNoTracking()
                .FirstOrDefaultAsync(p =>
                    p.Id == request.PropertyId
                    && p.UserId == request.UserId
                    && p.IsActive);

            if (property is null)
                return new Response<PropertyPhoto?>(null, 404, "Imóvel não encontrado");

            _context.Attach(property);

            if (!request.HttpRequest.HasFormContentType)
                return new Response<PropertyPhoto?>(null, 400, "Conteúdo do tipo
```

```
multipart/form-data esperado");

            var form = await request.HttpRequest.ReadFormAsync();
            var files = form.Files;

            if (files.Count == 0)
                return new Response<PropertyPhoto?>(null, 400, "Nenhum arquivo
encontrado");

            var photosToCreate = new List<PropertyPhoto>();
            var photosToUpdate = new List<PropertyPhoto>();

            foreach (var file in files)
            {
                var id = file.Headers["Id"].FirstOrDefault();
                var isThumbnail = bool.Parse(file.Headers["IsThumbnail"].FirstOrDefault()
?? "false");
                var idPhoto = string.IsNullOrEmpty(id) ? Guid.NewGuid().ToString() : id;
                var extension = Path.GetExtension(file.FileName);
                var fileName = $"{idPhoto}{extension}";
                var fullFileName = Path.Combine(Configuration.PhotosPath, fileName);

                await using var stream = new FileStream(fullFileName, FileMode.Create);
                await file.CopyToAsync(stream);

                var propertyPhoto = new PropertyPhoto
                {
                    Id = idPhoto,
                    Extension = extension,
                    IsThumbnail = isThumbnail,
                    PropertyId = request.PropertyId,
                    Property = property,
                    IsActive = true,
                    UserId = request.UserId
                };
                if (string.IsNullOrEmpty(id))
                    photosToCreate.Add(propertyPhoto);
                else
                    photosToUpdate.Add(propertyPhoto);
            }

            if (photosToCreate.Any(p => p.IsThumbnail))
            {
                await _context.PropertyPhotos
                    .Where(pi =>
                        pi.PropertyId == request.PropertyId
                        && pi.UserId == request.UserId
                        && pi.IsActive
                        && pi.IsThumbnail)
                    .ForEachAsync(pi =>
                    {
                        _context.Attach(pi);
                        pi.IsThumbnail = false;
                        pi.UpdatedAt = DateTime.UtcNow;
                    });
            }

            await _context.PropertyPhotos.AddRangeAsync(photosToCreate);
            _context.PropertyPhotos.UpdateRange(photosToUpdate);
            await _context.SaveChangesAsync();

            return new Response<PropertyPhoto?>(null, 201);
        }
        catch
        {
            return new Response<PropertyPhoto?>(null, 500, "Não foi possível criar as
fotos");
        }
    }

    /// <summary>
    /// Atualiza as informações de fotos de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método atualiza as informações de fotos existentes no banco de dados.
    /// </remarks>
```

```
    /// <param name="request">Requisição contendo as informações das fotos a serem
atualizadas.</param>
    /// <returns>Retorna uma resposta indicando o sucesso ou falha da operação.</returns>
    public async Task<Response<List<PropertyPhoto>?>>
UpdateAsync(UpdatePropertyPhotosRequest request)
    {
        try
        {
            if (request.Photos.Any(p => p.IsThumbnail))
            {
                await _context.PropertyPhotos
                    .Where(pi =>
                        pi.PropertyId == request.PropertyId
                        && pi.UserId == request.UserId
                        && pi.IsActive
                        && pi.IsThumbnail)
                    .ForEachAsync(pi =>
                    {
                        _context.Attach(pi);
                        pi.IsThumbnail = false;
                        pi.UpdatedAt = DateTime.UtcNow;
                    });
            }

            foreach (var photo in request.Photos)
            {
                var existingEntity = await _context
                    .PropertyPhotos
                    .FirstOrDefaultAsync(pi =>
                        pi.Id == photo.Id
                        && pi.UserId == request.UserId
                        && pi.IsActive);

                if (existingEntity is null)
                    continue;

                _context.Attach(existingEntity);

                existingEntity.IsThumbnail = photo.IsThumbnail;
                existingEntity.UpdatedAt = DateTime.UtcNow;
            }

            await _context.SaveChangesAsync();

            return new Response<List<PropertyPhoto>?>();
        }
        catch
        {
            return new Response<List<PropertyPhoto>?>(null, 500, "Não foi possível
atualizar as fotos");
        }
    }

    /// <summary>
    /// Exclui uma foto de um imóvel.
    /// </summary>
    /// <remarks>
    /// Este método marca uma foto como inativa no banco de dados.
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da foto a ser excluída.</param>
    /// <returns>Retorna uma resposta indicando o sucesso ou falha da operação.</returns>
    public async Task<Response<PropertyPhoto?>> DeleteAsync(DeletePropertyPhotoRequest
request)
    {
        try
        {
            var propertyPhoto = await _context
                .PropertyPhotos
                .FirstOrDefaultAsync(pi =>
                    pi.Id == request.Id
                    && pi.UserId == request.UserId
                    && pi.IsActive);

            if (propertyPhoto is null)
                return new Response<PropertyPhoto?>(null, 404, "Foto não encontrada");
```

```
            _context.Attach(propertyPhoto);

            propertyPhoto.IsActive = false;
            propertyPhoto.IsThumbnail = false;
            propertyPhoto.UpdatedAt = DateTime.UtcNow;

            await _context.SaveChangesAsync();

            return new Response<PropertyPhoto?>(null, 204, "Foto excluída com sucesso");
        }
        catch
        {
            return new Response<PropertyPhoto?>(null, 500, "Não foi possível deletar a
foto");
        }
    }

    /// <summary>
    /// Obtém todas as fotos de um imóvel específico.
    /// </summary>
    /// <remarks>
    /// Este método busca todas as fotos de um imóvel no banco de dados com base no ID do
imóvel.
    /// </remarks>
    /// <param name="request">Requisição contendo o ID do imóvel.</param>
    /// <returns>Retorna uma lista de fotos do imóvel ou um erro em caso de
falha.</returns>
    public async Task<Response<List<PropertyPhoto>?>> GetAllByPropertyAsync(
       GetAllPropertyPhotosByPropertyRequest request)
    {
        try
        {
            var propertyPhotos = await _context
                .PropertyPhotos
                .AsNoTracking()
                .Where(p =>
                    p.PropertyId == request.PropertyId
                    && p.UserId == request.UserId
                    && p.IsActive)
                .OrderBy(p => p.IsThumbnail)
                .ToListAsync();

            return new Response<List<PropertyPhoto>?>(propertyPhotos);
        }
        catch
        {
            return new Response<List<PropertyPhoto>?>(null, 500, "Não foi possível buscar
a foto");
        }
    }
}
```

.\RealtyHub.ApiService\Handlers\ViewingHandler.cs

```
using Microsoft.EntityFrameworkCore;
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;

namespace RealtyHub.ApiService.Handlers;

/// <summary>
/// Responsável pelas operações relacionadas às visitas de imóveis.
/// </summary>
/// <remarks>
/// Esta classe implementa a interface <see cref="IViewingHandler"/> e fornece
/// métodos para agendar, reagendar, finalizar e cancelar visitas,
/// além de buscar visitas específicas ou todas as visitas.
/// </remarks>
public class ViewingHandler : IViewingHandler
{
    /// <summary>
    /// Contexto do banco de dados para interação com visitas.
    /// </summary>
    /// <remarks>
    /// Este campo é utilizado para realizar operações CRUD nas visitas.
    /// </remarks>
    private readonly AppDbContext _context;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ViewingHandler"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor é utilizado para injetar o contexto do banco de dados
    /// necessário para realizar operações CRUD nas visitas.
    /// </remarks>
    /// <param name="context">Contexto do banco de dados para interação com as
visitas.</param>
    public ViewingHandler(AppDbContext context)
    {
        _context = context;
    }

    /// <summary>
    /// Agenda uma nova visita para um imóvel.
    /// </summary>
    /// <remarks>
    /// <para>Este método adiciona uma nova visita à base de dados e salva as alterações.
</para>
    /// <para>Ele verifica se o cliente e o imóvel existem e estão ativos antes de
agendar a visita.</para>
    /// <para>Se a visita já estiver agendada para a mesma data, retorna um erro.</para>
    /// <para>O status da visita é definido como "Pendente" por padrão.</para>
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para agendamento da
visita.</param>
    /// <returns>Retorna uma resposta com a visita agendada ou um erro em caso de
falha.</returns>
    public async Task<Response<Viewing?>> ScheduleAsync(Viewing request)
    {
        try
        {
            var customer = await _context
                .Customers
                .FirstOrDefaultAsync(c => c.Id == request.BuyerId
                                          && (string.IsNullOrEmpty(c.UserId) || c.UserId
== request.UserId)
                                          && c.IsActive);

            if (customer is null)
                return new Response<Viewing?>(null, 404, "Cliente não encontrado");

            var property = await _context
```

```
                .Properties
                .FirstOrDefaultAsync(p => p.Id == request.PropertyId
                                          && (string.IsNullOrEmpty(p.UserId) || p.UserId
== request.UserId)
                                          && p.IsActive);

            if (property is null)
                return new Response<Viewing?>(null, 404, "Imóvel não encontrado");

            var isViewingExist = await _context
                .Viewing
                .AnyAsync(v => v.PropertyId == request.PropertyId
                               && v.ViewingDate == request.ViewingDate);

            if (isViewingExist)
                return new Response<Viewing?>(null, 400, "Visita já agendada para esta
data");

            var viewing = new Viewing
            {
                ViewingDate = request.ViewingDate,
                ViewingStatus = request.ViewingStatus,
                BuyerId = request.BuyerId,
                Buyer = customer,
                PropertyId = request.PropertyId,
                Property = property,
                UserId = request.UserId
            };

            _context.Viewing.Add(viewing);
            await _context.SaveChangesAsync();

            return new Response<Viewing?>(viewing, 201, "Visita agendada com sucesso");
        }
        catch
        {
            return new Response<Viewing?>(null, 500, "Não foi possível agendar a
visita");
        }
    }

    /// <summary>
    /// Reagenda uma visita existente.
    /// </summary>
    /// <remarks>
    /// <para>Este método atualiza a data e o status de uma visita existente.</para>
    /// <para>Ele verifica se a visita existe e se não está cancelada ou
finalizada.</para>
    /// <para>Se a visita não existir ou já estiver cancelada ou finalizada, retorna um
erro.</para>
    /// </remarks>
    /// <param name="request">Objeto contendo as informações para reagendamento da
visita.</param>
    /// <returns>Retorna uma resposta com a visita reagendada ou um erro em caso de
falha.</returns>
    public async Task<Response<Viewing?>> RescheduleAsync(Viewing request)
    {
        try
        {
            var viewing = await _context
                .Viewing
                .Include(v => v.Buyer)
                .Include(v => v.Property)
                .FirstOrDefaultAsync(v => v.Id == request.Id
                                          && (string.IsNullOrEmpty(v.UserId) || v.UserId
== request.UserId));

            if (viewing is null)
                return new Response<Viewing?>(null, 404, "Visita não encontrada");

            switch (viewing.ViewingStatus)
            {
                case EViewingStatus.Canceled:
                    return new Response<Viewing?>(null, 400, "Não é possível reagendar
uma visita cancelada");
                case EViewingStatus.Done:
```

```
                    return new Response<Viewing?>(null, 400, "Não é possível reagendar
uma visita finalizada");
            }

            viewing.ViewingDate = request.ViewingDate;
            viewing.UpdatedAt = DateTime.UtcNow;

            _context.Viewing.Update(viewing);
            await _context.SaveChangesAsync();

            return new Response<Viewing?>(viewing, message: "Visita reagendada com
sucesso");
        }
        catch
        {
            return new Response<Viewing?>(null, 500, "Não foi possível reagendar a
visita");
        }
    }

    /// <summary>
    /// Finaliza uma visita existente.
    /// </summary>
    /// <remarks>
    /// <para>Este método atualiza o status de uma visita para "Finalizada".</para>
    /// <para>Ele verifica se a visita existe e se não está cancelada ou já
finalizada.</para>
    /// <para>Se a visita não existir ou já estiver cancelada ou finalizada, retorna um
erro.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da visita a ser
finalizada.</param>
    /// <returns>Retorna uma resposta com a visita finalizada ou um erro em caso de
falha.</returns>
    public async Task<Response<Viewing?>> DoneAsync(DoneViewingRequest request)
    {
        try
        {
            var viewing = await _context
                .Viewing
                .Include(v => v.Buyer)
                .Include(v => v.Property)
                .FirstOrDefaultAsync(v => v.Id == request.Id
                                          && (string.IsNullOrEmpty(v.UserId) || v.UserId
== request.UserId));

            if (viewing is null)
                return new Response<Viewing?>(null, 404, "Visita não encontrada");

            switch (viewing.ViewingStatus)
            {
                case EViewingStatus.Canceled:
                    return new Response<Viewing?>(null, 400, "Não é possível finalizar
uma visita cancelada");
                case EViewingStatus.Done:
                    return new Response<Viewing?>(null, 400, "Visita já finalizada");
            }

            viewing.ViewingStatus = EViewingStatus.Done;
            viewing.UpdatedAt = DateTime.UtcNow;

            await _context.SaveChangesAsync();

            return new Response<Viewing?>(viewing, message: "Visita finalizada com
sucesso");
        }
        catch
        {
            return new Response<Viewing?>(null, 500, "Não foi possível finalizar a
visita");
        }
    }

    /// <summary>
    /// Cancela uma visita existente.
    /// </summary>
```

```
    /// <remarks>
    /// <para>Este método atualiza o status de uma visita para "Cancelada".</para>
    /// <para>Ele verifica se a visita existe e se não está finalizada.</para>
    /// <para>Se a visita não existir ou já estiver finalizada, retorna um erro.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da visita a ser cancelada.</param>
    /// <returns>Retorna uma resposta com a visita cancelada ou um erro em caso de
falha.</returns>
    public async Task<Response<Viewing?>> CancelAsync(CancelViewingRequest request)
    {
        try
        {
            var viewing = await _context
                .Viewing
                .Include(v => v.Buyer)
                .Include(v => v.Property)
                .FirstOrDefaultAsync(v => v.Id == request.Id
                                          && (string.IsNullOrEmpty(v.UserId) || v.UserId
== request.UserId));

            if (viewing is null)
                return new Response<Viewing?>(null, 404, "Visita não encontrada");

            switch (viewing.ViewingStatus)
            {
                case EViewingStatus.Done:
                    return new Response<Viewing?>(null, 400, "Não é possível cancelar uma
visita finalizada");
                case EViewingStatus.Canceled:
                    return new Response<Viewing?>(null, 400, "Visita já cancelada");
            }

            viewing.ViewingStatus = EViewingStatus.Canceled;
            viewing.UpdatedAt = DateTime.UtcNow;

            await _context.SaveChangesAsync();

            return new Response<Viewing?>(viewing, message: "Visita cancelada com
sucesso");
        }
        catch
        {
            return new Response<Viewing?>(null, 500, "Não foi possível cancelar a
visita");
        }
    }

    /// <summary>
    /// Obtém uma visita específica pelo ID.
    /// </summary>
    /// <remarks>
    /// <para>Este método busca uma visita no banco de dados com base no ID
fornecido.</para>
    /// <para>Ele verifica se a visita existe e se não está cancelada ou
finalizada.</para>
    /// <para>Se a visita não existir ou já estiver cancelada ou finalizada, retorna um
erro.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo o ID da visita desejada.</param>
    /// <returns>Retorna a visita ou um erro caso não seja encontrada.</returns>
    public async Task<Response<Viewing?>> GetByIdAsync(GetViewingByIdRequest request)
    {
        try
        {
            var viewing = await _context
                .Viewing
                .AsNoTracking()
                .Include(v => v.Buyer)
                .Include(v => v.Property)
                .ThenInclude(p => p!.Seller)
                .FirstOrDefaultAsync(v => v.Id == request.Id
                                          && (string.IsNullOrEmpty(v.UserId) || v.UserId
== request.UserId));

            if (viewing is null)
                return new Response<Viewing?>(null, 404, "Visita não encontrada");
```

```
            viewing.ViewingDate = viewing.ViewingDate?.ToLocalTime();
            return new Response<Viewing?>(viewing);
        }
        catch
        {
            return new Response<Viewing?>(null, 500, "Não foi possível retornar a
visita");
        }
    }

    /// <summary>
    /// Retorna uma lista paginada de todas as visitas.
    /// </summary>
    /// <remarks>
    /// <para>Este método busca todas as visitas no banco de dados e aplica filtros de
data e paginação.</para>
    /// <para>Caso o filtro de data não seja fornecido, retorna todas as visitas.</para>
    /// </remarks>
    /// <param name="request">Requisição contendo parâmetros de paginação e
filtro.</param>
    /// <returns>Retorna uma resposta paginada com as visitas ou um erro em caso de
falha.</returns>
    public async Task<PagedResponse<List<Viewing>?>> GetAllAsync(GetAllViewingsRequest
request)
    {
        try
        {
            var query = _context
                .Viewing
                .AsNoTracking()
                .Include(v => v.Buyer)
                .Include(v => v.Property)
                .ThenInclude(p => p!.Seller)
                .Where(v => string.IsNullOrEmpty(v.UserId) || v.UserId ==
request.UserId);

            if (request.StartDate is not null && request.EndDate is not null)
            {
                var startDate = DateTime.Parse(request.StartDate).ToUniversalTime();
                var endDate = DateTime.Parse(request.EndDate).ToUniversalTime();

                query = query.Where(v => v.ViewingDate >= startDate
                                         && v.ViewingDate <= endDate);
            }

            query = query.OrderBy(v => v.ViewingDate);

            var viewings = await query
                .Skip((request.PageNumber - 1) * request.PageSize)
                .Take(request.PageSize)
                .ToListAsync();

            foreach (var viewing in viewings)
            {
                viewing.ViewingDate = viewing.ViewingDate?.ToLocalTime();
            }

            var count = await query.CountAsync();

            return new PagedResponse<List<Viewing>?>(
                viewings, count, request.PageNumber, request.PageSize);
        }
        catch
        {
            return new PagedResponse<List<Viewing>?>(null, 500, "Não foi possível
retornar as visitas");
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\20250207012644_v1.Designer.cs

```
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
using RealtyHub.ApiService.Data;

#nullable disable

namespace RealtyHub.ApiService.Migrations
{
    [DbContext(typeof(AppDbContext))]
    [Migration("20250207012644_v1")]
    partial class v1
    {
        /// <inheritdoc />
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "8.0.11")
                .HasAnnotation("Relational:MaxIdentifierLength", 63);

            NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Name")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<long?>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique();

                    b.HasIndex("UserId");

                    b.ToTable("IdentityRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
```

```
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.ToTable("IdentityRoleClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.Property<string>("LoginProvider")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderKey")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderDisplayName")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityUserLogin", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");
```

```
                    b.HasKey("UserId", "RoleId");

                    b.ToTable("IdentityUserRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<string>("LoginProvider")
                        .HasMaxLength(120)
                        .HasColumnType("character varying(120)");

                    b.Property<string>("Name")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("Value")
                        .HasColumnType("text");

                    b.HasKey("UserId", "LoginProvider", "Name");

                    b.ToTable("IdentityUserToken", (string)null);
                });

            modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<int>("AccessFailedCount")
                        .HasColumnType("integer");

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Creci")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<string>("Email")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<bool>("EmailConfirmed")
                        .HasColumnType("boolean");

                    b.Property<string>("GivenName")
                        .IsRequired()
                        .HasMaxLength(100)
                        .HasColumnType("character varying(100)");

                    b.Property<bool>("LockoutEnabled")
                        .HasColumnType("boolean");

                    b.Property<DateTimeOffset?>("LockoutEnd")
                        .HasColumnType("timestamp with time zone");

                    b.Property<string>("NormalizedEmail")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("NormalizedUserName")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("PasswordHash")
                        .HasColumnType("text");
```

```
                    b.Property<string>("PhoneNumber")
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<bool>("PhoneNumberConfirmed")
                        .HasColumnType("boolean");

                    b.Property<string>("SecurityStamp")
                        .HasColumnType("text");

                    b.Property<bool>("TwoFactorEnabled")
                        .HasColumnType("boolean");

                    b.Property<string>("UserName")
                        .HasMaxLength(160)
                        .HasColumnType("character varying(160)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedEmail")
                        .IsUnique();

                    b.HasIndex("NormalizedUserName")
                        .IsUnique();

                    b.ToTable("IdentityUser", (string)null);
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<decimal>("CondominiumValue")
                        .HasColumnType("numeric");

                    b.Property<DateTime>("CreatedAt")
                        .HasColumnType("timestamp with time zone");

                    b.Property<int>("Floors")
                        .HasColumnType("integer");

                    b.Property<bool>("HasElevator")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasFitnessRoom")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasPartyRoom")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasPlayground")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasSwimmingPool")
                        .HasColumnType("boolean");

                    b.Property<bool>("IsActive")
                        .HasColumnType("boolean");

                    b.Property<string>("Name")
                        .IsRequired()
                        .HasMaxLength(120)
                        .HasColumnType("character varying(120)");

                    b.Property<int>("Units")
                        .HasColumnType("integer");

                    b.Property<DateTime>("UpdatedAt")
                        .HasColumnType("timestamp with time zone");
```

```
                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.HasKey("Id");

                    b.ToTable("Condominium", (string)null);
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<long>("BuyerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<DateTime?>("EffectiveDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<string>("FileId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<bool>("IsActive")
                        .HasColumnType("boolean");

                    b.Property<DateTime?>("IssueDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<long>("OfferId")
                        .HasColumnType("bigint");

                    b.Property<long>("SellerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime?>("SignatureDate")
                        .HasColumnType("timestamp with time zone");

                    b.Property<DateTime?>("TermEndDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.HasKey("Id");

                    b.HasIndex("BuyerId");

                    b.HasIndex("OfferId")
                        .IsUnique();

                    b.HasIndex("SellerId");

                    b.ToTable("Contract", (string)null);
                });

            modelBuilder.Entity("RealtyHub.Core.Models.ContractTemplate", b =>
```

```
            {
                b.Property<string>("Id")
                    .HasColumnType("text");

                b.Property<string>("Extension")
                    .IsRequired()
                    .HasColumnType("text");

                b.Property<string>("Name")
                    .IsRequired()
                    .HasColumnType("text");

                b.Property<bool>("ShowInPage")
                    .HasColumnType("boolean");

                b.Property<int>("Type")
                    .HasColumnType("integer");

                b.HasKey("Id");

                b.ToTable("ContractTemplate", (string)null);

                b.HasData(
                    new
                    {
                        Id = "a2c16556-5098-4496-ae7a-1f9b6d0e8fcf",
                        Extension = ".docx",
                        Name = "Modelo de Contrato - PJPJ",
                        ShowInPage = false,
                        Type = 1
                    },
                    new
                    {
                        Id = "f7581a63-f4f0-4881-b6ed-6a4100b4182e",
                        Extension = ".docx",
                        Name = "Modelo de Contrato - PFPF",
                        ShowInPage = false,
                        Type = 2
                    },
                    new
                    {
                        Id = "2f4c556d-6850-4b3d-afe9-d7c2bd282718",
                        Extension = ".docx",
                        Name = "Modelo de Contrato - PFPJ",
                        ShowInPage = false,
                        Type = 3
                    },
                    new
                    {
                        Id = "fd7ed50d-8f42-4288-8877-3cb8095370e7",
                        Extension = ".docx",
                        Name = "Modelo de Contrato - PJPF",
                        ShowInPage = false,
                        Type = 4
                    },
                    new
                    {
                        Id = "2824aec3-3219-4d81-a97a-c3b80ca72844",
                        Extension = ".pdf",
                        Name = "Modelo Padrão",
                        ShowInPage = true,
                        Type = 0
                    });
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<string>("BusinessName")
                    .IsRequired()
```

```
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<DateTime>("CreatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<int>("CustomerType")
            .HasColumnType("integer");

        b.Property<string>("DocumentNumber")
            .IsRequired()
            .HasMaxLength(20)
            .HasColumnType("character varying(20)");

        b.Property<string>("Email")
            .IsRequired()
            .HasMaxLength(50)
            .HasColumnType("character varying(50)");

        b.Property<bool>("IsActive")
            .ValueGeneratedOnAdd()
            .HasColumnType("boolean")
            .HasDefaultValue(true);

        b.Property<string>("IssuingAuthority")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<int>("MaritalStatus")
            .HasColumnType("integer");

        b.Property<string>("Name")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<string>("Nationality")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<string>("Occupation")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<int>("PersonType")
            .HasColumnType("integer");

        b.Property<string>("Phone")
            .IsRequired()
            .HasMaxLength(30)
            .HasColumnType("character varying(30)");

        b.Property<string>("Rg")
            .IsRequired()
            .HasMaxLength(20)
            .HasColumnType("character varying(20)");

        b.Property<DateTime?>("RgIssueDate")
            .HasColumnType("timestamp with time zone");

        b.Property<DateTime>("UpdatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");
```

```
                b.ToTable("Customer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<int>("OfferStatus")
                    .HasColumnType("integer");

                b.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b.Property<DateTime?>("SubmissionDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("PropertyId");

                b.ToTable("Offer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<bool>("Installments")
                    .HasColumnType("boolean");

                b.Property<int>("InstallmentsCount")
                    .HasColumnType("integer");

                b.Property<bool>("IsActive")
                    .ValueGeneratedOnAdd()
```

```
                .HasColumnType("boolean")
                .HasDefaultValue(true);

            b.Property<long>("OfferId")
                .HasColumnType("bigint");

            b.Property<int>("PaymentType")
                .HasColumnType("integer");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("OfferId");

            b.ToTable("Payment", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
        {
            b.Property<long>("Id")
                .HasColumnType("bigint");

            b.Property<double>("Area")
                .HasColumnType("double precision");

            b.Property<int>("Bathroom")
                .HasColumnType("integer");

            b.Property<int>("Bedroom")
                .HasColumnType("integer");

            b.Property<long>("CondominiumId")
                .HasColumnType("bigint");

            b.Property<DateTime>("CreatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("Description")
                .IsRequired()
                .HasMaxLength(255)
                .HasColumnType("character varying(255)");

            b.Property<int>("Garage")
                .HasColumnType("integer");

            b.Property<bool>("IsActive")
                .HasColumnType("boolean");

            b.Property<bool>("IsNew")
                .HasColumnType("boolean");

            b.Property<decimal>("Price")
                .HasColumnType("numeric");

            b.Property<int>("PropertyType")
                .HasColumnType("integer");

            b.Property<string>("RegistryNumber")
                .IsRequired()
                .HasColumnType("text");

            b.Property<string>("RegistryRecord")
                .IsRequired()
                .HasColumnType("text");

            b.Property<long>("SellerId")
```

```
                .HasColumnType("bigint");

            b.Property<bool>("ShowInHome")
                .HasColumnType("boolean");

            b.Property<string>("Title")
                .IsRequired()
                .HasMaxLength(120)
                .HasColumnType("character varying(120)");

            b.Property<string>("TransactionsDetails")
                .IsRequired()
                .HasColumnType("text");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("CondominiumId");

            b.HasIndex("SellerId");

            b.ToTable("Property", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("text");

            b.Property<DateTime>("CreatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("Extension")
                .IsRequired()
                .HasColumnType("text");

            b.Property<bool>("IsActive")
                .HasColumnType("boolean");

            b.Property<bool>("IsThumbnail")
                .HasColumnType("boolean");

            b.Property<long>("PropertyId")
                .HasColumnType("bigint");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("PropertyId");

            b.ToTable("PropertyPhotos", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint");
```

```
NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<long>("BuyerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<long>("PropertyId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<DateTime?>("ViewingDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<int>("ViewingStatus")
                        .HasColumnType("integer");

                    b.HasKey("Id");

                    b.HasIndex("BuyerId");

                    b.HasIndex("PropertyId");

                    b.ToTable("Viewing", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany("Roles")
                        .HasForeignKey("UserId");
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });
```

```
            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
                {
                    b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
                        {
                            b1.Property<long>("CondominiumId")
                                .HasColumnType("bigint");

                            b1.Property<string>("City")
                                .IsRequired()
                                .HasMaxLength(80)
                                .HasColumnType("character varying(80)")
                                .HasColumnName("City");

                            b1.Property<string>("Complement")
                                .IsRequired()
                                .HasMaxLength(80)
                                .HasColumnType("character varying(80)")
                                .HasColumnName("Complement");

                            b1.Property<string>("Country")
                                .IsRequired()
                                .HasMaxLength(80)
                                .HasColumnType("character varying(80)")
                                .HasColumnName("Country");

                            b1.Property<string>("Neighborhood")
                                .IsRequired()
                                .HasMaxLength(80)
                                .HasColumnType("character varying(80)")
                                .HasColumnName("Neighborhood");

                            b1.Property<string>("Number")
                                .IsRequired()
                                .HasColumnType("text")
                                .HasColumnName("Number");

                            b1.Property<string>("State")
                                .IsRequired()
                                .HasMaxLength(80)
                                .HasColumnType("character varying(80)")
                                .HasColumnName("State");

                            b1.Property<string>("Street")
                                .IsRequired()
                                .HasMaxLength(80)
                                .HasColumnType("character varying(80)")
                                .HasColumnName("Street");

                            b1.Property<string>("ZipCode")
                                .IsRequired()
                                .HasMaxLength(20)
                                .HasColumnType("character varying(20)")
                                .HasColumnName("ZipCode");

                            b1.HasKey("CondominiumId");

                            b1.ToTable("Condominium");

                            b1.WithOwner()
                                .HasForeignKey("CondominiumId");
                        });

                    b.Navigation("Address")
                        .IsRequired();
                });
```

```
modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithOne("Contract")
            .HasForeignKey("RealtyHub.Core.Models.Contract", "OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany()
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Offer");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
    {
        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("CustomerId")
                    .HasColumnType("bigint");

                b1.Property<string>("City")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("State");

                b1.Property<string>("Street")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");
```

```
                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("CustomerId");

                b1.ToTable("Customer");

                b1.WithOwner()
                    .HasForeignKey("CustomerId");
            });

        b.Navigation("Address")
            .IsRequired();
    });

modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany()
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithMany("Payments")
            .HasForeignKey("OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Offer");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Condominium", "Condominium")
            .WithMany()
            .HasForeignKey("CondominiumId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Condominium", null)
            .WithMany("Properties")
            .HasForeignKey("Id")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany("Properties")
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b1.Property<string>("City")
                    .IsRequired()
                    .HasMaxLength(80)
```

```
                    .HasColumnType("character varying(80)")
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("State");

                b1.Property<string>("Street")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("PropertyId");

                b1.ToTable("Property");

                b1.WithOwner()
                    .HasForeignKey("PropertyId");
            });

        b.Navigation("Address")
            .IsRequired();

        b.Navigation("Condominium");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany("PropertyPhotos")
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
```

```
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();

                b.HasOne("RealtyHub.Core.Models.Property", "Property")
                    .WithMany()
                    .HasForeignKey("PropertyId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();

                b.Navigation("Buyer");

                b.Navigation("Property");
            });

        modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
            {
                b.Navigation("Roles");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.Navigation("Properties");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
            {
                b.Navigation("Properties");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Navigation("Contract");

                b.Navigation("Payments");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
            {
                b.Navigation("PropertyPhotos");
            });
#pragma warning restore 612, 618
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\20250207012644_v1.cs

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

#pragma warning disable CA1814 // Prefer jagged arrays over multidimensional

namespace RealtyHub.ApiService.Migrations
{
    /// <inheritdoc />
    public partial class v1 : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Condominium",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Name = table.Column<string>(type: "character varying(120)",
maxLength: 120, nullable: false),
                    Street = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Neighborhood = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Number = table.Column<string>(type: "text", nullable: false),
                    City = table.Column<string>(type: "character varying(80)", maxLength:
80, nullable: false),
                    State = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Country = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    ZipCode = table.Column<string>(type: "character varying(20)",
maxLength: 20, nullable: false),
                    Complement = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Units = table.Column<int>(type: "integer", nullable: false),
                    Floors = table.Column<int>(type: "integer", nullable: false),
                    HasElevator = table.Column<bool>(type: "boolean", nullable: false),
                    HasSwimmingPool = table.Column<bool>(type: "boolean", nullable:
false),
                    HasPartyRoom = table.Column<bool>(type: "boolean", nullable: false),
                    HasPlayground = table.Column<bool>(type: "boolean", nullable: false),
                    HasFitnessRoom = table.Column<bool>(type: "boolean", nullable:
false),
                    CondominiumValue = table.Column<decimal>(type: "numeric", nullable:
false),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    IsActive = table.Column<bool>(type: "boolean", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Condominium", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "ContractTemplate",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Extension = table.Column<string>(type: "text", nullable: false),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Type = table.Column<int>(type: "integer", nullable: false),
                    ShowInPage = table.Column<bool>(type: "boolean", nullable: false)
```

```
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_ContractTemplate", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Customer",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Name = table.Column<string>(type: "character varying(80)", maxLength:
80, nullable: false),
                    Email = table.Column<string>(type: "character varying(50)",
maxLength: 50, nullable: false),
                    Phone = table.Column<string>(type: "character varying(30)",
maxLength: 30, nullable: false),
                    DocumentNumber = table.Column<string>(type: "character varying(20)",
maxLength: 20, nullable: false),
                    Occupation = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Nationality = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    PersonType = table.Column<int>(type: "integer", nullable: false),
                    CustomerType = table.Column<int>(type: "integer", nullable: false),
                    Street = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Neighborhood = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Number = table.Column<string>(type: "text", nullable: false),
                    City = table.Column<string>(type: "character varying(80)", maxLength:
80, nullable: false),
                    State = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Country = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    ZipCode = table.Column<string>(type: "character varying(20)",
maxLength: 20, nullable: false),
                    Complement = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Rg = table.Column<string>(type: "character varying(20)", maxLength:
20, nullable: false),
                    IssuingAuthority = table.Column<string>(type: "character
varying(80)", maxLength: 80, nullable: false),
                    RgIssueDate = table.Column<DateTime>(type: "timestamp with time
zone", nullable: true),
                    BusinessName = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    MaritalStatus = table.Column<int>(type: "integer", nullable: false),
                    IsActive = table.Column<bool>(type: "boolean", nullable: false,
defaultValue: true),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Customer", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "IdentityRoleClaim",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    RoleId = table.Column<long>(type: "bigint", nullable: false),
                    ClaimType = table.Column<string>(type: "character varying(255)",
maxLength: 255, nullable: true),
                    ClaimValue = table.Column<string>(type: "character varying(255)",
maxLength: 255, nullable: true)
```

```
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityRoleClaim", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "IdentityUser",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    GivenName = table.Column<string>(type: "character varying(100)",
maxLength: 100, nullable: false),
                    Creci = table.Column<string>(type: "text", nullable: false),
                    UserName = table.Column<string>(type: "character varying(160)",
maxLength: 160, nullable: true),
                    NormalizedUserName = table.Column<string>(type: "character
varying(180)", maxLength: 180, nullable: true),
                    Email = table.Column<string>(type: "character varying(180)",
maxLength: 180, nullable: true),
                    NormalizedEmail = table.Column<string>(type: "character
varying(180)", maxLength: 180, nullable: true),
                    EmailConfirmed = table.Column<bool>(type: "boolean", nullable:
false),
                    PasswordHash = table.Column<string>(type: "text", nullable: true),
                    SecurityStamp = table.Column<string>(type: "text", nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "text", nullable:
true),
                    PhoneNumber = table.Column<string>(type: "character varying(20)",
maxLength: 20, nullable: true),
                    PhoneNumberConfirmed = table.Column<bool>(type: "boolean", nullable:
false),
                    TwoFactorEnabled = table.Column<bool>(type: "boolean", nullable:
false),
                    LockoutEnd = table.Column<DateTimeOffset>(type: "timestamp with time
zone", nullable: true),
                    LockoutEnabled = table.Column<bool>(type: "boolean", nullable:
false),
                    AccessFailedCount = table.Column<int>(type: "integer", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityUser", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Property",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false),
                    SellerId = table.Column<long>(type: "bigint", nullable: false),
                    CondominiumId = table.Column<long>(type: "bigint", nullable: false),
                    Title = table.Column<string>(type: "character varying(120)",
maxLength: 120, nullable: false),
                    Description = table.Column<string>(type: "character varying(255)",
maxLength: 255, nullable: false),
                    Price = table.Column<decimal>(type: "numeric", nullable: false),
                    PropertyType = table.Column<int>(type: "integer", nullable: false),
                    Bedroom = table.Column<int>(type: "integer", nullable: false),
                    Bathroom = table.Column<int>(type: "integer", nullable: false),
                    Garage = table.Column<int>(type: "integer", nullable: false),
                    Area = table.Column<double>(type: "double precision", nullable:
false),
                    TransactionsDetails = table.Column<string>(type: "text", nullable:
false),
                    Street = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Neighborhood = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    Number = table.Column<string>(type: "text", nullable: false),
                    City = table.Column<string>(type: "character varying(80)", maxLength:
80, nullable: false),
                    State = table.Column<string>(type: "character varying(80)",
```

```
maxLength: 80, nullable: false),
                    Country = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    ZipCode = table.Column<string>(type: "character varying(20)",
maxLength: 20, nullable: false),
                    Complement = table.Column<string>(type: "character varying(80)",
maxLength: 80, nullable: false),
                    IsNew = table.Column<bool>(type: "boolean", nullable: false),
                    RegistryNumber = table.Column<string>(type: "text", nullable: false),
                    RegistryRecord = table.Column<string>(type: "text", nullable: false),
                    IsActive = table.Column<bool>(type: "boolean", nullable: false),
                    ShowInHome = table.Column<bool>(type: "boolean", nullable: false),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Property", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Property_Condominium_CondominiumId",
                        column: x => x.CondominiumId,
                        principalTable: "Condominium",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Property_Condominium_Id",
                        column: x => x.Id,
                        principalTable: "Condominium",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Property_Customer_SellerId",
                        column: x => x.SellerId,
                        principalTable: "Customer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "IdentityClaim",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    UserId = table.Column<long>(type: "bigint", nullable: false),
                    ClaimType = table.Column<string>(type: "character varying(255)",
maxLength: 255, nullable: true),
                    ClaimValue = table.Column<string>(type: "character varying(255)",
maxLength: 255, nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityClaim", x => x.Id);
                    table.ForeignKey(
                        name: "FK_IdentityClaim_IdentityUser_UserId",
                        column: x => x.UserId,
                        principalTable: "IdentityUser",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "IdentityRole",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Name = table.Column<string>(type: "character varying(256)",
maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(type: "character varying(256)",
maxLength: 256, nullable: true),
```

```
                    ConcurrencyStamp = table.Column<string>(type: "text", nullable:
true),
                    UserId = table.Column<long>(type: "bigint", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityRole", x => x.Id);
                    table.ForeignKey(
                        name: "FK_IdentityRole_IdentityUser_UserId",
                        column: x => x.UserId,
                        principalTable: "IdentityUser",
                        principalColumn: "Id");
                });

            migrationBuilder.CreateTable(
                name: "IdentityUserLogin",
                columns: table => new
                {
                    LoginProvider = table.Column<string>(type: "character varying(128)",
maxLength: 128, nullable: false),
                    ProviderKey = table.Column<string>(type: "character varying(128)",
maxLength: 128, nullable: false),
                    ProviderDisplayName = table.Column<string>(type: "character
varying(255)", maxLength: 255, nullable: true),
                    UserId = table.Column<long>(type: "bigint", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityUserLogin", x => new { x.LoginProvider,
x.ProviderKey });
                    table.ForeignKey(
                        name: "FK_IdentityUserLogin_IdentityUser_UserId",
                        column: x => x.UserId,
                        principalTable: "IdentityUser",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "IdentityUserRole",
                columns: table => new
                {
                    UserId = table.Column<long>(type: "bigint", nullable: false),
                    RoleId = table.Column<long>(type: "bigint", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityUserRole", x => new { x.UserId, x.RoleId
});
                    table.ForeignKey(
                        name: "FK_IdentityUserRole_IdentityUser_UserId",
                        column: x => x.UserId,
                        principalTable: "IdentityUser",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "IdentityUserToken",
                columns: table => new
                {
                    UserId = table.Column<long>(type: "bigint", nullable: false),
                    LoginProvider = table.Column<string>(type: "character varying(120)",
maxLength: 120, nullable: false),
                    Name = table.Column<string>(type: "character varying(180)",
maxLength: 180, nullable: false),
                    Value = table.Column<string>(type: "text", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_IdentityUserToken", x => new { x.UserId,
x.LoginProvider, x.Name });
                    table.ForeignKey(
                        name: "FK_IdentityUserToken_IdentityUser_UserId",
                        column: x => x.UserId,
                        principalTable: "IdentityUser",
```

```
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "Offer",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Amount = table.Column<decimal>(type: "numeric", nullable: false),
                    PropertyId = table.Column<long>(type: "bigint", nullable: false),
                    BuyerId = table.Column<long>(type: "bigint", nullable: false),
                    SubmissionDate = table.Column<DateTime>(type: "timestamp with time
zone", nullable: false),
                    OfferStatus = table.Column<int>(type: "integer", nullable: false),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Offer", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Offer_Customer_BuyerId",
                        column: x => x.BuyerId,
                        principalTable: "Customer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Offer_Property_PropertyId",
                        column: x => x.PropertyId,
                        principalTable: "Property",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "PropertyPhotos",
                columns: table => new
                {
                    Id = table.Column<string>(type: "text", nullable: false),
                    Extension = table.Column<string>(type: "text", nullable: false),
                    IsThumbnail = table.Column<bool>(type: "boolean", nullable: false),
                    PropertyId = table.Column<long>(type: "bigint", nullable: false),
                    IsActive = table.Column<bool>(type: "boolean", nullable: false),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_PropertyPhotos", x => x.Id);
                    table.ForeignKey(
                        name: "FK_PropertyPhotos_Property_PropertyId",
                        column: x => x.PropertyId,
                        principalTable: "Property",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "Viewing",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    ViewingDate = table.Column<DateTime>(type: "timestamp with time
zone", nullable: false),
                    ViewingStatus = table.Column<int>(type: "integer", nullable: false),
```

```
                    BuyerId = table.Column<long>(type: "bigint", nullable: false),
                    PropertyId = table.Column<long>(type: "bigint", nullable: false),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Viewing", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Viewing_Customer_BuyerId",
                        column: x => x.BuyerId,
                        principalTable: "Customer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Viewing_Property_PropertyId",
                        column: x => x.PropertyId,
                        principalTable: "Property",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "Contract",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    SellerId = table.Column<long>(type: "bigint", nullable: false),
                    BuyerId = table.Column<long>(type: "bigint", nullable: false),
                    OfferId = table.Column<long>(type: "bigint", nullable: false),
                    IssueDate = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false),
                    EffectiveDate = table.Column<DateTime>(type: "timestamp with time
zone", nullable: false),
                    TermEndDate = table.Column<DateTime>(type: "timestamp with time
zone", nullable: false),
                    SignatureDate = table.Column<DateTime>(type: "timestamp with time
zone", nullable: true),
                    FileId = table.Column<string>(type: "text", nullable: false),
                    IsActive = table.Column<bool>(type: "boolean", nullable: false),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Contract", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Contract_Customer_BuyerId",
                        column: x => x.BuyerId,
                        principalTable: "Customer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Contract_Customer_SellerId",
                        column: x => x.SellerId,
                        principalTable: "Customer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Contract_Offer_OfferId",
                        column: x => x.OfferId,
                        principalTable: "Offer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "Payment",
```

```
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Amount = table.Column<decimal>(type: "numeric", nullable: false),
                    PaymentType = table.Column<int>(type: "integer", nullable: false),
                    Installments = table.Column<bool>(type: "boolean", nullable: false),
                    InstallmentsCount = table.Column<int>(type: "integer", nullable:
false),
                    OfferId = table.Column<long>(type: "bigint", nullable: false),
                    IsActive = table.Column<bool>(type: "boolean", nullable: false,
defaultValue: true),
                    UserId = table.Column<string>(type: "text", nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()"),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone",
nullable: false, defaultValueSql: "NOW()")
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Payment", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Payment_Offer_OfferId",
                        column: x => x.OfferId,
                        principalTable: "Offer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.InsertData(
                table: "ContractTemplate",
                columns: new[] { "Id", "Extension", "Name", "ShowInPage", "Type" },
                values: new object[,]
                {
                    { "2824aec3-3219-4d81-a97a-c3b80ca72844", ".pdf", "Modelo Padrão",
true, 0 },
                    { "2f4c556d-6850-4b3d-afe9-d7c2bd282718", ".docx", "Modelo de
Contrato - PFPJ", false, 3 },
                    { "a2c16556-5098-4496-ae7a-1f9b6d0e8fcf", ".docx", "Modelo de
Contrato - PJPJ", false, 1 },
                    { "f7581a63-f4f0-4881-b6ed-6a4100b4182e", ".docx", "Modelo de
Contrato - PFPF", false, 2 },
                    { "fd7ed50d-8f42-4288-8877-3cb8095370e7", ".docx", "Modelo de
Contrato - PJPF", false, 4 }
                });

            migrationBuilder.CreateIndex(
                name: "IX_Contract_BuyerId",
                table: "Contract",
                column: "BuyerId");

            migrationBuilder.CreateIndex(
                name: "IX_Contract_OfferId",
                table: "Contract",
                column: "OfferId",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_Contract_SellerId",
                table: "Contract",
                column: "SellerId");

            migrationBuilder.CreateIndex(
                name: "IX_IdentityClaim_UserId",
                table: "IdentityClaim",
                column: "UserId");

            migrationBuilder.CreateIndex(
                name: "IX_IdentityRole_NormalizedName",
                table: "IdentityRole",
                column: "NormalizedName",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_IdentityRole_UserId",
```

```
            table: "IdentityRole",
            column: "UserId");

        migrationBuilder.CreateIndex(
            name: "IX_IdentityUser_NormalizedEmail",
            table: "IdentityUser",
            column: "NormalizedEmail",
            unique: true);

        migrationBuilder.CreateIndex(
            name: "IX_IdentityUser_NormalizedUserName",
            table: "IdentityUser",
            column: "NormalizedUserName",
            unique: true);

        migrationBuilder.CreateIndex(
            name: "IX_IdentityUserLogin_UserId",
            table: "IdentityUserLogin",
            column: "UserId");

        migrationBuilder.CreateIndex(
            name: "IX_Offer_BuyerId",
            table: "Offer",
            column: "BuyerId");

        migrationBuilder.CreateIndex(
            name: "IX_Offer_PropertyId",
            table: "Offer",
            column: "PropertyId");

        migrationBuilder.CreateIndex(
            name: "IX_Payment_OfferId",
            table: "Payment",
            column: "OfferId");

        migrationBuilder.CreateIndex(
            name: "IX_Property_CondominiumId",
            table: "Property",
            column: "CondominiumId");

        migrationBuilder.CreateIndex(
            name: "IX_Property_SellerId",
            table: "Property",
            column: "SellerId");

        migrationBuilder.CreateIndex(
            name: "IX_PropertyPhotos_PropertyId",
            table: "PropertyPhotos",
            column: "PropertyId");

        migrationBuilder.CreateIndex(
            name: "IX_Viewing_BuyerId",
            table: "Viewing",
            column: "BuyerId");

        migrationBuilder.CreateIndex(
            name: "IX_Viewing_PropertyId",
            table: "Viewing",
            column: "PropertyId");
    }

    /// <inheritdoc />
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Contract");

        migrationBuilder.DropTable(
            name: "ContractTemplate");

        migrationBuilder.DropTable(
            name: "IdentityClaim");

        migrationBuilder.DropTable(
            name: "IdentityRole");
```

```
            migrationBuilder.DropTable(
                name: "IdentityRoleClaim");

            migrationBuilder.DropTable(
                name: "IdentityUserLogin");

            migrationBuilder.DropTable(
                name: "IdentityUserRole");

            migrationBuilder.DropTable(
                name: "IdentityUserToken");

            migrationBuilder.DropTable(
                name: "Payment");

            migrationBuilder.DropTable(
                name: "PropertyPhotos");

            migrationBuilder.DropTable(
                name: "Viewing");

            migrationBuilder.DropTable(
                name: "IdentityUser");

            migrationBuilder.DropTable(
                name: "Offer");

            migrationBuilder.DropTable(
                name: "Property");

            migrationBuilder.DropTable(
                name: "Condominium");

            migrationBuilder.DropTable(
                name: "Customer");
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\20250208140349_v2.Designer.cs

```
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
using RealtyHub.ApiService.Data;

#nullable disable

namespace RealtyHub.ApiService.Migrations
{
    [DbContext(typeof(AppDbContext))]
    [Migration("20250208140349_v2")]
    partial class v2
    {
        /// <inheritdoc />
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "8.0.11")
                .HasAnnotation("Relational:MaxIdentifierLength", 63);

            NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Name")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<long?>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique();

                    b.HasIndex("UserId");

                    b.ToTable("IdentityRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
```

```
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.ToTable("IdentityRoleClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.Property<string>("LoginProvider")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderKey")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderDisplayName")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityUserLogin", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");
```

```
                    b.HasKey("UserId", "RoleId");

                    b.ToTable("IdentityUserRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<string>("LoginProvider")
                        .HasMaxLength(120)
                        .HasColumnType("character varying(120)");

                    b.Property<string>("Name")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("Value")
                        .HasColumnType("text");

                    b.HasKey("UserId", "LoginProvider", "Name");

                    b.ToTable("IdentityUserToken", (string)null);
                });

            modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<int>("AccessFailedCount")
                        .HasColumnType("integer");

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Creci")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<string>("Email")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<bool>("EmailConfirmed")
                        .HasColumnType("boolean");

                    b.Property<string>("GivenName")
                        .IsRequired()
                        .HasMaxLength(100)
                        .HasColumnType("character varying(100)");

                    b.Property<bool>("LockoutEnabled")
                        .HasColumnType("boolean");

                    b.Property<DateTimeOffset?>("LockoutEnd")
                        .HasColumnType("timestamp with time zone");

                    b.Property<string>("NormalizedEmail")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("NormalizedUserName")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("PasswordHash")
                        .HasColumnType("text");
```

```
                    b.Property<string>("PhoneNumber")
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<bool>("PhoneNumberConfirmed")
                        .HasColumnType("boolean");

                    b.Property<string>("SecurityStamp")
                        .HasColumnType("text");

                    b.Property<bool>("TwoFactorEnabled")
                        .HasColumnType("boolean");

                    b.Property<string>("UserName")
                        .HasMaxLength(160)
                        .HasColumnType("character varying(160)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedEmail")
                        .IsUnique();

                    b.HasIndex("NormalizedUserName")
                        .IsUnique();

                    b.ToTable("IdentityUser", (string)null);
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<decimal>("CondominiumValue")
                        .HasColumnType("numeric");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<int>("Floors")
                        .HasColumnType("integer");

                    b.Property<bool>("HasElevator")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasFitnessRoom")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasPartyRoom")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasPlayground")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasSwimmingPool")
                        .HasColumnType("boolean");

                    b.Property<bool>("IsActive")
                        .HasColumnType("boolean");

                    b.Property<string>("Name")
                        .IsRequired()
                        .HasMaxLength(120)
                        .HasColumnType("character varying(120)");

                    b.Property<int>("Units")
                        .HasColumnType("integer");

                    b.Property<DateTime>("UpdatedAt")
```

```
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.ToTable("Condominium", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<DateTime?>("EffectiveDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<string>("FileId")
                    .IsRequired()
                    .HasColumnType("text");

                b.Property<bool>("IsActive")
                    .HasColumnType("boolean");

                b.Property<DateTime?>("IssueDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<long>("OfferId")
                    .HasColumnType("bigint");

                b.Property<long>("SellerId")
                    .HasColumnType("bigint");

                b.Property<DateTime?>("SignatureDate")
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime?>("TermEndDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("OfferId")
                    .IsUnique();

                b.HasIndex("SellerId");
```

```
            b.ToTable("Contract", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.ContractTemplate", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("text");

            b.Property<string>("Extension")
                .IsRequired()
                .HasColumnType("text");

            b.Property<string>("Name")
                .IsRequired()
                .HasColumnType("text");

            b.Property<bool>("ShowInPage")
                .HasColumnType("boolean");

            b.Property<int>("Type")
                .HasColumnType("integer");

            b.HasKey("Id");

            b.ToTable("ContractTemplate", (string)null);

            b.HasData(
                new
                {
                    Id = "a2c16556-5098-4496-ae7a-1f9b6d0e8fcf",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PJPJ",
                    ShowInPage = false,
                    Type = 1
                },
                new
                {
                    Id = "f7581a63-f4f0-4881-b6ed-6a4100b4182e",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PFPF",
                    ShowInPage = false,
                    Type = 2
                },
                new
                {
                    Id = "2f4c556d-6850-4b3d-afe9-d7c2bd282718",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PFPJ",
                    ShowInPage = false,
                    Type = 3
                },
                new
                {
                    Id = "fd7ed50d-8f42-4288-8877-3cb8095370e7",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PJPF",
                    ShowInPage = false,
                    Type = 4
                },
                new
                {
                    Id = "2824aec3-3219-4d81-a97a-c3b80ca72844",
                    Extension = ".pdf",
                    Name = "Modelo Padrão",
                    ShowInPage = true,
                    Type = 0
                });
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint");
```

```
NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("BusinessName")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<int>("CustomerType")
                        .HasColumnType("integer");

                    b.Property<string>("DocumentNumber")
                        .IsRequired()
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<string>("Email")
                        .IsRequired()
                        .HasMaxLength(50)
                        .HasColumnType("character varying(50)");

                    b.Property<bool>("IsActive")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("boolean")
                        .HasDefaultValue(true);

                    b.Property<string>("IssuingAuthority")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<int>("MaritalStatus")
                        .HasColumnType("integer");

                    b.Property<string>("Name")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<string>("Nationality")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<string>("Occupation")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<int>("PersonType")
                        .HasColumnType("integer");

                    b.Property<string>("Phone")
                        .IsRequired()
                        .HasMaxLength(30)
                        .HasColumnType("character varying(30)");

                    b.Property<string>("Rg")
                        .IsRequired()
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<DateTime?>("RgIssueDate")
                        .HasColumnType("timestamp with time zone");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
```

```
                    .HasColumnType("text");

                b.HasKey("Id");

                b.ToTable("Customer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<int>("OfferStatus")
                    .HasColumnType("integer");

                b.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b.Property<DateTime?>("SubmissionDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("PropertyId");

                b.ToTable("Offer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<bool>("Installments")
                    .HasColumnType("boolean");

                b.Property<int>("InstallmentsCount")
```

```
                .HasColumnType("integer");

            b.Property<bool>("IsActive")
                .ValueGeneratedOnAdd()
                .HasColumnType("boolean")
                .HasDefaultValue(true);

            b.Property<long>("OfferId")
                .HasColumnType("bigint");

            b.Property<int>("PaymentType")
                .HasColumnType("integer");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("OfferId");

            b.ToTable("Payment", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
        {
            b.Property<long>("Id")
                .HasColumnType("bigint");

            b.Property<double>("Area")
                .HasColumnType("double precision");

            b.Property<int>("Bathroom")
                .HasColumnType("integer");

            b.Property<int>("Bedroom")
                .HasColumnType("integer");

            b.Property<long>("CondominiumId")
                .HasColumnType("bigint");

            b.Property<DateTime>("CreatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("Description")
                .IsRequired()
                .HasMaxLength(255)
                .HasColumnType("character varying(255)");

            b.Property<int>("Garage")
                .HasColumnType("integer");

            b.Property<bool>("IsActive")
                .HasColumnType("boolean");

            b.Property<bool>("IsNew")
                .HasColumnType("boolean");

            b.Property<decimal>("Price")
                .HasColumnType("numeric");

            b.Property<int>("PropertyType")
                .HasColumnType("integer");

            b.Property<string>("RegistryNumber")
                .IsRequired()
                .HasColumnType("text");

            b.Property<string>("RegistryRecord")
```

```
                .IsRequired()
                .HasColumnType("text");

            b.Property<long>("SellerId")
                .HasColumnType("bigint");

            b.Property<bool>("ShowInHome")
                .HasColumnType("boolean");

            b.Property<string>("Title")
                .IsRequired()
                .HasMaxLength(120)
                .HasColumnType("character varying(120)");

            b.Property<string>("TransactionsDetails")
                .IsRequired()
                .HasColumnType("text");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("CondominiumId");

            b.HasIndex("SellerId");

            b.ToTable("Property", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("text");

            b.Property<DateTime>("CreatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("Extension")
                .IsRequired()
                .HasColumnType("text");

            b.Property<bool>("IsActive")
                .HasColumnType("boolean");

            b.Property<bool>("IsThumbnail")
                .HasColumnType("boolean");

            b.Property<long>("PropertyId")
                .HasColumnType("bigint");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("PropertyId");

            b.ToTable("PropertyPhotos", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
```

```
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<long>("BuyerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<long>("PropertyId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<DateTime?>("ViewingDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<int>("ViewingStatus")
                        .HasColumnType("integer");

                    b.HasKey("Id");

                    b.HasIndex("BuyerId");

                    b.HasIndex("PropertyId");

                    b.ToTable("Viewing", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany("Roles")
                        .HasForeignKey("UserId");
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
```

```
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
                    {
                        b1.Property<long>("CondominiumId")
                            .HasColumnType("bigint");

                        b1.Property<string>("City")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("City");

                        b1.Property<string>("Complement")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Complement");

                        b1.Property<string>("Country")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Country");

                        b1.Property<string>("Neighborhood")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Neighborhood");

                        b1.Property<string>("Number")
                            .IsRequired()
                            .HasColumnType("text")
                            .HasColumnName("Number");

                        b1.Property<string>("State")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("State");

                        b1.Property<string>("Street")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Street");

                        b1.Property<string>("ZipCode")
                            .IsRequired()
                            .HasMaxLength(20)
                            .HasColumnType("character varying(20)")
                            .HasColumnName("ZipCode");

                        b1.HasKey("CondominiumId");

                        b1.ToTable("Condominium");

                        b1.WithOwner()
                            .HasForeignKey("CondominiumId");
                    });
```

```
            b.Navigation("Address")
                .IsRequired();
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
        {
            b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
                .WithMany()
                .HasForeignKey("BuyerId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
                .WithOne("Contract")
                .HasForeignKey("RealtyHub.Core.Models.Contract", "OfferId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
                .WithMany()
                .HasForeignKey("SellerId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.Navigation("Buyer");

            b.Navigation("Offer");

            b.Navigation("Seller");
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
        {
            b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
                {
                    b1.Property<long>("CustomerId")
                        .HasColumnType("bigint");

                    b1.Property<string>("City")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)")
                        .HasColumnName("City");

                    b1.Property<string>("Complement")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)")
                        .HasColumnName("Complement");

                    b1.Property<string>("Country")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)")
                        .HasColumnName("Country");

                    b1.Property<string>("Neighborhood")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)")
                        .HasColumnName("Neighborhood");

                    b1.Property<string>("Number")
                        .IsRequired()
                        .HasColumnType("text")
                        .HasColumnName("Number");

                    b1.Property<string>("State")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)")
                        .HasColumnName("State");

                    b1.Property<string>("Street")
                        .IsRequired()
```

```
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("CustomerId");

                b1.ToTable("Customer");

                b1.WithOwner()
                    .HasForeignKey("CustomerId");
            });

        b.Navigation("Address")
            .IsRequired();
    });

modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany()
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithMany("Payments")
            .HasForeignKey("OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Offer");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Condominium", "Condominium")
            .WithMany()
            .HasForeignKey("CondominiumId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Condominium", null)
            .WithMany("Properties")
            .HasForeignKey("Id")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany("Properties")
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("PropertyId")
                    .HasColumnType("bigint");
```

```
                b1.Property<string>("City")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("State");

                b1.Property<string>("Street")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("PropertyId");

                b1.ToTable("Property");

                b1.WithOwner()
                    .HasForeignKey("PropertyId");
            });

        b.Navigation("Address")
            .IsRequired();

        b.Navigation("Condominium");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany("PropertyPhotos")
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
```

```
                {
                    b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
                        .WithMany()
                        .HasForeignKey("BuyerId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();

                    b.HasOne("RealtyHub.Core.Models.Property", "Property")
                        .WithMany()
                        .HasForeignKey("PropertyId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();

                    b.Navigation("Buyer");

                    b.Navigation("Property");
                });

            modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
                {
                    b.Navigation("Roles");
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
                {
                    b.Navigation("Properties");
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
                {
                    b.Navigation("Properties");
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
                {
                    b.Navigation("Contract");

                    b.Navigation("Payments");
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
                {
                    b.Navigation("PropertyPhotos");
                });
#pragma warning restore 612, 618
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\20250208140349_v2.cs

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace RealtyHub.ApiService.Migrations
{
    /// <inheritdoc />
    public partial class v2 : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AlterColumn<DateTime>(
                name: "UpdatedAt",
                table: "Condominium",
                type: "timestamp with time zone",
                nullable: false,
                defaultValueSql: "NOW()",
                oldClrType: typeof(DateTime),
                oldType: "timestamp with time zone");

            migrationBuilder.AlterColumn<DateTime>(
                name: "CreatedAt",
                table: "Condominium",
                type: "timestamp with time zone",
                nullable: false,
                defaultValueSql: "NOW()",
                oldClrType: typeof(DateTime),
                oldType: "timestamp with time zone");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AlterColumn<DateTime>(
                name: "UpdatedAt",
                table: "Condominium",
                type: "timestamp with time zone",
                nullable: false,
                oldClrType: typeof(DateTime),
                oldType: "timestamp with time zone",
                oldDefaultValueSql: "NOW()");

            migrationBuilder.AlterColumn<DateTime>(
                name: "CreatedAt",
                table: "Condominium",
                type: "timestamp with time zone",
                nullable: false,
                oldClrType: typeof(DateTime),
                oldType: "timestamp with time zone",
                oldDefaultValueSql: "NOW()");
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\20250208142858_v3.Designer.cs

```
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
using RealtyHub.ApiService.Data;

#nullable disable

namespace RealtyHub.ApiService.Migrations
{
    [DbContext(typeof(AppDbContext))]
    [Migration("20250208142858_v3")]
    partial class v3
    {
        /// <inheritdoc />
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "8.0.11")
                .HasAnnotation("Relational:MaxIdentifierLength", 63);

            NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Name")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<long?>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique();

                    b.HasIndex("UserId");

                    b.ToTable("IdentityRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
```

```
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.ToTable("IdentityRoleClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.Property<string>("LoginProvider")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderKey")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderDisplayName")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityUserLogin", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");
```

```
                    b.HasKey("UserId", "RoleId");

                    b.ToTable("IdentityUserRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<string>("LoginProvider")
                        .HasMaxLength(120)
                        .HasColumnType("character varying(120)");

                    b.Property<string>("Name")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("Value")
                        .HasColumnType("text");

                    b.HasKey("UserId", "LoginProvider", "Name");

                    b.ToTable("IdentityUserToken", (string)null);
                });

            modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<int>("AccessFailedCount")
                        .HasColumnType("integer");

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Creci")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<string>("Email")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<bool>("EmailConfirmed")
                        .HasColumnType("boolean");

                    b.Property<string>("GivenName")
                        .IsRequired()
                        .HasMaxLength(100)
                        .HasColumnType("character varying(100)");

                    b.Property<bool>("LockoutEnabled")
                        .HasColumnType("boolean");

                    b.Property<DateTimeOffset?>("LockoutEnd")
                        .HasColumnType("timestamp with time zone");

                    b.Property<string>("NormalizedEmail")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("NormalizedUserName")
                        .HasMaxLength(180)
                        .HasColumnType("character varying(180)");

                    b.Property<string>("PasswordHash")
                        .HasColumnType("text");
```

```
                    b.Property<string>("PhoneNumber")
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<bool>("PhoneNumberConfirmed")
                        .HasColumnType("boolean");

                    b.Property<string>("SecurityStamp")
                        .HasColumnType("text");

                    b.Property<bool>("TwoFactorEnabled")
                        .HasColumnType("boolean");

                    b.Property<string>("UserName")
                        .HasMaxLength(160)
                        .HasColumnType("character varying(160)");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedEmail")
                        .IsUnique();

                    b.HasIndex("NormalizedUserName")
                        .IsUnique();

                    b.ToTable("IdentityUser", (string)null);
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<decimal>("CondominiumValue")
                        .HasColumnType("numeric");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<int>("Floors")
                        .HasColumnType("integer");

                    b.Property<bool>("HasElevator")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasFitnessRoom")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasPartyRoom")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasPlayground")
                        .HasColumnType("boolean");

                    b.Property<bool>("HasSwimmingPool")
                        .HasColumnType("boolean");

                    b.Property<bool>("IsActive")
                        .HasColumnType("boolean");

                    b.Property<string>("Name")
                        .IsRequired()
                        .HasMaxLength(120)
                        .HasColumnType("character varying(120)");

                    b.Property<int>("Units")
                        .HasColumnType("integer");

                    b.Property<DateTime>("UpdatedAt")
```

```
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.ToTable("Condominium", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<DateTime?>("EffectiveDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<string>("FileId")
                    .IsRequired()
                    .HasColumnType("text");

                b.Property<bool>("IsActive")
                    .HasColumnType("boolean");

                b.Property<DateTime?>("IssueDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<long>("OfferId")
                    .HasColumnType("bigint");

                b.Property<long>("SellerId")
                    .HasColumnType("bigint");

                b.Property<DateTime?>("SignatureDate")
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime?>("TermEndDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("OfferId")
                    .IsUnique();

                b.HasIndex("SellerId");
```

```
            b.ToTable("Contract", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.ContractTemplate", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("text");

            b.Property<string>("Extension")
                .IsRequired()
                .HasColumnType("text");

            b.Property<string>("Name")
                .IsRequired()
                .HasColumnType("text");

            b.Property<bool>("ShowInPage")
                .HasColumnType("boolean");

            b.Property<int>("Type")
                .HasColumnType("integer");

            b.HasKey("Id");

            b.ToTable("ContractTemplate", (string)null);

            b.HasData(
                new
                {
                    Id = "a2c16556-5098-4496-ae7a-1f9b6d0e8fcf",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PJPJ",
                    ShowInPage = false,
                    Type = 1
                },
                new
                {
                    Id = "f7581a63-f4f0-4881-b6ed-6a4100b4182e",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PFPF",
                    ShowInPage = false,
                    Type = 2
                },
                new
                {
                    Id = "2f4c556d-6850-4b3d-afe9-d7c2bd282718",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PFPJ",
                    ShowInPage = false,
                    Type = 3
                },
                new
                {
                    Id = "fd7ed50d-8f42-4288-8877-3cb8095370e7",
                    Extension = ".docx",
                    Name = "Modelo de Contrato - PJPF",
                    ShowInPage = false,
                    Type = 4
                },
                new
                {
                    Id = "2824aec3-3219-4d81-a97a-c3b80ca72844",
                    Extension = ".pdf",
                    Name = "Modelo Padrão",
                    ShowInPage = true,
                    Type = 0
                });
        });

    modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
        {
            b.Property<long>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("bigint");
```

```
NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("BusinessName")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<int>("CustomerType")
                        .HasColumnType("integer");

                    b.Property<string>("DocumentNumber")
                        .IsRequired()
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<string>("Email")
                        .IsRequired()
                        .HasMaxLength(50)
                        .HasColumnType("character varying(50)");

                    b.Property<bool>("IsActive")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("boolean")
                        .HasDefaultValue(true);

                    b.Property<string>("IssuingAuthority")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<int>("MaritalStatus")
                        .HasColumnType("integer");

                    b.Property<string>("Name")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<string>("Nationality")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<string>("Occupation")
                        .IsRequired()
                        .HasMaxLength(80)
                        .HasColumnType("character varying(80)");

                    b.Property<int>("PersonType")
                        .HasColumnType("integer");

                    b.Property<string>("Phone")
                        .IsRequired()
                        .HasMaxLength(30)
                        .HasColumnType("character varying(30)");

                    b.Property<string>("Rg")
                        .IsRequired()
                        .HasMaxLength(20)
                        .HasColumnType("character varying(20)");

                    b.Property<DateTime?>("RgIssueDate")
                        .HasColumnType("timestamp with time zone");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
```

```
                    .HasColumnType("text");

                b.HasKey("Id");

                b.ToTable("Customer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<int>("OfferStatus")
                    .HasColumnType("integer");

                b.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b.Property<DateTime?>("SubmissionDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("PropertyId");

                b.ToTable("Offer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<bool>("Installments")
                    .HasColumnType("boolean");

                b.Property<int>("InstallmentsCount")
```

```
                    .HasColumnType("integer");

                b.Property<bool>("IsActive")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("boolean")
                    .HasDefaultValue(true);

                b.Property<long>("OfferId")
                    .HasColumnType("bigint");

                b.Property<int>("PaymentType")
                    .HasColumnType("integer");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("OfferId");

                b.ToTable("Payment", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<double>("Area")
                    .HasColumnType("double precision");

                b.Property<int>("Bathroom")
                    .HasColumnType("integer");

                b.Property<int>("Bedroom")
                    .HasColumnType("integer");

                b.Property<long>("CondominiumId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("Description")
                    .IsRequired()
                    .HasMaxLength(255)
                    .HasColumnType("character varying(255)");

                b.Property<int>("Garage")
                    .HasColumnType("integer");

                b.Property<bool>("IsActive")
                    .HasColumnType("boolean");

                b.Property<bool>("IsNew")
                    .HasColumnType("boolean");

                b.Property<decimal>("Price")
                    .HasColumnType("numeric");

                b.Property<int>("PropertyType")
                    .HasColumnType("integer");

                b.Property<string>("RegistryNumber")
```

```
            .IsRequired()
            .HasColumnType("text");

        b.Property<string>("RegistryRecord")
            .IsRequired()
            .HasColumnType("text");

        b.Property<long>("SellerId")
            .HasColumnType("bigint");

        b.Property<bool>("ShowInHome")
            .HasColumnType("boolean");

        b.Property<string>("Title")
            .IsRequired()
            .HasMaxLength(120)
            .HasColumnType("character varying(120)");

        b.Property<string>("TransactionsDetails")
            .IsRequired()
            .HasColumnType("text");

        b.Property<DateTime>("UpdatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");

        b.HasIndex("CondominiumId");

        b.HasIndex("SellerId");

        b.ToTable("Property", (string)null);
    });

modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
    {
        b.Property<string>("Id")
            .HasColumnType("text");

        b.Property<DateTime>("CreatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<string>("Extension")
            .IsRequired()
            .HasColumnType("text");

        b.Property<bool>("IsActive")
            .HasColumnType("boolean");

        b.Property<bool>("IsThumbnail")
            .HasColumnType("boolean");

        b.Property<long>("PropertyId")
            .HasColumnType("bigint");

        b.Property<DateTime>("UpdatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");

        b.HasIndex("PropertyId");
```

```
                b.ToTable("PropertyPhotos", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.Property<DateTime?>("ViewingDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<int>("ViewingStatus")
                    .HasColumnType("integer");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("PropertyId");

                b.ToTable("Viewing", (string)null);
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany("Roles")
                    .HasForeignKey("UserId");
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
```

```
=>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
                    {
                        b1.Property<long>("CondominiumId")
                            .HasColumnType("bigint");

                        b1.Property<string>("City")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("City");

                        b1.Property<string>("Complement")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Complement");

                        b1.Property<string>("Country")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Country");

                        b1.Property<string>("Neighborhood")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Neighborhood");

                        b1.Property<string>("Number")
                            .IsRequired()
                            .HasColumnType("text")
                            .HasColumnName("Number");

                        b1.Property<string>("State")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("State");

                        b1.Property<string>("Street")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Street");

                        b1.Property<string>("ZipCode")
                            .IsRequired()
                            .HasMaxLength(20)
                            .HasColumnType("character varying(20)")
                            .HasColumnName("ZipCode");

                        b1.HasKey("CondominiumId");

                        b1.ToTable("Condominium");
```

```
                b1.WithOwner()
                    .HasForeignKey("CondominiumId");
            });

        b.Navigation("Address")
            .IsRequired();
    });

modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithOne("Contract")
            .HasForeignKey("RealtyHub.Core.Models.Contract", "OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany()
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Offer");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
    {
        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("CustomerId")
                    .HasColumnType("bigint");

                b1.Property<string>("City")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
```

```
                    .HasColumnName("State");

                b1.Property<string>("Street")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("CustomerId");

                b1.ToTable("Customer");

                b1.WithOwner()
                    .HasForeignKey("CustomerId");
            });

        b.Navigation("Address")
            .IsRequired();
    });

modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany()
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithMany("Payments")
            .HasForeignKey("OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Offer");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Condominium", "Condominium")
            .WithMany("Properties")
            .HasForeignKey("CondominiumId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany("Properties")
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b1.Property<string>("City")
```

```
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("State");

                b1.Property<string>("Street")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("PropertyId");

                b1.ToTable("Property");

                b1.WithOwner()
                    .HasForeignKey("PropertyId");
            });

        b.Navigation("Address")
            .IsRequired();

        b.Navigation("Condominium");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany("PropertyPhotos")
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
```

```
                    .WithMany()
                    .HasForeignKey("BuyerId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();

                b.HasOne("RealtyHub.Core.Models.Property", "Property")
                    .WithMany()
                    .HasForeignKey("PropertyId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();

                b.Navigation("Buyer");

                b.Navigation("Property");
            });

        modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
            {
                b.Navigation("Roles");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.Navigation("Properties");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
            {
                b.Navigation("Properties");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Navigation("Contract");

                b.Navigation("Payments");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
            {
                b.Navigation("PropertyPhotos");
            });
#pragma warning restore 612, 618
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\20250208142858_v3.cs

```
using Microsoft.EntityFrameworkCore.Migrations;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace RealtyHub.ApiService.Migrations
{
    /// <inheritdoc />
    public partial class v3 : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropForeignKey(
                name: "FK_Property_Condominium_Id",
                table: "Property");

            migrationBuilder.AlterColumn<long>(
                name: "Id",
                table: "Property",
                type: "bigint",
                nullable: false,
                oldClrType: typeof(long),
                oldType: "bigint")
                .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn);
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AlterColumn<long>(
                name: "Id",
                table: "Property",
                type: "bigint",
                nullable: false,
                oldClrType: typeof(long),
                oldType: "bigint")
                .OldAnnotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn);

            migrationBuilder.AddForeignKey(
                name: "FK_Property_Condominium_Id",
                table: "Property",
                column: "Id",
                principalTable: "Condominium",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        }
    }
}
```

.\RealtyHub.ApiService\Migrations\AppDbContextModelSnapshot.cs

```
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
using RealtyHub.ApiService.Data;

#nullable disable

namespace RealtyHub.ApiService.Migrations
{
    [DbContext(typeof(AppDbContext))]
    partial class AppDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "8.0.11")
                .HasAnnotation("Relational:MaxIdentifierLength", 63);

            NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("ConcurrencyStamp")
                        .IsConcurrencyToken()
                        .HasColumnType("text");

                    b.Property<string>("Name")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<string>("NormalizedName")
                        .HasMaxLength(256)
                        .HasColumnType("character varying(256)");

                    b.Property<long?>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("NormalizedName")
                        .IsUnique();

                    b.HasIndex("UserId");

                    b.ToTable("IdentityRole", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");
```

```
                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.ToTable("IdentityRoleClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                    b.Property<string>("ClaimType")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<string>("ClaimValue")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("Id");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityClaim", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.Property<string>("LoginProvider")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderKey")
                        .HasMaxLength(128)
                        .HasColumnType("character varying(128)");

                    b.Property<string>("ProviderDisplayName")
                        .HasMaxLength(255)
                        .HasColumnType("character varying(255)");

                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.HasKey("LoginProvider", "ProviderKey");

                    b.HasIndex("UserId");

                    b.ToTable("IdentityUserLogin", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.Property<long>("UserId")
                        .HasColumnType("bigint");

                    b.Property<long>("RoleId")
                        .HasColumnType("bigint");

                    b.HasKey("UserId", "RoleId");
```

```
                b.ToTable("IdentityUserRole", (string)null);
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
            {
                b.Property<long>("UserId")
                    .HasColumnType("bigint");

                b.Property<string>("LoginProvider")
                    .HasMaxLength(120)
                    .HasColumnType("character varying(120)");

                b.Property<string>("Name")
                    .HasMaxLength(180)
                    .HasColumnType("character varying(180)");

                b.Property<string>("Value")
                    .HasColumnType("text");

                b.HasKey("UserId", "LoginProvider", "Name");

                b.ToTable("IdentityUserToken", (string)null);
            });

        modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<int>("AccessFailedCount")
                    .HasColumnType("integer");

                b.Property<string>("ConcurrencyStamp")
                    .IsConcurrencyToken()
                    .HasColumnType("text");

                b.Property<string>("Creci")
                    .IsRequired()
                    .HasColumnType("text");

                b.Property<string>("Email")
                    .HasMaxLength(180)
                    .HasColumnType("character varying(180)");

                b.Property<bool>("EmailConfirmed")
                    .HasColumnType("boolean");

                b.Property<string>("GivenName")
                    .IsRequired()
                    .HasMaxLength(100)
                    .HasColumnType("character varying(100)");

                b.Property<bool>("LockoutEnabled")
                    .HasColumnType("boolean");

                b.Property<DateTimeOffset?>("LockoutEnd")
                    .HasColumnType("timestamp with time zone");

                b.Property<string>("NormalizedEmail")
                    .HasMaxLength(180)
                    .HasColumnType("character varying(180)");

                b.Property<string>("NormalizedUserName")
                    .HasMaxLength(180)
                    .HasColumnType("character varying(180)");

                b.Property<string>("PasswordHash")
                    .HasColumnType("text");

                b.Property<string>("PhoneNumber")
                    .HasMaxLength(20)
```

```
                    .HasColumnType("character varying(20)");

                b.Property<bool>("PhoneNumberConfirmed")
                    .HasColumnType("boolean");

                b.Property<string>("SecurityStamp")
                    .HasColumnType("text");

                b.Property<bool>("TwoFactorEnabled")
                    .HasColumnType("boolean");

                b.Property<string>("UserName")
                    .HasMaxLength(160)
                    .HasColumnType("character varying(160)");

                b.HasKey("Id");

                b.HasIndex("NormalizedEmail")
                    .IsUnique();

                b.HasIndex("NormalizedUserName")
                    .IsUnique();

                b.ToTable("IdentityUser", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("CondominiumValue")
                    .HasColumnType("numeric");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<int>("Floors")
                    .HasColumnType("integer");

                b.Property<bool>("HasElevator")
                    .HasColumnType("boolean");

                b.Property<bool>("HasFitnessRoom")
                    .HasColumnType("boolean");

                b.Property<bool>("HasPartyRoom")
                    .HasColumnType("boolean");

                b.Property<bool>("HasPlayground")
                    .HasColumnType("boolean");

                b.Property<bool>("HasSwimmingPool")
                    .HasColumnType("boolean");

                b.Property<bool>("IsActive")
                    .HasColumnType("boolean");

                b.Property<string>("Name")
                    .IsRequired()
                    .HasMaxLength(120)
                    .HasColumnType("character varying(120)");

                b.Property<int>("Units")
                    .HasColumnType("integer");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");
```

```
                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.HasKey("Id");

                    b.ToTable("Condominium", (string)null);
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<long>("BuyerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<DateTime?>("EffectiveDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<string>("FileId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<bool>("IsActive")
                        .HasColumnType("boolean");

                    b.Property<DateTime?>("IssueDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<long>("OfferId")
                        .HasColumnType("bigint");

                    b.Property<long>("SellerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime?>("SignatureDate")
                        .HasColumnType("timestamp with time zone");

                    b.Property<DateTime?>("TermEndDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.HasKey("Id");

                    b.HasIndex("BuyerId");

                    b.HasIndex("OfferId")
                        .IsUnique();

                    b.HasIndex("SellerId");

                    b.ToTable("Contract", (string)null);
                });
```

```
            modelBuilder.Entity("RealtyHub.Core.Models.ContractTemplate", b =>
                {
                    b.Property<string>("Id")
                        .HasColumnType("text");

                    b.Property<string>("Extension")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<bool>("ShowInPage")
                        .HasColumnType("boolean");

                    b.Property<int>("Type")
                        .HasColumnType("integer");

                    b.HasKey("Id");

                    b.ToTable("ContractTemplate", (string)null);

                    b.HasData(
                        new
                        {
                            Id = "a2c16556-5098-4496-ae7a-1f9b6d0e8fcf",
                            Extension = ".docx",
                            Name = "Modelo de Contrato - PJPJ",
                            ShowInPage = false,
                            Type = 1
                        },
                        new
                        {
                            Id = "f7581a63-f4f0-4881-b6ed-6a4100b4182e",
                            Extension = ".docx",
                            Name = "Modelo de Contrato - PFPF",
                            ShowInPage = false,
                            Type = 2
                        },
                        new
                        {
                            Id = "2f4c556d-6850-4b3d-afe9-d7c2bd282718",
                            Extension = ".docx",
                            Name = "Modelo de Contrato - PFPJ",
                            ShowInPage = false,
                            Type = 3
                        },
                        new
                        {
                            Id = "fd7ed50d-8f42-4288-8877-3cb8095370e7",
                            Extension = ".docx",
                            Name = "Modelo de Contrato - PJPF",
                            ShowInPage = false,
                            Type = 4
                        },
                        new
                        {
                            Id = "2824aec3-3219-4d81-a97a-c3b80ca72844",
                            Extension = ".pdf",
                            Name = "Modelo Padrão",
                            ShowInPage = true,
                            Type = 0
                        });
                });

            modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<string>("BusinessName")
```

```
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<DateTime>("CreatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<int>("CustomerType")
            .HasColumnType("integer");

        b.Property<string>("DocumentNumber")
            .IsRequired()
            .HasMaxLength(20)
            .HasColumnType("character varying(20)");

        b.Property<string>("Email")
            .IsRequired()
            .HasMaxLength(50)
            .HasColumnType("character varying(50)");

        b.Property<bool>("IsActive")
            .ValueGeneratedOnAdd()
            .HasColumnType("boolean")
            .HasDefaultValue(true);

        b.Property<string>("IssuingAuthority")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<int>("MaritalStatus")
            .HasColumnType("integer");

        b.Property<string>("Name")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<string>("Nationality")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<string>("Occupation")
            .IsRequired()
            .HasMaxLength(80)
            .HasColumnType("character varying(80)");

        b.Property<int>("PersonType")
            .HasColumnType("integer");

        b.Property<string>("Phone")
            .IsRequired()
            .HasMaxLength(30)
            .HasColumnType("character varying(30)");

        b.Property<string>("Rg")
            .IsRequired()
            .HasMaxLength(20)
            .HasColumnType("character varying(20)");

        b.Property<DateTime?>("RgIssueDate")
            .HasColumnType("timestamp with time zone");

        b.Property<DateTime>("UpdatedAt")
            .ValueGeneratedOnAdd()
            .HasColumnType("timestamp with time zone")
            .HasDefaultValueSql("NOW()");

        b.Property<string>("UserId")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");
```

```
                b.ToTable("Customer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<long>("BuyerId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<int>("OfferStatus")
                    .HasColumnType("integer");

                b.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b.Property<DateTime?>("SubmissionDate")
                    .IsRequired()
                    .HasColumnType("timestamp with time zone");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("BuyerId");

                b.HasIndex("PropertyId");

                b.ToTable("Offer", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<decimal>("Amount")
                    .HasColumnType("numeric");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<bool>("Installments")
                    .HasColumnType("boolean");

                b.Property<int>("InstallmentsCount")
                    .HasColumnType("integer");

                b.Property<bool>("IsActive")
```

```
                    .ValueGeneratedOnAdd()
                    .HasColumnType("boolean")
                    .HasDefaultValue(true);

                b.Property<long>("OfferId")
                    .HasColumnType("bigint");

                b.Property<int>("PaymentType")
                    .HasColumnType("integer");

                b.Property<DateTime>("UpdatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("text");

                b.HasKey("Id");

                b.HasIndex("OfferId");

                b.ToTable("Payment", (string)null);
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
            {
                b.Property<long>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                b.Property<double>("Area")
                    .HasColumnType("double precision");

                b.Property<int>("Bathroom")
                    .HasColumnType("integer");

                b.Property<int>("Bedroom")
                    .HasColumnType("integer");

                b.Property<long>("CondominiumId")
                    .HasColumnType("bigint");

                b.Property<DateTime>("CreatedAt")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("timestamp with time zone")
                    .HasDefaultValueSql("NOW()");

                b.Property<string>("Description")
                    .IsRequired()
                    .HasMaxLength(255)
                    .HasColumnType("character varying(255)");

                b.Property<int>("Garage")
                    .HasColumnType("integer");

                b.Property<bool>("IsActive")
                    .HasColumnType("boolean");

                b.Property<bool>("IsNew")
                    .HasColumnType("boolean");

                b.Property<decimal>("Price")
                    .HasColumnType("numeric");

                b.Property<int>("PropertyType")
                    .HasColumnType("integer");

                b.Property<string>("RegistryNumber")
                    .IsRequired()
                    .HasColumnType("text");
```

```
            b.Property<string>("RegistryRecord")
                .IsRequired()
                .HasColumnType("text");

            b.Property<long>("SellerId")
                .HasColumnType("bigint");

            b.Property<bool>("ShowInHome")
                .HasColumnType("boolean");

            b.Property<string>("Title")
                .IsRequired()
                .HasMaxLength(120)
                .HasColumnType("character varying(120)");

            b.Property<string>("TransactionsDetails")
                .IsRequired()
                .HasColumnType("text");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("CondominiumId");

            b.HasIndex("SellerId");

            b.ToTable("Property", (string)null);
        });

    modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
        {
            b.Property<string>("Id")
                .HasColumnType("text");

            b.Property<DateTime>("CreatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("Extension")
                .IsRequired()
                .HasColumnType("text");

            b.Property<bool>("IsActive")
                .HasColumnType("boolean");

            b.Property<bool>("IsThumbnail")
                .HasColumnType("boolean");

            b.Property<long>("PropertyId")
                .HasColumnType("bigint");

            b.Property<DateTime>("UpdatedAt")
                .ValueGeneratedOnAdd()
                .HasColumnType("timestamp with time zone")
                .HasDefaultValueSql("NOW()");

            b.Property<string>("UserId")
                .IsRequired()
                .HasColumnType("text");

            b.HasKey("Id");

            b.HasIndex("PropertyId");

            b.ToTable("PropertyPhotos", (string)null);
        });
```

```
            modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
                {
                    b.Property<long>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("bigint");


NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

                    b.Property<long>("BuyerId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("CreatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<long>("PropertyId")
                        .HasColumnType("bigint");

                    b.Property<DateTime>("UpdatedAt")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("timestamp with time zone")
                        .HasDefaultValueSql("NOW()");

                    b.Property<string>("UserId")
                        .IsRequired()
                        .HasColumnType("text");

                    b.Property<DateTime?>("ViewingDate")
                        .IsRequired()
                        .HasColumnType("timestamp with time zone");

                    b.Property<int>("ViewingStatus")
                        .HasColumnType("integer");

                    b.HasKey("Id");

                    b.HasIndex("BuyerId");

                    b.HasIndex("PropertyId");

                    b.ToTable("Viewing", (string)null);
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<long>", b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany("Roles")
                        .HasForeignKey("UserId");
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<long>",
b =>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
                        .WithMany()
                        .HasForeignKey("UserId")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                });

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<long>", b
=>
                {
                    b.HasOne("RealtyHub.ApiService.Models.User", null)
```

```
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<long>",
b =>
            {
                b.HasOne("RealtyHub.ApiService.Models.User", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
                    {
                        b1.Property<long>("CondominiumId")
                            .HasColumnType("bigint");

                        b1.Property<string>("City")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("City");

                        b1.Property<string>("Complement")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Complement");

                        b1.Property<string>("Country")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Country");

                        b1.Property<string>("Neighborhood")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Neighborhood");

                        b1.Property<string>("Number")
                            .IsRequired()
                            .HasColumnType("text")
                            .HasColumnName("Number");

                        b1.Property<string>("State")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("State");

                        b1.Property<string>("Street")
                            .IsRequired()
                            .HasMaxLength(80)
                            .HasColumnType("character varying(80)")
                            .HasColumnName("Street");

                        b1.Property<string>("ZipCode")
                            .IsRequired()
                            .HasMaxLength(20)
                            .HasColumnType("character varying(20)")
                            .HasColumnName("ZipCode");

                        b1.HasKey("CondominiumId");

                        b1.ToTable("Condominium");

                        b1.WithOwner()
                            .HasForeignKey("CondominiumId");
```

```
            });

        b.Navigation("Address")
            .IsRequired();
    });

modelBuilder.Entity("RealtyHub.Core.Models.Contract", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithOne("Contract")
            .HasForeignKey("RealtyHub.Core.Models.Contract", "OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany()
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Offer");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
    {
        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("CustomerId")
                    .HasColumnType("bigint");

                b1.Property<string>("City")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("State");

                b1.Property<string>("Street")
```

```
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("CustomerId");

                b1.ToTable("Customer");

                b1.WithOwner()
                    .HasForeignKey("CustomerId");
            });

        b.Navigation("Address")
            .IsRequired();
    });

modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany()
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Buyer");

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Payment", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Offer", "Offer")
            .WithMany("Payments")
            .HasForeignKey("OfferId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Offer");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Condominium", "Condominium")
            .WithMany("Properties")
            .HasForeignKey("CondominiumId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.HasOne("RealtyHub.Core.Models.Customer", "Seller")
            .WithMany("Properties")
            .HasForeignKey("SellerId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.OwnsOne("RealtyHub.Core.Models.Address", "Address", b1 =>
            {
                b1.Property<long>("PropertyId")
                    .HasColumnType("bigint");

                b1.Property<string>("City")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
```

```
                    .HasColumnName("City");

                b1.Property<string>("Complement")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Complement");

                b1.Property<string>("Country")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Country");

                b1.Property<string>("Neighborhood")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Neighborhood");

                b1.Property<string>("Number")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("Number");

                b1.Property<string>("State")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("State");

                b1.Property<string>("Street")
                    .IsRequired()
                    .HasMaxLength(80)
                    .HasColumnType("character varying(80)")
                    .HasColumnName("Street");

                b1.Property<string>("ZipCode")
                    .IsRequired()
                    .HasMaxLength(20)
                    .HasColumnType("character varying(20)")
                    .HasColumnName("ZipCode");

                b1.HasKey("PropertyId");

                b1.ToTable("Property");

                b1.WithOwner()
                    .HasForeignKey("PropertyId");
            });

        b.Navigation("Address")
            .IsRequired();

        b.Navigation("Condominium");

        b.Navigation("Seller");
    });

modelBuilder.Entity("RealtyHub.Core.Models.PropertyPhoto", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Property", "Property")
            .WithMany("PropertyPhotos")
            .HasForeignKey("PropertyId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("Property");
    });

modelBuilder.Entity("RealtyHub.Core.Models.Viewing", b =>
    {
        b.HasOne("RealtyHub.Core.Models.Customer", "Buyer")
            .WithMany()
            .HasForeignKey("BuyerId")
            .OnDelete(DeleteBehavior.Cascade)
```

```
                    .IsRequired();

                b.HasOne("RealtyHub.Core.Models.Property", "Property")
                    .WithMany()
                    .HasForeignKey("PropertyId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();

                b.Navigation("Buyer");

                b.Navigation("Property");
            });

        modelBuilder.Entity("RealtyHub.ApiService.Models.User", b =>
            {
                b.Navigation("Roles");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Condominium", b =>
            {
                b.Navigation("Properties");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Customer", b =>
            {
                b.Navigation("Properties");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Offer", b =>
            {
                b.Navigation("Contract");

                b.Navigation("Payments");
            });

        modelBuilder.Entity("RealtyHub.Core.Models.Property", b =>
            {
                b.Navigation("PropertyPhotos");
            });
#pragma warning restore 612, 618
        }
    }
}
```

.\RealtyHub.ApiService\Models\User.cs

```
using Microsoft.AspNetCore.Identity;

namespace RealtyHub.ApiService.Models;

/// <summary>
/// Representa um usuário do sistema.
/// </summary>
/// <remarks>
/// Esta classe herda de IdentityUser e adiciona propriedades específicas do domínio.
/// </remarks>
public class User : IdentityUser<long>
{
    /// <summary>
    /// Nome completo do usuário.
    /// </summary>
    /// <value>Nome completo do usuário.</value>
    public string GivenName { get; set; } = string.Empty;

    /// <summary>
    /// Identificador do Conselho Regional de Corretores de Imóveis (CRECI).
    /// </summary>
    /// <value>Identificador do Conselho Regional de Corretores de Imóveis.</value>
    public string Creci { get; set; } = string.Empty;

    /// <summary>
    /// Cargos do usuário no sistema.
    /// </summary>
    /// <value>Cargos do usuário no sistema.</value>
    public List<IdentityRole<long>>? Roles { get; set; }
}
```

.\RealtyHub.ApiService\Program.cs

```
using RealtyHub.ApiService;
using RealtyHub.ApiService.Common.Api;
using RealtyHub.ApiService.Endpoints;
using System.Globalization;
using QuestPDF.Infrastructure;

var builder = WebApplication.CreateBuilder(args);

CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("pt-BR");
CultureInfo.DefaultThreadCurrentUICulture = new CultureInfo("pt-BR");

QuestPDF.Settings.License = LicenseType.Community;

builder.AddDirectories();
builder.Logging.AddConsole();
builder.AddConfiguration();
builder.AddSecurity();
builder.AddDataContexts();
builder.AddCrossOrigin();
builder.AddDocumentation();
builder.AddServices();

builder.Services.Configure<Microsoft.AspNetCore.Http.Json.JsonOptions>(options =>
{
    options.SerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
});

var app = builder.Build();

app.ApplyMigrations();

app.UseSwagger();
app.UseSwaggerUI();
app.MapSwagger().RequireAuthorization();

if (app.Environment.IsDevelopment())
    app.ConfigureDevEnvironment();

app.UseStaticFiles();
app.UseExceptionHandler();
app.UseCors(Configuration.CorsPolicyName);
app.UseSecurity();
app.MapEndpoints();

app.Run();

namespace RealtyHub.ApiService
{
    public partial class Program;
}
```

.\RealtyHub.ApiService\Properties\ServiceDependencies\realtyhub-api - Web Deploy\profile.arm.jso

```
{
  "$schema":
"https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.js
on#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_dependencyType": "compute.appService.windows"
  },
  "parameters": {
    "resourceGroupName": {
      "type": "string",
      "defaultValue": "realtyhub",
      "metadata": {
        "description": "Name of the resource group for the resource. It is recommended to
put resources under same resource group for better tracking."
      }
    },
    "resourceGroupLocation": {
      "type": "string",
      "defaultValue": "brazilsouth",
      "metadata": {
        "description": "Location of the resource group. Resource groups could have
different location than resources, however by default we use API versions from latest
hybrid profile which support all locations for resource types we support."
      }
    },
    "resourceName": {
      "type": "string",
      "defaultValue": "realtyhub-api",
      "metadata": {
        "description": "Name of the main resource to be created by this template."
      }
    },
    "resourceLocation": {
      "type": "string",
      "defaultValue": "[parameters('resourceGroupLocation')]",
      "metadata": {
        "description": "Location of the resource. By default use resource group's
location, unless the resource provider is not supported there."
      }
    }
  },
  "variables": {
    "appServicePlan_name": "[concat('Plan',
uniqueString(concat(parameters('resourceName'), subscription().subscriptionId)))]",
    "appServicePlan_ResourceId": "[concat('/subscriptions/',
subscription().subscriptionId, '/resourceGroups/', parameters('resourceGroupName'),
'/providers/Microsoft.Web/serverFarms/', variables('appServicePlan_name'))]"
  },
  "resources": [
    {
      "type": "Microsoft.Resources/resourceGroups",
      "name": "[parameters('resourceGroupName')]",
      "location": "[parameters('resourceGroupLocation')]",
      "apiVersion": "2019-10-01"
    },
    {
      "type": "Microsoft.Resources/deployments",
      "name": "[concat(parameters('resourceGroupName'), 'Deployment',
uniqueString(concat(parameters('resourceName'), subscription().subscriptionId)))]",
      "resourceGroup": "[parameters('resourceGroupName')]",
      "apiVersion": "2019-10-01",
      "dependsOn": [
        "[parameters('resourceGroupName')]"
      ],
      "properties": {
        "mode": "Incremental",
        "template": {
          "$schema":
"http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
          "contentVersion": "1.0.0.0",
          "resources": [
```

```
            {
              "location": "[parameters('resourceLocation')]",
              "name": "[parameters('resourceName')]",
              "type": "Microsoft.Web/sites",
              "apiVersion": "2015-08-01",
              "tags": {
                "[concat('hidden-related:', variables('appServicePlan_ResourceId'))]":
"empty"
              },
              "dependsOn": [
                "[variables('appServicePlan_ResourceId')]"
              ],
              "kind": "app",
              "properties": {
                "name": "[parameters('resourceName')]",
                "kind": "app",
                "httpsOnly": true,
                "reserved": false,
                "serverFarmId": "[variables('appServicePlan_ResourceId')]",
                "siteConfig": {
                  "metadata": [
                    {
                      "name": "CURRENT_STACK",
                      "value": "dotnetcore"
                    }
                  ]
                }
              },
              "identity": {
                "type": "SystemAssigned"
              }
            },
            {
              "location": "[parameters('resourceLocation')]",
              "name": "[variables('appServicePlan_name')]",
              "type": "Microsoft.Web/serverFarms",
              "apiVersion": "2015-08-01",
              "sku": {
                "name": "S1",
                "tier": "Standard",
                "family": "S",
                "size": "S1"
              },
              "properties": {
                "name": "[variables('appServicePlan_name')]"
              }
            }
          ]
        }
      }
    }
  ]
}
```

.\RealtyHub.ApiService\Properties\ServiceDependencies\realtyhub-api - Zip Deploy\profile.arm.json


```
{
  "$schema":
"https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.js
on#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_dependencyType": "compute.function.windows.appService"
  },
  "parameters": {
    "resourceGroupName": {
      "type": "string",
      "defaultValue": "realtyhub",
      "metadata": {
        "description": "Name of the resource group for the resource. It is recommended to
put resources under same resource group for better tracking."
      }
    },
    "resourceGroupLocation": {
      "type": "string",
      "defaultValue": "brazilsouth",
      "metadata": {
        "description": "Location of the resource group. Resource groups could have
different location than resources, however by default we use API versions from latest
hybrid profile which support all locations for resource types we support."
      }
    },
    "resourceName": {
      "type": "string",
      "defaultValue": "realtyhub-api",
      "metadata": {
        "description": "Name of the main resource to be created by this template."
      }
    },
    "resourceLocation": {
      "type": "string",
      "defaultValue": "[parameters('resourceGroupLocation')]",
      "metadata": {
        "description": "Location of the resource. By default use resource group's
location, unless the resource provider is not supported there."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Resources/resourceGroups",
      "name": "[parameters('resourceGroupName')]",
      "location": "[parameters('resourceGroupLocation')]",
      "apiVersion": "2019-10-01"
    },
    {
      "type": "Microsoft.Resources/deployments",
      "name": "[concat(parameters('resourceGroupName'), 'Deployment',
uniqueString(concat(parameters('resourceName'), subscription().subscriptionId)))]",
      "resourceGroup": "[parameters('resourceGroupName')]",
      "apiVersion": "2019-10-01",
      "dependsOn": [
        "[parameters('resourceGroupName')]"
      ],
      "properties": {
        "mode": "Incremental",
        "expressionEvaluationOptions": {
          "scope": "inner"
        },
        "parameters": {
          "resourceGroupName": {
            "value": "[parameters('resourceGroupName')]"
          },
          "resourceGroupLocation": {
            "value": "[parameters('resourceGroupLocation')]"
          },
          "resourceName": {
            "value": "[parameters('resourceName')]"
```

```
          },
          "resourceLocation": {
            "value": "[parameters('resourceLocation')]"
          }
        },
        "template": {
          "$schema":
"http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
          "contentVersion": "1.0.0.0",
          "parameters": {
            "resourceGroupName": {
              "type": "string"
            },
            "resourceGroupLocation": {
              "type": "string"
            },
            "resourceName": {
              "type": "string"
            },
            "resourceLocation": {
              "type": "string"
            }
          },
          "variables": {
            "storage_name": "[toLower(concat('storage',
uniqueString(concat(parameters('resourceName'), subscription().subscriptionId))))]",
            "appServicePlan_name": "[concat('Plan',
uniqueString(concat(parameters('resourceName'), subscription().subscriptionId)))]",
            "storage_ResourceId": "[concat('/subscriptions/',
subscription().subscriptionId, '/resourceGroups/', parameters('resourceGroupName'),
'/providers/Microsoft.Storage/storageAccounts/', variables('storage_name'))]",
            "appServicePlan_ResourceId": "[concat('/subscriptions/',
subscription().subscriptionId, '/resourceGroups/', parameters('resourceGroupName'),
'/providers/Microsoft.Web/serverFarms/', variables('appServicePlan_name'))]",
            "function_ResourceId": "[concat('/subscriptions/',
subscription().subscriptionId, '/resourceGroups/', parameters('resourceGroupName'),
'/providers/Microsoft.Web/sites/', parameters('resourceName'))]"
          },
          "resources": [
            {
              "location": "[parameters('resourceLocation')]",
              "name": "[parameters('resourceName')]",
              "type": "Microsoft.Web/sites",
              "apiVersion": "2015-08-01",
              "tags": {
                "[concat('hidden-related:', variables('appServicePlan_ResourceId'))]":
"empty"
              },
              "dependsOn": [
                "[variables('appServicePlan_ResourceId')]",
                "[variables('storage_ResourceId')]"
              ],
              "kind": "functionapp",
              "properties": {
                "name": "[parameters('resourceName')]",
                "kind": "functionapp",
                "httpsOnly": true,
                "reserved": false,
                "serverFarmId": "[variables('appServicePlan_ResourceId')]",
                "siteConfig": {
                  "alwaysOn": true
                }
              },
              "identity": {
                "type": "SystemAssigned"
              },
              "resources": [
                {
                  "name": "appsettings",
                  "type": "config",
                  "apiVersion": "2015-08-01",
                  "dependsOn": [
                    "[variables('function_ResourceId')]"
                  ],
                  "properties": {
                    "AzureWebJobsStorage":
```

```
"[concat('DefaultEndpointsProtocol=https;AccountName=', variables('storage_name'),
';AccountKey=', listKeys(variables('storage_ResourceId'), '2017-10-01').keys[0].value,
';EndpointSuffix=', 'core.windows.net')]",
                    "FUNCTIONS_EXTENSION_VERSION": "~3",
                    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
                  }
                }
              ]
            },
            {
              "location": "[parameters('resourceGroupLocation')]",
              "name": "[variables('storage_name')]",
              "type": "Microsoft.Storage/storageAccounts",
              "apiVersion": "2017-10-01",
              "tags": {
                "[concat('hidden-related:', concat('/providers/Microsoft.Web/sites/',
parameters('resourceName')))]": "empty"
              },
              "properties": {
                "supportsHttpsTrafficOnly": true
              },
              "sku": {
                "name": "Standard_LRS"
              },
              "kind": "Storage"
            },
            {
              "location": "[parameters('resourceGroupLocation')]",
              "name": "[variables('appServicePlan_name')]",
              "type": "Microsoft.Web/serverFarms",
              "apiVersion": "2015-08-01",
              "sku": {
                "name": "S1",
                "tier": "Standard",
                "family": "S",
                "size": "S1"
              },
              "properties": {
                "name": "[variables('appServicePlan_name')]"
              }
            }
          ]
        }
      }
    }
  ]
}
```

.\RealtyHub.ApiService\Properties\launchSettings.json

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "http://localhost:5538",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

.\RealtyHub.ApiService\Services\ContractGenerator.cs

```
using System.IO.Compression;
using GrapeCity.Documents.Word;
using GrapeCity.Documents.Word.Layout;
using RealtyHub.Core.Models;
using Xceed.Document.NET;
using Xceed.Words.NET;

namespace RealtyHub.ApiService.Services;

/// <summary>
/// Classe responsável por gerar contratos utilizando templates e mapeamentos de
variáveis.
/// </summary>
/// <remarks>
/// A classe <c><see cref="ContractGenerator"/></c> utiliza um template em formato DOCX
para substituir
/// as variáveis de contrato e gerar um documento final em PDF.
/// </remarks>
public class ContractGenerator
{
    /// <summary>
    /// Gera um contrato a partir de um objeto <c><see cref="Contract"/></c>
    /// e de um template opcional <c><see cref="ContractTemplate"/></c>.
    /// </summary>
    /// <remarks>
    /// Este método realiza as seguintes etapas:
    /// <para>1. Carrega o template DOCX do contrato.</para>
    /// <para>2. Substitui as variáveis no documento de acordo com
    /// os mapeamentos gerados a partir do objeto <c><see cref="Contract"/></c>.</para>
    /// <para>3. Salva o documento modificado como DOCX.</para>
    /// <para>4. Converte o arquivo DOCX para PDF utilizando a biblioteca
GrapeCity.</para>
    /// <para>5. Exclui o arquivo DOCX temporário.</para>
    /// </remarks>
    /// <param name="contract">Objeto <c><see cref="Contract"/></c>
    /// contendo os dados do contrato a ser gerado.</param>
    /// <param name="model">Objeto opcional do tipo <c><see cref="ContractTemplate"/></c>
    /// que define o template do contrato.</param>
    public void GenerateContract(Contract contract, ContractTemplate? model)
    {
        var pathOutputDocx = Path.Combine(Configuration.ContractsPath,
$"{contract.FileId}.docx");
        var pathTemplateDocx = Path.Combine(Configuration.ContractTemplatesPath,
$"{model?.Id}{model?.Extension}");
        using var doc = DocX.Load(pathTemplateDocx);

        var mappings = new VariablesContractMappings(contract);
        var fiels = mappings.GetFields();

        foreach (var kvp in fiels)
        {
            doc.ReplaceText(new StringReplaceTextOptions
            {
                SearchValue = $"%{kvp.Key}%",
                NewValue = kvp.Value,
            });
        }

        doc.SaveAs(pathOutputDocx);

        var wordDoc = new GcWordDocument();
        wordDoc.Load(pathOutputDocx);

        using (var layoyt = new GcWordLayout(wordDoc))
        {
            var pdfOutpuSettings = new PdfOutputSettings
            {
                CompressionLevel = CompressionLevel.Fastest,
                ConformanceLevel = GrapeCity.Documents.Pdf.PdfAConformanceLevel.PdfA1a
            };

            layoyt.SaveAsPdf(Path.Combine(Configuration.ContractsPath,
```

```
$"{contract.FileId}.pdf"), null, pdfOutpuSettings);
        }
        File.Delete(pathOutputDocx);
    }
}
```

.\RealtyHub.ApiService\Services\EmailService.cs

```
using RealtyHub.Core.Requests.Emails;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;
using System.Net;
using System.Net.Mail;

namespace RealtyHub.ApiService.Services;

/// <summary>
/// Classe responsável pelo envio de e-mails utilizando o protocolo SMTP.
/// </summary>
/// <remarks>
/// A classe <c><see cref="EmailService"/></c> implementa a interface
/// <c><see cref="IEmailService"/></c> e fornece métodos para enviar e-mails
/// </remarks>
public class EmailService : IEmailService
{
    /// <summary>
    /// Envia um link de confirmação para o e-mail do usuário.
    /// </summary>
    /// <remarks>
    /// Este método configura um cliente SMTP e envia um e-mail
    /// formatado em HTML contendo um link para confirmação de e-mail.
    /// </remarks>
    /// <param name="message">
    /// O objeto do tipo <c><see cref="ConfirmEmailMessage"/></c>
    /// que contém o e-mail de destino e o link de confirmação.
    /// </param>
    /// <returns>
    /// Um objeto <c><see cref="Response{TData}"/></c> indicando o resultado do envio do
e-mail.
    /// </returns>
    public Task<Response<bool>> SendConfirmationLinkAsync(ConfirmEmailMessage message)
    {
        var smtpClient = ConfigureClient();
        var mailMessage = ConfigureMailMessage();

        mailMessage.To.Add(message.EmailTo);
        mailMessage.Subject = "Confirme seu email";
        mailMessage.Body = $"<h3>Por favor, confirme seu email clicando neste link:</h3>"
+
                           $"<h3> <a href='{message.ConfirmationLink}'>Confirmar
email</a></h3>";
        mailMessage.IsBodyHtml = true;

        try
        {
            smtpClient.Send(mailMessage);
            return Task.FromResult(new Response<bool>(true));
        }
        catch (Exception ex)
        {
            return Task.FromResult(new Response<bool>(false, 500, ex.Message));
        }
    }

    /// <summary>
    /// Envia um link para redefinição de senha para o e-mail do usuário.
    /// </summary>
    /// <remarks>
    /// Este método configura um cliente SMTP e envia um e-mail formatado
    /// em HTML contendo um link para redefinir a senha.
    /// </remarks>
    /// <param name="message">
    /// O objeto do tipo <c><see cref="ResetPasswordMessage"/></c>
    /// que contém os detalhes do e-mail e o link para redefinição.
    /// </param>
    /// <returns>
    /// Um objeto <c><see cref="Response{TData}"/></c> indicando se o envio do e-mail foi
realizado com sucesso.
    /// </returns>
    public Task<Response<bool>> SendResetPasswordLinkAsync(ResetPasswordMessage message)
```

```
    {
        var smtpClient = ConfigureClient();
        var mailMessage = ConfigureMailMessage();

        mailMessage.To.Add(message.EmailTo);
        mailMessage.Subject = "Redefinir senha";
        mailMessage.Body = $"<h3>Para redefinir sua senha, clique no link abaixo:</h3>" +
                           $"<h3> <a href='{message.ResetPasswordLink}'>Redefinir
senha</a></h3>";
        mailMessage.IsBodyHtml = true;

        try
        {
            smtpClient.Send(mailMessage);
            return Task.FromResult(new Response<bool>(true));
        }
        catch (Exception ex)
        {
            return Task.FromResult(new Response<bool>(false, 500, ex.Message));
        }
    }

    /// <summary>
    /// Envia um contrato como anexo para o e-mail do usuário.
    /// </summary>
    /// <remarks>
    /// Este método configura um cliente SMTP e envia um e-mail com um anexo,
    /// utilizando os detalhes fornecidos na mensagem.
    /// </remarks>
    /// <param name="message">
    /// O objeto do tipo <c><see cref="AttachmentMessage"/></c> que contém
    /// o caminho do anexo, assunto e corpo do e-mail.
    /// </param>
    /// <returns>
    /// Um objeto <c><see cref="Response{TData}"/></c> indicando se o envio
    /// do e-mail com anexo foi realizado com sucesso.
    /// </returns>
    public Task<Response<bool>> SendContractAsync(AttachmentMessage message)
    {
        var smtpClient = ConfigureClient();
        var mailMessage = ConfigureMailMessage();

        var attachment = new Attachment(message.AttachmentPath);
        mailMessage.Attachments.Add(attachment);

        mailMessage.To.Add(message.EmailTo);
        mailMessage.Subject = message.Subject;
        mailMessage.Body = message.Body;
        mailMessage.IsBodyHtml = true;

        try
        {
            smtpClient.Send(mailMessage);
            return Task.FromResult(new Response<bool>(true));
        }
        catch (Exception ex)
        {
            return Task.FromResult(new Response<bool>(false, 500, ex.Message));
        }
    }

    /// <summary>
    /// Configura e retorna um cliente SMTP.
    /// </summary>
    /// <remarks>
    /// Este método configura o cliente SMTP com as credenciais
    /// e as configurações definidas em <c><see cref="Configuration.EmailSettings"/></c>.
    /// </remarks>
    /// <returns>
    /// Um objeto <c>SmtpClient</c> configurado.
    /// </returns>
    private static SmtpClient ConfigureClient()
    {
        return new SmtpClient
        {
            Host = "smtp.gmail.com",
```

```
            EnableSsl = true,
            Port = 587,
            UseDefaultCredentials = false,
            Credentials = new NetworkCredential(
                Configuration.EmailSettings.EmailFrom,
                Configuration.EmailSettings.EmailPassword)
        };
    }

    /// <summary>
    /// Configura e retorna uma mensagem de e-mail.
    /// </summary>
    /// <remarks>
    /// Este método cria um objeto <c><see cref="MailMessage"/></c> definindo o remetente
    /// com base nas configurações em <c><see cref="Configuration.EmailSettings"/></c>.
    /// </remarks>
    /// <returns>
    /// Um objeto <c><see cref="MailMessage"/></c> configurado.
    /// </returns>
    private static MailMessage ConfigureMailMessage()
    {
        return new MailMessage
        {
            From = new MailAddress(Configuration.EmailSettings.EmailFrom, "Realty Hub")
        };
    }
}
```

.\RealtyHub.ApiService\Services\Reports\OfferReportService.cs

```
using QuestPDF.Fluent;
using QuestPDF.Infrastructure;
using RealtyHub.Core.Extensions;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Services.Reports;

/// <summary>
/// Serviço que gera um relatório de propostas a imóveis utilizando a biblioteca
QuestPDF.
/// </summary>
public class OfferReportService : IDocument
{
    /// <summary>
    /// Lista de ofertas a serem incluídas no relatório.
    /// </summary>
    private readonly List<Offer> _offers;

    /// <summary>
    /// Inicializa uma nova instância de <c><see cref="OfferReportService"/></c> com a
lista de ofertas especificada.
    /// </summary>
    /// <param name="offers">A lista de ofertas para inclusão no relatório.</param>
    public OfferReportService(List<Offer> offers)
    {
        _offers = offers;
    }

    /// <summary>
    /// Obtém os metadados do documento.
    /// </summary>
    /// <remarks>
    /// Os metadados deste documento permitem identificar o título e a autoria do
relatório gerado.
    /// </remarks>
    /// <returns>
    /// Um objeto <c><see cref="DocumentMetadata"/></c> contendo o título e o autor do
relatório.
    /// </returns>
    public DocumentMetadata GetMetadata() => new()
    {
        Title = "Relatório de Propostas a Imóveis",
        Author = "RealtyHub",
    };

    /// <summary>
    /// Compoe o documento configurando as seções de cabeçalho, conteúdo e rodapé.
    /// </summary>
    /// <param name="container">O contêiner do documento a ser configurado.</param>
    /// <remarks>
    /// Este método organiza o layout do relatório utilizando as funcionalidades da
biblioteca QuestPDF.
    /// </remarks>
    public void Compose(IDocumentContainer container)
    {
        container.Page(page =>
        {
            page.Margin(30);
            page.DefaultTextStyle(x => x.FontSize(8));
            page.Header().Element(ComposeHeader);
            page.Content().Element(ComposeContent);
            page.Footer().Element(ComposeFooter);
        });
    }

    /// <summary>
    /// Compoe o cabeçalho do relatório.
    /// </summary>
    /// <param name="container">O contêiner onde o cabeçalho será construído.</param>
    /// <remarks>
    /// O cabeçalho inclui o título do relatório e o logotipo da aplicação.
    /// </remarks>
```

```
    private void ComposeHeader(IContainer container)
    {
        var pathLogo = Path.Combine(Configuration.LogosPath, "white-logo-nobg.png");
        container.Row(row =>
        {
            row.RelativeItem().Text("Relatório de Propostas a Imóveis")
                .FontSize(16)
                .Bold();

            row.ConstantItem(100).Height(50)
                .Image(pathLogo);
        });
    }

    /// <summary>
    /// Compoe o rodapé do relatório, exibindo o número da página atual e o total de
páginas.
    /// </summary>
    /// <param name="container">O contêiner onde o rodapé será construído.</param>
    /// <remarks>
    /// O rodapé é centralizado e atualiza dinamicamente o número atual e o total de
páginas.
    /// </remarks>
    private void ComposeFooter(IContainer container)
    {
        container.AlignCenter().Text(txt =>
        {
            txt.CurrentPageNumber();
            txt.Span(" / ");
            txt.TotalPages();
        });
    }

    /// <summary>
    /// Compoe o conteúdo principal do relatório, montando uma tabela com informações das
ofertas.
    /// </summary>
    /// <param name="container">O contêiner onde o conteúdo será construído.</param>
    /// <remarks>
    /// Este método cria uma tabela que exibe dados importantes de cada oferta, como
identificação, data, status, valor, comprador e vendedor.
    /// </remarks>
    private void ComposeContent(IContainer container)
    {
        container.Section("Main").Table(table =>
        {
            table.ColumnsDefinition(columns =>
            {
                columns.ConstantColumn(50);
                columns.ConstantColumn(70);
                columns.ConstantColumn(60);
                columns.RelativeColumn();
                columns.RelativeColumn();
                columns.RelativeColumn();
            });

            table.Header(header =>
            {
                header.Cell().Background("#EEE").Text("Proposta");
                header.Cell().Background("#EEE").Text("Data");
                header.Cell().Background("#EEE").Text("Status");
                header.Cell().Background("#EEE").Text("Valor proposto");
                header.Cell().Background("#EEE").Text("Comprador");
                header.Cell().Background("#EEE").Text("Vendedor");
            });

            foreach (var offer in _offers)
            {
                table.Cell().Text($"{offer.PropertyId}");
                table.Cell().Text($"{offer.SubmissionDate:dd/MM/yyyy}");
                table.Cell().Text($"{offer.OfferStatus.GetDisplayName()}");
                table.Cell().Text(offer.Amount.ToString("C"));
                table.Cell().Text(offer.Buyer!.Name);
                table.Cell().Text(offer.Property!.Seller!.Name);
            }
        });
```

```
    }
}
```

.\RealtyHub.ApiService\Services\Reports\PropertyReportService.cs

```
using QuestPDF.Fluent;
using QuestPDF.Infrastructure;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Services.Reports;

/// <summary>
/// Serviço que gera um relatório de imóveis utilizando a biblioteca QuestPDF.
/// </summary>
public class PropertyReportService : IDocument
{
    /// <summary>
    /// Lista de imóveis a serem incluídos no relatório.
    /// </summary>
    private readonly List<Property> _properties;

    /// <summary>
    /// Inicializa uma nova instância de <c><see cref="PropertyReportService"/></c> com a
lista de imóveis especificada.
    /// </summary>
    /// <remarks>
    /// Este construtor recebe a lista de imóveis que será utilizada para gerar o
relatório.
    /// </remarks>
    /// <param name="properties">A lista de imóveis para inclusão no relatório.</param>
    public PropertyReportService(List<Property> properties)
    {
        _properties = properties;
    }

    /// <summary>
    /// Obtém os metadados do documento.
    /// </summary>
    /// <remarks>
    /// Os metadados deste documento permitem identificar o título e a autoria do
relatório gerado.
    /// </remarks>
    /// <returns>
    /// Um objeto <c><see cref="DocumentMetadata"/></c> contendo o título e o autor do
relatório.
    /// </returns>
    public DocumentMetadata GetMetadata() => new()
    {
        Title = "Relatório de Imóveis",
        Author = "RealtyHub",
    };

    /// <summary>
    /// Compoe o documento configurando suas seções de cabeçalho, conteúdo e rodapé.
    /// </summary>
    /// <remarks>
    /// Este método organiza o layout do relatório utilizando as funcionalidades da
biblioteca QuestPDF.
    /// </remarks>
    /// <param name="container">O contêiner do documento a ser configurado.</param>
    public void Compose(IDocumentContainer container)
    {
        container.Page(page =>
        {
            page.Margin(30);
            page.DefaultTextStyle(x => x.FontSize(9));
            page.Header().Element(ComposeHeader);
            page.Content().Element(ComposeContent);
            page.Footer().Element(ComposeFooter);
        });
    }

    /// <summary>
    /// Compoe o cabeçalho do relatório.
    /// </summary>
    /// <remarks>
    /// O cabeçalho inclui o título do relatório e o logotipo da aplicação.
```

```
    /// </remarks>
    /// <param name="container">O contêiner onde o cabeçalho será construído.</param>
    private void ComposeHeader(IContainer container)
    {
        var pathLogo = Path.Combine(Configuration.LogosPath, "white-logo-nobg.png");
        container.Row(row =>
        {
            row.RelativeItem().Text("Relatório de Imóveis")
                .FontSize(16)
                .Bold();

            row.ConstantItem(100).Height(50)
                .Image(pathLogo);
        });
    }

    /// <summary>
    /// Compoe o rodapé do relatório, exibindo o número da página atual juntamente com o
total de páginas.
    /// </summary>
    /// <remarks>
    /// O rodapé é centralizado e atualiza dinamicamente o número da página atual e o
total de páginas.
    /// </remarks>
    /// <param name="container">O contêiner onde o rodapé será construído.</param>
    private void ComposeFooter(IContainer container)
    {
        container.AlignCenter().Text(txt =>
        {
            txt.CurrentPageNumber();
            txt.Span(" / ");
            txt.TotalPages();
        });
    }

    /// <summary>
    /// Compoe o conteúdo principal do relatório, montando uma tabela com as informações
dos imóveis.
    /// </summary>
    /// <remarks>
    /// Este método cria uma tabela que exibe dados importantes de cada imóvel, como o
identificador, datas de cadastro e atualização,
    /// preço, quantidade de quartos, banheiros, vagas, área e os status de novo e ativo.
    /// </remarks>
    /// <param name="container">O contêiner onde o conteúdo será construído.</param>
    private void ComposeContent(IContainer container)
    {
        container.Section("Main").Table(table =>
        {
            table.ColumnsDefinition(columns =>
            {
                columns.ConstantColumn(50);
                columns.ConstantColumn(70);
                columns.ConstantColumn(80);
                columns.ConstantColumn(70);
                columns.RelativeColumn(15);
                columns.RelativeColumn(15);
                columns.RelativeColumn(15);
                columns.RelativeColumn(15);
                columns.RelativeColumn(10);
                columns.RelativeColumn(10);
            });

            table.Header(header =>
            {
                header.Cell().Background("#EEE").Text("Imóvel");
                header.Cell().Background("#EEE").Text("Cadastro");
                header.Cell().Background("#EEE").Text("Última atualização");
                header.Cell().Background("#EEE").Text("Valor");
                header.Cell().Background("#EEE").Text("Quartos");
                header.Cell().Background("#EEE").Text("Banheiros");
                header.Cell().Background("#EEE").Text("Garagem");
                header.Cell().Background("#EEE").Text("Área(m²)");
                header.Cell().Background("#EEE").Text("Novo");
                header.Cell().Background("#EEE").Text("Ativo");
            });
```

```
            foreach (var property in _properties)
            {
                table.Cell().Text($"{property.Id}");
                table.Cell().Text($"{property.CreatedAt:dd/MM/yyyy}");
                table.Cell().Text($"{property.UpdatedAt:dd/MM/yyyy}");
                table.Cell().Text($"{property.Price:C}");
                table.Cell().Text($"{property.Bedroom}");
                table.Cell().Text($"{property.Bathroom}");
                table.Cell().Text($"{property.Garage}");
                table.Cell().Text($"{property.Area}");
                table.Cell().Text(property.IsNew ? "Sim" : "Não");
                table.Cell().Text(property.IsActive ? "Sim" : "Não");
            }
        });
    }
}
```

.\RealtyHub.ApiService\Services\VariablesContractMappings.cs

```
using System.Globalization;
using RealtyHub.Core.Extensions;
using RealtyHub.Core.Models;

namespace RealtyHub.ApiService.Services;

/// <summary>
/// Classe responsável por mapear as variáveis de contrato para substituição em
templates.
/// </summary>
/// <remarks>
/// <para>A classe <c><see cref="VariablesContractMappings"/></c>
/// gera um dicionário de campos que serão utilizados para substituir
/// as variáveis em um template de contrato.</para>
/// <para>Ela concatena campos padrão, de pessoa jurídica e pessoa física,
/// além dos dados dos pagamentos.</para>
/// </remarks>
public class VariablesContractMappings
{
    /// <summary>
    /// Retorna um dicionário com todas as variáveis mapeadas para o contrato.
    /// </summary>
    /// <remarks>
    /// Este método concatena o dicionário de campos padrão, os campos de pessoa
jurídica, os campos de pessoa física
    /// e os textos referentes aos pagamentos, retornando um dicionário que mapeia cada
variável ao seu valor correspondente.
    /// </remarks>
    /// <returns>
    /// Um objeto <c><see cref="Dictionary{TKey, TValue}"/></c> contendo as chaves e os
respectivos valores para substituição.
    /// </returns>
    public Dictionary<string, string> GetFields()
    {
        var paymentsFields = GetPaymentsText(_contract.Offer!.Payments);
        return _defaultFields
            .Concat(_fieldsPj)
            .Concat(_fieldsPf)
            .Concat(paymentsFields)
            .ToDictionary(kvp => kvp.Key, kvp => kvp.Value);
    }

    /// <summary>
    /// Dicionário contendo os campos padrão mapeados para o contrato.
    /// </summary>
    /// <remarks>
    private readonly Dictionary<string, string> _defaultFields;

    /// <summary>
    /// Dicionário contendo os campos de pessoa jurídica mapeados para o contrato.
    /// </summary>
    private readonly Dictionary<string, string> _fieldsPj;

    /// <summary>
    /// Dicionário contendo os campos de pessoa física mapeados para o contrato.
    /// </summary>
    private readonly Dictionary<string, string> _fieldsPf;

    /// <summary>
    /// Objeto do tipo <c><see cref="Contract"/></c> que contém os dados do contrato.
    /// </summary>
    private readonly Contract _contract;

    /// <summary>
    /// Inicializa uma nova instância de <c><see cref="VariablesContractMappings"/></c>
com o contrato especificado.
    /// </summary>
    /// <remarks>
    /// O construtor recebe um objeto do tipo <c><see cref="Contract"/></c> e inicializa
os dicionários de campos padrão,
    /// de pessoa jurídica e de pessoa física, mapeando os dados pertinentes do contrato.
    /// </remarks>
```

```
    /// <param name="contract">Um objeto do tipo <c><see cref="Contract"/></c> contendo
os dados do contrato.</param>
    public VariablesContractMappings(Contract contract)
    {
        _contract = contract;
        _defaultFields = new Dictionary<string, string>
        {
            { "enderecoCompleto_cliente_proprietario",
                $"Rua: {contract.Seller!.Address.Street}, " +
                (string.IsNullOrEmpty(contract.Seller.Address.Complement) ? ""
                    : $"Complemento: {contract.Seller.Address.Complement}, ") +
                $"Número: {contract.Seller.Address.Number}, " +
                $"Bairro: {contract.Seller.Address.Neighborhood}, " +
                $"Cidade: {contract.Seller.Address.City}, " +
                $"Estado: {contract.Seller.Address.State}, " +
                $"País: {contract.Seller.Address.Country}, " +
                $"CEP: {contract.Seller.Address.ZipCode} "
            },
            { "enderecoCompleto_cliente_comprador",
                $"Rua: {contract.Buyer!.Address.Street}, " +
                (string.IsNullOrEmpty(contract.Buyer.Address.Complement) ? ""
                    : $"Complemento: {contract.Buyer.Address.Complement}, ") +
                $"Número: {contract.Buyer.Address.Number}, " +
                $"Bairro: {contract.Buyer.Address.Neighborhood}, " +
                $"Cidade: {contract.Buyer.Address.City}, " +
                $"Estado: {contract.Buyer.Address.State}, " +
                $"País: {contract.Buyer.Address.Country}, " +
                $"CEP: {contract.Buyer.Address.ZipCode} "
            },
            { "tipo_imovel", contract.Offer!.Property!.PropertyType.GetDisplayName() },
            { "enderecoCompleto_imovel",
                $"Rua: {contract.Offer.Property!.Address.Street}, " +
                (string.IsNullOrEmpty(contract.Offer.Property!.Address.Complement) ? ""
                    : $"Complemento: {contract.Offer.Property!.Address.Complement}, ") +
                $"Número: {contract.Offer.Property!.Address.Number}, " +
                $"Bairro: {contract.Offer.Property!.Address.Neighborhood}, " +
                $"Cidade: {contract.Offer.Property!.Address.City}, " +
                $"Estado: {contract.Offer.Property!.Address.State}, " +
                $"País: {contract.Offer.Property!.Address.Country}, " +
                $"CEP: {contract.Offer.Property!.Address.ZipCode} "
            },
            { "area_imovel",
contract.Offer!.Property!.Area.ToString(CultureInfo.CurrentCulture) },
            { "numeroMatriculaCartorio_imovel", contract.Offer!.Property!.RegistryNumber
},
            { "cartorioRegistro_imovel", contract.Offer!.Property!.RegistryRecord },
            { "valorOferecido_proposta",
contract.Offer!.Amount.ToString(CultureInfo.CurrentCulture) },
            { "cidade_imovel", contract.Offer!.Property!.Address.City },
            { "estado_imovel", contract.Offer!.Property!.Address.State },
            { "data_assinatura_contrato", contract.SignatureDate?.ToString("dd/MM/yyyy")!
},
        };
        _fieldsPj = new Dictionary<string, string>
        {
            { "razaoSocial_cliente_proprietario", contract.Seller!.BusinessName },
            { "cnpj_cliente_proprietario", contract.Seller.DocumentNumber },
            { "razaoSocial_cliente_comprador", contract.Buyer!.BusinessName },
            { "cnpj_cliente_comprador", contract.Buyer!.DocumentNumber }
        };
        _fieldsPf = new Dictionary<string, string>
        {
            { "nome_cliente_proprietario", contract.Seller.Name },
            { "nome_cliente_comprador", contract.Buyer.Name },
            { "nacionalidade_cliente_proprietario", contract.Seller.Nationality },
            { "nacionalidade_cliente_comprador", contract.Buyer.Nationality },
            { "estadoCivil_cliente_proprietario",
contract.Seller.MaritalStatus.GetDisplayName() },
            { "estadoCivil_cliente_comprador",
contract.Buyer.MaritalStatus.GetDisplayName() },
            { "profissao_cliente_proprietario", contract.Seller.Occupation },
            { "profissao_cliente_comprador", contract.Buyer.Occupation },
            { "cpf_cliente_proprietario", contract.Seller.DocumentNumber },
            { "cpf_cliente_comprador", contract.Buyer.DocumentNumber },
            { "rg_cliente_proprietario", contract.Seller.Rg },
            { "rg_cliente_comprador", contract.Buyer.Rg },
```

```
            { "orgaoExpeditorRg_cliente_proprietario", contract.Seller.IssuingAuthority
},
            { "orgaoExpeditorRg_cliente_comprador", contract.Seller.IssuingAuthority },
            { "dataemissaorg_cliente_proprietario",
contract.Seller.RgIssueDate?.ToString("dd/MM/yyyy")! },
            { "dataemissaorg_cliente_comprador",
contract.Seller.RgIssueDate?.ToString("dd/MM/yyyy")! }
        };
    }

    /// <summary>
    /// Gera uma representação textual dos pagamentos do contrato.
    /// </summary>
    /// <remarks>
    /// Este método percorre a lista de pagamentos do contrato, criando um mapeamento de
forma
    /// que cada pagamento seja representado por uma chave no formato "formaPagamentoX",
onde X é um índice.
    /// Caso haja menos de 5 pagamentos, as chaves restantes são preenchidas com uma
string vazia.
    /// </remarks>
    /// <param name="payments">A lista de pagamentos do contrato.</param>
    /// <returns>
    /// Um objeto <c><see cref="Dictionary{TKey, TValue}"/></c> representando os
pagamentos mapeados para substituição.
    /// </returns>
    private Dictionary<string, string> GetPaymentsText(List<Payment> payments)
    {
        var dictPayments = new Dictionary<string, string>();
        var index = 1;
        foreach (var payment in payments)
        {
            dictPayments.Add($"formaPagamento{index}", payment.ToString());
            index++;
        }

        for (var i = index; i <= 5; i++)
        {
            dictPayments.Add($"formaPagamento{i}", "");
        }
        return dictPayments;
    }
}
```

.\RealtyHub.ApiService\appsettings.Development.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

.\RealtyHub.ApiService\appsettings.json

```
{
  "EmailPassword": "",
  "BackendUrl": "http://localhost:5538",
  "FrontendUrl": "http://localhost:5187",
  "ConnectionStrings": {
    "DefaultConnection":
"Host=localhost;Port=5432;Database=realtyhub;Username=root;Password=masterkey"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

.\RealtyHub.AppHost\Program.cs

```
var builder = DistributedApplication.CreateBuilder(args);

var apiService = builder.AddProject<Projects.RealtyHub_ApiService>("apiservice");

builder.AddProject<Projects.RealtyHub_Web>("webfrontend")
    .WithExternalHttpEndpoints()
    .WithReference(apiService)
    .WaitFor(apiService);

builder.Build().Run();
```

.\RealtyHub.AppHost\Properties\launchSettings.json


```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "http://localhost:15276",
      "environmentVariables": {
        "ASPIRE_ALLOW_UNSECURED_TRANSPORT": "true",
        "ASPNETCORE_ENVIRONMENT": "Development",
        "DOTNET_ENVIRONMENT": "Development",
        "DOTNET_DASHBOARD_OTLP_ENDPOINT_URL": "http://localhost:19040",
        "DOTNET_RESOURCE_SERVICE_ENDPOINT_URL": "http://localhost:20112"
      }
    }
  }
}
```

.\RealtyHub.AppHost\appsettings.Development.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

.\RealtyHub.AppHost\appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Aspire.Hosting.Dcp": "Warning"
    }
  }
}
```

.\RealtyHub.Core\Configuration.cs

```
namespace RealtyHub.Core;

/// <summary>
/// Representa as configurações da aplicação.
/// </summary>
public class Configuration
{
    /// <summary>
    /// Obtém o código de status padrão para as respostas.
    /// </summary>
    /// <value>Um inteiro representando o código de status padrão.</value>
    public const int DefaultStatusCode = 200;

    /// <summary>
    /// Obtém o número da página padrão para paginação.
    /// </summary>
    /// <value>Um inteiro representando o número da página padrão.</value>
    public const int DefaultPageNumber = 1;

    /// <summary>
    /// Obtém o locale utilizado pela aplicação.
    /// </summary>
    /// <value>Uma string representando o locale, por exemplo, "pt_BR".</value>
    public const string Locale = "pt_BR";

    /// <summary>
    /// Obtém o tamanho padrão da página para paginação.
    /// </summary>
    /// <value>Um inteiro representando o tamanho da página padrão.</value>
    public const int DefaultPageSize = 25;

    /// <summary>
    /// Obtém ou define a string de conexão utilizada para acessar o banco de dados.
    /// </summary>
    /// <value>Uma string representando a string de conexão.</value>
    public static string ConnectionString { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a URL do backend utilizada para chamadas à API.
    /// </summary>
    /// <value>Uma string representando a URL do backend.</value>
    public static string BackendUrl { get; set; } = "http://localhost:5538";

    /// <summary>
    /// Obtém ou define a URL do frontend utilizada para a interface web.
    /// </summary>
    /// <value>Uma string representando a URL do frontend.</value>
    public static string FrontendUrl { get; set; } = "http://localhost:5187";
}
```

.\RealtyHub.Core\Enums\EContractModelType.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os tipos de contrato.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes tipos de contrato que podem existir.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o tipo de contrato.</para>
/// </remarks>
public enum EContractModelType
{

    /// <summary>
    /// Tipo de contrato não definido.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o tipo de contrato não é especificado.
    /// </remarks>
    /// <value>0</value>
    [Display(Name = "Não definido")]
    None,

    /// <summary>
    /// Tipo de contrato entre pessoa jurídica e pessoa jurídica.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o contrato é entre duas pessoas jurídicas.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Pessoa Jurídica para Pessoa Jurídica")]
    PJPJ = 1,

    /// <summary>
    /// Tipo de contrato entre pessoa física e pessoa física.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o contrato é entre duas pessoas físicas.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Pessoa Física para Pessoa Física")]
    PFPF = 2,

    /// <summary>
    /// Tipo de contrato entre pessoa física e pessoa jurídica.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o contrato é entre uma pessoa física e uma pessoa
jurídica.
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Pessoa Física para Pessoa Jurídica")]
    PFPJ = 3,

    /// <summary>
    /// Tipo de contrato entre pessoa jurídica e pessoa física.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o contrato é entre uma pessoa jurídica e uma pessoa
física.
    /// </remarks>
    /// <value>4</value>
    [Display(Name = "Pessoa Jurídica para Pessoa Física")]
    PJPF = 4
}
```

.\RealtyHub.Core\Enums\ECustomerType.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os tipos de cliente.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes tipos de cliente que podem existir.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o tipo de cliente.</para>
/// </remarks>
public enum ECustomerType
{
    /// <summary>
    /// Tipo de cliente vendedor
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é um vendedor, ou seja, está vendendo um
imóvel.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Vendedor")]
    Seller = 1,

    /// <summary>
    /// Tipo de cliente comprador
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é um comprador, ou seja, está comprando
um imóvel.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Comprador")]
    Buyer = 2,

    /// <summary>
    /// Tipo de cliente vendedor e comprador
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é tanto vendedor quanto comprador, ou
seja, está vendendo e comprando um imóvel.
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Ambos")]
    BuyerSeller = 3
}
```

.\RealtyHub.Core\Enums\EMaritalStatus.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os estados civis dos clientes.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes estados civis que um cliente pode ter.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o estado civil.</para>
/// </remarks>
public enum EMaritalStatus
{
    /// <summary>
    /// Estado civil solteiro(a).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é solteiro(a).
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Solteiro(a)")]
    Single = 1,

    /// <summary>
    /// Estado civil casado(a).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é casado(a).
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Casado(a)")]
    Married = 2,

    /// <summary>
    /// Estado civil divorciado(a).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é divorciado(a).
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Divorciado(a)")]
    Divorced = 3,

    /// <summary>
    /// Estado civil viúvo(a).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é viúvo(a).
    /// </remarks>
    /// <value>4</value>
    [Display(Name = "Viúvo(a)")]
    Widowed = 4,

    /// <summary>
    /// Estado civil noivo(a).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o cliente é noivo(a).
    /// </remarks>
    /// <value>5</value>
    [Display(Name = "Noivo(a)")]
    Engaged = 5
}
```

.\RealtyHub.Core\Enums\EOfferStatus.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os status das ofertas.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes status que uma oferta pode ter.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o status da oferta.</para>
/// </remarks>
public enum EOfferStatus
{
    /// <summary>
    /// Status da oferta em análise.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando a oferta está em análise.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Em Análise")]
    Analysis = 1,

    /// <summary>
    /// Status da oferta aceita.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando a oferta foi aceita.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Aceita")]
    Accepted = 2,

    /// <summary>
    /// Status da oferta rejeitada.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando a oferta foi rejeitada.
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Rejeitada")]
    Rejected = 3
}
```

.\RealtyHub.Core\Enums\EPaymentType.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os tipos de pagamento.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes tipos de pagamento que podem ser
utilizados.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o tipo de pagamento.</para>
/// </remarks>
public enum EPaymentType
{
    /// <summary>
    /// Tipo de pagamento boleto
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito por meio de boleto bancário.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Boleto")]
    BankSlip = 1,

    /// <summary>
    /// Tipo de pagamento transferência bancária
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito por meio de transferência
bancária.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Transferência Bancária")]
    BankTranfer = 2,

    /// <summary>
    /// Tipo de pagamento cheque
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito por meio de cheque.
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Cheque")]
    Check = 3,

    /// <summary>
    /// Tipo de pagamento em dinheiro.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito em espécie.
    /// </remarks>
    /// <value>4</value>
    [Display(Name = "Dinheiro")]
    Cash = 4,

    /// <summary>
    /// Tipo de pagamento Pix.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito via Pix.
    /// </remarks>
    /// <value>5</value>
    [Display(Name = "Pix")]
    Pix = 5,

    /// <summary>
    /// Tipo de pagamento financiamento.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito por meio de financiamento.
    /// </remarks>
```

```
    /// <value>6</value>
    [Display(Name = "Financiamento")]
    Financing = 6,

    /// <summary>
    /// Tipo de pagamento cartão de crédito.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito por meio de cartão de crédito.
    /// </remarks>
    /// <value>7</value>
    [Display(Name = "Cartão de Crédito")]
    CreditCard = 7,

    /// <summary>
    /// Tipo de pagamento via FGTS.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o pagamento é feito utilizando os recursos do FGTS.
    /// </remarks>
    /// <value>8</value>
    [Display(Name = "FGTS")]
    Fgts = 8,

    /// <summary>
    /// Tipo de pagamento outros.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para pagamentos que não se enquadram nos tipos anteriores.
    /// </remarks>
    /// <value>9</value>
    [Display(Name = "Outros")]
    Others = 9
}
```

.\RealtyHub.Core\Enums\EPersonType.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os tipos de pessoa.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes tipos de pessoa que podem ser utilizados na
aplicação.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o tipo de pessoa.</para>
/// </remarks>
public enum EPersonType
{
    /// <summary>
    /// Tipo de pessoa individual (Pessoa Física).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o registro refere-se a uma pessoa física.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Pessoa Física")]
    Individual = 1,

    /// <summary>
    /// Tipo de pessoa jurídica (Pessoa Jurídica).
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando o registro refere-se a uma pessoa jurídica.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Pessoa Jurídica")]
    Business = 2
}
```

.\RealtyHub.Core\Enums\EPropertyType.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os tipos de propriedade.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes tipos de propriedade disponíveis na
aplicação.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o tipo de propriedade.</para>
/// </remarks>
public enum EPropertyType
{
    /// <summary>
    /// Tipo de propriedade Casa.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar uma residência do tipo casa.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Casa")]
    House = 1,

    /// <summary>
    /// Tipo de propriedade Apartamento.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar uma residência do tipo apartamento.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Apartamento")]
    Apartment = 2,

    /// <summary>
    /// Tipo de propriedade Ponto Comercial.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar imóveis de uso comercial.
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Ponto Comercial")]
    Commercial = 3,

    /// <summary>
    /// Tipo de propriedade Terreno.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar um terreno.
    /// </remarks>
    /// <value>4</value>
    [Display(Name = "Terreno")]
    Land = 4,

    /// <summary>
    /// Tipo de propriedade Kitnet.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar um imóvel compacto, como uma kitnet.
    /// </remarks>
    /// <value>5</value>
    [Display(Name = "Kitnet")]
    Kitnet = 5,

    /// <summary>
    /// Tipo de propriedade Fazenda.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar uma propriedade rural, como uma fazenda.
    /// </remarks>
    /// <value>6</value>
```

```
    [Display(Name = "Fazenda")]
    Farm = 6,

    /// <summary>
    /// Tipo de propriedade Outros.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado para designar outros tipos de propriedade que não se
enquadram nas categorias anteriores.
    /// </remarks>
    /// <value>7</value>
    [Display(Name = "Outros")]
    Others = 7
}
```

.\RealtyHub.Core\Enums\EViewingStatus.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Enums;

/// <summary>
/// Enumerador para os status de visita.
/// </summary>
/// <remarks>
/// <para>O enum representa os diferentes status que uma visita pode assumir na
aplicação.</para>
/// <para>Cada valor do enum é associado a um número inteiro e possui um atributo
/// Display que fornece uma descrição legível para o status.</para>
/// </remarks>
public enum EViewingStatus
{
    /// <summary>
    /// Status de visita agendada.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando a visita está marcada para ocorrer.
    /// </remarks>
    /// <value>1</value>
    [Display(Name = "Agendada")]
    Scheduled = 1,

    /// <summary>
    /// Status de visita finalizada.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando a visita foi concluída.
    /// </remarks>
    /// <value>2</value>
    [Display(Name = "Finalizada")]
    Done = 2,

    /// <summary>
    /// Status de visita cancelada.
    /// </summary>
    /// <remarks>
    /// Este valor é utilizado quando a visita foi cancelada.
    /// </remarks>
    /// <value>3</value>
    [Display(Name = "Cancelada")]
    Canceled = 3
}
```

.\RealtyHub.Core\Extensions\DateTimeExtension.cs

```
namespace RealtyHub.Core.Extensions;

/// <summary>
/// Métodos de extensão para a classe DateTime.
/// </summary>
/// <remarks>
/// Esta classe contém métodos de extensão para a classe DateTime, permitindo
/// realizar operações comuns de forma mais conveniente.
/// </remarks>
public static class DateTimeExtension
{
    /// <summary>
    /// Obtém o primeiro dia do mês de uma data específica.
    /// </summary>
    /// <remarks>
    /// Este método retorna o primeiro dia do mês de uma data específica.
    /// Se o ano e o mês não forem fornecidos, o ano e o mês da data serão usados.
    /// </remarks>
    /// <param name="date">A data para a qual o primeiro dia do mês será obtido.</param>
    /// <param name="year">O ano desejado. Se não for fornecido, o ano da data será
usado.</param>
    /// <param name="month">O mês desejado. Se não for fornecido, o mês da data será
usado.</param>
    /// <returns>O primeiro dia do mês especificado.</returns>
    public static DateTime GetFirstDay(this DateTime date, int? year = null, int? month =
null)
    {
        return new(year ?? date.Year, month ?? date.Month, 1);
    }

    /// <summary>
    /// Obtém o último dia do mês de uma data específica.
    /// </summary>
    /// <remarks>
    /// Este método retorna o último dia do mês de uma data específica.
    /// Se o ano e o mês não forem fornecidos, o ano e o mês da data serão usados.
    /// </remarks>
    /// <param name="date">A data para a qual o último dia do mês será obtido.</param>
    /// <param name="year">O ano desejado. Se não for fornecido, o ano da data será
usado.</param>
    /// <param name="month">O mês desejado. Se não for fornecido, o mês da data será
usado.</param>
    /// <returns>O último dia do mês especificado.</returns>
    public static DateTime GetLastDay(this DateTime date, int? year = null, int? month =
null)
    {
        return new DateTime(year ?? date.Year, month ?? date.Month, 1)
                .AddMonths(1)
                .AddDays(-1);
    }

    /// <summary>
    /// Converte uma data para uma string no formato UTC.
    /// </summary>
    /// <remarks>
    /// Este método converte uma data para uma string no formato UTC.
    /// Se a data for nula, retorna uma string vazia.
    /// </remarks>
    /// <param name="date">A data a ser convertida.</param>
    /// <returns>Uma string representando a data no formato UTC.</returns>
    public static string ToUtcString(this DateTime? date)
    {
        return date is null
            ? string.Empty
            : date.Value.ToUniversalTime().ToString("o");
    }

    /// <summary>
    /// Converte uma data para o fim do dia.
    /// </summary>
    /// <remarks>
    /// Este método converte uma data para o fim do dia, definindo a hora como 23:59:59.
```

```
    /// </remarks>
    /// <param name="date">A data a ser convertida.</param>
    /// <returns>A data convertida para o fim do dia.</returns>
    public static DateTime ToEndOfDay(this DateTime date)
    {
        return date.Date
            .AddHours(23)
            .AddMinutes(59)
            .AddSeconds(59);
    }
}
```

.\RealtyHub.Core\Extensions\EnumExtension.cs

```
using System.ComponentModel.DataAnnotations;
using System.Reflection;

namespace RealtyHub.Core.Extensions;
/// <summary>
/// Métodos de extensão para a classe Enum.
/// </summary>
/// <remarks>
/// Esta classe contém métodos de extensão para a classe Enum, permitindo
/// obter o nome de exibição associado a um valor de enumeração.
/// </remarks>
public static class EnumExtension
{
    /// <summary>
    /// Obtém o nome de exibição associado a um valor de enumeração.
    /// </summary>
    /// <remarks>
    /// Este método retorna o nome de exibição associado a um valor de enumeração,
    /// utilizando o atributo DisplayAttribute, se estiver presente.
    /// Caso contrário, retorna o nome do valor da enumeração como string.
    /// </remarks>
    /// <param name="value">O valor da enumeração para o qual o nome de exibição será
obtido.</param>
    /// <returns>
    /// O nome de exibição associado ao valor da enumeração, ou o nome do valor
    /// da enumeração como string, se o atributo DisplayAttribute não estiver presente.
    /// </returns>
    public static string GetDisplayName(this Enum value)
    {
        var field = value.GetType().GetField(value.ToString());
        var displayAttribute = field?.GetCustomAttribute<DisplayAttribute>();
        return displayAttribute?.Name ?? value.ToString();
    }
}
```

.\RealtyHub.Core\Extensions\IQueryableExtension.cs

```
using System.Linq.Dynamic.Core;

namespace RealtyHub.Core.Extensions;

/// <summary>
/// Métodos de extensão para IQueryable
/// </summary>
/// <remarks>
/// Esta classe contém métodos de extensão para IQueryable, permitindo
/// realizar operações comuns de forma mais conveniente.
/// </remarks>
public static class IQueryableExtension
{
    /// <summary>
    /// Filtra uma IQueryable com base em um termo de pesquisa e uma propriedade.
    /// </summary>
    /// <remarks>
    /// Este método filtra uma IQueryable com base em um termo de pesquisa e uma
propriedade.
    /// Se a propriedade não for encontrada ou o tipo não for suportado, retorna a
IQueryable original.
    /// </remarks>
    /// <typeparam name="T">O tipo de entidade da IQueryable.</typeparam>
    /// <param name="query">A IQueryable a ser filtrada.</param>
    /// <param name="searchTerm">O termo de pesquisa a ser usado para filtrar.</param>
    /// <param name="filterBy">O nome da propriedade pela qual filtrar.</param>
    /// <returns>A IQueryable filtrada.</returns>
    public static IQueryable<T> FilterByProperty<T>(this IQueryable<T> query, string
searchTerm, string filterBy)
    {
        var propertyInfo = typeof(T).GetNestedProperty(filterBy);
        var propertyType = propertyInfo?.PropertyType;

        if (propertyType == null) return query;

        if (propertyType == typeof(string))
        {
            query = query.Where($"{filterBy}.ToLower().Contains(@0.ToLower())",
searchTerm);
        }
        else if (propertyType == typeof(bool))
        {
            var boolValue = bool.TryParse(searchTerm, out var parsedBool) && parsedBool;
            query = query.Where($"{filterBy} == @0", !boolValue);
        }
        else if (propertyType == typeof(int))
        {
            searchTerm = string.IsNullOrEmpty(searchTerm) ? "0" : searchTerm;
            var searchValue = Convert.ChangeType(searchTerm, propertyType);
            query = query.Where($"{filterBy} == @0", searchValue);
        }
        return query;
    }
}
```

.\RealtyHub.Core\Extensions\TypeExtension.cs

```
using System.Reflection;

namespace RealtyHub.Core.Extensions;

/// <summary>
/// Métodos de extensão para a classe Type.
/// </summary>
/// <remarks>
/// Esta classe contém métodos de extensão para a classe Type, permitindo
/// realizar operações comuns de forma mais conveniente.
/// </remarks>
public static class TypeExtension
{
    /// <summary>
    /// Obtém uma propriedade aninhada de um tipo com base em um caminho de propriedade.
    /// </summary>
    /// <remarks>
    /// Este método obtém uma propriedade aninhada de um tipo com base em um caminho de
propriedade.
    /// Se a propriedade não for encontrada, retorna null.
    /// </remarks>
    /// <param name="type">O tipo do qual obter a propriedade.</param>
    /// <param name="propertyPath">O caminho da propriedade aninhada.</param>
    /// <returns>A propriedade aninhada encontrada ou null se não for
encontrada.</returns>
    public static PropertyInfo? GetNestedProperty(this Type type, string propertyPath)
    {
        var properties = propertyPath.Split('.');
        var currentType = type;
        PropertyInfo? property = null;

        foreach (var prop in properties)
        {
            property = currentType.GetProperty(prop);
            if (property == null) return null;
            currentType = property.PropertyType;
        }
        return property;
    }
}
```

.\RealtyHub.Core\Handlers\IAccountHandler.cs

```
using RealtyHub.Core.Models.Account;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de contas de usuário.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com as contas de usuário, como login, registro, confirmação de e-mail,
/// recuperação de senha, etc.
/// </remarks>
public interface IAccountHandler
{
    /// <summary>
    /// Realiza o login de um usuário no sistema.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por autenticar um usuário com base nas credenciais
fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="LoginRequest"/> contendo as
credenciais do usuário.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="string"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<string>> LoginAsync(LoginRequest request);

    /// <summary>
    /// Realiza o registro de um novo usuário no sistema.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por criar uma nova conta de usuário com base nas
informações fornecidas.
    /// </remarks>
    /// <param name="request"> Instância de <see cref="RegisterRequest"/> contendo as
informações do novo usuário.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="string"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<string>> RegisterAsync(RegisterRequest request);

    /// <summary>
    /// Confirma o e-mail de um usuário após o registro.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por validar o e-mail do usuário com base no token
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="ConfirmEmailRequest"/> contendo o
token de confirmação e Id do usuário.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="string"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<string>> ConfirmEmailAsync(ConfirmEmailRequest request);

    /// <summary>
    /// Recupera a senha de um usuário.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por enviar um e-mail de recuperação de senha para o
```

```
usuário.
    /// </remarks>
    /// <param name="request">>Instância de <see cref="ForgotPasswordRequest"/> contendo
o e-mail do usuário.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="string"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<string>> ForgotPasswordAsync(ForgotPasswordRequest request);

    /// <summary>
    /// Redefine a senha de um usuário.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar a senha do usuário com base no token
fornecido.
    /// </remarks>
    /// <param name="request">>Instância de <see cref="ResetPasswordRequest"/> contendo o
token e a nova senha do usuário.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="string"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<string>> ResetPasswordAsync(ResetPasswordRequest request);

    /// <summary>
    /// Realiza o logout de um usuário do sistema.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por encerrar a sessão do usuário no sistema.
    /// </remarks>
    Task LogoutAsync();
}
```

.\RealtyHub.Core\Handlers\ICondominiumHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de condomínios.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com os condomínios, como criação, atualização, exclusão e recuperação de informações.
/// </remarks>
public interface ICondominiumHandler
{
    /// <summary>
    /// Cria um novo condomínio.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por criar um novo condomínio com base nas informações
fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Condominium"/> contendo as
informações do novo condomínio.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Condominium"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Condominium?>> CreateAsync(Condominium request);

    /// <summary>
    /// Atualiza as informações de um condomínio existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar as informações de um condomínio existente
com base nas informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Condominium"/> contendo as
informações atualizadas do condomínio.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Condominium"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Condominium?>> UpdateAsync(Condominium request);

    /// <summary>
    /// Exclui um condomínio existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por excluir um condomínio existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="DeleteCondominiumRequest"/>
contendo o Id do condomínio a ser excluído.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Condominium"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Condominium?>> DeleteAsync(DeleteCondominiumRequest request);

    /// <summary>
    /// Recupera as informações de um condomínio existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar as informações de um condomínio existente
com base no Id fornecido.
```

```
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetCondominiumByIdRequest"/>
contendo o Id do condomínio a ser recuperado.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Condominium"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Condominium?>> GetByIdAsync(GetCondominiumByIdRequest request);

    /// <summary>
    /// Recupera todos os condomínios existentes.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar todos os condomínios existentes com base
nos parâmetros de paginação fornecidos.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllCondominiumsRequest"/>
contendo os parâmetros de paginação.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista de <see cref="Condominium"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<PagedResponse<List<Condominium>?>> GetAllAsync(GetAllCondominiumsRequest
request);
}
```

.\RealtyHub.Core\Handlers\IContractHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de contratos.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com os contratos, como criação, atualização, exclusão e recuperação de informações.
/// </remarks>
public interface IContractHandler
{
    /// <summary>
    /// Cria um novo contrato.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por criar um novo contrato com base nas informações
fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Contract"/> contendo as informações
do novo contrato.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Contract"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Contract?>> CreateAsync(Contract request);

    /// <summary>
    /// Atualiza as informações de um contrato existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar as informações de um contrato existente
com base nas informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Contract"/> contendo as informações
atualizadas do contrato.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Contract"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Contract?>> UpdateAsync(Contract request);

    /// <summary>
    /// Exclui um contrato existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por excluir um contrato existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="DeleteContractRequest"/> contendo o
Id do contrato a ser excluído.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Contract"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Contract?>> DeleteAsync(DeleteContractRequest request);

    /// <summary>
    /// Recupera as informações de um contrato existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar as informações de um contrato existente
com base no Id fornecido.
```

```
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetContractByIdRequest"/> contendo
o Id do contrato a ser recuperado.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Contract"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Contract?>> GetByIdAsync(GetContractByIdRequest request);

    /// <summary>
    /// Recupera uma lista de contratos com base nos critérios de pesquisa fornecidos.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar uma lista de contratos com base nos
critérios de pesquisa fornecidos.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllContractsRequest"/> contendo
os critérios de pesquisa.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista de <see cref="Contract"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<PagedResponse<List<Contract>?>> GetAllAsync(GetAllContractsRequest request);
}
```

.\RealtyHub.Core\Handlers\IContractTemplateHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de modelos de contrato.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com os modelos de contrato, como a recuperação de todos os modelos disponíveis.
/// </remarks>
public interface IContractTemplateHandler
{
    /// <summary>
    /// Recupera todos os modelos de contrato disponíveis.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por buscar todos os modelos de contrato
    /// armazenados no sistema e retornar uma lista com os resultados.
    /// </remarks>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> contendo uma lista de modelos de
contrato.
    /// O objeto TData é uma lista do tipo <see cref="ContractTemplate"/> e pode ser nulo
se a operação falhar.
    /// </returns>
    Task<Response<List<ContractTemplate>?>> GetAllAsync();
}
```

.\RealtyHub.Core\Handlers\ICustomerHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de clientes.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com os clientes, como criação, atualização, exclusão e recuperação de informações.
/// </remarks>
public interface ICustomerHandler
{
    /// <summary>
    /// Cria um novo cliente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por criar um novo cliente com base nas informações
fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Customer"/> contendo as informações
do novo cliente.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Customer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Customer?>> CreateAsync(Customer request);

    /// <summary>
    /// Atualiza um cliente existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar as informações de um cliente existente
com base nas informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Customer"/> contendo as informações
atualizadas do cliente.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Customer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Customer?>> UpdateAsync(Customer request);

    /// <summary>
    /// Exclui um cliente existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por excluir um cliente existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="DeleteCustomerRequest"/> contendo o
Id do cliente a ser excluído.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Customer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Customer?>> DeleteAsync(DeleteCustomerRequest request);

    /// <summary>
    /// Obtém um cliente pelo ID.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar as informações de um cliente existente
com base no Id fornecido.
```

```
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetCustomerByIdRequest"/> contendo
o Id do cliente a ser recuperado.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Customer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Customer?>> GetByIdAsync(GetCustomerByIdRequest request);

    /// <summary>
    /// Obtém todos os clientes com paginação e filtro.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar todos os clientes existentes com base nos
filtros fornecidos.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllCustomersRequest"/> contendo
os filtros e paginação.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista de <see cref="Customer"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<PagedResponse<List<Customer>?>> GetAllAsync(GetAllCustomersRequest request);
}
```

.\RealtyHub.Core\Handlers\IOfferHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de propostas.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com as propostas, como criação, atualização, rejeição, aceitação e recuperação de
informações.
/// </remarks>
public interface IOfferHandler
{
    /// <summary>
    /// Cria uma nova propostas.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por criar uma nova proposta com base nas informações
fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Offer"/> contendo as informações da
nova proposta.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Offer?>> CreateAsync(Offer request);

    /// <summary>
    /// Atualiza as informações de uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar as informações de uma proposta existente
com base nas informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Offer"/> contendo as informações
atualizadas da proposta.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Offer?>> UpdateAsync(Offer request);

    /// <summary>
    /// Rejeita uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por rejeitar uma proposta existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="RejectOfferRequest"/> contendo o Id
da proposta a ser rejeitada.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Offer?>> RejectAsync(RejectOfferRequest request);

    /// <summary>
    /// Aceita uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por aceitar uma proposta existente com base no Id
```

```
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="AcceptOfferRequest"/> contendo o Id
da proposta a ser aceita.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Offer?>> AcceptAsync(AcceptOfferRequest request);

    /// <summary>
    /// Recupera as informações de uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar as informações de uma proposta existente
com base no Id fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetOfferByIdRequest"/> contendo o
Id da proposta a ser recuperada.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Offer?>> GetByIdAsync(GetOfferByIdRequest request);

    /// <summary>
    /// Recupera as informações de uma proposta existente com base no Id do imóvel.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar as informações de uma proposta existente
com base no Id do imóvel fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetOfferAcceptedByProperty"/>
contendo o Id do imóvel.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Offer?>> GetAcceptedByProperty(GetOfferAcceptedByProperty request);

    /// <summary>
    /// Recupera todas as propostas de um determinado imóvel.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar todas as propostas de um imóvel
específico com base nos parâmetros de paginação fornecidos.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllOffersByPropertyRequest"/>
contendo os parâmetros de paginação.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista de <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<PagedResponse<List<Offer>?>>
GetAllOffersByPropertyAsync(GetAllOffersByPropertyRequest request);

    /// <summary>
    /// Recupera todas as propostas de um determinado cliente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar todas as propostas de um cliente
específico com base nos parâmetros de paginação fornecidos.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllOffersByCustomerRequest"/>
contendo os parâmetros de paginação.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
```

```
    /// O objeto TData é uma lista de <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<PagedResponse<List<Offer>?>>
GetAllOffersByCustomerAsync(GetAllOffersByCustomerRequest request);

    /// <summary>
    /// Recupera todas as propostas existentes.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar todas as propostas existentes com base
nos parâmetros de paginação fornecidos.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllOffersRequest"/> contendo os
parâmetros de paginação.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista de <see cref="Offer"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<PagedResponse<List<Offer>?>> GetAllAsync(GetAllOffersRequest request);
}
```

.\RealtyHub.Core\Handlers\IPropertyHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de imóveis.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com os imóveis, como criação, atualização, exclusão e recuperação de informações.
/// </remarks>
public interface IPropertyHandler
{
    /// <summary>
    /// Cria um novo imóvel.
    ///  </summary>
    /// <remarks>
    /// Este método é responsável por criar um novo imóvel com base nas informações
fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Property"/> contendo as informações
do novo imóvel.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Property"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Property?>> CreateAsync(Property request);

    /// <summary>
    /// Atualiza as informações de um imóvel existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar as informações de um imóvel existente com
base nas informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Property"/> contendo as informações
atualizadas do imóvel.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Property"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Property?>> UpdateAsync(Property request);

    /// <summary>
    /// Exclui um imóvel existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por excluir um imóvel existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="DeletePropertyRequest"/> contendo o
Id do imóvel a ser excluído.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Property"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Property?>> DeleteAsync(DeletePropertyRequest request);

    /// <summary>
    /// Recupera um imóvel existente com base no Id fornecido.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por buscar um imóvel existente com base no Id
fornecido.
```

```
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetPropertyByIdRequest"/> contendo
o Id do imóvel a ser recuperado.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> contendo as informações do
imóvel.
    /// O objeto TData é do tipo <see cref="Property"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Property?>> GetByIdAsync(GetPropertyByIdRequest request);

    /// <summary>
    /// Recupera todos os imóveis disponíveis.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por buscar todos os imóveis armazenados no sistema e
retornar uma lista com os resultados.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllPropertiesRequest"/> contendo
os parâmetros de filtragem.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> contendo uma lista de
imóveis.
    /// O objeto TData é uma lista do tipo <see cref="Property"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<PagedResponse<List<Property>?>> GetAllAsync(GetAllPropertiesRequest request);

    /// <summary>
    /// Recupera todos os imóveis disponíveis com base no Id do usuário.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por buscar todos os imóveis armazenados no sistema e
retornar uma lista com os resultados.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllViewingsByPropertyRequest"/>
contendo os parâmetros de filtragem.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> contendo uma lista de
imóveis.
    /// O objeto TData é uma lista do tipo <see cref="Viewing"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<PagedResponse<List<Viewing>?>>
GetAllViewingsAsync(GetAllViewingsByPropertyRequest request);
}
```

.\RealtyHub.Core\Handlers\IPropertyPhotosHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de fotos de imóveis.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com as fotos de imóveis, como criação, atualização, exclusão e recuperação de
informações.
/// </remarks>
public interface IPropertyPhotosHandler
{
    /// <summary>
    /// Cria uma nova foto de imóvel.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por criar uma nova foto de imóvel com base nas
informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="CreatePropertyPhotosRequest"/>
contendo as informações da nova foto.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="PropertyPhoto"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<Response<PropertyPhoto?>> CreateAsync(CreatePropertyPhotosRequest request);

    /// <summary>
    /// Atualiza as informações de uma foto de imóvel existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por atualizar as informações de uma foto de imóvel
existente com base nas informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="UpdatePropertyPhotosRequest"/>
contendo as informações atualizadas da foto.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="PropertyPhoto"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<Response<List<PropertyPhoto>?>> UpdateAsync(UpdatePropertyPhotosRequest
request);

    /// <summary>
    /// Exclui uma foto de imóvel existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por excluir uma foto de imóvel existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="DeletePropertyPhotoRequest"/>
contendo o Id da foto a ser excluída.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="PropertyPhoto"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<Response<PropertyPhoto?>> DeleteAsync(DeletePropertyPhotoRequest request);

    /// <summary>
    /// Recupera uma foto de imóvel existente.
    /// </summary>
    /// <remarks>
```

```
    /// Este método é responsável por recuperar uma foto de imóvel existente com base no
Id fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see
cref="GetAllPropertyPhotosByPropertyRequest"/> contendo o Id da foto a ser
recuperada.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista do tipo <see cref="PropertyPhoto"/> e pode ser nulo se
a operação falhar.
    /// </returns>
    Task<Response<List<PropertyPhoto>?>> GetAllByPropertyAsync(
        GetAllPropertyPhotosByPropertyRequest request);
}
```

.\RealtyHub.Core\Handlers\IViewingHandler.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Handlers;

/// <summary>
/// Interface que define os métodos para manipulação de visitas a imóveis.
/// </summary>
/// <remarks>
/// Esta interface é responsável por definir as operações que podem ser realizadas
/// com as visitas de imóveis, como agendamento, reagendamento, conclusão e cancelamento.
/// </remarks>
public interface IViewingHandler
{
    /// <summary>
    /// Agenda uma nova visita.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por agendar uma nova visita a um imóvel com base nas
informações fornecidas.
    /// </remarks>
    Task<Response<Viewing?>> ScheduleAsync(Viewing request);

    /// <summary>
    /// Reagenda uma visita existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por reagendar uma visita existente com base nas
informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="Viewing"/> contendo as informações
da visita a ser reagendada.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Viewing"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Viewing?>> RescheduleAsync(Viewing request);

    /// <summary>
    /// Conclui uma visita existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por concluir uma visita existente com base nas
informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="DoneViewingRequest"/> contendo as
informações da visita a ser concluída.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Viewing"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Viewing?>> DoneAsync(DoneViewingRequest request);

    /// <summary>
    /// Cancela uma visita existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por cancelar uma visita existente com base nas
informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="CancelViewingRequest"/> contendo as
informações da visita a ser cancelada.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Viewing"/> e pode ser nulo se a operação
falhar.
```

```
    /// </returns>
    Task<Response<Viewing?>> CancelAsync(CancelViewingRequest request);

    /// <summary>
    /// Recupera uma visita existente.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar uma visita existente com base no Id
fornecido.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetViewingByIdRequest"/> contendo o
Id da visita a ser recuperada.</param>
    /// <returns>
    /// Retorna um objeto <see cref="Response{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é do tipo <see cref="Viewing"/> e pode ser nulo se a operação
falhar.
    /// </returns>
    Task<Response<Viewing?>> GetByIdAsync(GetViewingByIdRequest request);

    /// <summary>
    /// Recupera todas as visitas existentes.
    /// </summary>
    /// <remarks>
    /// Este método é responsável por recuperar todas as visitas existentes com base nas
informações fornecidas.
    /// </remarks>
    /// <param name="request">Instância de <see cref="GetAllViewingsRequest"/> contendo
as informações para recuperação das visitas.</param>
    /// <returns>
    /// Retorna um objeto <see cref="PagedResponse{TData}"/> indicando o resultado da
operação.
    /// O objeto TData é uma lista do tipo <see cref="Viewing"/> e pode ser nulo se a
operação falhar.
    /// </returns>
    Task<PagedResponse<List<Viewing>?>> GetAllAsync(GetAllViewingsRequest request);
}
```

.\RealtyHub.Core\Models\Account\ConfirmEmailRequest.cs

```
namespace RealtyHub.Core.Models.Account;

/// <summary>
/// Representa a requisição para confirmar o e-mail do usuário.
/// </summary>
public class ConfirmEmailRequest
{
    /// <summary>
    /// ID do usuário que está solicitando a confirmação de e-mail.
    /// </summary>
    /// <value>O ID do usuário.</value>
    public long UserId { get; set; }

    /// <summary>
    /// Token de confirmação de e-mail.
    /// </summary>
    /// <value>O token de confirmação.</value>
    public string Token { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Models\Account\PasswordResetModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models.Account;

/// <summary>
/// Representa o modelo de redefinição de senha do usuário.
/// </summary>
public class PasswordResetModel
{
    /// <summary>
    /// Senha do usuário.
    /// </summary>
    /// <value>A senha do usuário, mínimo de 6 caracteres</value>
    [Required(ErrorMessage = "Senha é obrigatória")]
    [MinLength(6, ErrorMessage = "Mínimo de 6 caracteres")]
    public string Password { get; set; } = string.Empty;

    /// <summary>
    /// Confirmação da senha do usuário.
    /// </summary>
    /// <value>A confirmação da senha do usuário, mínimo de 6 caracteres</value>
    public string ConfirmPassword { get; set; } = string.Empty;

    /// <summary>
    /// Verifica se a senha e a confirmação da senha são iguais.
    /// </summary>
    /// <value>Retorna true se as senhas forem iguais, caso contrário, false.</value>
    public bool IsEqual => Password == ConfirmPassword;

    /// <summary>
    /// Mensagem de erro caso as senhas não sejam iguais.
    /// </summary>
    /// <value>Mensagem de erro se as senhas não coincidirem.</value>
    public string Message => IsEqual ? string.Empty : "As senhas não coincidem";
}
```

.\RealtyHub.Core\Models\Account\RoleClaim.cs

```
namespace RealtyHub.Core.Models.Account;

/// <summary>
/// Representa uma declaração (claim) associada a uma role no ASP .NET Core Identity.
/// </summary>
/// <remarks>
/// Cada instância desta classe corresponde a um registro na tabela
<c>AspNetRoleClaims</c>
/// e reflete a estrutura de um <see cref="System.Security.Claims.Claim"/> usada para
autorização.
/// </remarks>
public class RoleClaim
{
    /// <summary>
    /// Obtém ou define o emissor desta claim.
    /// </summary>
    /// <value>
    /// Identificador da autoridade que emitiu a claim.
    /// </value>
    public string? Issuer { get; set; }

    /// <summary>
    /// Obtém ou define o emissor original desta claim.
    /// </summary>
    /// <value>
    /// Emissor inicial antes de qualquer reemissão ou transformação da claim.
    /// </value>
    public string? OriginalIssuer { get; set; }

    /// <summary>
    /// Obtém ou define o tipo desta claim.
    /// </summary>
    /// <value>
    /// Nome categórico da claim (por exemplo, <c>"permission"</c> ou <c>"role"</c>).
    /// </value>
    public string? Type { get; set; }

    /// <summary>
    /// Obtém ou define o valor desta claim.
    /// </summary>
    /// <value>
    /// Valor associado à claim (por exemplo, <c>"edit-articles"</c>).
    /// </value>
    public string? Value { get; set; }

    /// <summary>
    /// Obtém ou define o tipo de dado do valor desta claim.
    /// </summary>
    /// <value>
    /// Tipo de valor conforme <see cref="System.Security.Claims.ClaimValueTypes"/>,
    /// como <c>ClaimValueTypes.String</c>.
    /// </value>
    public string? ValueType { get; set; }
}
```

.\RealtyHub.Core\Models\Account\User.cs

```
namespace RealtyHub.Core.Models.Account;

/// <summary>
/// Representa um usuário no sistema.
/// </summary>
public class User
{
    /// <summary>
    /// E-mail do usuário.
    /// </summary>
    /// <value>O endereço de e-mail do usuário.</value>
    public string Email { get; set; } = string.Empty;

    /// <summary>
    /// Nome do usuário.
    /// </summary>
    /// <value>O nome completo do usuário.</value>
    public string GivenName { get; set; } = string.Empty;

    /// <summary>
    /// Número de registro profissional (Creci) do usuário.
    /// </summary>
    /// <value>O número de registro profissional do usuário.</value>
    public string Creci { get; set; } = string.Empty;

    /// <summary>
    /// Dicionário de claims do usuário.
    /// </summary>
    /// <remarks>
    /// As claims são pares chave-valor que representam informações adicionais
    /// sobre o usuário, como permissões e roles.
    /// </remarks>
    /// <value>
    /// Um dicionário onde a chave é o tipo da claim e o valor é o valor da claim.
    /// </value>
    public Dictionary<string, string> Claims { get; set; } = new();
}
```

.\RealtyHub.Core\Models\Address.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um endereço no sistema.
/// </summary>
public class Address
{
    /// <summary>
    /// Logradouro do endereço.
    /// </summary>
    /// <value>O logradouro do endereço.</value>
    [Required(ErrorMessage = "Logradouro é um campo obrigatório")]
    public string Street { get; set; } = string.Empty;

    /// <summary>
    /// Bairro do endereço.
    /// </summary>
    /// <value>O bairro do endereço.</value>
    [Required(ErrorMessage = "Bairro é um campo obrigatório")]
    public string Neighborhood { get; set; } = string.Empty;

    /// <summary>
    /// Número do endereço.
    /// </summary>
    /// <value>O número do endereço.</value>
    [Required(ErrorMessage = "Número é um campo obrigatório")]
    public string Number { get; set; } = string.Empty;

    /// <summary>
    /// Cidade do endereço.
    /// </summary>
    /// <value>O nome da cidade do endereço.</value>
    [Required(ErrorMessage = "Cidade é um campo obrigatório")]
    public string City { get; set; } = string.Empty;

    /// <summary>
    /// Estado do endereço.
    /// </summary>
    /// <value>O nome do estado do endereço.</value>
    [Required(ErrorMessage = "Estado é um campo obrigatório")]
    public string State { get; set; } = string.Empty;

    /// <summary>
    /// País do endereço.
    /// </summary>
    /// <value>O nome do país do endereço.</value>
    [Required(ErrorMessage = "País é um campo obrigatório")]
    public string Country { get; set; } = string.Empty;

    /// <summary>
    /// Código postal (CEP) do endereço.
    /// </summary>
    /// <value>O código postal do endereço.</value>
    [Required(ErrorMessage = "Cep é um campo obrigatório")]
    public string ZipCode { get; set; } = string.Empty;

    /// <summary>
    /// Complemento do endereço.
    /// </summary>
    /// <value>Informações adicionais sobre o endereço.</value>
    public string Complement { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Models\Condominium.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um condomínio no sistema.
/// </summary>
/// <remarks>
/// Essa classe herda de <c><see cref="Entity"/></c>
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Condominium : Entity
{
    /// <summary>
    /// Obtém ou define o ID do condomínio.
    /// </summary>
    /// <value>Um valor inteiro representando o ID.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define o nome do condomínio.
    /// </summary>
    /// <value>Uma string contendo o nome do condomínio.</value>
    [Required(ErrorMessage = "Nome é um campo obrigatório")]
    [MaxLength(120, ErrorMessage = "O Nome deve conter até 120 caracteres")]
    public string Name { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o endereço do condomínio.
    /// </summary>
    /// <value>Um objeto <see cref="Address"/> que representa o endereço.</value>
    [ValidateComplexType]
    public Address Address { get; set; } = new();

    /// <summary>
    /// Obtém ou define o número de unidades no condomínio.
    /// </summary>
    /// <value>Um valor inteiro representando o número de unidades.</value>
    public int Units { get; set; }

    /// <summary>
    /// Obtém ou define o número de andares do condomínio.
    /// </summary>
    /// <value>Um valor inteiro representando o número de andares.</value>
    public int Floors { get; set; }

    /// <summary>
    /// Indica se o condomínio possui elevador.
    /// </summary>
    /// <value><c>true</c> se possui elevador; caso contrário, <c>false</c>.</value>
    public bool HasElevator { get; set; }

    /// <summary>
    /// Indica se o condomínio possui piscina.
    /// </summary>
    /// <value><c>true</c> se possui piscina; caso contrário, <c>false</c>.</value>
    public bool HasSwimmingPool { get; set; }

    /// <summary>
    /// Indica se o condomínio possui salão de festas.
    /// </summary>
    /// <value><c>true</c> se possui salão de festas; caso contrário,
<c>false</c>.</value>
    public bool HasPartyRoom { get; set; }

    /// <summary>
    /// Indica se o condomínio possui playground.
    /// </summary>
    /// <value><c>true</c> se possui playground; caso contrário, <c>false</c>.</value>
    public bool HasPlayground { get; set; }

    /// <summary>
```

```
    /// Indica se o condomínio possui academia.
    /// </summary>
    /// <value><c>true</c> se possui academia; caso contrário, <c>false</c>.</value>
    public bool HasFitnessRoom { get; set; }

    /// <summary>
    /// Obtém ou define o valor do condomínio.
    /// </summary>
    /// <value>Um valor decimal representando o valor do condomínio.</value>
    public decimal CondominiumValue { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado ao condomínio.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Indica se o condomínio está ativo.
    /// </summary>
    /// <value><c>true</c> se está ativo; caso contrário, <c>false</c>.</value>
    public bool IsActive { get; set; }

    /// <summary>
    /// Obtém ou define a coleção de propriedades associadas ao condomínio.
    /// </summary>
    /// <value>Uma coleção de objetos <see cref="Property"/> que representam as
propriedades associadas.</value>
    public ICollection<Property> Properties { get; set; } = [];
}
```

.\RealtyHub.Core\Models\Contract.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um contrato no sistema.
/// </summary>
/// <remarks>
/// Essa classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Contract : Entity
{
    /// <summary>
    /// Obtém ou define o ID do contrato.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do contrato.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define o ID do vendedor associado ao contrato.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do vendedor.</value>
    public long SellerId { get; set; }

    /// <summary>
    /// Obtém ou define o ID do comprador associado ao contrato.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do comprador.</value>
    public long BuyerId { get; set; }

    /// <summary>
    /// Obtém ou define o ID da oferta associada ao contrato.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da oferta.</value>
    public long OfferId { get; set; }

    /// <summary>
    /// Obtém ou define a data de emissão do contrato.
    /// </summary>
    /// <value>Uma data representando quando o contrato foi emitido.</value>
    [Required(ErrorMessage = "A data de emissão é obrigatoria")]
    public DateTime? IssueDate { get; set; }

    /// <summary>
    /// Obtém ou define a data de vigência do contrato.
    /// </summary>
    /// <value>Uma data representando quando o contrato entra em vigor.</value>
    [Required(ErrorMessage = "A data de vigência é obrigatoria")]
    public DateTime? EffectiveDate { get; set; }

    /// <summary>
    /// Obtém ou define a data de término do contrato.
    /// </summary>
    /// <value>Uma data representando quando o contrato termina.</value>
    [Required(ErrorMessage = "A data de término é obrigatoria")]
    public DateTime? TermEndDate { get; set; }

    /// <summary>
    /// Obtém ou define a data de assinatura do contrato.
    /// </summary>
    /// <value>Uma data representando quando o contrato foi assinado.</value>
    public DateTime? SignatureDate { get; set; }

    /// <summary>
    /// Obtém ou define o ID do arquivo associado ao contrato.
    /// </summary>
    /// <value>Uma string representando o ID do arquivo.</value>
    public string FileId { get; set; } = string.Empty;

    /// <summary>
    /// Indica se o contrato está ativo.
```

```
    /// </summary>
    /// <value><c>true</c> se o contrato estiver ativo; caso contrário,
<c>false</c>.</value>
    public bool IsActive { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado ao contrato.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a oferta associada ao contrato.
    /// </summary>
    /// <value>Um objeto <see cref="Offer"/> representando a oferta. Pode ser
nulo.</value>
    public Offer? Offer { get; set; } = new();

    /// <summary>
    /// Obtém ou define o cliente vendedor associado ao contrato.
    /// </summary>
    /// <value>Um objeto <see cref="Customer"/> representando o vendedor. Pode ser
nulo.</value>
    public Customer? Seller { get; set; }

    /// <summary>
    /// Obtém ou define o cliente comprador associado ao contrato.
    /// </summary>
    /// <value>Um objeto <see cref="Customer"/> representando o comprador. Pode ser
nulo.</value>
    public Customer? Buyer { get; set; }

    /// <summary>
    /// Obtém o caminho completo para o arquivo PDF do contrato.
    /// </summary>
    /// <value>Uma string representando a URL completa onde o PDF do contrato está
disponível.</value>
    public string FilePath =>
        $"{Configuration.BackendUrl}/contracts/{FileId}.pdf";
}
```

.\RealtyHub.Core\Models\ContractTemplate.cs

```
using RealtyHub.Core.Enums;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um template de contrato no sistema.
/// </summary>
public class ContractTemplate
{
    /// <summary>
    /// Obtém ou define o identificador do template.
    /// </summary>
    /// <value>Uma string representando o ID do template.</value>
    public string Id { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a extensão do arquivo do template.
    /// </summary>
    /// <value>Uma string representando a extensão do arquivo (por exemplo,
".pdf").</value>
    public string Extension { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o nome do template.
    /// </summary>
    /// <value>Uma string contendo o nome do template.</value>
    public string Name { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o tipo do modelo de contrato.
    /// </summary>
    /// <value>Um valor do enum <see cref="EContractModelType"/> representando o tipo do
modelo.</value>
    public EContractModelType Type { get; set; }

    /// <summary>
    /// Indica se o template deve ser exibido na página.
    /// </summary>
    /// <value><c>true</c> se o template deve ser exibido; caso contrário,
<c>false</c>.</value>
    public bool ShowInPage { get; set; }

    /// <summary>
    /// Obtém o caminho completo para o arquivo do template de contrato.
    /// </summary>
    /// <value>
    /// Uma string representando a URL completa construída a partir do backend.
    /// </value>
    public string Path =>
        $"{Configuration.BackendUrl}/contracts-templates/{Id}{Extension}";
}
```

.\RealtyHub.Core\Models\Customer.cs

```
using RealtyHub.Core.Enums;
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um cliente no sistema.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Customer : Entity
{
    /// <summary>
    /// Obtém ou define o ID do cliente.
    /// </summary>
    /// <value>Um valor inteiro representando o ID.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define o nome do cliente.
    /// </summary>
    /// <value>Uma string contendo o nome do cliente.</value>
    [Required(ErrorMessage = "Nome é um campo obrigatório")]
    [MaxLength(80, ErrorMessage = "O nome deve conter até 80 caracteres")]
    public string Name { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o e-mail do cliente.
    /// </summary>
    /// <value>Uma string representando o e-mail do cliente.</value>
    [Required(ErrorMessage = "Email é um campo obrigatório")]
    [EmailAddress(ErrorMessage = "Email inválido")]
    [MaxLength(50, ErrorMessage = "O e-mail deve conter até 80 caracteres")]
    public string Email { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o telefone do cliente.
    /// </summary>
    /// <value>Uma string contendo o telefone do cliente.</value>
    [Required(ErrorMessage = "Telefone é um campo obrigatório")]
    [Phone(ErrorMessage = "Telefone inválido")]
    [MaxLength(30, ErrorMessage = "O telefone deve conter até 80 caracteres")]
    public string Phone { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o número do documento do cliente.
    /// </summary>
    /// <value>Uma string representando o número do documento.</value>
    [Required(ErrorMessage = "Documento é um campo obrigatório")]
    [MaxLength(20, ErrorMessage = "O documento deve conter até 20 caracteres")]
    public string DocumentNumber { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a ocupação do cliente.
    /// </summary>
    /// <value>Uma string contendo a ocupação.</value>
    public string Occupation { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a nacionalidade do cliente.
    /// </summary>
    /// <value>Uma string representando a nacionalidade.</value>
    public string Nationality { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o tipo de pessoa do cliente.
    /// </summary>
    /// <value>Um valor do enum <see cref="EPersonType"/> indicando o tipo de
pessoa.</value>
    public EPersonType PersonType { get; set; }
```

```
    /// <summary>
    /// Obtém ou define o tipo de cliente.
    /// </summary>
    /// <value>Um valor do enum <see cref="ECustomerType"/> indicando o tipo de cliente.
O padrão é <see cref="ECustomerType.Buyer"/>.</value>
    public ECustomerType CustomerType { get; set; } = ECustomerType.Buyer;

    /// <summary>
    /// Obtém ou define o endereço do cliente.
    /// </summary>
    /// <value>Um objeto <see cref="Address"/> representando o endereço. A validação
complexa é aplicada para validar suas propriedades.</value>
    [ValidateComplexType]
    public Address Address { get; set; } = new();

    /// <summary>
    /// Obtém ou define o RG do cliente.
    /// </summary>
    /// <value>Uma string representando o RG.</value>
    public string Rg { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a autoridade emissora do RG.
    /// </summary>
    /// <value>Uma string contendo a autoridade emissora.</value>
    public string IssuingAuthority { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a data de emissão do RG.
    /// </summary>
    /// <value>Uma data representando quando o RG foi emitido.</value>
    public DateTime? RgIssueDate { get; set; }

    /// <summary>
    /// Obtém ou define o nome empresarial, caso o cliente possua empresa associada.
    /// </summary>
    /// <value>Uma string representando o nome empresarial.</value>
    public string BusinessName { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o estado civil do cliente.
    /// </summary>
    /// <value>Um valor do enum <see cref="EMaritalStatus"/> representando o estado
civil. O padrão é <see cref="EMaritalStatus.Single"/>.</value>
    public EMaritalStatus MaritalStatus { get; set; } = EMaritalStatus.Single;

    /// <summary>
    /// Indica se o cliente está ativo.
    /// </summary>
    /// <value><c>true</c> se o cliente estiver ativo; caso contrário,
<c>false</c>.</value>
    public bool IsActive { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado ao cliente.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a lista de propriedades associadas ao cliente.
    /// </summary>
    /// <value>Uma lista de objetos <see cref="Property"/> representando as propriedades
associadas.</value>
    public List<Property> Properties { get; set; } = [];
}
```

.\RealtyHub.Core\Models\Entity.cs

```
namespace RealtyHub.Core.Models;

/// <summary>
/// Representa uma entidade base com propriedades de data de criação e atualização.
/// </summary>
/// <remarks>
/// Esta classe é utilizada como base para outras entidades no sistema,
/// fornecendo um padrão para o gerenciamento de datas de criação e atualização.
/// </remarks>
public class Entity
{

    /// <summary>
    /// Obtém ou define a data de criação da entidade.
    /// </summary>
    /// <value>
    /// Uma <c><see cref="DateTime"/></c> representando a data e hora em que a entidade
foi criada.
    /// </value>
    public DateTime CreatedAt { get; set; }

    /// <summary>
    /// Obtém ou define a data de atualização da entidade.
    /// </summary>
    /// <value>
    /// Uma <c><see cref="DateTime"/></c> representando a data e hora da última
atualização da entidade.
    /// </value>
    public DateTime UpdatedAt { get; set; }
}
```

.\RealtyHub.Core\Models\FileData.cs

```
namespace RealtyHub.Core.Models;

/// <summary>
/// Representa os dados de um arquivo, incluindo seu conteúdo e metadados.
/// </summary>
public class FileData
{
    /// <summary>
    /// Obtém ou define o identificador do arquivo.
    /// </summary>
    /// <value>Uma string representando o ID do arquivo.</value>
    public string Id { get; set; } = string.Empty;

    /// <summary>
    /// Indica se o arquivo é uma miniatura.
    /// </summary>
    /// <value><c>true</c> se o arquivo for uma miniatura; caso contrário,
<c>false</c>.</value>
    public bool IsThumbnail { get; set; }

    /// <summary>
    /// Obtém ou define o conteúdo binário do arquivo.
    /// </summary>
    /// <value>Um array de bytes representando o conteúdo do arquivo.</value>
    public byte[] Content { get; set; } = [];

    /// <summary>
    /// Obtém ou define o tipo de conteúdo (MIME type) do arquivo.
    /// </summary>
    /// <value>Uma string representando o tipo de conteúdo, por exemplo,
"image/png".</value>
    public string ContentType { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o nome do arquivo.
    /// </summary>
    /// <value>Uma string contendo o nome do arquivo.</value>
    public string Name { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Models\FilterOption.cs

```
namespace RealtyHub.Core.Models;

/// <summary>
/// Representa uma opção de filtro para exibição e mapeamento de propriedades.
/// </summary>
public class FilterOption
{
    /// <summary>
    /// Obtém ou define o nome que será exibido para esta opção de filtro.
    /// </summary>
    /// <value>Uma string contendo o nome a ser exibido.</value>
    public string DisplayName { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o nome da propriedade que será filtrada.
    /// </summary>
    /// <value>Uma string representando o nome da propriedade.</value>
    public string PropertyName { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Models\Offer.cs

```
using RealtyHub.Core.Enums;
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa uma proposta de compra de um imóvel no sistema.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Offer : Entity
{
    /// <summary>
    /// Obtém ou define o ID da proposta.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da proposta.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define o valor da proposta.
    /// </summary>
    /// <value>Um valor decimal representando o montante da proposta.</value>
    [Required(ErrorMessage = "O valor da proposta é obrigatório")]
    public decimal Amount { get; set; }

    /// <summary>
    /// Obtém ou define o ID do imóvel associado à proposta.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do imóvel.</value>
    [Required(ErrorMessage = "O imóvel é obrigatório")]
    public long PropertyId { get; set; }

    /// <summary>
    /// Obtém ou define o ID do cliente que está fazendo a proposta.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do cliente.</value>
    [Required(ErrorMessage = "O cliente é obrigatório")]
    public long BuyerId { get; set; }

    /// <summary>
    /// Obtém ou define a lista de pagamentos relacionados à proposta.
    /// </summary>
    /// <value>Uma lista de objetos <see cref="Payment"/> representando os
pagamentos.</value>
    public List<Payment> Payments { get; set; } = [];

    /// <summary>
    /// Obtém ou define a data de submissão da proposta.
    /// </summary>
    /// <value>Uma data representando quando a proposta foi submetida.</value>
    public DateTime? SubmissionDate { get; set; } = DateTime.UtcNow;

    /// <summary>
    /// Obtém ou define o status da proposta.
    /// </summary>
    /// <value>Um valor do enum <see cref="EOfferStatus"/> representando o status atual
da proposta.</value>
    public EOfferStatus OfferStatus { get; set; } = EOfferStatus.Analysis;

    /// <summary>
    /// Obtém ou define o ID do usuário associado à proposta.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o cliente associado à proposta.
    /// </summary>
    /// <value>Um objeto <see cref="Customer"/> representando o cliente. Pode ser
nulo.</value>
```

```
    public Customer? Buyer { get; set; }

    /// <summary>
    /// Obtém ou define o imóvel associado à proposta.
    /// </summary>
    /// <value>Um objeto <see cref="Property"/> representando o imóvel. Pode ser
nulo.</value>
    public Property? Property { get; set; }

    /// <summary>
    /// Obtém ou define o contrato associado à proposta.
    /// </summary>
    /// <value>Um objeto <see cref="Contract"/> representando o contrato. Pode ser
nulo.</value>
    public Contract? Contract { get; set; }
}
```

.\RealtyHub.Core\Models\Payment.cs

```
using RealtyHub.Core.Enums;
using System.ComponentModel.DataAnnotations;
using System.Globalization;
using System.Text;
using System.Text.Json.Serialization;
using RealtyHub.Core.Extensions;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um pagamento relacionado a uma proposta.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Payment : Entity
{
    /// <summary>
    /// Obtém ou define o ID do pagamento.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do pagamento.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define o valor do pagamento.
    /// </summary>
    /// <value>Um valor decimal representando o montante do pagamento.</value>
    [Required(ErrorMessage = "O valor do pagamento é obrigatório")]
    public decimal Amount { get; set; }

    /// <summary>
    /// Obtém ou define o tipo de pagamento.
    /// </summary>
    /// <value>Um valor do enum <see cref="EPaymentType"/> representando a forma de
pagamento.</value>
    [Required(ErrorMessage = "O tipo de pagamento é obrigatório")]
    public EPaymentType PaymentType { get; set; } = EPaymentType.Cash;

    /// <summary>
    /// Indica se o pagamento será realizado em parcelas.
    /// </summary>
    /// <value><c>true</c> se o pagamento for parcelado; caso contrário,
<c>false</c>.</value>
    public bool Installments { get; set; }

    /// <summary>
    /// Obtém ou define o número de parcelas.
    /// </summary>
    /// <value>
    /// Um valor inteiro representando o número de parcelas, que deverá estar entre 1 e
24.
    /// </value>
    [Range(1, 24)]
    public int InstallmentsCount { get; set; } = 1;

    /// <summary>
    /// Obtém ou define o ID da proposta associada ao pagamento.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da proposta.</value>
    public long OfferId { get; set; }

    /// <summary>
    /// Indica se o pagamento está ativo.
    /// </summary>
    /// <value><c>true</c> se o pagamento estiver ativo; caso contrário,
<c>false</c>.</value>
    public bool IsActive { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado ao pagamento.
    /// </summary>
```

```
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a proposta associada ao pagamento.
    /// </summary>
    /// <value>Um objeto <see cref="Offer"/> representando a proposta associada.</value>
    [JsonIgnore]
    public Offer Offer { get; set; } = null!;

    /// <summary>
    /// Converte os dados do pagamento para uma representação em string.
    /// </summary>
    /// <returns>
    /// Uma string contendo o valor do pagamento, a forma de pagamento e, se aplicável, o
número de parcelas.
    /// </returns>
    public override string ToString()
    {
        var text = new StringBuilder();
        text.Append($"Valor: {Amount.ToString(CultureInfo.CurrentCulture)}, ");
        text.Append($"Forma: {PaymentType.GetDisplayName()}, ");
        if (Installments)
            text.Append($"Parcelas: {InstallmentsCount}");
        return text.ToString();
    }
}
```

.\RealtyHub.Core\Models\PhotoItem.cs

```
namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um item de foto associado a uma propriedade.
/// </summary>
public class PhotoItem
{
    /// <summary>
    /// Obtém ou define o identificador da foto.
    /// </summary>
    /// <value>Uma string representando o ID da foto.</value>
    public string Id { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o identificador da propriedade associada a esta foto.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da propriedade.</value>
    public long PropertyId { get; set; }

    /// <summary>
    /// Indica se a foto é uma miniatura.
    /// </summary>
    /// <value><c>true</c> se for uma miniatura; caso contrário, <c>false</c>.</value>
    public bool IsThumbnail { get; set; }

    /// <summary>
    /// Obtém ou define a URL de exibição da foto.
    /// </summary>
    /// <value>Uma string representando a URL para exibição da foto.</value>
    public string DisplayUrl { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o conteúdo binário da foto.
    /// </summary>
    /// <value>Um array de bytes representando o conteúdo da foto. Pode ser nulo.</value>
    public byte[]? Content { get; set; } = [];

    /// <summary>
    /// Obtém ou define o tipo de conteúdo (MIME type) da foto.
    /// </summary>
    /// <value>Uma string representando o tipo de conteúdo, por exemplo, "image/jpeg".
Pode ser nulo.</value>
    public string? ContentType { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o nome original do arquivo da foto.
    /// </summary>
    /// <value>Uma string contendo o nome original do arquivo. Pode ser nulo.</value>
    public string? OriginalFileName { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Models\Property.cs

```
using RealtyHub.Core.Enums;
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um imóvel no sistema.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Property : Entity
{
    /// <summary>
    /// Obtém ou define o ID do imóvel.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do imóvel.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define o ID do vendedor associado ao imóvel.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do vendedor.</value>
    public long SellerId { get; set; }

    /// <summary>
    /// Obtém ou define o ID do condomínio associado ao imóvel.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do condomínio.</value>
    public long CondominiumId { get; set; }

    /// <summary>
    /// Obtém ou define o título do imóvel.
    /// </summary>
    /// <value>Uma string contendo o título do imóvel.</value>
    [Required(ErrorMessage = "Título é um campo obrigatório")]
    [MaxLength(120, ErrorMessage = "O Título deve conter até 120 caracteres")]
    public string Title { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a descrição do imóvel.
    /// </summary>
    /// <value>Uma string contendo a descrição do imóvel.</value>
    [Required(ErrorMessage = "Descrição é um campo obrigatório")]
    [MaxLength(255, ErrorMessage = "A descrição deve conter até 255 caracteres")]
    public string Description { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o preço do imóvel.
    /// </summary>
    /// <value>Um valor decimal representando o preço do imóvel.</value>
    [Required(ErrorMessage = "Preço é um campo obrigatório")]
    public decimal Price { get; set; }

    /// <summary>
    /// Obtém ou define o tipo do imóvel.
    /// </summary>
    /// <value>Um valor do enum <see cref="EPropertyType"/> representando o tipo do
imóvel. O padrão é <see cref="EPropertyType.Apartment"/>.</value>
    [Required(ErrorMessage = "Tipo de propriedade é um campo obrigatório")]
    public EPropertyType PropertyType { get; set; } = EPropertyType.Apartment;

    /// <summary>
    /// Obtém ou define a quantidade de quartos do imóvel.
    /// </summary>
    /// <value>Um valor inteiro representando o número de quartos.</value>
    [Required(ErrorMessage = "Quartos é um campo obrigatório")]
    public int Bedroom { get; set; }

    /// <summary>
    /// Obtém ou define a quantidade de banheiros do imóvel.
```

```
    /// </summary>
    /// <value>Um valor inteiro representando o número de banheiros.</value>
    [Required(ErrorMessage = "Banheiros é um campo obrigatório")]
    public int Bathroom { get; set; }

    /// <summary>
    /// Obtém ou define a quantidade de vagas de garagem do imóvel.
    /// </summary>
    /// <value>Um valor inteiro representando o número de vagas na garagem.</value>
    [Required(ErrorMessage = "Garagem é um campo obrigatório")]
    public int Garage { get; set; }

    /// <summary>
    /// Obtém ou define a área (em metros quadrados) do imóvel.
    /// </summary>
    /// <value>Um valor double representando a área do imóvel.</value>
    [Required(ErrorMessage = "Área é um campo obrigatório")]
    public double Area { get; set; }

    /// <summary>
    /// Obtém ou define os detalhes da transação relativos ao imóvel.
    /// </summary>
    /// <value>Uma string contendo informações sobre a transação do imóvel.</value>
    [Required(ErrorMessage = "Detalhes de transações é um campo obrigatório")]
    public string TransactionsDetails { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o endereço do imóvel.
    /// </summary>
    /// <value>Um objeto <see cref="Address"/> representando o endereço do
imóvel.</value>
    [ValidateComplexType]
    public Address Address { get; set; } = new();

    /// <summary>
    /// Indica se o imóvel é novo.
    /// </summary>
    /// <value><c>true</c> se o imóvel for novo; caso contrário, <c>false</c>.</value>
    [Required(ErrorMessage = "Propriedade nova é um campo obrigatório")]
    public bool IsNew { get; set; }

    /// <summary>
    /// Obtém ou define o número de matrícula no cartório.
    /// </summary>
    /// <value>Uma string representando o número de matrícula.</value>
    [Required(ErrorMessage = "Matriculá no Cartório é um campo obrigatório")]
    public string RegistryNumber { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o registro do imóvel no cartório.
    /// </summary>
    /// <value>Uma string representando o registro do cartório.</value>
    [Required(ErrorMessage = "Registro do Cartório é um campo obrigatório")]
    public string RegistryRecord { get; set; } = string.Empty;

    /// <summary>
    /// Indica se o imóvel está ativo.
    /// </summary>
    /// <value><c>true</c> se o imóvel estiver ativo; caso contrário,
<c>false</c>.</value>
    public bool IsActive { get; set; }

    /// <summary>
    /// Indica se o imóvel deverá ser exibido na página inicial.
    /// </summary>
    /// <value><c>true</c> se o imóvel deve ser exibido na home; caso contrário,
<c>false</c>.</value>
    public bool ShowInHome { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado ao imóvel.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
```

```
    /// Obtém ou define a lista de fotos associadas ao imóvel.
    /// </summary>
    /// <value>Uma lista de objetos <see cref="PropertyPhoto"/> que representam as fotos
do imóvel.</value>
    public List<PropertyPhoto> PropertyPhotos { get; set; } = [];

    /// <summary>
    /// Obtém ou define o vendedor associado ao imóvel.
    /// </summary>
    /// <value>Um objeto <see cref="Customer"/> representando o vendedor. Pode ser
nulo.</value>
    public Customer? Seller { get; set; }

    /// <summary>
    /// Obtém ou define o condomínio associado ao imóvel.
    /// </summary>
    /// <value>Um objeto <see cref="Condominium"/> representando o condomínio ao qual o
imóvel pertence.</value>
    public Condominium Condominium { get; set; } = new();
}
```

.\RealtyHub.Core\Models\PropertyPhoto.cs

```
using System.Text.Json.Serialization;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa uma foto associada a um imóvel.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class PropertyPhoto : Entity
{
    /// <summary>
    /// Obtém ou define o identificador único da foto.
    /// </summary>
    /// <value>Uma string representando o ID da foto.</value>
    public string Id { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a extensão do arquivo da foto.
    /// </summary>
    /// <value>Uma string representando a extensão, por exemplo, ".jpg" ou
".png".</value>
    public string Extension { get; set; } = string.Empty;

    /// <summary>
    /// Indica se a foto é uma miniatura.
    /// </summary>
    /// <value><c>true</c> se a foto for uma miniatura; caso contrário,
<c>false</c>.</value>
    public bool IsThumbnail { get; set; }

    /// <summary>
    /// Obtém ou define o ID do imóvel associada a esta foto.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da imóvel.</value>
    public long PropertyId { get; set; }

    /// <summary>
    /// Indica se a foto está ativa.
    /// </summary>
    /// <value><c>true</c> se a foto estiver ativa; caso contrário, <c>false</c>.</value>
    public bool IsActive { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado à foto.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o imóvel associada a esta foto.
    /// </summary>
    /// <value>Um objeto <see cref="Property"/> representando a imóvel à qual a foto
pertence.</value>
    [JsonIgnore]
    public Property Property { get; set; } = null!;

    /// <summary>
    /// Retorna uma representação em forma de string dos dados da foto.
    /// </summary>
    /// <returns>
    /// Uma string contendo o ID, a extensão, se é miniatura, o ID do imóvel, status
ativo e o ID do usuário.
    /// </returns>
    public override string ToString()
    {
        return $"Id: {Id}, " +
            $"Extension: {Extension}, " +
            $"IsThumbnail: {IsThumbnail}, " +
            $"PropertyId: {PropertyId}, " +
```

```
            $"IsActive: {IsActive}, " +
            $"UserId: {UserId}";
    }
}
```

.\RealtyHub.Core\Models\Report.cs

```
namespace RealtyHub.Core.Models;

/// <summary>
/// Representa um relatório no sistema.
/// </summary>
public class Report
{
    /// <summary>
    /// Obtém ou define a URL do relatório.
    /// </summary>
    /// <value>Uma string representando a URL onde o relatório pode ser acessado.</value>
    public string Url { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Models\Viewing.cs

```
using RealtyHub.Core.Enums;
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Models;

/// <summary>
/// Representa uma visita a um imóvel no sistema.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Entity"/></c>,
/// que contém propriedades comuns a todas as entidades do sistema.
/// </remarks>
public class Viewing : Entity
{
    /// <summary>
    /// Obtém ou define o ID da visita.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da visita.</value>
    public long Id { get; set; }

    /// <summary>
    /// Obtém ou define a data e hora da visita.
    /// </summary>
    /// <value>Uma data/hora representando quando a visita ocorrerá. O valor padrão é a
data e hora atual.</value>
    [Required(ErrorMessage = "Data da visita é um campo obrigatório")]
    [DataType(DataType.DateTime)]
    public DateTime? ViewingDate { get; set; } = DateTime.Now;

    /// <summary>
    /// Obtém ou define o status da visita.
    /// </summary>
    /// <value>Um valor do enum <see cref="EViewingStatus"/> representando o status da
visita. O padrão é <see cref="EViewingStatus.Scheduled"/>.</value>
    [Required(ErrorMessage = "Status da visita é um campo obrigatório")]
    public EViewingStatus ViewingStatus { get; set; } = EViewingStatus.Scheduled;

    /// <summary>
    /// Obtém ou define o ID do comprador associado à visita.
    /// </summary>
    /// <value>Um valor inteiro representando o ID do comprador.</value>
    [Required(ErrorMessage = "Cliente é um campo obrigatório")]
    public long BuyerId { get; set; }

    /// <summary>
    /// Obtém ou define o ID da imóvel que será visitada.
    /// </summary>
    /// <value>Um valor inteiro representando o ID da imóvel.</value>
    [Required(ErrorMessage = "Imóvel é um campo obrigatório")]
    public long PropertyId { get; set; }

    /// <summary>
    /// Obtém ou define o ID do usuário associado a esta visita.
    /// </summary>
    /// <value>Uma string representando o ID do usuário.</value>
    public string UserId { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a imóvel que será visitada.
    /// </summary>
    /// <value>Um objeto <see cref="Property"/> representando a imóvel. Pode ser
nulo.</value>
    public Property? Property { get; set; }

    /// <summary>
    /// Obtém ou define o cliente comprador associado à visita.
    /// </summary>
    /// <value>Um objeto <see cref="Customer"/> representando o comprador. Pode ser
nulo.</value>
    public Customer? Buyer { get; set; }
}
```

.\RealtyHub.Core\Requests\Account\ForgotPasswordRequest.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Requests.Account;

/// <summary>
/// Classe que representa uma requisição de recuperação de senha.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class ForgotPasswordRequest : Request
{
    /// <summary>
    /// Email do usuário que está solicitando a recuperação de senha.
    /// </summary>
    /// <value>O endereço de email do usuário.</value>
    [Required(ErrorMessage = "Email é obrigatório")]
    public string Email { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Account\LoginRequest.cs

```
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Core.Requests.Account;

/// <summary>
/// Classe que representa uma requisição de login de usuário.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class LoginRequest : Request
{
    /// <summary>
    /// Email do usuário que está tentando fazer login.
    /// </summary>
    /// <value>O endereço de email do usuário.</value>
    [Required(ErrorMessage = "E-mail é obrigatório")]
    [EmailAddress(ErrorMessage = "E-mail inválido")]
    public string Email { get; set; } = string.Empty;

    /// <summary>
    /// Senha do usuário que está tentando fazer login.
    /// </summary>
    /// <value> A senha do usuário.</value>
    [Required(ErrorMessage = "Senha inválida")]
    [MinLength(6, ErrorMessage = "A senha deve ter no mínimo 6 caracteres")]
    public string Password { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Account\RegisterRequest.cs

```
using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;

namespace RealtyHub.Core.Requests.Account;

/// <summary>
/// Classe que representa uma requisição de registro de usuário.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class RegisterRequest : Request
{
    /// <summary>
    /// Número do CRECI do usuário.
    /// </summary>
    /// <remarks>
    /// O número do CRECI é um registro profissional necessário para corretores de
imóveis.
    /// </remarks>
    /// <value>O número do CRECI do usuário.</value>
    [Required(ErrorMessage = "Creci é obrigatório")]
    public string Creci { get; set; } = string.Empty;

    /// <summary>
    /// Nome do usuário.
    /// </summary>
    /// <value>O nome completo do usuário.</value>
    [Required(ErrorMessage = "Nome é obrigatório")]
    public string GivenName { get; set; } = string.Empty;

    /// <summary>
    /// E-mail do usuário.
    /// </summary>
    /// <value>O endereço de e-mail do usuário.</value>
    [Required(ErrorMessage = "E-mail é obrigatório")]
    [EmailAddress(ErrorMessage = "E-mail inválido")]
    public string Email { get; set; } = string.Empty;

    /// <summary>
    /// Senha do usuário.
    /// </summary>
    /// <value>A senha do usuário.</value>
    [Required(ErrorMessage = "Senha inválida")]
    [MinLength(6, ErrorMessage = "A senha deve ter no mínimo 6 caracteres")]
    public string Password { get; set; } = string.Empty;

    /// <summary>
    /// Confirmação da senha do usuário.
    /// </summary>
    /// <value>A confirmação da senha do usuário.</value>
    [Required(ErrorMessage = "Senha inválida")]
    [MinLength(6, ErrorMessage = "A senha deve ter no mínimo 6 caracteres")]
    [JsonIgnore]
    public string ConfirmPassword { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Account\ResetPasswordRequest.cs

```
using System.ComponentModel.DataAnnotations;
using RealtyHub.Core.Models.Account;

namespace RealtyHub.Core.Requests.Account;

/// <summary>
/// Classe que representa uma requisição de redefinição de senha.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class ResetPasswordRequest : Request
{
    /// <summary>
    /// Token de redefinição de senha.
    /// </summary>
    /// <remarks>
    /// O token é usado para verificar a autenticidade da solicitação de redefinição de
senha.
    /// </remarks>
    /// <value>O token de redefinição de senha.</value>
    public string Token { get; set; } = string.Empty;

    /// <summary>
    /// E-mail do usuário.
    /// </summary>
    /// <value>O endereço de e-mail do usuário.</value>
    public string Email { get; set; } = string.Empty;

    /// <summary>
    /// Modelo de redefinição de senha.
    /// </summary>
    /// <value>O modelo de redefinição de senha.</value>
    [ValidateComplexType]
    public PasswordResetModel PasswordResetModel { get; set; } = new();
}
```

.\RealtyHub.Core\Requests\Condominiums\DeleteCondominiumRequest.cs

```
namespace RealtyHub.Core.Requests.Condominiums;

/// <summary>
/// Classe que representa uma requisição para excluir um condomínio.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class DeleteCondominiumRequest : Request
{
    /// <summary>
    /// Identificador do condomínio a ser excluído.
    /// </summary>
    /// <value>O Id do condomínio a ser excluído.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Condominiums\GetAllCondominiumsRequest.cs

```
namespace RealtyHub.Core.Requests.Condominiums;

/// <summary>
/// Classe que representa uma requisição para obter todos os condomínios com paginação.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades comuns
para requisições paginadas.
/// </remarks>
public class GetAllCondominiumsRequest : PagedRequest { }
```

.\RealtyHub.Core\Requests\Condominiums\GetCondominiumByIdRequest.cs

```
namespace RealtyHub.Core.Requests.Condominiums;

/// <summary>
/// Classe que representa uma requisição para obter um condomínio específico pelo Id.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetCondominiumByIdRequest : Request
{
    /// <summary>
    /// Identificador do condomínio a ser recuperado.
    /// </summary>
    /// <value>O Id do condomínio a ser recuperado.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Contracts\DeleteContractRequest.cs

```
namespace RealtyHub.Core.Requests.Contracts;

/// <summary>
/// Classe que representa uma requisição para excluir um contrato.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class DeleteContractRequest : Request
{

    /// <summary>
    /// Identificador do contrato a ser excluído.
    /// </summary>
    /// <value> O Id do contrato a ser excluído </value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Contracts\GetAllContractsRequest.cs

```
namespace RealtyHub.Core.Requests.Contracts;

/// <summary>
/// Classe que representa uma requisição para obter todos os contratos com paginação.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades comuns
para requisições paginadas.
/// </remarks>
public class GetAllContractsRequest : PagedRequest { }
```

.\RealtyHub.Core\Requests\Contracts\GetContractByIdRequest.cs

```
namespace RealtyHub.Core.Requests.Contracts;

/// <summary>
/// Classe que representa uma requisição para obter um contrato específico pelo seu Id.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetContractByIdRequest : Request
{
    /// <summary>
    /// Identificador do contrato a ser recuperado.
    /// </summary>
    /// <value>O Id do contrato a ser recuperado.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Customers\DeleteCustomerRequest.cs

```
namespace RealtyHub.Core.Requests.Customers;

/// <summary>
/// Classe que representa uma requisição para excluir um cliente.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class DeleteCustomerRequest : Request
{
    /// <summary>
    /// Identificador do cliente a ser excluído.
    /// </summary>
    /// <value>O Id do cliente a ser excluído.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Customers\GetAllCustomersRequest.cs

```
namespace RealtyHub.Core.Requests.Customers;

/// <summary>
/// Classe que representa uma solicitação para obter todos os clientes com paginação.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades comuns
para solicitações paginadas.
/// </remarks>
public class GetAllCustomersRequest : PagedRequest { }
```

.\RealtyHub.Core\Requests\Customers\GetCustomerByIdRequest.cs

```
namespace RealtyHub.Core.Requests.Customers;

/// <summary>
/// Classe que representa uma requisição para obter um cliente específico pelo seu Id.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetCustomerByIdRequest : Request
{
    /// <summary>
    /// Identificador do cliente a ser recuperado.
    /// </summary>
    /// <value>O Id do cliente a ser recuperado.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Emails\AttachmentMessage.cs

```
namespace RealtyHub.Core.Requests.Emails;

/// <summary>
/// Classe que representa uma mensagem de e-mail com um anexo específico, associada a um
contrato.
/// </summary>
/// <remarks>
/// A classe herda de <see cref="EmailMessage"/>, que contém propriedades comuns para
mensagens de e-mail.
/// </remarks>
public class AttachmentMessage : EmailMessage
{
    /// <summary>
    /// Identificador do contrato associado à mensagem de e-mail.
    /// </summary>
    /// <value>O Id do contrato associado à mensagem de e-mail.</value>
    public long ContractId { get; set; }
}
```

.\RealtyHub.Core\Requests\Emails\ConfirmEmailMessage.cs

```
namespace RealtyHub.Core.Requests.Emails;

/// <summary>
/// Classe que representa uma mensagem de e-mail para confirmação de conta
/// </summary>
/// <remarks>
/// A classe herda de <see cref="EmailMessage"/>, que contém propriedades comuns para
mensagens de e-mail.
/// </remarks>
public class ConfirmEmailMessage : EmailMessage
{
    /// <summary>
    /// Link de confirmação para o e-mail
    /// </summary>
    /// <value>O link de confirmação para o e-mail.</value>
    public string ConfirmationLink { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Emails\EmailMesage.cs

```
namespace RealtyHub.Core.Requests.Emails;

/// <summary>
/// Classe abstrata que representa uma mensagem de e-mail genérica.
/// </summary>
public abstract class EmailMessage
{
    /// <summary>
    /// Endereço de e-mail do destinatário.
    /// </summary>
    /// <value>O endereço de e-mail do destinatário.</value>
    public string EmailTo { get; set; } = string.Empty;

    /// <summary>
    /// Assunto da mensagem de e-mail.
    /// </summary>
    /// <value>O assunto da mensagem de e-mail.</value>
    public string Subject { get; set; } = string.Empty;

    /// <summary>
    /// Corpo da mensagem de e-mail.
    /// </summary>
    /// <value>O corpo da mensagem de e-mail.</value>
    public string Body { get; set; } = string.Empty;

    /// <summary>
    /// Caminho do anexo da mensagem de e-mail.
    /// </summary>
    /// <value>O caminho do anexo da mensagem de e-mail.</value>
    public string AttachmentPath { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Emails\ResetPasswordMessage.cs

```
namespace RealtyHub.Core.Requests.Emails;

/// <summary>
/// Classe que representa uma mensagem de e-mail para redefinição de senha
/// </summary>
/// <remarks>
/// A classe herda de <see cref="EmailMessage"/>, que contém propriedades comuns para
mensagens de e-mail.
/// </remarks>
public class ResetPasswordMessage : EmailMessage
{
    /// <summary>
    /// Link de redefinição de senha
    /// </summary>
    /// <value>O link de redefinição de senha.</value>
    public string ResetPasswordLink { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Offers\AcceptOfferRequest.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa um pedido para aceitar uma oferta
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class AcceptOfferRequest : Request
{
    /// <summary>
    /// Identificador da oferta a ser aceita
    /// </summary>
    /// <value>O Id da oferta a ser aceita.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Offers\GetAllOffersByCustomerRequest.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa uma requisição para obter todas as propostas de um cliente
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades
comuns para requisições paginadas.
/// </remarks>
public class GetAllOffersByCustomerRequest : PagedRequest
{
    /// <summary>
    /// Identificador do cliente cujas propostas serão obtidas
    /// </summary>
    /// <value>O Id do cliente.</value>
    public long CustomerId { get; set; }
}
```

.\RealtyHub.Core\Requests\Offers\GetAllOffersByPropertyRequest.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa uma requisição para obter todas as propostas de um imóvel
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades
comuns para requisições paginadas.
/// </remarks>
public class GetAllOffersByPropertyRequest : PagedRequest
{
    /// <summary>
    /// Identificador do imóvel cujas propostas serão obtidas
    /// </summary>
    /// <value>O Id do imóvel.</value>
    public long PropertyId { get; set; }
}
```

.\RealtyHub.Core\Requests\Offers\GetAllOffersRequest.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa uma requisição para obter todas as propostas
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades
comuns para requisições paginadas.
/// </remarks>
public class GetAllOffersRequest : PagedRequest { }
```

.\RealtyHub.Core\Requests\Offers\GetOfferAcceptedByProperty.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa uma requisição para obter todas as propostas aceitas de um
imóvel
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetOfferAcceptedByProperty : Request
{
    /// <summary>
    /// Identificador do imóvel cujas propostas aceitas serão obtidas
    /// </summary>
    /// <value>O Id do imóvel.</value>
    public long PropertyId { get; set; }
}
```

.\RealtyHub.Core\Requests\Offers\GetOfferByIdRequest.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa uma requisição para obter uma proposta específica
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetOfferByIdRequest : Request
{
    /// <summary>
    /// Identificador da proposta a ser obtida
    /// </summary>
    /// <value>O Id da proposta.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Offers\RejectOfferRequest.cs

```
namespace RealtyHub.Core.Requests.Offers;

/// <summary>
/// Classe que representa uma reqquisição para rejeitar uma proposta
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class RejectOfferRequest : Request
{
    /// <summary>
    /// Identificador da proposta a ser rejeitada
    /// </summary>
    /// <value>O Id da proposta.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\PagedRequest.cs

```
namespace RealtyHub.Core.Requests;

/// <summary>
/// Classe base para solicitações paginadas.
/// </summary>
/// <remarks>
/// <para>Esta classe é utilizada como base para todas as solicitações paginadas</para>
/// <para>Herda da classe <c><seealso cref="Request"/></c>.</para>
/// </remarks>
public abstract class PagedRequest : Request
{
    /// <summary>
    /// Número da página atual.
    /// </summary>
    /// <value>O número da página atual.</value>
    public int PageNumber { get; set; } = Configuration.DefaultPageNumber;

    /// <summary>
    /// Tamanho da página.
    /// </summary>
    /// <value>O tamanho da página.</value>
    public int PageSize { get; set; } = Configuration.DefaultPageSize;

    /// <summary>
    /// Termo de pesquisa.
    /// </summary>
    /// <value>O termo de pesquisa.</value>
    public string SearchTerm { get; set; } = string.Empty;

    /// <summary>
    /// Campo de filtro onde o termo de pesquisa será aplicado.
    /// </summary>
    /// <value>O campo de filtro.</value>
    public string FilterBy { get; set; } = string.Empty;

    /// <summary>
    /// Data de início do filtro.
    /// </summary>
    /// <value>A data de início do filtro.</value>
    public string? StartDate { get; set; }

    /// <summary>
    /// Data de término do filtro.
    /// </summary>
    /// <value>A data de término do filtro.</value>
    public string? EndDate { get; set; }
}
```

.\RealtyHub.Core\Requests\PropertiesPhotos\CreatePropertyPhotosRequest.cs

```
using Microsoft.AspNetCore.Http;
using RealtyHub.Core.Models;

namespace RealtyHub.Core.Requests.PropertiesPhotos;

/// <summary>
/// Representa uma requisição para criar fotos de um imóvel.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class CreatePropertyPhotosRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único do imóvel.
    /// </summary>
    /// <value>O Id do imóvel cujas fotos serão criadas.</value>
    public long PropertyId { get; set; }

    /// <summary>
    /// Obtém ou define a requisição HTTP associada.
    /// </summary>
    /// <value>Instância do <c><see cref="HttpRequest"/></c> que contém os detalhes da
requisição HTTP, podendo ser nula.</value>
    public HttpRequest? HttpRequest { get; set; }

    /// <summary>
    /// Obtém ou define a lista de arquivos em formato de bytes.
    /// </summary>
    /// <value>Uma lista de <c><see cref="FileData"/></c> representando os arquivos
enviados, ou nula se nenhum arquivo for enviado.</value>
    public List<FileData>? FileBytes { get; set; }
}
```

.\RealtyHub.Core\Requests\PropertiesPhotos\DeletePropertyPhotoRequest.cs

```
namespace RealtyHub.Core.Requests.PropertiesPhotos;

/// <summary>
/// Representa uma requisição para excluir uma foto de um imóvel.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class DeletePropertyPhotoRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único da foto.
    /// </summary>
    /// <value>Uma string representando o identificador da foto a ser excluída.</value>
    public string Id { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o identificador único do imóvel associado à foto.
    /// </summary>
    /// <value>O identificador do imóvel ao qual a foto pertence.</value>
    public long PropertyId { get; set; }
}
```

.\RealtyHub.Core\Requests\PropertiesPhotos\GetAllPropertyPhotosByPropertyRequest.cs

```
namespace RealtyHub.Core.Requests.PropertiesPhotos;

/// <summary>
/// Representa uma requisição para obter todas as fotos de um imóvel.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetAllPropertyPhotosByPropertyRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único do imóvel.
    /// </summary>
    /// <value>O identificador do imóvel cujas fotos serão retornadas.</value>
    public long PropertyId { get; set; }
}
```

.\RealtyHub.Core\Requests\PropertiesPhotos\UpdatePorpertyPhotosRequest.cs

```
using RealtyHub.Core.Models;

namespace RealtyHub.Core.Requests.PropertiesPhotos;

/// <summary>
/// Representa uma requisição para atualizar as fotos de um imóvel.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class UpdatePropertyPhotosRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único do imóvel.
    /// </summary>
    /// <value>O identificador do imóvel cujas fotos serão atualizadas.</value>
    public long PropertyId { get; set; }

    /// <summary>
    /// Obtém ou define a lista de fotos do imóvel.
    /// </summary>
    /// <value>
    /// Uma lista de <c><see cref="PropertyPhoto"/></c> representando as fotos que serão
atualizadas.
    /// </value>
    public List<PropertyPhoto> Photos { get; set; } = [];

    /// <summary>
    /// Retorna uma representação em string dos detalhes da requisição.
    /// </summary>
    /// <returns>
    /// Uma string contendo o identificador do imóvel e os detalhes das fotos.
    /// </returns>
    public override string ToString()
    {
        var property = $"PropertyId: {PropertyId}";
        var photos = string.Join(", ", Photos.Select(p => p.ToString()));
        return string.Join(", ", photos, property);
    }
}
```

.\RealtyHub.Core\Requests\Properties\DeletePropertyRequest.cs

```
namespace RealtyHub.Core.Requests.Properties;

/// <summary>
/// Representa uma requisição para excluir um imóvel.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class DeletePropertyRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único do imóvel.
    /// </summary>
    /// <value> O Id do imóvel a ser excluído. </value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Properties\GetAllPropertiesRequest.cs

```
namespace RealtyHub.Core.Requests.Properties;

/// <summary>
/// Representa uma requisição para obter uma lista paginada de imóveis.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades
comuns para requisições paginadas.
/// </remarks>
public class GetAllPropertiesRequest : PagedRequest { }
```

.\RealtyHub.Core\Requests\Properties\GetAllViewingsByPropertyRequest.cs

```
namespace RealtyHub.Core.Requests.Properties;

/// <summary>
/// Representa uma requisição para obter todas as visitas de um imóvel.
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades
comuns para requisições paginadas.
/// </remarks>
public class GetAllViewingsByPropertyRequest : PagedRequest
{
    /// <summary>
    /// Obtém ou define o identificador único do imóvel.
    /// </summary>
    /// <value> O Id do imóvel cujas visitas serão obtidas. </value>
    public long PropertyId { get; set; }
}
```

.\RealtyHub.Core\Requests\Properties\GetPropertyByIdRequest.cs

```
namespace RealtyHub.Core.Requests.Properties;

/// <summary>
/// Classe que representa uma requisição para obter todas as visitas de um imóvel
/// </summary>
/// <remarks>
/// Esta classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns
para requisições.
/// </remarks>
public class GetPropertyByIdRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único do imóvel.
    /// </summary>
    /// <value> O Id do imóvel a ser obtido. </value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Request.cs

```
using System.Text.Json.Serialization;

namespace RealtyHub.Core.Requests;

/// <summary>
/// Classe base para todas as requisições.
/// </summary>
public abstract class Request
{
    /// <summary>
    /// Identificador do usuário que está fazendo a requisição.
    /// </summary>
    /// <remarks>
    /// Essa propriedade é usada para fins de auditoria e controle de acesso.
    /// </remarks>
    /// <value>O ID do usuário que está fazendo a requisição.</value>
    [JsonIgnore]
    public string UserId { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Requests\Viewings\CancelViewingRequest.cs

```
namespace RealtyHub.Core.Requests.Viewings;

/// <summary>
/// Representa uma requisição para cancelar uma visita.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class CancelViewingRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único da visita.
    /// </summary>
    /// <value>O identificador da visita a ser cancelada.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Viewings\DoneViewingRequest.cs

```
namespace RealtyHub.Core.Requests.Viewings;

/// <summary>
/// Representa uma requisição para marcar uma visita como concluída.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class DoneViewingRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único da visita.
    /// </summary>
    /// <value>O identificador da visita a ser marcada como concluída.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Requests\Viewings\GetAllViewingsRequest.cs

```
namespace RealtyHub.Core.Requests.Viewings;

/// <summary>
/// Representa uma requisição para obter uma lista paginada de visitas.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="PagedRequest"/></c>, que contém propriedades comuns
para requisições paginadas.
/// </remarks>
public class GetAllViewingsRequest : PagedRequest { }
```

.\RealtyHub.Core\Requests\Viewings\GetViewingByIdRequest.cs

```
namespace RealtyHub.Core.Requests.Viewings;

/// <summary>
/// Representa uma requisição para obter uma visita a partir do seu identificador.
/// </summary>
/// <remarks>
/// A classe herda de <c><see cref="Request"/></c>, que contém propriedades comuns para
requisições.
/// </remarks>
public class GetViewingByIdRequest : Request
{
    /// <summary>
    /// Obtém ou define o identificador único da visita.
    /// </summary>
    /// <value>O identificador da visita que será retornada.</value>
    public long Id { get; set; }
}
```

.\RealtyHub.Core\Responses\PagedResponse.cs

```
using System.Text.Json.Serialization;

namespace RealtyHub.Core.Responses;

/// <summary>
/// Classe genérica para encapsular a resposta de uma operação paginada.
/// </summary>
/// <typeparam name="TData">Tipo de dados a serem retornados na resposta.</typeparam>
public class PagedResponse<TData> : Response<TData>
{
    /// <summary>
    /// Construtor padrão da classe <see cref="PagedResponse{TData}"/>.
    /// </summary>
    /// <param name="data">Dados a serem retornados na resposta.</param>
    /// <param name="totalCount">Total de itens disponíveis.</param>
    /// <param name="currentPage">Número da página atual.</param>
    /// <param name="pageSize">Tamanho da página.</param>
    /// <remarks>
    /// Este construtor inicializa a resposta paginada com os dados, total de itens,
    /// número da página atual e tamanho da página.
    /// <para> O valor padrão de <c><paramref name="currentPage"/></c> e de <c><paramref
name="pageSize"/></c> é
    /// definido na classe <c><see cref="Configuration"/></c>.</para>
    /// </remarks>
    [JsonConstructor]
    public PagedResponse(TData? data, int totalCount,
        int currentPage = Configuration.DefaultPageNumber,
        int pageSize = Configuration.DefaultPageSize) : base(data)
    {
        Data = data;
        TotalCount = totalCount;
        CurrentPage = currentPage;
        PageSize = pageSize;
    }

    /// <summary>
    /// Construtor da classe <see cref="PagedResponse{TData}"/> com dados, código de
status e mensagem.
    /// </summary>
    /// <remarks>
    /// Este construtor permite inicializar a resposta com dados específicos,
    /// um código de status e uma mensagem personalizada.
    /// </remarks>
    /// <param name="data">Dados a serem retornados na resposta.</param>
    /// <param name="code">Código de status HTTP da resposta.</param>
    /// <param name="message">Mensagem adicional sobre o resultado da operação.</param>
    public PagedResponse(TData? data, int code = Configuration.DefaultStatusCode,
        string? message = null) : base(data, code, message) { }

    /// <summary>
    /// Número da página atual.
    /// </summary>
    /// <value>Valor em inteiro da página atual</value>
    public int CurrentPage { get; set; }

    /// <summary>
    /// Obtém ou define o número total de páginas disponíveis.
    /// </summary>
    /// <remarks> O campo é calculado dividindo
    /// o <c><see cref="TotalCount"/></c> por <c><see cref="PageSize"/></c> .
    /// </remarks>
    /// <value>Valor em inteiro do número total de páginas</value>
    public int TotalPages => (int)Math.Ceiling(TotalCount / (double)PageSize);

    /// <summary>
    /// Obtém ou define o tamanho da página.
    /// </summary>
    /// <remarks> O valor padrão é 25.</remarks>
    /// <value>Valor em inteiro do tamanho da página</value>
    public int PageSize { get; set; } = Configuration.DefaultPageSize;

    /// <summary>
```

```
    /// Obtém ou define o número total de itens disponíveis.
    /// </summary>
    /// <value>Valor em inteiro do número total de itens</value>
    public int TotalCount { get; set; }
}
```

.\RealtyHub.Core\Responses\Response.cs

```
using System.Text.Json.Serialization;

namespace RealtyHub.Core.Responses;

/// <summary>
/// Classe genérica para encapsular a resposta de uma operação.
/// </summary>
/// <typeparam name="TData">Tipo de dados a serem retornados na resposta.</typeparam>
public class Response<TData>
{
    /// <summary>
    /// Código de status HTTP da resposta.
    /// </summary>
    private readonly int _code;

    /// <summary>
    /// Construtor padrão da classe <see cref="Response{TData}"/>.
    /// </summary>
    /// <remarks>
    /// Este construtor inicializa o código de status com o valor padrão definido na
configuração.
    /// </remarks>
    [JsonConstructor]
    public Response() => _code = Configuration.DefaultStatusCode;

    /// <summary>
    /// Construtor da classe <see cref="Response{TData}"/> com dados, código de status e
mensagem.
    /// </summary>
    /// <remarks>
    /// Este construtor permite inicializar a resposta com dados específicos,
    /// um código de status e uma mensagem personalizada.
    /// </remarks>
    /// <param name="data">Dados a serem retornados na resposta.</param>
    /// <param name="code">Código de status HTTP da resposta.</param>
    /// <param name="message">Mensagem adicional sobre o resultado da operação.</param>

    public Response(TData? data, int code = Configuration.DefaultStatusCode, string?
message = null)
    {
        Data = data;
        Message = message;
        _code = code;
    }
    /// <summary>
    /// Dados a serem retornados na resposta.
    /// </summary>
    /// <remarks>
    /// Este campo pode conter qualquer tipo de dado, dependendo do contexto da operação.
    /// </remarks>
    /// <value>Um objeto do tipo <typeparamref name="TData"/> representando os dados da
resposta.</value>
    public TData? Data { get; set; }

    /// <summary>
    /// Mensagem adicional sobre o resultado da operação.
    /// </summary>
    /// <remarks>
    /// Esta mensagem pode conter informações sobre erros, avisos ou sucesso da operação.
    /// </remarks>
    /// <value>Uma string contendo a mensagem.</value>
    public string? Message { get; set; }

    /// <summary>
    /// Campo que indica se a operação foi bem-sucedida.
    /// </summary>
    /// <remarks>
    /// Este campo é calculado com base no código de status HTTP.
    /// </remarks>
    /// <value> <c>true</c> se o código de status estiver entre 200 e 299; caso
contrário, <c>false</c>.</value>
    [JsonIgnore]
```

```
    public bool IsSuccess => _code is >= 200 and <= 299;
}
```

.\RealtyHub.Core\Responses\UserResponse.cs

```
namespace RealtyHub.Core.Responses;

/// <summary>
/// Representa a resposta do usuário no sistema.
/// </summary>
public class UserResponse
{
    /// <summary>
    /// Obtém ou define o nome do usuário.
    /// </summary>
    /// <value>Uma string contendo o nome do usuário.</value>
    public string GivenName { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o CRECI do usuário.
    /// </summary>
    /// <value>Uma string representando o CRECI do usuário.</value>
    public string Creci { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o e-mail do usuário.
    /// </summary>
    /// <value>Uma string contendo o e-mail do usuário.</value>
    public string Email { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o nome de usuário.
    /// </summary>
    /// <value>Uma string contendo o nome de usuário.</value>
    public string UserName { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define as claims associadas ao usuário.
    /// </summary>
    /// <value>Um dicionário que mapeia os nomes das claims aos seus respectivos
valores.</value>
    public Dictionary<string, string> Claims { get; set; } = new();
}
```

.\RealtyHub.Core\Responses\ViaCepResponse.cs

```
using System.Text.Json.Serialization;

namespace RealtyHub.Core.Responses;

/// <summary>
/// Representa a resposta da API ViaCep contendo informações do endereço.
/// </summary>
public class ViaCepResponse
{
    /// <summary>
    /// Obtém ou define o logradouro.
    /// </summary>
    /// <value>Uma string contendo o logradouro.</value>
    [JsonPropertyName("logradouro")]
    public string Street { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o complemento do endereço.
    /// </summary>
    /// <value>Uma string contendo o complemento, se houver.</value>
    [JsonPropertyName("complemento")]
    public string Complement { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o bairro.
    /// </summary>
    /// <value>Uma string contendo o bairro.</value>
    [JsonPropertyName("bairro")]
    public string Neighborhood { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define a cidade.
    /// </summary>
    /// <value>Uma string contendo a cidade.</value>
    [JsonPropertyName("localidade")]
    public string City { get; set; } = string.Empty;

    /// <summary>
    /// Obtém ou define o estado.
    /// </summary>
    /// <value>Uma string contendo o estado.</value>
    [JsonPropertyName("estado")]
    public string State { get; set; } = string.Empty;
}
```

.\RealtyHub.Core\Services\IEmailService.cs

```
using RealtyHub.Core.Requests.Emails;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Services;

/// <summary>
/// Interface que define os métodos para o serviço de envio de e-mails.
/// </summary>
public interface IEmailService
{
    /// <summary>
    /// Envia um link de confirmação para o e-mail do usuário.
    /// </summary>
    /// <param name="message">A mensagem contendo os detalhes do link de
confirmação.</param>
    /// <returns>Um objeto Response indicando se o envio foi realizado com
sucesso.</returns>
    Task<Response<bool>> SendConfirmationLinkAsync(ConfirmEmailMessage message);

    /// <summary>
    /// Envia um link para redefinição de senha para o e-mail do usuário.
    /// </summary>
    /// <param name="message">A mensagem contendo os detalhes do link de redefinição de
senha.</param>
    /// <returns>Um objeto Response indicando se o envio foi realizado com
sucesso.</returns>
    Task<Response<bool>> SendResetPasswordLinkAsync(ResetPasswordMessage message);

    /// <summary>
    /// Envia um contrato como anexo para o e-mail do usuário.
    /// </summary>
    /// <param name="message">A mensagem contendo os detalhes do contrato a ser
enviado.</param>
    /// <returns>Um objeto Response indicando se o envio foi realizado com
sucesso.</returns>
    Task<Response<bool>> SendContractAsync(AttachmentMessage message);
}
```

.\RealtyHub.Core\Services\ISearchCep.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.Core.Services;

/// <summary>
/// Define os métodos para buscar informações de endereço via CEP utilizando a API
ViaCep.
/// </summary>
public interface IViaCepService
{
    /// <summary>
    /// Busca o endereço associado ao CEP especificado utilizando a API ViaCep.
    /// </summary>
    /// <param name="cep">O CEP a ser pesquisado.</param>
    /// <returns>
    /// Um objeto <c><see cref="Response{T}"/></c> contendo um
    /// <c><see cref="ViaCepResponse"/></c> com as informações do endereço, se
encontrado.
    /// </returns>
    Task<Response<ViaCepResponse?>> SearchAddressAsync(string cep);

    /// <summary>
    /// Obtém o endereço a partir do CEP especificado, retornando um objeto do tipo
<c><see cref="Address"/></c>.
    /// </summary>
    /// <param name="cep">O CEP a ser pesquisado.</param>
    /// <returns>
    /// Um objeto <c><see cref="Response{T}"/></c> contendo um <c><see
cref="Address"/></c> com
    /// as informações do endereço, se encontrado.
    /// </returns>
    Task<Response<Address?>> GetAddressAsync(string cep);
}
```

.\RealtyHub.Core\Utilities\FakeEntities\AddressFake.cs

```
using Bogus;
using RealtyHub.Core.Models;

namespace RealtyHub.Core.Utilities.FakeEntities;

/// <summary>
/// Fornece métodos para gerar dados falsos para a entidade <c><see cref="Address"/></c>.
/// </summary>
public class AddressFake
{
    /// <summary>
    /// Gera um endereço falso para a entidade <c><see cref="Address"/></c> utilizando a
biblioteca Bogus.
    /// </summary>
    /// <returns>Um objeto <c><see cref="Address"/></c> com dados gerados
aleatoriamente.</returns>
    public static Address GetFakeAddress()
    {
        return new Faker<Address>(Configuration.Locale)
            .RuleFor(a => a.Street, a => a.Address.StreetName())
            .RuleFor(a => a.Neighborhood, a => a.Address.SecondaryAddress())
            .RuleFor(a => a.Number, a => a.Address.BuildingNumber())
            .RuleFor(a => a.City, a => a.Address.City())
            .RuleFor(a => a.State, a => a.Address.State())
            .RuleFor(a => a.Country, a => a.Address.Country())
            .RuleFor(a => a.ZipCode, a => a.Address.ZipCode("#####-###"))
            .RuleFor(a => a.Complement, a => a.Address.SecondaryAddress());
    }
}
```

.\RealtyHub.Core\Utilities\FakeEntities\CustomerFake.cs

```
using Bogus;
using Bogus.Extensions.Brazil;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Models;

namespace RealtyHub.Core.Utilities.FakeEntities;

/// <summary>
/// Fornece métodos para gerar dados falsos para a entidade <c><see
cref="Customer"/></c>.
/// </summary>
public class CustomerFake
{
    /// <summary>
    /// Gera uma lista de clientes empresariais falsos.
    /// </summary>
    /// <param name="quantity">A quantidade de clientes a serem gerados.</param>
    /// <returns>Uma lista de objetos <c><see cref="Customer"/></c> com dados gerados
aleatoriamente para clientes empresariais.</returns>
    public static List<Customer> GetFakeBusinessCustomers(int quantity)
    {
        var businessCustomerFake = new Faker<Customer>(Configuration.Locale)
            .RuleFor(c => c.Name, c => c.Company.CompanyName())
            .RuleFor(c => c.Email, c => c.Person.Email)
            .RuleFor(c => c.Phone, c => c.Phone.PhoneNumber("###########"))
            .RuleFor(c => c.DocumentNumber, c => c.Company.Cnpj(false))
            .RuleFor(c => c.PersonType, EPersonType.Business)
            .RuleFor(c => c.CustomerType, ECustomerType.Seller)
            .RuleFor(c => c.Address, AddressFake.GetFakeAddress)
            .RuleFor(c => c.BusinessName, c => c.Company.CompanyName())
            .RuleFor(c => c.IsActive, true);

        return businessCustomerFake.Generate(quantity);
    }

    /// <summary>
    /// Gera uma lista de clientes individuais falsos.
    /// </summary>
    /// <param name="quantity">A quantidade de clientes a serem gerados.</param>
    /// <returns>Uma lista de objetos <c><see cref="Customer"/></c> com dados gerados
aleatoriamente para clientes individuais.</returns>
    public static List<Customer> GetFakeIndividualCustomers(int quantity)
    {
        var individualCustomerFake = new Faker<Customer>(Configuration.Locale)
            .RuleFor(c => c.Name, c => c.Person.FullName)
            .RuleFor(c => c.Email, c => c.Person.Email)
            .RuleFor(c => c.Phone, c => c.Phone.PhoneNumber("###########"))
            .RuleFor(c => c.DocumentNumber, c => c.Person.Cpf(false))
            .RuleFor(c => c.PersonType, EPersonType.Individual)
            .RuleFor(c => c.CustomerType, ECustomerType.Seller)
            .RuleFor(c => c.Address, AddressFake.GetFakeAddress)
            .RuleFor(c => c.Rg, c => c.Person.Cpf(false))
            .RuleFor(c => c.IssuingAuthority, c => c.Address.City())
            .RuleFor(c => c.RgIssueDate, c => c.Date.Past().ToUniversalTime())
            .RuleFor(c => c.MaritalStatus, c => c.Random.Enum<EMaritalStatus>())
            .RuleFor(c => c.Occupation, c => c.Company.CompanyName())
            .RuleFor(c => c.Nationality, c => c.Address.Country())
            .RuleFor(c => c.IsActive, true);

        return individualCustomerFake.Generate(quantity);
    }
}
```

.\RealtyHub.Core\Utilities\FakeEntities\PropertyFake.cs

```
using Bogus;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Models;

namespace RealtyHub.Core.Utilities.FakeEntities;

/// <summary>
/// Fornece métodos para gerar dados falsos para a entidade <c><see
cref="Property"/></c>.
/// </summary>
public class PropertyFake
{
    /// <summary>
    /// Gera uma lista de imóveis falsas.
    /// </summary>
    /// <param name="quantity">A quantidade de imóveis a serem geradas.</param>
    /// <param name="customerId">O ID do cliente (vendedor) associado a cada
propriedade.</param>
    /// <param name="condominiumId">O ID do condomínio associado a cada
propriedade.</param>
    /// <returns>Uma lista de objetos <c><see cref="Property"/></c> com dados gerados
aleatoriamente.</returns>
    public static List<Property> GetFakeProperties(int quantity, int customerId, int
condominiumId)
    {
        var propertyFake = new Faker<Property>(Configuration.Locale)
            .RuleFor(p => p.Title, p => p.Commerce.ProductName())
            .RuleFor(p => p.Description, p => p.Commerce.ProductDescription())
            .RuleFor(p => p.Price, p => decimal.Parse(p.Commerce.Price()))
            .RuleFor(p => p.PropertyType, p => p.PickRandom<EPropertyType>())
            .RuleFor(p => p.Bedroom, p => p.Random.Int(1, 5))
            .RuleFor(p => p.Bathroom, p => p.Random.Int(1, 5))
            .RuleFor(p => p.Garage, p => p.Random.Int(1, 5))
            .RuleFor(p => p.Area, p => double.Parse(p.Random.Double(50,
500).ToString("F2")))
            .RuleFor(p => p.TransactionsDetails, p => p.Commerce.ProductMaterial())
            .RuleFor(p => p.Address, AddressFake.GetFakeAddress)
            .RuleFor(p => p.SellerId, customerId)
            .RuleFor(p => p.CondominiumId, condominiumId)
            .RuleFor(p => p.RegistryNumber, p => p.Random.UInt().ToString())
            .RuleFor(p => p.RegistryRecord, p => p.Random.UInt().ToString())
            .RuleFor(p => p.IsNew, true)
            .RuleFor(p => p.ShowInHome, true)
            .RuleFor(p => p.IsActive, true);

        return propertyFake.Generate(quantity);
    }
}
```

.\RealtyHub.Tests\Customers\CustomerTests.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using System.Net.Http.Json;

namespace RealtyHub.Tests.Customers;

public class CustomerTests
{
    [Fact]
    public async Task GetCustomersReturnsOkStatusCode()
    {
        // Arrange
        await using var application = new RealtyHubApiTests();

        // Act
        await MockData.CreateCustomers(application, true, 5, 5);
        var client = await MockData.CreateClient(application);
        var result = await client.GetAsync("/v1/customers");
        var customers = await
result.Content.ReadFromJsonAsync<Response<List<Customer>>>();

        // Assert
        Assert.Equal(HttpStatusCode.OK, result.StatusCode);
        Assert.NotNull(customers?.Data);
        Assert.Equal(10, customers.Data.Count);
    }

    [Fact]
    public async Task GetCustomerByIdReturnsOkStatusCode()
    {
        // Arrange
        await using var application = new RealtyHubApiTests();
        // Act
        await MockData.CreateCustomers(application, true, 1, 0);
        var client = await MockData.CreateClient(application);
        var result = await client.GetAsync("/v1/customers/1");
        var customer = await result.Content.ReadFromJsonAsync<Response<Customer>>();
        // Assert
        Assert.Equal(HttpStatusCode.OK, result.StatusCode);
        Assert.NotNull(customer?.Data);
        Assert.Equal(1, customer.Data.Id);
    }
}
```

.\RealtyHub.Tests\MockData.cs

```
using RealtyHub.ApiService.Data;
using RealtyHub.Core.Utilities.FakeEntities;
using System.Net.Http.Json;

namespace RealtyHub.Tests;

public class MockData
{
    public static async Task CreateCustomers(RealtyHubApiTests application,
        bool create, int quantityBusiness, int quantityIndividual)
    {
        using var scoce = application.Services.CreateScope();
        var provider = scoce.ServiceProvider;
        await using var dbContext = provider.GetRequiredService<AppDbContext>();
        await dbContext.Database.EnsureCreatedAsync();

        if (create)
        {
            var customersBusinessToCreate =
CustomerFake.GetFakeBusinessCustomers(quantityBusiness);
            var customersIndividualToCreate =
CustomerFake.GetFakeIndividualCustomers(quantityIndividual);
            await dbContext.Customers.AddRangeAsync(customersBusinessToCreate);
            await dbContext.Customers.AddRangeAsync(customersIndividualToCreate);
            await dbContext.SaveChangesAsync();
        }
    }
    public static async Task<HttpClient> CreateClient(RealtyHubApiTests application)
    {
        var client = application.CreateClient();

        const string registerUrl = "/v1/identity/register";
        const string loginUrl =
"/v1/identity/login?useCookies=true&useSessionCookies=true";

        var login = new Login("israel@gmail.com", "!W92X+!Q@rOwC48+v.V3");

        var registerResponse = await client.PostAsJsonAsync(registerUrl, login);
        var loginResponse = await client.PostAsJsonAsync(loginUrl, login);

        return client;
    }
    private record Login
    {
        public Login(string Email, string Password)
        {
            email = Email;
            password = Password;
        }

        public string email { get; init; }
        public string password { get; init; }

        public void Deconstruct(out string email, out string password)
        {
            email = this.email;
            password = this.password;
        }
    }
}
```

.\RealtyHub.Tests\RealtyHubApiTests.cs

```
using Microsoft.AspNetCore.Mvc.Testing;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Storage;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Hosting;
using RealtyHub.ApiService.Data;

namespace RealtyHub.Tests;

public class RealtyHubApiTests : WebApplicationFactory<ApiService.Program>
{
    protected override IHost CreateHost(IHostBuilder builder)
    {
        var root = new InMemoryDatabaseRoot();

        builder.ConfigureServices(services =>
        {
            services.RemoveAll(typeof(DbContextOptions<AppDbContext>));

            services.AddDbContext<AppDbContext>(options =>
            {
                options.UseInMemoryDatabase("InMemoryDbForTesting", root);
            });
        });

        return base.CreateHost(builder);
    }
}
```

.\RealtyHub.Web\App.razor

```
<CascadingAuthenticationState>
    <Router AppAssembly="@typeof(App).Assembly">
        <Found Context="routeData">
            <AuthorizeRouteView RouteData="@routeData"
DefaultLayout="@typeof(MainLayout)">
                <NotAuthorized>
                    <PageTitle>Not authorized</PageTitle>
                    <LayoutView Layout="@typeof(MainLayout)">
                        <h1>401</h1>
                        <p>Você não tem permissão para acessar esta página</p>
                    </LayoutView>
                </NotAuthorized>
            </AuthorizeRouteView>
        </Found>
        <NotFound>
            <PageTitle>Not found</PageTitle>
            <LayoutView Layout="@typeof(MainLayout)">
                <h1>404</h1>
                <p>Desculpe, não há nada aqui...</p>
            </LayoutView>
        </NotFound>
    </Router>
</CascadingAuthenticationState>
```

.\RealtyHub.Web\Components\Common\AddressInput.razor

```
@using RealtyHub.Core.Models
@using RealtyHub.Core.Services

<MudText Class="mt-6 mb-6" Typo="Typo.h6">Endereço</MudText>

<MudGrid>
    <MudItem lg="2" xs="12" sm="6">
        <MudTextField T="string"
                      Label="CEP"
                      @bind-Value="InputModel.ZipCode"
                      For="@(() => InputModel.ZipCode)"
                      Mask="@Utility.Masks.ZipCode"
                      OnBlur="SearchAddressAsync"
                      InputType="InputType.Text" />
    </MudItem>

    <MudItem lg="4" xs="12" sm="6">
        <MudTextField T="string"
                      Label="Logradouro"
                      @bind-Value="InputModel.Street"
                      For="@(() => InputModel.Street)"
                      InputType="InputType.Text" />
    </MudItem>

    <MudItem lg="2" xs="12" sm="6">
        <MudTextField T="string"
                      Label="Número"
                      @bind-Value="InputModel.Number"
                      For="@(() => InputModel.Number)"
                      InputType="InputType.Text"
                      MaxLength="10" />
    </MudItem>

    <MudItem lg="4" xs="12" sm="6">
        <MudTextField T="string"
                      Label="Complemento"
                      @bind-Value="InputModel.Complement"
                      For="@(() => InputModel.Complement)"
                      InputType="InputType.Text" />
    </MudItem>

    <MudItem lg="3" xs="12" sm="6">
        <MudTextField T="string"
                      Label="Bairro"
                      @bind-Value="InputModel.Neighborhood"
                      For="@(() => InputModel.Neighborhood)"
                      InputType="InputType.Text" />
    </MudItem>

    <MudItem lg="3" xs="12" sm="6">
        <MudTextField T="string"
                      Label="Cidade"
                      @bind-Value="InputModel.City"
                      For="@(() => InputModel.City)"
                      InputType="InputType.Text" />
    </MudItem>

    <MudItem lg="3" xs="12" sm="6">
        <MudTextField T="string"
                      Label="Estado"
                      @bind-Value="InputModel.State"
                      For="@(() => InputModel.State)"
                      InputType="InputType.Text" />
    </MudItem>

    <MudItem lg="3" xs="12" sm="6">
        <MudTextField T="string"
                      Label="País"
                      @bind-Value="InputModel.Country"
                      For="@(() => InputModel.Country)"
                      InputType="InputType.Text" />
    </MudItem>
```

```
</MudGrid>

@code
{
    [Parameter, EditorRequired]
    public Address InputModel { get; set; } = null!;

    [Parameter]
    public EventCallback<Address> InputModelChanged { get; set; }

    [CascadingParameter]
    public EditContext CurrentEditContext { get; set; } = null!;

    [Inject]
    public IViaCepService CepService { get; set; } = null!;

    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    public async Task SearchAddressAsync(FocusEventArgs focusEventArgs)
    {
        try
        {
            if (string.IsNullOrWhiteSpace(InputModel.ZipCode))
                return;

            var result = await CepService.GetAddressAsync(InputModel.ZipCode);
            if (result.Data is null || !result.IsSuccess) return;

            InputModel.Street = result.Data.Street;
            InputModel.Number = result.Data.Number;
            InputModel.Complement = result.Data.Complement;
            InputModel.Neighborhood = result.Data.Neighborhood;
            InputModel.City = result.Data.City;
            InputModel.State = result.Data.State;
            InputModel.Country = result.Data.Country;
            await InputModelChanged.InvokeAsync(InputModel);
            StateHasChanged();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    protected override async Task OnInitializedAsync()
    {
        await base.OnParametersSetAsync();
        await base.OnInitializedAsync();
    }
}
```

.\RealtyHub.Web\Components\Common\DialogConfirm.razor

```
<MudDialog>
    <DialogContent>
        <MudText>@((MarkupString)ContentText)</MudText>
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="Cancel">Cancelar</MudButton>
        <MudButton OnClick="Submit" Color="@ButtonColor"
Variant="Variant.Filled">@ButtonText</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter] MudDialogInstance MudDialog { get; set; } = new();
    [Parameter] public string ContentText { get; set; } = string.Empty;
    [Parameter] public string ButtonText { get; set; } = "Confirmar";
    [Parameter] public Color ButtonColor { get; set; } = Color.Success;

    void Submit() => MudDialog.Close(DialogResult.Ok(true));
    void Cancel() => MudDialog.Cancel();
}
```

.\RealtyHub.Web\Components\Common\EnumSelect.razor

```
@using System.Reflection
@typeparam TEnum

<MudSelect T="TEnum"
           HelperText="@HelperText"
           Class="@Class"
           Label="@Label"
           Value="SelectedValue"
           ReadOnly="ReadOnly"
           ValueChanged="OnSelectedValueChanged">
    @foreach (var value in Enum.GetValues(typeof(TEnum)).Cast<TEnum>())
    {
        <MudSelectItem Value="value">@GetDisplayName(value)</MudSelectItem>
    }
</MudSelect>

@code {
    [Parameter] public required TEnum SelectedValue { get; set; }
    [Parameter] public EventCallback<TEnum> SelectedValueChanged { get; set; }
    [Parameter] public string Label { get; set; } = string.Empty;
    [Parameter] public string Class { get; set; } = string.Empty;
    [Parameter] public string HelperText { get; set; } = string.Empty;
    [Parameter] public bool ReadOnly { get; set; }

    private string GetDisplayName(TEnum value)
    {
        var field = value!.GetType().GetField(value.ToString() ?? string.Empty);
        var displayAttribute = field?.GetCustomAttribute<DisplayAttribute>();
        return displayAttribute?.Name ?? value.ToString()!;
    }

    private async Task OnSelectedValueChanged(TEnum value)
    {
        SelectedValue = value;
        await SelectedValueChanged.InvokeAsync(value);
        StateHasChanged();
    }
}
```

.\RealtyHub.Web\Components\Common\ImageDialog.razor

```
<MudDialog>
    <DialogContent>
        <MudImage Src="@ImageUrl" Fluid="true" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Error">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter] MudDialogInstance MudDialog { get; set; } = null!;
    [Parameter] public string ImageUrl { get; set; } = null!;

    void CloseDialog() => MudDialog.Close();
}
```

.\RealtyHub.Web\Components\Common\Logo.razor

```
<div class="d-flex justify-center pb-8 pt-8">
    <img src="@Configuration.SrcLogo"
         alt="RealtyHub - Portal Imobiliário"
         width="@Width" />
</div>

@code
{
    [Parameter]
    public string Width { get; set; } = "200";
}
```

.\RealtyHub.Web\Components\Common\NavMenu.razor

```
<Logo Width="100" />

<MudNavMenu Bordered="true" @key="MudNavMenuKey">
    <MudText Align="Align.Center" Typo="Typo.h6" Class="px-4">RealtyHub</MudText>
    <MudDivider Class="my-2" />
    <div class="d-flex">
        <MudIcon Icon="@Icons.Material.Filled.RealEstateAgent" Class="ml-4"></MudIcon>
        <MudText Typo="Typo.body1" Class="px-4">@Configuration.GivenName</MudText>
    </div>
    <MudDivider Class="my-2" />
    <MudNavGroup Icon="@Icons.Material.Filled.Person" Title="Clientes"
@bind-Expanded="ExpandedCustomers">
        <MudNavLink Icon="@Icons.Material.Filled.PersonAddAlt1"
Href="/clientes/adicionar" Match="NavLinkMatch.All">Cadastro</MudNavLink>
        <MudNavLink Icon="@Icons.Material.Filled.Person" Href="/clientes"
Match="NavLinkMatch.All">Listar</MudNavLink>
    </MudNavGroup>
    <MudNavGroup Icon="fa-solid fa-building" Title="Condomínios"
@bind-Expanded="ExpandedCondominiums">
        <MudNavLink Icon="fa-solid fa-building-circle-arrow-right"
Href="/condominios/adicionar" Match="NavLinkMatch.All">Cadastro</MudNavLink>
        <MudNavLink Icon="fa-solid fa-building" Href="/condominios"
Match="NavLinkMatch.All">Listar</MudNavLink>
    </MudNavGroup>
    <MudNavGroup Icon="@Icons.Material.Filled.Home" Title="Imóveis"
@bind-Expanded="ExpandedProperties">
        <MudNavLink Icon="@Icons.Material.Filled.AddHome" Href="/imoveis/adicionar"
Match="NavLinkMatch.All">Cadastro</MudNavLink>
        <MudNavLink Icon="@Icons.Material.Filled.Home" Href="/imoveis"
Match="NavLinkMatch.All">Listar</MudNavLink>
        <MudNavLink Icon="@Icons.Material.Filled.Home" Href="/listar-imoveis"
Match="NavLinkMatch.All">Visualizar como visitante</MudNavLink>
    </MudNavGroup>
    <MudNavGroup Icon="@Icons.Material.Filled.Pageview" Title="Visitas"
@bind-Expanded="ExpandedViewings">
        <MudNavLink Icon="@Icons.Material.Filled.ScheduleSend" Href="/visitas/agendar"
Match="NavLinkMatch.All">Agendar</MudNavLink>
        <MudNavLink Icon="@Icons.Material.Filled.Schedule" Href="/visitas"
Match="NavLinkMatch.All">Listar</MudNavLink>
    </MudNavGroup>
    <MudNavLink Icon="@Icons.Material.Filled.LocalOffer" Href="/propostas"
Match="NavLinkMatch.All">Propostas</MudNavLink>
    <MudNavGroup Icon="fa-solid fa-file-contract" Title="Contratos"
@bind-Expanded="ExpandedContracts">
        <MudNavLink Icon="fa-solid fa-file-signature" Href="/contratos/emitir"
Match="NavLinkMatch.All">Emitir</MudNavLink>
        <MudNavLink Icon="fa-solid fa-file-contract" Href="/contratos"
Match="NavLinkMatch.All">Listar</MudNavLink>
        <MudNavLink Icon="@Icons.Material.Filled.Settings" Href="/contratos/modelos"
Match="NavLinkMatch.All">Modelos</MudNavLink>
    </MudNavGroup>

</MudNavMenu>

@code
{
    [Parameter]
    public int MudNavMenuKey { get; set; }

    public bool ExpandedCustomers { get; set; }
    public bool ExpandedCondominiums { get; set; }
    public bool ExpandedProperties { get; set; }
    public bool ExpandedViewings { get; set; }
    public bool ExpandedContracts { get; set; }
}
```

.\RealtyHub.Web\Components\Common\NavMenu.razor.css

```
.navbar-toggler {
    background-color: rgba(255, 255, 255, 0.1);
}

.top-row {
    height: 3.5rem;
    background-color: rgba(0,0,0,0.4);
}

.navbar-brand {
    font-size: 1.1rem;
}

.oi {
    width: 2rem;
    font-size: 1.1rem;
    vertical-align: text-top;
    top: -2px;
}

.nav-item {
    font-size: 0.9rem;
    padding-bottom: 0.5rem;
}

    .nav-item:first-of-type {
        padding-top: 1rem;
    }

    .nav-item:last-of-type {
        padding-bottom: 1rem;
    }

    .nav-item ::deep a {
        color: #d7d7d7;
        border-radius: 4px;
        height: 3rem;
        display: flex;
        align-items: center;
        line-height: 3rem;
    }

.nav-item ::deep a.active {
    background-color: rgba(255,255,255,0.25);
    color: white;
}

.nav-item ::deep a:hover {
    background-color: rgba(255,255,255,0.1);
    color: white;
}

@media (min-width: 641px) {
    .navbar-toggler {
        display: none;
    }

    .collapse {
        /* Never collapse the sidebar for wide screens */
        display: block;
    }
}
```

.\RealtyHub.Web\Components\Common\SkeletonWave.razor

```
<MudPaper Class="pa-8 mt-4">
    <MudSkeleton Animation="Animation.Wave" Height="55px" />
    <MudSkeleton Animation="Animation.Wave" Height="110px" />
    <MudSkeleton Animation="Animation.Wave" Height="55px" />
    <MudSkeleton Animation="Animation.Wave" Height="110px" />
    <MudSkeleton Animation="Animation.Wave" Height="55px" />
</MudPaper>
```

.\RealtyHub.Web\Components\Condominiums\CondominiumDialog.razor

```
@using RealtyHub.Core.Models
@using RealtyHub.Web.Pages.Condominiums

<MudDialog>
    <DialogContent>
        <List OnCondominiumSelected="SelectCondominium" RowStyle="cursor: pointer" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code
{
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public EventCallback<Condominium> OnCondominiumSelected { get; set; }

    void CloseDialog() => MudDialog.Close();

    public async Task SelectCondominium(Condominium condominium)
    {
        if (OnCondominiumSelected.HasDelegate)
            await OnCondominiumSelected.InvokeAsync(condominium);

        MudDialog.Close(DialogResult.Ok(condominium));
    }
}
```

.\RealtyHub.Web\Components\Condominiums\CondominiumForm.razor

```
@inherits CondominiumFormComponent

<PageTitle>@Operation Condomínio</PageTitle>

<MudText Typo="Typo.h4">@Operation Condomínio</MudText>

@if (IsBusy)
{
    <SkeletonWave />
}
else
{
    <MudPaper Class="pa-8 mt-4" Elevation="2">
        <EditForm OnValidSubmit="OnValidSubmitAsync" Model="InputModel">
            <ObjectGraphDataAnnotationsValidator />

            <MudText Typo="Typo.h6">Informações sobre o condomínio</MudText>

            <MudGrid Class="mb-4" Justify="Justify.FlexStart">
                <MudItem lg="5" md="7" xs="12">
                    <MudTextField T="string"
                                  Label="Nome"
                                  @bind-Value="InputModel.Name"
                                  For="@(() => InputModel.Name)"
                                  InputType="InputType.Text" />
                </MudItem>

                <MudItem lg="2" md="5" xs="12">
                    <MudNumericField Label="Valor mensal"
                                     @bind-Value="InputModel.CondominiumValue"
                                     For="@(() => InputModel.CondominiumValue)"
                                     InputMode="InputMode.numeric" />
                </MudItem>
            </MudGrid>

            <MudGrid Justify="Justify.FlexStart">
                <MudItem lg="1" md="2" xs="12">
                    <MudCheckBox Label="Elevador"
                                 @bind-Value="InputModel.HasElevator"
                                 For="@(() => InputModel.HasElevator)" />
                </MudItem>

                <MudItem lg="1" md="2" xs="12">
                    <MudCheckBox Label="Piscina"
                                 @bind-Value="InputModel.HasSwimmingPool"
                                 For="@(() => InputModel.HasSwimmingPool)" />
                </MudItem>

                <MudItem lg="1" md="2" xs="12">
                    <MudCheckBox Label="Playground"
                                 @bind-Value="InputModel.HasPlayground"
                                 For="@(() => InputModel.HasPlayground)" />
                </MudItem>

                <MudItem lg="1" md="2" xs="12">
                    <MudCheckBox Label="Academia"
                                 @bind-Value="InputModel.HasFitnessRoom"
                                 For="@(() => InputModel.HasFitnessRoom)" />
                </MudItem>

                <MudItem lg="2" md="3" xs="12">
                    <MudCheckBox Label="Salão de Festas"
                                 @bind-Value="InputModel.HasPartyRoom"
                                 For="@(() => InputModel.HasPartyRoom)" />
                </MudItem>
            </MudGrid>

            <AddressInput @bind-InputModel="InputModel.Address" />

            <MudButton ButtonType="ButtonType.Submit"
                       Variant="Variant.Filled"
                       Color="Color.Primary"
```

```
                        Class="mt-6"
                        StartIcon="@Icons.Material.Filled.Save">
                    Salvar
                </MudButton>
            </EditForm>
        </MudPaper>
}
```

.\RealtyHub.Web\Components\Condominiums\CondominiumForm.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;
using System.Text.RegularExpressions;

namespace RealtyHub.Web.Components.Condominiums;

/// <summary>
/// Componente responsável pelo formulário de cadastro e edição de condomínios.
/// </summary>
public class CondominiumFormComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do condomínio. Se for diferente de zero, o formulário entra em modo
de edição.
    /// </summary>
    [Parameter]
    public long Id { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Indica a operação atual do formulário ("Editar" ou "Cadastrar").
    /// </summary>
    public string Operation => Id != 0
        ? "Editar" : "Cadastrar";

    /// <summary>
    /// Indica se o formulário está em estado de carregamento ou processamento.
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Modelo de entrada utilizado para o binding dos campos do formulário.
    /// </summary>
    public Condominium InputModel { get; set; } = new();

    /// <summary>
    /// Expressão regular utilizada para remover caracteres não numéricos do CEP.
    /// </summary>
    public string Pattern = @"\D";

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Serviço de navegação para redirecionamento de páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de condomínio.
    /// </summary>
    [Inject]
    public ICondominiumHandler Handler { get; set; } = null!;

    #endregion
```

```
    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário, realizando a criação ou atualização do
condomínio.
    /// </summary>
    /// <returns>Task assíncrona representando a operação.</returns>
    public async Task OnValidSubmitAsync()
    {
        IsBusy = true;
        try
        {
            // Remove caracteres não numéricos do CEP
            InputModel.Address.ZipCode = Regex.Replace(InputModel.Address.ZipCode,
Pattern, "");

            Response<Condominium?> result;
            if (InputModel.Id > 0)
                result = await Handler.UpdateAsync(InputModel);
            else
                result = await Handler.CreateAsync(InputModel);

            if (!result.IsSuccess)
            {
                Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
                return;
            }

            Snackbar.Add("Condomínio cadastrado com sucesso", Severity.Success);
            NavigationManager.NavigateTo("/condominios");
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente, buscando os dados do condomínio caso esteja em modo de
edição.
    /// </summary>
    /// <returns>Task assíncrona representando a operação.</returns>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        GetCondominiumByIdRequest? request = null;
        try
        {
            request = new GetCondominiumByIdRequest { Id = Id };
        }
        catch
        {
            Snackbar.Add("Parâmetro inválido", Severity.Error);
        }

        if (request is null) return;

        try
        {
            if (Id != 0)
            {
                var result = await Handler.GetByIdAsync(request);
                if (result is { IsSuccess: true, Data: not null })
                    InputModel = result.Data;
                else
                    Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
            }
```

```
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Contracts\ContractForm.razor

```
@inherits ContractFormComponent

<PageTitle>@Operation Contrato</PageTitle>

@if (IsBusy)
{
    <SkeletonWave />
}
else
{
    <MudPaper Class="pa-8" Elevation="4">
        <EditForm Model="InputModel" OnValidSubmit="OnValidSubmitAsync"
@key="EditFormKey">
            <ObjectGraphDataAnnotationsValidator />

            <MudText Class="mt-5 mb-3" Typo="Typo.h6">Informações do contrato</MudText>

            <MudGrid>
                <MudItem lg="3" md="4" sm="6">
                    <MudDatePicker Class="mb-3"
                                   Label="Data de emissão"
                                   HelperText="Data de emissão do contrato"
                                   @bind-Date="@InputModel.IssueDate"
                                   For="@(() => InputModel.IssueDate)" />
                </MudItem>
                <MudItem lg="3" md="4" sm="6">
                    <MudDatePicker Class="mb-3"
                                   Label="Data de vigência"
                                   HelperText="Data de vigência do contrato"
                                   @bind-Date="InputModel.EffectiveDate"
                                   For="@(() => InputModel.EffectiveDate)" />
                </MudItem>
                <MudItem lg="3" md="4" sm="6">
                    <MudDatePicker Class="mb-3"
                                   Label="Data de término"
                                   HelperText="Data de término do contrato"
                                   @bind-Date="InputModel.TermEndDate"
                                   For="@(() => InputModel.TermEndDate)" />
                </MudItem>

                <MudItem lg="3" md="4" sm="6">
                    <MudDatePicker Class="mb-3"
                                   Label="Data da assinatura"
                                   HelperText="Data de assinatura do contrato"
                                   @bind-Date="InputModel.SignatureDate"
                                   For="@(() => InputModel.SignatureDate)" />
                </MudItem>

                <MudItem lg="6" md="8" sm="12">
                    <MudTextField Class="mb-3"
                                  ReadOnly="true"
                                  T="string"
                                  Label="@(InputModel.OfferId == 0 ? "Clique aqui para
pesquisar a proposta" : "Id da proposta selecionada")"
                                  HelperText="Informe a proposta que irá gerar o
contrato"
                                  AdornmentIcon="@Icons.Material.Filled.Search"
                                  OnClearButtonClick="ClearOfferObjets"
                                  Clearable="true"
                                  @onclick="@OpenOfferDialog"
                                  OnAdornmentClick="OpenOfferDialog"
                                  Adornment="Adornment.End"
                                  Value="@(InputModel.OfferId == 0 ? "" :
InputModel.Offer?.Id.ToString())" />
                </MudItem>

                <MudItem lg="6" md="8" sm="12">
                    <MudTextField Class="mb-3"
                                  ReadOnly="true"
                                  Label="@(InputModel.BuyerId == 0 ? "Clique aqui para
pesquisar o comprador" : "Comprador selecionado")"
                                  HelperText="Informe o comprador do imóvel"
```

```
                                    AdornmentIcon="@Icons.Material.Filled.PersonSearch"
                                    OnClearButtonClick="ClearBuyerObjects"
                                    Clearable="true"
                                    @onclick="@OpenBuyerDialog"
                                    OnAdornmentClick="OpenBuyerDialog"
                                    Adornment="Adornment.End"
                                    Value="@InputModel.Buyer?.Name" />
                </MudItem>

                <MudItem lg="6" md="8" sm="12">
                    <MudTextField Class="mb-3"
                                    ReadOnly="true"
                                    Label="@(InputModel.SellerId == 0 ? "Clique aqui para
pesquisar o vendedor" : "Vendedor selecionado")"
                                    HelperText="Informe o vendedor do imóvel"
                                    AdornmentIcon="@Icons.Material.Filled.PersonSearch"
                                    OnClearButtonClick="ClearSellerObjects"
                                    Clearable="true"
                                    @onclick="@OpenSellerDialog"
                                    OnAdornmentClick="OpenSellerDialog"
                                    Adornment="Adornment.End"
                                    Value="@InputModel.Seller?.Name" />
                </MudItem>

                <MudItem lg="6" md="8" sm="12">
                    <MudTextField Class="mb-3"
                                    ReadOnly="true"
                                    Label="@(InputModel.Offer?.PropertyId == 0 ? "Clique
aqui para pesquisar o imóvel" : "Imóvel selecionado")"
                                    HelperText="Informe o imóvel que será objeto do
contrato"
                                    AdornmentIcon="@Icons.Material.Filled.Search"
                                    OnClearButtonClick="ClearPropertyObjects"
                                    Clearable="true"
                                    @onclick="@OpenPropertyDialog"
                                    OnAdornmentClick="OpenPropertyDialog"
                                    Adornment="Adornment.End"
                                    Value="@InputModel.Offer?.Property?.Title" />
                </MudItem>
            </MudGrid>
            <MudButton ButtonType="ButtonType.Submit"
                       Variant="Variant.Filled"
                       Color="Color.Primary"
                       Class="mt-6"
                       StartIcon="@(ContractId == 0 ? Icons.Material.Filled.Send :
Icons.Material.Filled.Save)">
                @Operation
            </MudButton>
        </EditForm>
    </MudPaper>
}
```

.\RealtyHub.Web\Components\Contracts\ContractForm.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Web.Components.Common;
using RealtyHub.Web.Components.Customers;
using RealtyHub.Web.Components.Email;
using RealtyHub.Web.Components.Offers;
using RealtyHub.Web.Components.Properties;

namespace RealtyHub.Web.Components.Contracts;

/// <summary>
/// Componente responsável pelo formulário de emissão e edição de contratos.
/// </summary>
public partial class ContractFormComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do contrato. Se diferente de zero, o formulário entra em modo de
edição.
    /// </summary>
    [Parameter]
    public long ContractId { get; set; }

    /// <summary>
    /// Identificador da proposta associada ao contrato.
    /// </summary>
    [Parameter]
    public long OfferId { get; set; }

    /// <summary>
    /// Identificador do imóvel associado ao contrato.
    /// </summary>
    [Parameter]
    public long PropertyId { get; set; }

    /// <summary>
    /// Indica se o formulário está em modo somente leitura.
    /// </summary>
    [Parameter]
    public bool ReadOnly { get; set; }

    /// <summary>
    /// Indica se a busca de imóveis está bloqueada.
    /// </summary>
    [Parameter]
    public bool LockPropertySearch { get; set; }

    /// <summary>
    /// Indica se a busca de clientes está bloqueada.
    /// </summary>
    [Parameter]
    public bool LockCustomerSearch { get; set; }

    /// <summary>
    /// Evento disparado ao clicar no botão de submissão do formulário.
    /// </summary>
    [Parameter]
    public EventCallback OnSubmitButtonClicked { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Indica se o formulário está em estado de carregamento ou processamento.
    /// </summary>
    public bool IsBusy { get; set; }
```

```
    /// <summary>
    /// Chave para forçar a atualização do formulário.
    /// </summary>
    public int EditFormKey { get; set; }

    /// <summary>
    /// Modelo de entrada utilizado para o binding dos campos do formulário.
    /// </summary>
    public Contract InputModel { get; set; } = new();

    /// <summary>
    /// Indica a operação atual do formulário ("Emitir" ou "Editar").
    /// </summary>
    public string Operation => ContractId == 0 ? "Emitir" : "Editar";

    #endregion

    #region Services

    /// <summary>
    /// Handler responsável pelas operações de propostas.
    /// </summary>
    [Inject]
    public IOfferHandler OfferHandler { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de contratos.
    /// </summary>
    [Inject]
    public IContractHandler ContractHandler { get; set; } = null!;

    /// <summary>
    /// Serviço de navegação para redirecionamento de páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Serviço de diálogos para exibição de modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário, realizando a criação ou atualização do
contrato.
    /// </summary>
    /// <remarks>
    /// Este método verifica se o formulário está em modo de emissão ou edição, chama o
handler apropriado para criar ou atualizar o contrato,
    /// exibe mensagens de sucesso ou erro via Snackbar, dispara o evento de submissão e,
em caso de sucesso, abre o diálogo para envio do contrato por e-mail.
    /// </remarks>
    public async Task OnValidSubmitAsync()
    {
        IsBusy = true;
        try
        {
            bool success;
            if (ContractId == 0)
            {
                var response = await ContractHandler.CreateAsync(InputModel);
                success = response.IsSuccess;
                if (response.IsSuccess)
                {
```

```
                    Snackbar.Add("Contrato emitido com sucesso", Severity.Success);
                    await OnSubmitButtonClickedAsync();
                }
                else
                    Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            }
            else
            {
                var response = await ContractHandler.UpdateAsync(InputModel);
                success = response.IsSuccess;
                if (response.IsSuccess)
                {
                    Snackbar.Add("Contrato alterado com sucesso", Severity.Success);
                    NavigationManager.NavigateTo("/contratos");
                    await OnSubmitButtonClickedAsync();
                }
                else
                    Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            }

            if (success)
                await OpenEmailDialog();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    /// <summary>
    /// Abre o diálogo para confirmação e envio do contrato por e-mail.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo de confirmação para o envio do contrato por e-mail. Caso o
usuário confirme, abre um segundo diálogo para informar os e-mails do comprador e
vendedor.
    /// Utiliza o serviço de diálogos (<see cref="IDialogService"/>) para exibir os
modais.
    /// </remarks>
    private async Task OpenEmailDialog()
    {
        var parametersConfirm = new DialogParameters
        {
            { "ContentText", "Deseja enviar os contrados via e-mail? " },
            { "ButtonText", "Confirmar" },
            { "ButtonColor", Color.Success }
        };

        var optionsConfirm = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Small
        };

        var dialogConfirm = await DialogService
            .ShowAsync<DialogConfirm>("Emails", parametersConfirm, optionsConfirm);
        var confirmResult = await dialogConfirm.Result;

        if (confirmResult is { Canceled: true }) return;

        var parameters = new DialogParameters
        {
            { "ContractId", InputModel.Id },
            { "BuyerEmail", InputModel.Buyer?.Email },
            { "SellerEmail", InputModel.Seller?.Email }
        };
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Medium,
            FullWidth = true
        };
```

```
        var dialog = await DialogService
            .ShowAsync<EmailDialog>("Enviar contrato por e-mail", parameters, options);
    }

    /// <summary>
    /// Dispara o evento de clique no botão de submissão, se houver delegate.
    /// </summary>
    /// <remarks>
    /// Verifica se existe um delegate associado ao evento <see
cref="OnSubmitButtonClicked"/> e o executa de forma assíncrona.
    /// Permite que componentes pais sejam notificados após a submissão do formulário.
    /// </remarks>
    private async Task OnSubmitButtonClickedAsync()
    {
        if (OnSubmitButtonClicked.HasDelegate)
            await OnSubmitButtonClicked.InvokeAsync();
    }

    /// <summary>
    /// Abre o diálogo para seleção de imóvel.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo modal para seleção de um imóvel. Ao selecionar, atualiza o
modelo do contrato com o imóvel escolhido.
    /// O diálogo só é aberto se <see cref="LockPropertySearch"/> for falso.
    /// </remarks>
    public async Task OpenPropertyDialog()
    {
        if (LockPropertySearch) return;
        var parameters = new DialogParameters
        {
            { "OnPropertySelected", EventCallback.Factory
                .Create<Property>(this, SelectedProperty) }
        };
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Large,
            FullWidth = true
        };
        var dialog = await DialogService
            .ShowAsync<PropertyDialog>("Informe o Imóvel", parameters, options);

        var result = await dialog.Result;

        if (result is { Canceled: false, Data: Property selectedProperty })
            SelectedProperty(selectedProperty);
    }

    /// <summary>
    /// Manipula a seleção de um imóvel no diálogo.
    /// </summary>
    /// <remarks>
    /// Atualiza o modelo do contrato com o imóvel selecionado e força a atualização do
formulário.
    /// </remarks>
    private void SelectedProperty(Property property)
    {
        InputModel.Offer!.Property = property;
        InputModel.Offer.PropertyId = property.Id;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Limpa os dados do imóvel selecionado.
    /// </summary>
    /// <remarks>
    /// Remove o imóvel do modelo do contrato e reseta os campos relacionados.
    /// </remarks>
    public void ClearPropertyObjects()
    {
        InputModel.Offer!.Property = new Property();
        InputModel.Offer.PropertyId = 0;
        EditFormKey++;
        StateHasChanged();
```

```
    }

    /// <summary>
    /// Abre o diálogo para seleção de proposta.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo modal para seleção de uma proposta aceita. Ao selecionar,
atualiza o modelo do contrato com a proposta escolhida.
    /// O diálogo só é aberto se <see cref="LockPropertySearch"/> for falso.
    /// </remarks>
    public async Task OpenOfferDialog()
    {
        if (LockPropertySearch) return;
        var parameters = new DialogParameters
        {
            { "OnOfferSelected", EventCallback.Factory
                .Create<Offer>(this, SelectedOffer) },
            { "OnlyAccepted", true }
        };
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.ExtraLarge,
            FullWidth = true
        };
        var dialog = await DialogService
            .ShowAsync<OfferListDialog>("Selecione a proposta", parameters, options);
        var result = await dialog.Result;
        if (result is { Canceled: false, Data: Offer offer })
            SelectedOffer(offer);
    }

    /// <summary>
    /// Manipula a seleção de uma proposta no diálogo.
    /// </summary>
    /// <remarks>
    /// Atualiza o modelo do contrato com a proposta selecionada, preenchendo também os
dados do comprador e vendedor.
    /// </remarks>
    private void SelectedOffer(Offer offer)
    {
        InputModel.OfferId = offer.Id;
        InputModel.Offer = offer;
        InputModel.Seller = offer.Property!.Seller;
        InputModel.SellerId = offer.Property.SellerId;
        InputModel.Buyer = offer.Buyer;
        InputModel.BuyerId = offer.BuyerId;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Limpa os dados da proposta selecionada.
    /// </summary>
    /// <remarks>
    /// Remove a proposta do modelo do contrato e reseta os campos de comprador e
vendedor.
    /// </remarks>
    public void ClearOfferObjets()
    {
        InputModel.Offer = new Offer();
        InputModel.OfferId = 0;
        InputModel.Seller = new Customer();
        InputModel.SellerId = 0;
        InputModel.Buyer = new Customer();
        InputModel.BuyerId = 0;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Abre o diálogo para seleção do comprador.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo modal para seleção de um cliente como comprador. O diálogo só é
aberto se <see cref="LockCustomerSearch"/> for falso.
```

```
    /// </remarks>
    public async Task OpenBuyerDialog()
    {
        if (LockCustomerSearch) return;
        var parameters = new DialogParameters
        {
            { "OnCustomerSelected", EventCallback.Factory
                .Create<Customer>(this, SelectedBuyer) }
        };
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Large,
            FullWidth = true
        };
        var dialog = await DialogService
            .ShowAsync<CustomerDialog>("Informe o comprador", parameters, options);
        var result = await dialog.Result;

        if (result is { Canceled: false, Data: Customer buyer })
            SelectedBuyer(buyer);
    }

    /// <summary>
    /// Manipula a seleção do comprador no diálogo.
    /// </summary>
    /// <remarks>
    /// Atualiza o modelo do contrato com o comprador selecionado.
    /// </remarks>
    private void SelectedBuyer(Customer buyer)
    {
        InputModel.Buyer = buyer;
        InputModel.BuyerId = buyer.Id;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Limpa os dados do comprador selecionado.
    /// </summary>
    /// <remarks>
    /// Remove o comprador do modelo do contrato e reseta os campos relacionados.
    /// </remarks>
    public void ClearBuyerObjects()
    {
        InputModel.Buyer = new Customer();
        InputModel.BuyerId = 0;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Abre o diálogo para seleção do vendedor.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo modal para seleção de um cliente como vendedor. O diálogo só é
aberto se <see cref="LockCustomerSearch"/> for falso.
    /// </remarks>
    public async Task OpenSellerDialog()
    {
        if (LockCustomerSearch) return;
        var parameters = new DialogParameters
        {
            { "OnCustomerSelected", EventCallback.Factory
                .Create<Customer>(this, SelectedSeller) }
        };
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Large,
            FullWidth = true
        };
        var dialog = await DialogService
            .ShowAsync<CustomerDialog>("Informe o vendedor", parameters, options);
        var result = await dialog.Result;
```

```
        if (result is { Canceled: false, Data: Customer seller })
            SelectedSeller(seller);
    }

    /// <summary>
    /// Manipula a seleção do vendedor no diálogo.
    /// </summary>
    /// <remarks>
    /// Atualiza o modelo do contrato com o vendedor selecionado.
    /// </remarks>
    private void SelectedSeller(Customer seller)
    {
        InputModel.Seller = seller;
        InputModel.SellerId = seller.Id;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Limpa os dados do vendedor selecionado.
    /// </summary>
    /// <remarks>
    /// Remove o vendedor do modelo do contrato e reseta os campos relacionados.
    /// </remarks>
    public void ClearSellerObjects()
    {
        InputModel.Seller = new Customer();
        InputModel.SellerId = 0;
        EditFormKey++;
        StateHasChanged();
    }

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente, buscando os dados do contrato, proposta ou imóvel
conforme os parâmetros informados.
    /// </summary>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
        {
            if (ContractId != 0)
            {
                var request = new GetContractByIdRequest { Id = ContractId };
                var response = await ContractHandler.GetByIdAsync(request);
                if (response is { IsSuccess: true, Data: not null })
                {
                    InputModel = response.Data;
                    return;
                }
                Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
                return;
            }
            if (OfferId != 0)
            {
                var request = new GetOfferByIdRequest { Id = OfferId };
                var response = await OfferHandler.GetByIdAsync(request);
                if (response is { IsSuccess: true, Data: not null })
                {
                    InputModel.Offer = response.Data;
                    InputModel.OfferId = OfferId;
                    InputModel.Seller = response.Data.Property!.Seller;
                    InputModel.SellerId = response.Data.Property.SellerId;
                    InputModel.Buyer = response.Data.Buyer;
                    InputModel.BuyerId = response.Data.BuyerId;
                    return;
                }
                Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
                return;
            }
            if (PropertyId != 0)
            {
```

```
                var request = new GetOfferAcceptedByProperty { PropertyId = PropertyId };
                var response = await OfferHandler.GetAcceptedByProperty(request);
                if (response is { IsSuccess: true, Data: not null })
                {
                    InputModel.Offer = response.Data;
                    InputModel.OfferId = response.Data.Id;
                    InputModel.Seller = response.Data.Property!.Seller;
                    InputModel.SellerId = response.Data.Property.SellerId;
                    InputModel.Buyer = response.Data.Buyer;
                    InputModel.BuyerId = response.Data.BuyerId;
                    return;
                }
                Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            }
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Customers\CustomerDialog.razor

```
@using RealtyHub.Core.Models
@using RealtyHub.Web.Pages.Customers

<MudDialog>
    <DialogContent>
        <List OnCustomerSelected="SelectCustomer" RowStyle="cursor: pointer" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public EventCallback<Customer> OnCustomerSelected { get; set; }

    void CloseDialog() => MudDialog.Close();

    public async Task SelectCustomer(Customer customer)
    {
        if (OnCustomerSelected.HasDelegate)
            await OnCustomerSelected.InvokeAsync(customer);

        MudDialog.Close(DialogResult.Ok(customer));
    }
}
```

.\RealtyHub.Web\Components\Customers\CustomerForm.razor

```
@using RealtyHub.Core.Extensions
@inherits CustomerFormComponent

<PageTitle>@Operation Cliente</PageTitle>

<MudText Typo="Typo.h4">@Operation Cliente</MudText>

@if (IsBusy)
{
    <SkeletonWave />
}
else
{
    <MudPaper Class="pa-8 mt-4" Elevation="2">
        <EditForm OnValidSubmit="OnValidSubmitAsync" EditContext="@EditContext">
            <ObjectGraphDataAnnotationsValidator />

            <MudText Class="mb-6" Typo="Typo.h6">Dados Pessoais</MudText>

            <MudGrid>
                <MudItem lg="4" md="5" xs="12">
                    <MudRadioGroup T="EPersonType"
                                   Value="InputModel.PersonType"
                                   ValueChanged="OnPersonTypeChanged"
                                   Class="flex-center">
                        <MudStack Row="true" Spacing="1">
                            <MudStack Row="true" AlignItems="AlignItems.Center"
Spacing="1">
                                <MudIcon Icon="@Icons.Material.Filled.Person"></MudIcon>
                                <MudRadio Value="EPersonType.Individual">
                                    @(EPersonType.Individual.GetDisplayName())
                                </MudRadio>
                            </MudStack>
                            <MudStack Row="true" AlignItems="AlignItems.Center"
Spacing="1">
                                <MudIcon
Icon="@Icons.Material.Filled.Business"></MudIcon>
                                <MudRadio Value="EPersonType.Business">
                                    @(EPersonType.Business.GetDisplayName())
                                </MudRadio>
                            </MudStack>
                        </MudStack>
                    </MudRadioGroup>
                </MudItem>

                @if (InputModel.PersonType == EPersonType.Individual)
                {
                    <MudItem lg="3" md="4" xs="12">
                        <EnumSelect TEnum="EMaritalStatus"
                                    Label="Estado Civil"
                                    @bind-SelectedValue="InputModel.MaritalStatus"
                                    Class="ml-4">
                        </EnumSelect>
                    </MudItem>
                }

                <MudItem lg="3" md="5" xs="12">
                    <EnumSelect TEnum="ECustomerType"
                                Label="Tipo do cliente"
                                @bind-SelectedValue="InputModel.CustomerType"
                                Class="ml-4">
                    </EnumSelect>
                </MudItem>
            </MudGrid>

            <MudGrid Class="mt-2">
                <MudItem lg="2" md="3" xs="12">
                    <MudTextField T="string"
                                  Label="@DocumentType"
                                  @bind-Value="InputModel.DocumentNumber"
                                  For="@(() => InputModel.DocumentNumber)"
                                  Mask="DocumentCustomerMask"
```

```
                        OnBlur="OnBlurDocumentTextField"
                        ErrorText="@ErrorText"
                        Error="Error"
                        Immediate="true"
                        @key="DocumentCustomerMask"
                        InputType="InputType.Text" />
            </MudItem>

            <MudItem lg="5" md="6" xs="12">
                <MudTextField T="string"
                        Label="@(InputModel.PersonType ==
                                 EPersonType.Individual ? "Nome" : "Razão
Social")"
                        @bind-Value="InputModel.Name"
                        For="@(() => InputModel.Name)"
                        InputType="InputType.Text" />
            </MudItem>

            @if (InputModel.PersonType == EPersonType.Individual)
            {
                <MudItem lg="3" md="5" xs="12">
                    <MudTextField T="string"
                                  Label="RG"
                                  @bind-Value="InputModel.Rg"
                                  For="@(() => InputModel.Rg)"
                                  MaxLength="20"
                                  InputType="InputType.Text" />
                </MudItem>

                <MudItem lg="3" md="4" xs="12">
                    <MudTextField T="string"
                                  Label="Orgão expedidor"
                                  @bind-Value="InputModel.IssuingAuthority"
                                  For="@(() => InputModel.IssuingAuthority)"
                                  InputType="InputType.Text" />
                </MudItem>

                <MudItem lg="3" md="4" xs="12">
                    <MudDatePicker Label="Data da expedição do RG"
                                   @bind-Date="InputModel.RgIssueDate"
                                   For="@(() => InputModel.RgIssueDate)"/>
                </MudItem>

                <MudItem lg="3" md="4" xs="12">
                    <MudTextField T="string"
                                  Label="Nacionalidade"
                                  @bind-Value="InputModel.Nationality"
                                  For="@(() => InputModel.Nationality)"
                                  MaxLength="80"
                                  InputType="InputType.Text" />
                </MudItem>

                <MudItem lg="3" md="4" xs="12">
                    <MudTextField T="string"
                                  Label="Ocupação"
                                  @bind-Value="InputModel.Occupation"
                                  For="@(() => InputModel.Occupation)"
                                  MaxLength="80"
                                  InputType="InputType.Text" />
                </MudItem>
            }
            else
            {
                <MudItem lg="5" md="6" xs="12">
                    <MudTextField T="string"
                                  Label="Nome Fantasia"
                                  @bind-Value="InputModel.BusinessName"
                                  For="@(() => InputModel.BusinessName)"
                                  InputType="InputType.Text" />
                </MudItem>
            }

            <MudItem lg="3" md="4" xs="12">
                <MudTextField T="string"
                              Label="Telefone"
                              @bind-Value="InputModel.Phone"
```

```
                                For="@(() => InputModel.Phone)"
                                Mask="@Utility.Masks.Phone"
                                InputType="InputType.Text" />
                    </MudItem>

                    <MudItem lg="5" md="6" xs="12">
                        <MudTextField T="string"
                                Label="E-mail"
                                @bind-Value="InputModel.Email"
                                For="@(() => InputModel.Email)"
                                InputType="InputType.Email" />
                    </MudItem>
                </MudGrid>

                <AddressInput @bind-InputModel="InputModel.Address" />

                <MudButton ButtonType="ButtonType.Submit"
                           Variant="Variant.Filled"
                           Color="Color.Primary"
                           Class="mt-6"
                           StartIcon="@Icons.Material.Filled.Save">
                    Salvar
                </MudButton>
            </EditForm>
        </MudPaper>
}
```

.\RealtyHub.Web\Components\Customers\CustomerForm.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Forms;
using MudBlazor;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;
using RealtyHub.Web.Services;
using System.Text.RegularExpressions;

namespace RealtyHub.Web.Components.Customers;

/// <summary>
/// Componente responsável pelo formulário de cadastro e edição de clientes.
/// </summary>
public partial class CustomerFormComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Parâmetro que representa o ID do cliente a ser editado.
    /// </summary>
    [Parameter]
    public long Id { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Contexto de edição do formulário.
    /// </summary>
    public EditContext EditContext = null!;

    /// <summary>
    /// Armazena mensagens de validação do formulário.
    /// </summary>
    private ValidationMessageStore? MessageStore;

    /// <summary>
    /// Texto do botão de ação do formulário.
    /// </summary>
    public string Operation => Id != 0
        ? "Editar" : "Cadastrar";

    /// <summary>
    /// Indica se o formulário está em estado de carregamento ou processamento.
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Instancia de <c><see cref="Customer"/></c> que representa o modelo de entrada do
formulário.
    /// </summary>
    public Customer InputModel { get; set; } = new();

    /// <summary>
    /// Máscara de entrada para o campo de documento do cliente.
    /// </summary>
    /// <remarks>
    /// A máscara é definida com base no tipo de pessoa (física ou jurídica).
    /// </remarks>
    public PatternMask DocumentCustomerMask { get; set; } = null!;

    /// <summary>
    /// Tipo de documento do cliente, que pode ser CPF ou CNPJ.
    /// </summary>
    public string DocumentType =>
        InputModel.PersonType == EPersonType.Individual ? "CPF" : "CNPJ";

    /// <summary>
```

```
    /// Expressão regular utilizada para validar campos de texto.
    /// </summary>
    public string Pattern = @"\D";

    /// <summary>
    /// Texto de erro a ser exibido quando o documento do cliente é inválido.
    /// </summary>
    public string? ErrorText { get; set; }

    /// <summary>
    /// Indica se há erro de validação no campo de documento.
    /// </summary>
    public bool Error => !string.IsNullOrEmpty(ErrorText);

    #endregion

    #region Services

    /// <summary>
    /// Handler responsável por gerenciar operações relacionadas a clientes.
    /// </summary>
    [Inject]
    public ICustomerHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço de navegação para redirecionamento de páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Serviço para validação de documentos (CPF/CNPJ).
    /// </summary>
    [Inject]
    public DocumentValidator DocumentValidator { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Método chamado quando o formulário é enviado e os dados são válidos.
    /// </summary>
    /// <remarks>
    /// Esse método valida os campos do formulário, formata os dados de entrada e chama o
handler para criar ou atualizar o cliente.
    /// Em caso de sucesso, redireciona para a lista de clientes.
    /// Caso contrário, exibe uma mensagem de erro.
    /// </remarks>
    /// <returns>Uma tarefa assíncrona.</returns>
    public async Task OnValidSubmitAsync()
    {
        if (IsIndividualPersonFieldsInvalid()) return;
        IsBusy = true;
        try
        {
            InputModel.Phone = Regex.Replace(InputModel.Phone, Pattern, "");
            InputModel.DocumentNumber = Regex.Replace(InputModel.DocumentNumber, Pattern,
"");
            InputModel.Address.ZipCode = Regex.Replace(InputModel.Address.ZipCode,
Pattern, "");
            InputModel.RgIssueDate = InputModel.RgIssueDate?.ToUniversalTime();

            Response<Customer?> result;
            if (InputModel.Id > 0)
                result = await Handler.UpdateAsync(InputModel);
            else
                result = await Handler.CreateAsync(InputModel);

            var resultMessage = result.Message ?? string.Empty;
```

```
            if (result.IsSuccess)
            {
                Snackbar.Add(resultMessage, Severity.Success);
                NavigationManager.NavigateTo("/clientes");
            }
            else
                Snackbar.Add(resultMessage, Severity.Error);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    /// <summary>
    /// Método chamado quando o campo de documento perde o foco.
    /// </summary>
    /// <remarks>
    /// Esse método valida o documento do cliente e exibe mensagens de erro, se
necessário.
    /// </remarks>
    public void OnBlurDocumentTextField()
        => ValidateDocument();

    /// <summary>
    /// Método chamado quando o campo de documento é alterado.
    /// </summary>
    /// <remarks>
    /// Esse método valida o documento do cliente e exibe mensagens de erro, se
necessário.
    /// </remarks>
    public void ValidateDocument()
    {
        if (string.IsNullOrEmpty(InputModel.DocumentNumber)) return;
        MessageStore?.Clear();
        if (InputModel.PersonType is EPersonType.Individual)
        {
            ErrorText =
                !DocumentValidator.IsValidCpf(InputModel.DocumentNumber)
                    ? "CPF inválido" : null;
        }
        else
        {
            ErrorText =
                !DocumentValidator.IsValidCnpj(InputModel.DocumentNumber)
                    ? "CNPJ inválido" : null;
        }

        if (string.IsNullOrEmpty(ErrorText)) return;

        MessageStore?.Add(() => InputModel.DocumentNumber, ErrorText);
        EditContext.NotifyValidationStateChanged();
    }

    /// <summary>
    /// Método chamado quando o tipo de pessoa é alterado.
    /// </summary>
    /// <remarks>
    /// Esse método altera a máscara do documento e valida o documento do cliente.
    /// </remarks>
    public void OnPersonTypeChanged(EPersonType newPersonType)
    {
        InputModel.PersonType = newPersonType;
        ChangeDocumentMask(newPersonType);
        ValidateDocument();
    }

    /// <summary>
    /// Valida os campos obrigatórios para pessoas físicas.
    /// </summary>
    /// <remarks>
    /// Esse método verifica se os campos obrigatórios estão preenchidos e exibe
```

```
mensagens de erro, se necessário.
    /// </remarks>
    /// <returns><c>true</c> caso os campos sejam inválidos, caso contrario retorna
<c>false</c> .</returns>
    private bool IsIndividualPersonFieldsInvalid()
    {
        if (InputModel.PersonType is not EPersonType.Individual) return false;
        MessageStore?.Clear();
        var isInvalid = false;

        if (string.IsNullOrEmpty(InputModel.Nationality))
        {
            MessageStore?.Add(() => InputModel.Nationality, "Campo obrigatório");
            isInvalid = true;
        }

        if (string.IsNullOrEmpty(InputModel.Occupation))
        {
            MessageStore?.Add(() => InputModel.Occupation, "Campo obrigatório");
            isInvalid = true;
        }

        if (string.IsNullOrEmpty(InputModel.Rg))
        {
            MessageStore?.Add(() => InputModel.Rg, "Campo obrigatório");
            isInvalid = true;
        }

        if (string.IsNullOrEmpty(InputModel.IssuingAuthority))
        {
            MessageStore?.Add(() => InputModel.IssuingAuthority, "Campo obrigatório");
            isInvalid = true;
        }

        if (InputModel.RgIssueDate is null)
        {
            MessageStore?.Add(() => InputModel.RgIssueDate!, "Campo obrigatório");
            isInvalid = true;
        }

        EditContext.NotifyValidationStateChanged();
        MessageStore?.Clear();

        return isInvalid;
    }

    /// <summary>
    /// Altera a máscara do documento com base no tipo de pessoa.
    /// </summary>
    /// <param name="personType"> Enumerador <see cref="EPersonType"/> que representa o
tipo de pessoa.</param>
    private void ChangeDocumentMask(EPersonType personType)
    {
        DocumentCustomerMask = personType switch
        {
            EPersonType.Individual => Utility.Masks.Cpf,
            EPersonType.Business => Utility.Masks.Cnpj,
            _ => DocumentCustomerMask
        };
        StateHasChanged();
    }

    /// <summary>
    /// Carrega os dados do cliente a partir do ID fornecido.
    /// </summary>
    /// <remarks>
    /// Esse método faz uma requisição para obter os dados do cliente e preenche o modelo
de entrada.
    /// Caso ocorra algum erro, exibe uma mensagem de erro e redireciona para a lista de
clientes.
    /// </remarks>
    private async Task LoadCustomerAsync()
    {
        GetCustomerByIdRequest? request = null;

        try
```

```
        {
            request = new GetCustomerByIdRequest { Id = Id };
        }
        catch
        {
            Snackbar.Add("Parâmetro inválido", Severity.Error);
        }

        if (request is null) return;

        var response = await Handler.GetByIdAsync(request);
        if (response is { IsSuccess: true, Data: not null })
        {
            InputModel.Id = response.Data.Id;
            InputModel.Name = response.Data.Name;
            InputModel.Email = response.Data.Email;
            InputModel.Phone = response.Data.Phone;
            InputModel.DocumentNumber = response.Data.DocumentNumber;
            InputModel.PersonType = response.Data.PersonType;
            InputModel.CustomerType = response.Data.CustomerType;
            InputModel.Occupation = response.Data.Occupation;
            InputModel.Nationality = response.Data.Nationality;
            InputModel.MaritalStatus = response.Data.MaritalStatus;
            InputModel.Address.ZipCode = response.Data.Address.ZipCode;
            InputModel.Address.Street = response.Data.Address.Street;
            InputModel.Address.Number = response.Data.Address.Number;
            InputModel.Address.Complement = response.Data.Address.Complement;
            InputModel.Address.Neighborhood = response.Data.Address.Neighborhood;
            InputModel.Address.City = response.Data.Address.City;
            InputModel.Address.State = response.Data.Address.State;
            InputModel.Address.Country = response.Data.Address.Country;
            InputModel.Rg = response.Data.Rg;
            InputModel.IssuingAuthority = response.Data.IssuingAuthority;
            InputModel.RgIssueDate = response.Data.RgIssueDate;
            InputModel.BusinessName = response.Data.BusinessName;
            InputModel.IsActive = response.Data.IsActive;
            InputModel.UserId = response.Data.UserId;
        }
        else
        {
            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            NavigationManager.NavigateTo("/clientes");
        }
        ValidateDocument();
    }

    /// <summary>
    /// Redireciona para a página de criação de cliente.
    /// </summary>
    private void RedirectToCreateCustomer()
    {
        InputModel.PersonType = EPersonType.Individual;
        InputModel.MaritalStatus = EMaritalStatus.Single;
        NavigationManager.NavigateTo("/clientes/adicionar");
    }

    /// <summary>
    /// Executa as validações iniciais do formulário.
    /// </summary>
    private void ExecuteValidations()
    {
        EditContext = new EditContext(InputModel);
        MessageStore = new ValidationMessageStore(EditContext);
        ChangeDocumentMask(InputModel.PersonType);
    }

    #endregion

    #region Overrides

    /// <summary>
    /// Método chamado quando o componente é inicializado.
    /// </summary>
    /// <remarks>
    /// Esse método inicializa o contexto de edição, executa as validações iniciais e
carrega os dados do cliente, se necessário.
```

```
    /// Caso ocorra algum erro, exibe uma mensagem de erro.
    /// </remarks>
    /// <returns>Uma tarefa assíncrona.</returns>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
        {
            if (Id != 0)
                await LoadCustomerAsync();
            else
                RedirectToCreateCustomer();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            ExecuteValidations();
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Email\EmailDialog.razor

```
<MudDialog>
    <DialogContent>
        <EmailForm ContractId="ContractId"
                   BuyerEmail="@BuyerEmail"
                   SellerEmail="@SellerEmail"
                   OnEmailSent="@CloseDialog"/>
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public long ContractId { get; set; }

    [Parameter]
    public string BuyerEmail { get; set; } = string.Empty;

    [Parameter]
    public string SellerEmail { get; set; } = string.Empty;

    void CloseDialog() => MudDialog.Close();
}
```

.\RealtyHub.Web\Components\Email\EmailForm.razor

```
@using RealtyHub.Core.Requests.Emails
@using RealtyHub.Core.Services

<MudPaper Class="pa-4">
    <MudGrid>
        <MudItem lg="6" md="8" xs="12">
            <MudTextField T="string"
            Label="E-mail do comprador"
            @bind-Value="BuyerEmail"
            For="@(() => BuyerEmail)"
            InputType="InputType.Email" />
        </MudItem>
        <MudItem lg="6" md="8" xs="12">
            <MudTextField T="string"
            Label="E-mail do vendedor"
            @bind-Value="SellerEmail"
            For="@(() => SellerEmail)"
            InputType="InputType.Email" />
        </MudItem>
    </MudGrid>
    <div class="d-flex mt-8">
        @if (IsBusy)
        {
            <MudProgressLinear Color="Color.Info" Indeterminate="true" />
        }
        else
        {
            <MudButton Class="mt-4"
            Variant="Variant.Filled"
            Color="Color.Primary"
            OnClick="OnClickAsync">
                Enviar e-mails
            </MudButton>
        }
    </div>

</MudPaper>

@code
{
    [Parameter]
    public EventCallback OnEmailSent { get; set; }

    [Parameter]
    public string BuyerEmail { get; set; } = string.Empty;

    [Parameter]
    public string SellerEmail { get; set; } = string.Empty;

    [Parameter]
    public long ContractId { get; set; }

    [Inject]
    public IEmailService EmailService { get; set; } = null!;

    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    public bool IsBusy { get; set; }
    private bool _success;

    public async Task OnClickAsync()
    {
        IsBusy = true;
        if (string.IsNullOrWhiteSpace(BuyerEmail) ||
string.IsNullOrWhiteSpace(SellerEmail))
        {
            Snackbar.Add("Os e-mails são obrigatórios", Severity.Error);
            return;
        }
        try
        {
```

```
            _success = await SendEmailToBuyerAsync();
            _success = await SendEmailToSellerAsync();
            if (_success)
                Snackbar.Add("E-mails enviados com sucesso", Severity.Success);

            if (OnEmailSent.HasDelegate)
                await OnEmailSent.InvokeAsync();
        }
        catch
        {
            Snackbar.Add("Erro ao enviar e-mail", Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    private async Task<bool> SendEmailToBuyerAsync()
    {
        var request = new AttachmentMessage
        {
            ContractId = ContractId,
            EmailTo = BuyerEmail,
            Subject = "Contrato de compra e venda",
            Body = "Segue em anexo o contrato referente a aquisição do seu novo imóvel"
        };
        var result = await EmailService.SendContractAsync(request);
        if (!result.IsSuccess)
            Snackbar.Add("Erro ao enviar e-mail para o comprador", Severity.Error);

        return result.IsSuccess;
    }

    private async Task<bool> SendEmailToSellerAsync()
    {
        var request = new AttachmentMessage
        {
            ContractId = ContractId,
            EmailTo = SellerEmail,
            Subject = "Contrato de compra e venda",
            Body = "Segue em anexo o contrato referente a venda do seu imóvel"
        };
        var result = await EmailService.SendContractAsync(request);
        if (!result.IsSuccess)
            Snackbar.Add("Erro ao enviar e-mail para o vendedor", Severity.Error);

        return result.IsSuccess;
    }

    protected override async Task OnInitializedAsync()
    {
        await base.OnParametersSetAsync();
        await base.OnInitializedAsync();
    }
}
```

.\RealtyHub.Web\Components\Home\CardHome.razor

```
@inherits CardHomeComponent

<MudItem lg="3" md="6" xs="12">
    <MudPaper Elevation="2">
        <MudCard Class="@("mb-2 " + Class)">
            <MudCardHeader>
                <CardHeaderContent>
                    <MudText Typo="Typo.body1">@Property.Title</MudText>
                </CardHeaderContent>
            </MudCardHeader>
            <MudCardMedia Image="@GetSrcPhoto()" Height="300" />
            <MudCardContent>
                <MudText Typo="Typo.body2">@Property.Description</MudText> <br />
                <MudText Typo="Typo.body2">@Property.Bedroom @(Property.Bedroom > 1 ?
"Quartos" : "Quarto")</MudText> <br />
                <MudText Typo="Typo.body2">@Property.Bathroom @(Property.Bathroom > 1 ?
"Banheiros" : "Banheiro")</MudText> <br />
                <MudText Typo="Typo.body2">@Property.Garage @(Property.Garage > 1 ?
"Garagens" : "Garagem")</MudText> <br />
                <MudText Typo="Typo.body2">@Property.Area M²: </MudText> <br />
                <MudText Typo="Typo.body2">@Property.Price Preço</MudText> <br />
            </MudCardContent>
            <MudCardActions>
                <MudTooltip Text="Enviar proposta">
                    <MudIconButton Icon="@Icons.Material.Filled.Send"
                                   Color="Color.Primary"
                                   OnClick="OnSendOfferClickedAsync" />
                </MudTooltip>
                <MudTooltip Text="Mais informações">
                    <MudIconButton Icon="@Icons.Material.Filled.Info"
                                   Color="Color.Primary"
                                   Href="@($"/imoveis/detalhes/{Property.Id}")" />
                </MudTooltip>
            </MudCardActions>
        </MudCard>
    </MudPaper>
</MudItem>
```

.\RealtyHub.Web\Components\Home\CardHome.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Models;
using RealtyHub.Web.Components.Offers;

namespace RealtyHub.Web.Components.Home;

/// <summary>
/// Componente responsável por exibir o card de um imóvel na página inicial, incluindo
foto, informações e ação para envio de proposta.
/// </summary>
public partial class CardHomeComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Classe CSS personalizada para o card.
    /// </summary>
    [Parameter]
    public string Class { get; set; } = string.Empty;

    /// <summary>
    /// Imóvel a ser exibido no card.
    /// </summary>
    [Parameter]
    public Property Property { get; set; } = new();

    #endregion

    #region Services

    /// <summary>
    /// Serviço de diálogo para exibição de modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Obtém a URL da foto principal (thumbnail) do imóvel, ou a primeira foto
disponível.
    /// </summary>
    /// <remarks>
    /// Busca a foto marcada como thumbnail em <see cref="Property.PropertyPhotos"/>. Se
não houver, retorna a primeira foto.
    /// Monta a URL completa para exibição da imagem no card.
    /// </remarks>
    /// <returns>URL da foto do imóvel.</returns>
    public string GetSrcPhoto()
    {
        var photo = Property
                        .PropertyPhotos
                        .FirstOrDefault(p => p.IsThumbnail)
                    ?? Property.PropertyPhotos.FirstOrDefault();

        var fullName = $"{photo?.Id}{photo?.Extension}";

        return $"{Configuration.BackendUrl}/photos/{fullName}";
    }

    /// <summary>
    /// Abre o diálogo para envio de proposta para o imóvel exibido no card.
    /// </summary>
    /// <remarks>
    /// Exibe um modal utilizando <see cref="OfferDialog"/>, passando o ID do imóvel como
parâmetro.
    /// </remarks>
    public async Task OnSendOfferClickedAsync()
    {
```

```
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Medium,
            CloseOnEscapeKey = true,
            FullWidth = true
        };

        var parameters = new DialogParameters
        {
            { "PropertyId", Property.Id }
        };

        await DialogService.ShowAsync<OfferDialog>("Enviar proposta", parameters,
options);
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Home\HomeRedirect.razor

```
@inject NavigationManager NavigationManager

<h3>Redirecionando para a página principal...</h3>

@code
{
    protected override void OnInitialized()
    {
        NavigationManager.NavigateTo("/home");
    }
}
```

.\RealtyHub.Web\Components\Home\PropertyDetails.razor

```
@page "/imoveis/detalhes/{id:long}"
@using RealtyHub.Core.Models
@inherits PropertyDetailsPage
@layout PublicLayout

<PageTitle>Detalhes - @InputModel.Title</PageTitle>

<MudPaper Elevation="2">
    <MudGrid @key="GridKey">
        <MudItem xs="24" sm="12">
            <MudCarousel ItemsSource="@InputModel.PropertyPhotos"
                         ShowArrows="true"
                         ShowBullets="true"
                         AutoCycle="false"
                         Style="height: 70vh; width: 100%;"
                         TData="PropertyPhoto">

                <ItemTemplate>
                    <div class="d-flex justify-center align-center" style="height:
100%;">
                        <MudImage
Src="@($"{Configuration.BackendUrl}/photos/{context.Id}{context.Extension}")"
                                  Style="max-width: 95%; max-height: 100%;"
                                  ObjectFit="ObjectFit.Contain" />
                    </div>
                </ItemTemplate>
            </MudCarousel>
        </MudItem>
    </MudGrid>

    <MudText Class="ml-5 mb-10 mt-10" Typo="Typo.h6">@InputModel.Title</MudText>

    <MudCard Elevation="3" Class="mb-10 mt-10">
        <MudCardHeader>
            <CardHeaderContent>
                <MudText Typo="Typo.h6">Descrição</MudText>
            </CardHeaderContent>
        </MudCardHeader>
        <MudCardContent>
            <MudText Class="mb-2" Typo="Typo.body1">@InputModel.Description</MudText>
            <MudText Class="mb-2">@InputModel.Bedroom Quartos</MudText>
            <MudText Class="mb-2">@InputModel.Bathroom Banheiros</MudText>
            <MudText Class="mb-2">@InputModel.Garage Garagem</MudText>
            <MudText Class="mb-2">@InputModel.Area M²</MudText>
            <MudText Class="mb-2">@InputModel.TransactionsDetails</MudText>
        </MudCardContent>
    </MudCard>

    <MudCard Elevation="3" Class="mb-10 mt-10">
        <MudCardHeader>
            <CardHeaderContent>
                <MudText Typo="Typo.h6">Preço</MudText>
            </CardHeaderContent>
        </MudCardHeader>
        <MudCardContent>
            <MudText Typo="Typo.body1">@InputModel.Price.ToString("C")</MudText>
        </MudCardContent>
    </MudCard>

    <MudCard Elevation="3" Class="mb-10 mt-10">
        <MudCardHeader>
            <CardHeaderContent>
                <MudText Typo="Typo.h6">Localização</MudText>
            </CardHeaderContent>
        </MudCardHeader>
        <MudCardContent>
            <MudText Typo="Typo.body1">@InputModel.Address.Neighborhood</MudText>
        </MudCardContent>
    </MudCard>
</MudPaper>
```

.\RealtyHub.Web\Components\Home\PropertyDetails.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;

namespace RealtyHub.Web.Components.Home;

/// <summary>
/// Página responsável por exibir os detalhes de um imóvel na área pública do site.
/// </summary>
public partial class PropertyDetailsPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do imóvel a ser exibido.
    /// </summary>
    [Parameter]
    public long Id { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Modelo do imóvel exibido na página.
    /// </summary>
    public Property InputModel { get; set; } = new();

    /// <summary>
    /// Indica se a página está em estado de carregamento.
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Chave para forçar atualização do grid de fotos ou informações.
    /// </summary>
    public int GridKey { get; set; }

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de imóveis.
    /// </summary>
    [Inject]
    public IPropertyHandler PropertyHandler { get; set; } = null!;

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente, buscando os detalhes do imóvel pelo ID informado.
    /// </summary>
    /// <remarks>
    /// Realiza a requisição ao handler para obter os dados do imóvel. Em caso de
sucesso, preenche o modelo e atualiza a interface.
    /// Em caso de erro, exibe mensagem via Snackbar.
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
```

```
        {
            var request = new GetPropertyByIdRequest { Id = Id };
            var result = await PropertyHandler.GetByIdAsync(request);
            if (result is { IsSuccess: true, Data: not null })
            {
                InputModel = result.Data;
                GridKey++;
            }
            else
            {
                Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
            }
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Identity\LoginRedirect.razor

```
@inject NavigationManager NavigationManager

<h3>Redirecionando para o login...</h3>

@code
{
    protected override void OnInitialized()
    {
        NavigationManager.NavigateTo("/login");
    }
}
```

.\RealtyHub.Web\Components\Identity\PasswordInput.razor

```
@using RealtyHub.Core.Models.Account

<MudTextField T="string"
              Label="Senha"
              InputType="InputType.Password"
              For="@(() => InputModel.Password)"
              @bind-Value="InputModel.Password"
              Class="mb-4"
              AdornmentIcon="@Icons.Material.Filled.Password"
              Adornment="Adornment.Start" />

<MudTextField T="string"
              Label="Confirmar Senha"
              InputType="InputType.Password"
              For="@(() => InputModel.ConfirmPassword)"
              @bind-Value="InputModel.ConfirmPassword"
              OnBlur="IsPasswordEqual"
              Error="@(!InputModel.IsEqual)"
              ErrorText="@InputModel.Message"
              Class="mb-4"
              AdornmentIcon="@Icons.Material.Filled.Password"
              Adornment="Adornment.Start" />

@code
{
    [Parameter, EditorRequired]
    public PasswordResetModel InputModel { get; set; } = new();

    [CascadingParameter]
    public EditContext CurrentEditContext { get; set; } = null!;

    [Parameter]
    public EventCallback<PasswordResetModel> InputModelChanged { get; set; }

    protected override async Task OnInitializedAsync()
    {
        await base.OnParametersSetAsync();
        await base.OnInitializedAsync();
    }

    public void IsPasswordEqual()
    {
        if (InputModel.IsEqual) return;
        InputModelChanged.InvokeAsync(InputModel);
        StateHasChanged();
    }
}
```

.\RealtyHub.Web\Components\Offers\OfferDialog.razor

```
<MudDialog>
    <DialogContent>
        <OfferForm PropertyId="@PropertyId"
                   OfferId="@OfferId"
                   OnSubmitButtonClicked="OnSubmitButtonClickedAsync"
                   ReadOnly="@ReadyOnly"
                   ShowPayments="@ShowPayments" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public EventCallback OnSubmitButtonClicked { get; set; }

    [Parameter]
    public long PropertyId { get; set; }

    [Parameter]
    public long OfferId { get; set; }

    [Parameter]
    public bool ReadyOnly { get; set; }

    [Parameter]
    public bool ShowPayments { get; set; }

    public void CloseDialog() => MudDialog.Close();

    public async Task OnSubmitButtonClickedAsync()
    {
        if (OnSubmitButtonClicked.HasDelegate)
            await OnSubmitButtonClicked.InvokeAsync();

        CloseDialog();
    }
}
```

.\RealtyHub.Web\Components\Offers\OfferForm.razor

```
@inherits OfferFormComponent

<PageTitle>@Operation proposta</PageTitle>

@if (IsBusy)
{
    <SkeletonWave />
}
else
{
    <MudPaper Class="pa-8" Elevation="4">
        <EditForm OnValidSubmit="OnValidSubmitAsync" Model="InputModel">
            <ObjectGraphDataAnnotationsValidator />

            <MudText Class="mt-5 mb-3" Typo="Typo.h6">Informações da proposta</MudText>

            <MudGrid>
                <MudItem xs="8" sm="4">
                    <MudTextField Label="Valor total"
                                  T="decimal"
                                  Variant="Variant.Text"
                                  HelperText="Informe quando deseja pagar no imóvel"
                                  Format="C"
                                  @bind-Value="InputModel.Amount"
                                  For="@(() => InputModel.Amount)"
                                  ReadOnly="ReadOnly" />
                </MudItem>
            </MudGrid>

            @if (OfferId == 0 || ShowPayments)
            {
                <MudGrid Class="mt-1">
                    <MudItem xs="12">
                        <MudText Class="mb-2" Typo="Typo.h6">Informações de
pagamento</MudText>
                    </MudItem>
                    @foreach (var payment in InputModel.Payments)
                    {
                        <MudItem lg="12" xs="16" sm="10">
                            <PaymentForm InputModel="payment"
                                         @bind-Payments="InputModel.Payments"
                                         ReadOnly="ReadOnly" />
                        </MudItem>
                    }
                </MudGrid>

               @if (ReadOnly == false)
                {
                    <MudItem xs="12">
                        <MudButton ButtonType="ButtonType.Button"
                                   Variant="Variant.Outlined"
                                   Color="Color.Primary"
                                   Class="mt-5"
                                   StartIcon="@Icons.Material.Filled.Add"
                                   OnClick="AddPayment"
                                   Disabled="DisableAddPayment">
                            Adicionar pagamento
                        </MudButton>
                    </MudItem>
                }
            }

            <MudText Class="mt-5" Typo="Typo.h6">Informações para contato</MudText>

            <MudGrid>
                <MudItem xs="12" sm="6">
                    <MudTextField Label="Nome"
                                  T="string"
                                  Variant="Variant.Text"
                                  HelperText="Informe seu nome"
                                  @bind-Value="InputModel.Buyer!.Name"
                                  For="@(() => InputModel.Buyer!.Name)"
```

```
                                ReadOnly="ReadOnly" />
                    </MudItem>
                    <MudItem xs="12" sm="6">
                        <MudTextField Label="E-mail"
                                      T="string"
                                      Variant="Variant.Text"
                                      HelperText="Informe seu e-mail para contato"
                                      @bind-Value="InputModel.Buyer!.Email"
                                      For="@(() => InputModel.Buyer!.Email)"
                                      ReadOnly="ReadOnly" />
                    </MudItem>

                    <MudItem xs="12" sm="6">
                        <MudTextField Label="Telefone"
                                      T="string"
                                      Variant="Variant.Text"
                                      HelperText="Informe seu telefone para contato"
                                      @bind-Value="InputModel.Buyer!.Phone"
                                      For="@(() => InputModel.Buyer!.Phone)"
                                      Mask="@Utility.Masks.Phone"
                                      ReadOnly="ReadOnly" />
                    </MudItem>
                </MudGrid>

                @if (ReadOnly == false)
                {
                    <MudButton ButtonType="ButtonType.Submit"
                               Variant="Variant.Filled"
                               Color="Color.Primary"
                               Class="mt-5"
                               StartIcon="@Icons.Material.Filled.Send">
                        @Operation proposta
                    </MudButton>
                }
            </EditForm>
        </MudPaper>
}
```

.\RealtyHub.Web\Components\Offers\OfferForm.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Requests.Properties;
using System.Text.RegularExpressions;

namespace RealtyHub.Web.Components.Offers;

/// <summary>
/// Componente responsável pelo formulário de envio e edição de propostas de compra de
imóvel.
/// </summary>
public partial class OfferFormComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do imóvel para o qual a proposta será enviada.
    /// </summary>
    [Parameter]
    public long PropertyId { get; set; }

    /// <summary>
    /// Identificador da proposta. Se diferente de zero, o formulário entra em modo de
edição.
    /// </summary>
    [Parameter]
    public long OfferId { get; set; }

    /// <summary>
    /// Indica se o formulário está em modo somente leitura.
    /// </summary>
    [Parameter]
    public bool ReadOnly { get; set; }

    /// <summary>
    /// Indica se a seção de pagamentos deve ser exibida.
    /// </summary>
    [Parameter]
    public bool ShowPayments { get; set; }

    /// <summary>
    /// Evento disparado ao clicar no botão de submissão do formulário.
    /// </summary>
    [Parameter]
    public EventCallback OnSubmitButtonClicked { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Indica se o formulário está em estado de carregamento ou processamento.
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Modelo de entrada utilizado para o binding dos campos do formulário.
    /// </summary>
    public Offer InputModel { get; set; } = new();

    /// <summary>
    /// Indica a operação atual do formulário ("Enviar" ou "Editar").
    /// </summary>
    public string Operation => OfferId == 0 ? "Enviar" : "Editar";

    /// <summary>
    /// Indica se o botão de adicionar pagamento deve estar desabilitado (máximo de 5
pagamentos).
```

```
    /// </summary>
    public bool DisableAddPayment => InputModel.Payments.Count >= 5;

    /// <summary>
    /// Expressão regular utilizada para remover caracteres não numéricos dos campos
relevantes.
    /// </summary>
    public string Pattern = @"\D";

    #endregion

    #region Services

    /// <summary>
    /// Handler responsável pelas operações de propostas.
    /// </summary>
    [Inject]
    public IOfferHandler OfferHandler { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de imóveis.
    /// </summary>
    [Inject]
    public IPropertyHandler PropertyHandler { get; set; } = null!;

    /// <summary>
    /// Serviço de navegação para redirecionamento de páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário, realizando a criação ou atualização da
proposta.
    /// </summary>
    /// <remarks>
    /// Remove caracteres não numéricos do telefone do comprador, chama o handler para
criar ou atualizar a proposta,
    /// exibe mensagens de sucesso ou erro via Snackbar, dispara o evento de submissão e
redireciona para a listagem de imóveis em caso de sucesso.
    /// </remarks>
    public async Task OnValidSubmitAsync()
    {
        IsBusy = true;
        try
        {
            string message;
            InputModel.Buyer!.Phone = Regex.Replace(InputModel.Buyer.Phone, Pattern, "");
            if (OfferId == 0)
            {
                var response = await OfferHandler.CreateAsync(InputModel);
                if (response.IsSuccess)
                {
                    Snackbar.Add("Proposta enviada com sucesso", Severity.Success);
                    Snackbar.Add("Aguarde nosso contato", Severity.Success);
                    await OnSubmitButtonClickedAsync();
                    NavigationManager.NavigateTo("/listar-imoveis");
                    return;
                }
                message = response.Message ?? string.Empty;
            }
            else
            {
                var response = await OfferHandler.UpdateAsync(InputModel);
                if (response.IsSuccess)
                {
```

```
                    Snackbar.Add("Proposta alterada com sucesso", Severity.Success);
                    await OnSubmitButtonClickedAsync();
                    return;
                }
                message = response.Message ?? string.Empty;
            }
            Snackbar.Add(message, Severity.Error);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    /// <summary>
    /// Dispara o evento de clique no botão de submissão, se houver delegate.
    /// </summary>
    /// <remarks>
    /// Verifica se existe um delegate associado ao evento <see
cref="OnSubmitButtonClicked"/> e o executa de forma assíncrona.
    /// Permite que componentes pais sejam notificados após a submissão do formulário.
    /// </remarks>
    private async Task OnSubmitButtonClickedAsync()
    {
        if (OnSubmitButtonClicked.HasDelegate)
            await OnSubmitButtonClicked.InvokeAsync();
    }

    /// <summary>
    /// Adiciona um novo pagamento à lista de pagamentos da proposta, limitado a 5
pagamentos.
    /// </summary>
    /// <remarks>
    /// Cada novo pagamento é criado com o tipo padrão <see
cref="EPaymentType.BankSlip"/>.
    /// </remarks>
    public void AddPayment()
    {
        if (InputModel.Payments.Count < 5)
        {
            InputModel.Payments.Add(new Payment
            {
                PaymentType = EPaymentType.BankSlip
            });
        }
    }

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente, carregando os dados da proposta para edição ou
preparando para envio de nova proposta.
    /// </summary>
    /// <remarks>
    /// Se <see cref="OfferId"/> for diferente de zero, busca os dados da proposta para
edição.
    /// Caso contrário, busca os dados do imóvel e prepara o modelo para envio de nova
proposta.
    /// Exibe mensagens de erro via Snackbar em caso de falha.
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
        {
            if (OfferId != 0)
            {
                var request = new GetOfferByIdRequest { Id = OfferId };
                var response = await OfferHandler.GetByIdAsync(request);
                if (response is { IsSuccess: true, Data: not null })
```

```
                {
                    InputModel = response.Data;
                    return;
                }
                Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            }
            else
            {
                var request = new GetPropertyByIdRequest { Id = PropertyId };
                var response = await PropertyHandler.GetByIdAsync(request);
                if (response is { IsSuccess: true, Data: not null })
                {
                    InputModel.Property = response.Data;
                    InputModel.PropertyId = response.Data.Id;
                    InputModel.Buyer = new Customer();
                    InputModel.BuyerId = 0;
                    return;
                }

                Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            }
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Offers\OfferListDialog.razor

```
@using RealtyHub.Core.Models
@using RealtyHub.Web.Pages.Offers

<MudDialog>
    <DialogContent>
        <List OnOfferSelected="SelectOffer"
              RowStyle="cursor: pointer"
              OnlyAccepted="OnlyAccepted" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code
{
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public EventCallback<Offer> OnOfferSelected { get; set; }

    [Parameter]
    public bool OnlyAccepted { get; set; }

    void CloseDialog() => MudDialog.Close();

    public async Task SelectOffer(Offer offer)
    {
        if (OnOfferSelected.HasDelegate)
            await OnOfferSelected.InvokeAsync(offer);

        MudDialog.Close(DialogResult.Ok(offer));
    }
}
```

.\RealtyHub.Web\Components\Offers\OfferStatus.razor

```
@using RealtyHub.Core.Extensions

@switch (Status)
{
    case EOfferStatus.Accepted:
        <MudChip T="Enum" Color="Color.Success">@Status.GetDisplayName()</MudChip>
        break;

    case EOfferStatus.Rejected:
        <MudChip T="Enum" Color="Color.Error">@Status.GetDisplayName()</MudChip>
        break;

    case EOfferStatus.Analysis:
        <MudChip T="Enum" Color="Color.Primary">@Status.GetDisplayName()</MudChip>
        break;
}

@code
{
    [Parameter, EditorRequired]
    public EOfferStatus Status { get; set; }
}
```

.\RealtyHub.Web\Components\Offers\PaymentForm.razor

```
@using RealtyHub.Core.Models

<EditForm Model="InputModel">
    <ObjectGraphDataAnnotationsValidator />

    <MudStack Row="true">
        <EnumSelect TEnum="EPaymentType"
                    Label="Tipo do pagamento"
                    @bind-SelectedValue="InputModel.PaymentType"
                    HelperText="Informe o tipo de pagamento"
                    ReadOnly="ReadOnly" />

        <MudTextField Label="Valor"
                      T="decimal"
                      Variant="Variant.Text"
                      Format="C"
                      @bind-Value="InputModel.Amount"
                      For="@(() => InputModel.Amount)"
                      HelperText="Valor deste pagamento"
                      ReadOnly="ReadOnly" />

        @if (InputModel.Installments)
        {
            <MudSelect T="int"
                       Label="Parcelas"
                       ReadOnly="ReadOnly"
                       @bind-Value="InputModel.InstallmentsCount"
                       For="@(() => InputModel.InstallmentsCount)">
                @for (int i = 1; i <= 24; i++)
                {
                    var i1 = i;
                    <MudSelectItem Value="@i">@($"{i1}x")</MudSelectItem>
                }
            </MudSelect>
        }

        @if (InputModel.PaymentType != EPaymentType.Check
             && InputModel.PaymentType != EPaymentType.Fgts)
        {
            <MudCheckBox Label="Parcelado"
                         @bind-Value="InputModel.Installments"
                         For="@(() => InputModel.Installments)"
                         ReadOnly="ReadOnly">
            </MudCheckBox>
        }
    </MudStack>
</EditForm>

@code
{
    [CascadingParameter]
    public EditContext CurrentEditContext { get; set; } = null!;

    [Parameter]
    public EventCallback<List<Payment>> PaymentsChanged { get; set; }

    [Parameter]
    public long OfferId { get; set; }

    [Parameter]
    public Payment InputModel { get; set; } = null!;

    [Parameter]
    public List<Payment> Payments { get; set; } = null!;

    [Parameter]
    public bool ReadOnly { get; set; }

    protected override async Task OnInitializedAsync()
    {
        await base.OnParametersSetAsync();
        await base.OnInitializedAsync();
```

```
    }
}
```

.\RealtyHub.Web\Components\Properties\PropertyDialog.razor

```
@using RealtyHub.Core.Models
@using RealtyHub.Web.Pages.Properties

<MudDialog>
    <DialogContent>
        <List OnPropertySelected="SelectProperty" RowStyle="cursor: pointer" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public EventCallback<Property> OnPropertySelected { get; set; }

    void CloseDialog() => MudDialog.Close();

    public async Task SelectProperty(Property property)
    {
        if (OnPropertySelected.HasDelegate)
            await OnPropertySelected.InvokeAsync(property);

        MudDialog.Close(DialogResult.Ok(property));
    }
}
```

.\RealtyHub.Web\Components\Properties\PropertyForm.razor

```
@using RealtyHub.Core.Models
@inherits PropertyFormComponent

<PageTitle>@Operation Imóvel</PageTitle>

<MudText Typo="Typo.h4">@Operation Imóvel</MudText>

@if (IsBusy)
{
    <SkeletonWave />
}
else
{
    <MudPaper Class="pa-8 mt-4 d-flex justify-center align-center" Elevation="2">
        <EditForm Model="@InputModel" OnValidSubmit="OnValidSubmitAsync">
            <ObjectGraphDataAnnotationsValidator />

            <MudText Class="mb-6" Typo="Typo.h6">Dados do Imóvel</MudText>

            <MudGrid Class="mb-4">
                <MudItem lg="2" xs="12" sm="6">
                    <EnumSelect TEnum="EPropertyType"
                                Label="Tipo do Imóvel"
                                @bind-SelectedValue="InputModel.PropertyType">
                    </EnumSelect>
                </MudItem>

                <MudItem xs="3" sm="2">
                    <MudCheckBox @bind-Value="InputModel.IsNew"
                                 For="@(() => InputModel.IsNew)"
                                 Label="Imóvel Novo">
                    </MudCheckBox>
                </MudItem>

                <MudItem xs="6" sm="3">
                    <MudCheckBox @bind-Value="InputModel.ShowInHome"
                                 For="@(() => InputModel.ShowInHome)"
                                 Label="Exibir na página principal">
                    </MudCheckBox>
                </MudItem>
            </MudGrid>

            <MudGrid Class="mb-4">
                <MudItem lg="4" xs="12" sm="6">
                    <MudTextField Label="Título"
                                  @bind-Value="InputModel.Title"
                                  For="@(() => InputModel.Title)"
                                  InputType="InputType.Text" />
                </MudItem>

                <MudItem lg="8" xs="12" sm="6">
                    <MudTextField Label="Descrição"
                                  @bind-Value="InputModel.Description"
                                  For="@(() => InputModel.Description)"
                                  AutoGrow="true"
                                  InputType="InputType.Text"
                                  MaxLength="255" />
                </MudItem>
            </MudGrid>

            <MudGrid>
                <MudItem lg="3" xs="12" sm="6">
                    <MudTextField Label="Preço"
                                  T="decimal"
                                  Variant="Variant.Text"
                                  Format="C"
                                  @bind-Value="InputModel.Price"
                                  For="@(() => InputModel.Price)" />
                </MudItem>

                <MudItem lg="8" xs="12" sm="6">
                    <MudTextField Label="Detalhes da transação"
```

```
                                @bind-Value="InputModel.TransactionsDetails"
                                For="@(() => InputModel.TransactionsDetails)"
                                AutoGrow="true"
                                MaxLength="255"
                                InputType="InputType.Text" />
                    </MudItem>
                </MudGrid>

                <MudGrid Class="mt-1">
                    <MudItem lg="1" xs="12" sm="6">
                        <MudNumericField Label="Quartos"
                                         @bind-Value="InputModel.Bedroom"
                                         For="@(() => InputModel.Bedroom)"
                                         InputMode="InputMode.numeric"
                                         Max="999"
                                         Min="0" />
                    </MudItem>

                    <MudItem lg="1" xs="12" sm="6">
                        <MudNumericField Label="Banheiros"
                                         @bind-Value="InputModel.Bathroom"
                                         For="@(() => InputModel.Bathroom)"
                                         InputMode="InputMode.numeric"
                                         Max="999"
                                         Min="0" />
                    </MudItem>

                    <MudItem lg="1" xs="12" sm="6">
                        <MudNumericField Label="Garagens"
                                         @bind-Value="InputModel.Garage"
                                         For="@(() => InputModel.Garage)"
                                         InputMode="InputMode.numeric"
                                         Max="999"
                                         Min="0" />
                    </MudItem>

                    <MudItem lg="2" xs="12" sm="6">
                        <MudNumericField Label="Área(m²)"
                                         T="double"
                                         @bind-Value="InputModel.Area"
                                         For="@(() => InputModel.Area)"
                                         InputMode="InputMode.numeric"
                                         Max="9999999"
                                         Min="1" />
                    </MudItem>

                    <MudItem lg="3" xs="12" sm="6">
                        <MudTextField Label="Matrícula no Cartório"
                                      T="string"
                                      @bind-Value="InputModel.RegistryNumber"
                                      For="@(() => InputModel.RegistryNumber)"
                                      HelperText="Número de registro do imóvel"
                                      InputType="InputType.Text" />
                    </MudItem>

                    <MudItem lg="3" xs="12" sm="6">
                        <MudTextField Label="Registro do Cartório"
                                      T="string"
                                      @bind-Value="InputModel.RegistryRecord"
                                      For="@(() => InputModel.RegistryRecord)"
                                      HelperText="Número de registro do cartório"
                                      InputType="InputType.Text" />
                    </MudItem>
                </MudGrid>

                <MudText Class="mb-4 mt-4" Typo="Typo.h6">Proprietário/Vendedor</MudText>
                <MudGrid>
                    <MudItem lg="8" xs="12" sm="6">
                        <MudTextField Class="mb-2"
                                      ReadOnly="true"
                                      Label="@(InputModel.Seller == null ? "Clique aqui para
pesquisar o cliente" : "Cliente selecionado")"
                                      AdornmentIcon="@Icons.Material.Filled.PersonSearch"
                                      @onclick="@OpenSellerDialog"
                                      OnAdornmentClick="OpenSellerDialog"
                                      Adornment="Adornment.End"
```

```
                                Value="@InputModel.Seller?.Name" />
                </MudItem>
            </MudGrid>

            <MudText Class="mb-4 mt-4" Typo="Typo.h6">Condomínio</MudText>
            <MudGrid>
                <MudItem lg="8" md="10" xs="12">
                    <MudTextField Class="mb-2"
                                  ReadOnly="true"
                                  Label="@(InputModel.Condominium == null ? "Clique aqui
para pesquisar o condomínio" : "Condomínio selecionado")"
                                  AdornmentIcon="@Icons.Material.Filled.Search"
                                  @onclick="OpenCondominiumDialog"
                                  OnAdornmentClick="OpenCondominiumDialog"
                                  Adornment="Adornment.End"
                                  Value="InputModel.Condominium?.Name"/>
                </MudItem>
            </MudGrid>

            <AddressInput @bind-InputModel="InputModel.Address" />

            <MudText Class="mb-4 mt-4" Typo="Typo.h6">Fotos do imóvel</MudText>

            <MudGrid Class="mt-2 mb-4" Spacing="2" @key="@DataGridBtnPhotosKey">
                <MudItem lg="8" xs="12" sm="10">
                    <MudStack Row="true" Spacing="2">
                        <MudFileUpload T="IReadOnlyList<IBrowserFile>"
FilesChanged="OnFilesChange">
                            <ActivatorContent>
                                <MudButton Variant="Variant.Filled"
                                           Color="Color.Primary"
                                           StartIcon="@Icons.Material.Filled.AddAPhoto"
                                           Style="white-space: nowrap;">
                                    Selecionar fotos
                                </MudButton>
                            </ActivatorContent>
                        </MudFileUpload>
                        @if (AllPhotos.Count > 0)
                        {
                            <MudButton Color="Color.Error"
                                       Variant="Variant.Filled"
                                       Style="white-space: nowrap;"
                                       StartIcon="@Icons.Material.Filled.DeleteForever"
                                       OnClick="RemoveAllPhotos">
                                Excluir todas as fotos
                            </MudButton>
                            <MudButton Color="Color.Error"
                                       Variant="Variant.Filled"
                                       Style="white-space: nowrap;"
                                       StartIcon="@Icons.Material.Filled.Delete"
                                       OnClick="RemovePhoto">
                                Excluir a foto selecionada
                            </MudButton>
                        }
                    </MudStack>
                </MudItem>
            </MudGrid>

            @if (AllPhotos.Count > 0)
            {
                <MudGrid Class="d-flex justify-center align-center mb-8" Style="height:
40vh;">
                    <MudItem xs="12" sm="10" md="8" lg="6" xl="4">
                        <MudCarousel ItemsSource="@AllPhotos"
                                     ShowArrows="true"
                                     ShowBullets="true"
                                     AutoCycle="false"
                                     Class="mud-width-full align-center"
                                     Style="width: 100%; height: 105vh;"
                                     TData="PhotoItem"
                                     @bind-SelectedIndex="SelectedIndex"
                                     @key="@CarouselKey">

                            <ItemTemplate Context="photoItem">
                                <div class="d-flex flex-column align-items-center mt-4">
                                    <MudItem md="9" xs="6" sm="3">
```

```
                                        <MudButton ButtonType="ButtonType.Button"
                                                   Color="Color.Primary"
                                                   Variant="Variant.Filled"
StartIcon="@Icons.Material.Filled.Image"
                                                   Disabled="@photoItem.IsThumbnail"
                                                   OnClick="@(() =>
UpdateThumbnailsAsync(photoItem))">
                                            @(photoItem.IsThumbnail ? "Foto já definida
como principal" : "Definir como foto principal")
                                        </MudButton>
                                    </MudItem>
                                    <MudText Typo="Typo.caption" Class="mt-2">
                                        A foto ficará em exibição na página principal
                                    </MudText>
                                    <MudImage Src="@photoItem.DisplayUrl"
                                              @onclick="@(() =>
OpenImageDialog(photoItem.DisplayUrl))"
                                              Class="cursor-pointer"
                                              Style="width: 80%; height: 60%; align-self:
center"
                                              Fluid="true"
                                              ObjectFit="ObjectFit.Cover" />
                                </div>
                            </ItemTemplate>
                        </MudCarousel>
                    </MudItem>
                </MudGrid>
            }

            <MudButton ButtonType="ButtonType.Submit"
                       Variant="Variant.Filled"
                       Color="Color.Primary"
                       Class="mt-10"
                       StartIcon="@Icons.Material.Filled.Save">
                Salvar
            </MudButton>
            <MudText Class="mb-10" />
        </EditForm>
    </MudPaper>
}
```

.\RealtyHub.Web\Components\Properties\PropertyForm.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Forms;
using Microsoft.AspNetCore.Components.Web;
using MudBlazor;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;
using RealtyHub.Web.Components.Common;
using RealtyHub.Web.Components.Condominiums;
using RealtyHub.Web.Components.Customers;

namespace RealtyHub.Web.Components.Properties;

/// <summary>
/// Componente responsável pelo formulário de cadastro e edição de imóveis, incluindo
upload, remoção e definição de fotos.
/// </summary>
public partial class PropertyFormComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do imóvel. Se diferente de zero, o formulário entra em modo de
edição.
    /// </summary>
    [Parameter]
    public long Id { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Indica a operação atual do formulário ("Editar" ou "Cadastrar").
    /// </summary>
    public string Operation => Id != 0
        ? "Editar" : "Cadastrar";

    /// <summary>
    /// Indica se o formulário está em estado de carregamento ou processamento.
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Modelo de entrada utilizado para o binding dos campos do formulário.
    /// </summary>
    public Property InputModel { get; set; } = new();

    /// <summary>
    /// Lista de fotos do imóvel carregadas do servidor.
    /// </summary>
    public List<PropertyPhoto> PropertyPhotos { get; set; } = [];

    /// <summary>
    /// Lista de todas as fotos (novas e existentes) exibidas no formulário.
    /// </summary>
    public List<PhotoItem> AllPhotos { get; set; } = [];

    /// <summary>
    /// Índice da foto atualmente selecionada no carrossel.
    /// </summary>
    public int SelectedIndex { get; set; }

    /// <summary>
    /// Chave para forçar atualização do carrossel de fotos.
    /// </summary>
    public int CarouselKey { get; set; }

    /// <summary>
```

```
    /// Chave para forçar atualização do botão de fotos no DataGrid.
    /// </summary>
    public int DataGridBtnPhotosKey { get; set; }

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de fotos de imóveis.
    /// </summary>
    [Inject]
    public IPropertyPhotosHandler PropertyPhotosHandler { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de imóveis.
    /// </summary>
    [Inject]
    public IPropertyHandler PropertyHandler { get; set; } = null!;

    /// <summary>
    /// Serviço de navegação para redirecionamento de páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Serviço de diálogo para exibição de modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário, realizando a criação ou atualização do
imóvel e o gerenciamento das fotos.
    /// </summary>
    /// <remarks>
    /// Cria ou atualiza o imóvel via handler, gerencia o upload, atualização e remoção
de fotos no servidor,
    /// exibe mensagens de sucesso ou erro via Snackbar e redireciona para a listagem de
imóveis em caso de sucesso.
    /// </remarks>
    public async Task OnValidSubmitAsync()
    {
        IsBusy = true;
        try
        {
            Response<Property?> result;

            if (InputModel.Id > 0)
                result = await PropertyHandler.UpdateAsync(InputModel);
            else
                result = await PropertyHandler.CreateAsync(InputModel);

            if (!result.IsSuccess)
            {
                Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
                return;
            }

            InputModel.Id = result.Data?.Id ?? 0;

            await DeleteRemovedServerPhotos();
            await UpdateServerPhotos();

            Snackbar.Add(result.Message ?? string.Empty, Severity.Success);
```

```
            NavigationManager.NavigateTo("/imoveis");
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    /// <summary>
    /// Remove todas as fotos do imóvel após confirmação do usuário.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo de confirmação antes de limpar a lista de fotos exibidas no
formulário.
    /// </remarks>
    public async Task RemoveAllPhotos()
    {
        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja realmente excluir todas as fotos do imóvel?" },
            { "ButtonColor", Color.Error }
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        var result = await dialog.Result;
        if (result is { Canceled: true }) return;

        CarouselKey++;
        AllPhotos.Clear();
        StateHasChanged();
    }

    /// <summary>
    /// Remove a foto atualmente selecionada do imóvel após confirmação do usuário.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo de confirmação antes de remover a foto selecionada da lista.
    /// </remarks>
    public async Task RemovePhoto()
    {
        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja realmente excluir a foto selecionada?" },
            { "ButtonColor", Color.Error }
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        var result = await dialog.Result;
        if (result is { Canceled: true }) return;

        if (SelectedIndex >= 0 && SelectedIndex < AllPhotos.Count)
        {
            AllPhotos.RemoveAt(SelectedIndex);

            SelectedIndex = AllPhotos.Count > 0
                ? Math.Min(SelectedIndex, AllPhotos.Count - 1)
                : 0;
        }

        CarouselKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Abre um diálogo para exibir a imagem em tamanho ampliado.
    /// </summary>
    /// <param name="photoUrl">URL da foto a ser exibida.</param>
    public async Task OpenImageDialog(string photoUrl)
    {
        var parameters = new DialogParameters { ["ImageUrl"] = photoUrl };
```

```
        var options = new DialogOptions
        {
            CloseOnEscapeKey = true,
            MaxWidth = MaxWidth.Large,
            FullWidth = true
        };
        await DialogService.ShowAsync<ImageDialog>(null, parameters, options);
    }

    /// <summary>
    /// Manipula o upload de arquivos de imagem, convertendo-os para base64 e adicionando
à lista de fotos.
    /// </summary>
    /// <param name="files">Lista de arquivos selecionados pelo usuário.</param>
    public async Task OnFilesChange(IReadOnlyList<IBrowserFile>? files)
    {
        if (files is null) return;
        foreach (var file in files)
        {
            using var ms = new MemoryStream();
            await file.OpenReadStream(long.MaxValue).CopyToAsync(ms);

            var newBytes = ms.ToArray();
            var base64 = Convert.ToBase64String(newBytes);
            var dataUri = $"data:{file.ContentType};base64,{base64}";

            AllPhotos.Add(new PhotoItem
            {
                DisplayUrl = dataUri,
                Content = newBytes,
                ContentType = file.ContentType,
                OriginalFileName = file.Name
            });
        }

        CarouselKey++;
        DataGridBtnPhotosKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Carrega as fotos do imóvel a partir do servidor e popula a lista de fotos
exibidas.
    /// </summary>
    public async Task LoadPhotosFromServerAsync()
    {
        try
        {
            var request = new GetAllPropertyPhotosByPropertyRequest { PropertyId = Id };
            var response = await PropertyPhotosHandler.GetAllByPropertyAsync(request);
            if (response is { IsSuccess: true, Data: not null })
            {
                PropertyPhotos = response.Data;
                foreach (var photo in PropertyPhotos)
                {
                    AllPhotos.Add(new PhotoItem
                    {
                        Id = photo.Id,
                        IsThumbnail = photo.IsThumbnail,
                        DisplayUrl =
$"{Configuration.BackendUrl}/photos/{photo.Id}{photo.Extension}",
                    });
                }
            }
            else
                Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    /// <summary>
    /// Atualiza a foto principal (thumbnail) do imóvel no servidor e na interface.
    /// </summary>
```

```
/// <param name="item">Foto a ser definida como principal.</param>
public async Task UpdateThumbnailsAsync(PhotoItem item)
{
    if (string.IsNullOrEmpty(item.Id))
    {
        foreach (var photoItem in AllPhotos)
        {
            photoItem.IsThumbnail = photoItem.Id == item.Id;
        }
        return;
    }
    IsBusy = true;
    try
    {
        var request = new UpdatePropertyPhotosRequest
        {
            PropertyId = InputModel.Id,
            Photos =
            [
                new()
                {
                    Id = item.Id,
                    IsThumbnail = true,
                    PropertyId = InputModel.Id,
                }
            ]
        };

        var result = await PropertyPhotosHandler.UpdateAsync(request);
        if (!result.IsSuccess)
        {
            Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
            return;
        }

        Snackbar.Add("Foto definida como principal", Severity.Success);

        foreach (var photoItem in AllPhotos)
        {
            photoItem.IsThumbnail = photoItem.Id == item.Id;
        }

        CarouselKey++;
        StateHasChanged();
    }
    catch (Exception e)
    {
        Snackbar.Add(e.Message, Severity.Error);
    }
    finally
    {
        IsBusy = false;
    }
}

/// <summary>
/// Envia as novas fotos adicionadas no formulário para o servidor.
/// </summary>
private async Task UpdateServerPhotos()
{
    var newPhotos = AllPhotos.Where(p => string.IsNullOrEmpty(p.Id)).ToList();
    if (newPhotos.Count > 0)
    {
        var request = new CreatePropertyPhotosRequest
        {
            PropertyId = InputModel.Id,
            FileBytes = newPhotos.Select(p => new FileData
            {
                Id = p.Id,
                Content = p.Content!,
                ContentType = p.ContentType!,
                Name = p.OriginalFileName!,
                IsThumbnail = p.IsThumbnail
            }).ToList()
        };
```

```
            var resultPhotos = await PropertyPhotosHandler.CreateAsync(request);
            if (!resultPhotos.IsSuccess)
                Snackbar.Add(resultPhotos.Message ?? string.Empty, Severity.Error);
        }
    }

    /// <summary>
    /// Remove do servidor as fotos que foram excluídas no formulário.
    /// </summary>
    private async Task DeleteRemovedServerPhotos()
    {
        var oldServerPhotosIds = PropertyPhotos
            .Select(p => p.Id)
            .ToList();

        var currentServerPhotosIds = AllPhotos
            .Where(p => !string.IsNullOrEmpty(p.Id))
            .Select(p => p.Id)
            .ToList();

        var removedIds = oldServerPhotosIds.Except(currentServerPhotosIds).ToList();

        foreach (var photoId in removedIds)
        {
            var deleteReq = new DeletePropertyPhotoRequest { Id = photoId, PropertyId =
InputModel.Id };
            var resp = await PropertyPhotosHandler.DeleteAsync(deleteReq);
            if (!resp.IsSuccess)
            {
                Snackbar.Add(resp.Message ?? $"Erro ao excluir foto {photoId}",
Severity.Error);
            }
        }
    }

    /// <summary>
    /// Carrega os dados do imóvel para edição a partir do ID informado.
    /// </summary>
    /// <remarks>
    /// Busca os dados do imóvel via handler, preenche o modelo de entrada e carrega as
fotos do servidor.
    /// Em caso de erro, exibe mensagem e redireciona para a listagem de imóveis.
    /// </remarks>
    private async Task LoadPropertyAsync()
    {
        GetPropertyByIdRequest? request = null;

        try
        {
            request = new GetPropertyByIdRequest { Id = Id };
        }
        catch
        {
            Snackbar.Add("Parâmetro inválido", Severity.Error);
        }

        if (request is null) return;

        var response = await PropertyHandler.GetByIdAsync(request);
        if (response is { IsSuccess: true, Data: not null })
        {
            InputModel = response.Data;
            await LoadPhotosFromServerAsync();
        }
        else
        {
            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            NavigationManager.NavigateTo("/imoveis");
        }
    }

    /// <summary>
    /// Redireciona para a tela de cadastro de novo imóvel, preenchendo valores padrão.
    /// </summary>
    private void RedirectToCreateProperty()
    {
```

```
    InputModel.PropertyType = EPropertyType.House;
    NavigationManager.NavigateTo("/imoveis/adicionar");
}

/// <summary>
/// Abre o diálogo para seleção do vendedor do imóvel.
/// </summary>
public async Task OpenSellerDialog()
{
    var parameters = new DialogParameters
    {
        { "OnCustomerSelected", EventCallback.Factory
            .Create<Customer>(this, SelectedSeller) }
    };
    var options = new DialogOptions
    {
        CloseButton = true,
        MaxWidth = MaxWidth.Large,
        FullWidth = true
    };
    var dialog = await DialogService
        .ShowAsync<CustomerDialog>("Informe o vendedor", parameters, options);
    var result = await dialog.Result;

    if (result is { Canceled: false, Data: Customer selectedCustomer })
        SelectedSeller(selectedCustomer);
}

/// <summary>
/// Manipula a seleção do vendedor no diálogo, atualizando o modelo do imóvel.
/// </summary>
/// <param name="seller">Vendedor selecionado.</param>
private void SelectedSeller(Customer seller)
{
    InputModel.Seller = seller;
    InputModel.SellerId = seller.Id;
    StateHasChanged();
}

/// <summary>
/// Abre o diálogo para seleção do condomínio do imóvel.
/// </summary>
public async Task OpenCondominiumDialog()
{
    var parameters = new DialogParameters
    {
        { "OnCondominiumSelected", EventCallback.Factory
            .Create<Condominium>(this, SelectedCondominium) }
    };
    var options = new DialogOptions
    {
        CloseButton = true,
        MaxWidth = MaxWidth.Large,
        FullWidth = true
    };
    var dialog = await DialogService
        .ShowAsync<CondominiumDialog>("Informe o condomínio", parameters, options);
    var result = await dialog.Result;

    if (result is { Canceled: false, Data: Condominium selectedCondominium })
        SelectedCondominium(selectedCondominium);
}

/// <summary>
/// Manipula a seleção do condomínio no diálogo, atualizando o modelo do imóvel.
/// </summary>
/// <param name="condominium">Condomínio selecionado.</param>
private void SelectedCondominium(Condominium condominium)
{
    InputModel.Condominium = condominium;
    InputModel.CondominiumId = condominium.Id;
    StateHasChanged();
}

#endregion
```

```
    #region Overrides

    /// <summary>
    /// Inicializa o componente, carregando os dados do imóvel para edição ou preparando
para cadastro.
    /// </summary>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
        {
            if (Id != 0)
                await LoadPropertyAsync();
            else
                RedirectToCreateProperty();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Viewings\ViewingDialog.razor

```
@using RealtyHub.Core.Models

<MudDialog>
    <DialogContent>
        <ViewingForm
            Id="Id"
            Customer="Customer"
            Property="Property"
            LockCustomerSearch="LockCustomerSearch"
            LockPropertySearch="LockPropertySearch"
            RedirectToPageList="RedirectToPageList"
            OnSubmitButtonClicked="OnSubmitButtonClickedAsync" />
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="CloseDialog" Color="Color.Primary">Fechar</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter]
    MudDialogInstance MudDialog { get; set; } = null!;

    [Parameter]
    public Property? Property { get; set; }

    [Parameter]
    public Customer? Customer { get; set; }

    [Parameter]
    public long Id { get; set; }

    [Parameter]
    public bool LockCustomerSearch { get; set; }

    [Parameter]
    public bool LockPropertySearch { get; set; }

    [Parameter]
    public bool RedirectToPageList { get; set; }

    [Parameter]
    public EventCallback<Property> OnPropertySelected { get; set; }

    [Parameter]
    public EventCallback OnSubmitButtonClicked { get; set; }

    public async Task SelectProperty(Property property)
    {
        if (OnPropertySelected.HasDelegate)
            await OnPropertySelected.InvokeAsync(property);

        MudDialog.Close(DialogResult.Ok(property));
    }

    public async Task OnSubmitButtonClickedAsync()
    {
        if (OnSubmitButtonClicked.HasDelegate)
            await OnSubmitButtonClicked.InvokeAsync();

        CloseDialog();
    }

    void CloseDialog() => MudDialog.Close();
}
```

.\RealtyHub.Web\Components\Viewings\ViewingForm.razor

```
@inherits ViewingFormComponent

<PageTitle>@Operation Visita</PageTitle>

<MudText Typo="Typo.h4">@Operation Visita</MudText>

@if (IsBusy)
{
    <SkeletonWave />
}
else
{
    <MudPaper Class="pa-8 mt-4" Elevation="2">
        <EditForm OnValidSubmit="OnValidSubmitAsync" Model="InputModel"
@key="EditFormKey">
            <ObjectGraphDataAnnotationsValidator />

            <MudDatePicker Class="mb-2"
                           Label="Data da Visita"
                           @bind-Date="@InputModel.ViewingDate"
                           For="@(() => InputModel.ViewingDate)" />

            <MudTimePicker Class="mb-2"
                           Label="Hora da Visita"
                           @bind-Time="@ViewingTime"
                           For="@(() => ViewingTime)" />

            <MudTextField Class="mb-2"
                          ReadOnly="true"
                          Label="@(InputModel.Buyer == null ? "Clique aqui para pesquisar
o cliente" : "Cliente Selecionado")"
                          AdornmentIcon="@Icons.Material.Filled.PersonSearch"
                          @onclick="@OpenCustomerDialog"
                          OnAdornmentClick="OpenCustomerDialog"
                          Adornment="Adornment.End"
                          Value="@InputModel.Buyer?.Name" />

            <MudTextField Class="mb-2"
                          ReadOnly="true"
                          Label="@(InputModel.Property == null ? "Clique aqui para
pesquisar o imóvel" : "Imóvel Selecionado")"
                          AdornmentIcon="@Icons.Material.Filled.Search"
                          @onclick="@OpenPropertyDialog"
                          OnAdornmentClick="OpenPropertyDialog"
                          Adornment="Adornment.End"
                          Value="@InputModel.Property?.Title" />

            <div>
                <MudButton ButtonType="ButtonType.Submit"
                           Variant="Variant.Filled"
                           Color="Color.Primary"
                           Class="mt-6"
                           StartIcon="@Icons.Material.Filled.Save">
                    @Operation
                </MudButton>
                @if (Id != 0)
                {
                    <MudButton Variant="Variant.Filled"
                               ButtonType="ButtonType.Button"
                               Color="Color.Primary"
                               Class="mt-6 ml-2"
                               StartIcon="@Icons.Material.Filled.Check"
                               OnClick="OnClickDoneViewing">
                        Finalizar Visita
                    </MudButton>
                    <MudButton Variant="Variant.Filled"
                               ButtonType="ButtonType.Button"
                               Color="Color.Error"
                               Class="mt-6 ml-2"
                               StartIcon="@Icons.Material.Filled.Cancel"
                               OnClick="OnClickCancelViewing">
                        Cancelar Visita
```

```
                    </MudButton>
                }
            </div>
        </EditForm>
    </MudPaper>
}
```

.\RealtyHub.Web\Components\Viewings\ViewingForm.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;
using RealtyHub.Web.Components.Customers;
using RealtyHub.Web.Components.Properties;
using System.ComponentModel.DataAnnotations;

namespace RealtyHub.Web.Components.Viewings;

/// <summary>
/// Componente responsável pelo formulário de agendamento e reagendamento de visitas.
/// </summary>
public partial class ViewingFormComponent : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador da visita. Se diferente de zero, o formulário entra em modo de
edição/reagendamento.
    /// </summary>
    [Parameter]
    public long Id { get; set; }

    /// <summary>
    /// Cliente associado à visita (opcional).
    /// </summary>
    [Parameter]
    public Customer? Customer { get; set; }

    /// <summary>
    /// Imóvel associado à visita (opcional).
    /// </summary>
    [Parameter]
    public Property? Property { get; set; }

    /// <summary>
    /// Indica se a busca de imóveis está bloqueada.
    /// </summary>
    [Parameter]
    public bool LockPropertySearch { get; set; }

    /// <summary>
    /// Indica se a busca de clientes está bloqueada.
    /// </summary>
    [Parameter]
    public bool LockCustomerSearch { get; set; }

    /// <summary>
    /// Indica se deve redirecionar para a listagem após submissão.
    /// </summary>
    [Parameter]
    public bool RedirectToPageList { get; set; } = true;

    /// <summary>
    /// Evento disparado ao clicar no botão de submissão do formulário.
    /// </summary>
    [Parameter]
    public EventCallback OnSubmitButtonClicked { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Indica a operação atual do formulário ("Reagendar" ou "Agendar").
    /// </summary>
    public string Operation => Id != 0
        ? "Reagendar" : "Agendar";
```

```
    /// <summary>
    /// Modelo de entrada utilizado para o binding dos campos do formulário.
    /// </summary>
    public Viewing InputModel { get; set; } = new();

    /// <summary>
    /// Horário selecionado para a visita.
    /// </summary>
    [DataType(DataType.Time)]
    public TimeSpan? ViewingTime { get; set; } = DateTime.Now.TimeOfDay;

    /// <summary>
    /// Indica se o formulário está em estado de carregamento ou processamento.
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Chave para forçar atualização do formulário.
    /// </summary>
    public int EditFormKey { get; set; }

    #endregion

    #region Services

    /// <summary>
    /// Handler responsável pelas operações de visitas.
    /// </summary>
    [Inject]
    public IViewingHandler ViewingHandler { get; set; } = null!;

    /// <summary>
    /// Serviço de navegação para redirecionamento de páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de mensagens e notificações.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Serviço de diálogo para exibição de modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário, realizando o agendamento ou reagendamento
da visita.
    /// </summary>
    /// <remarks>
    /// Valida os campos obrigatórios, ajusta a data/hora da visita, chama o handler para
agendar ou reagendar,
    /// exibe mensagens de sucesso ou erro via Snackbar, dispara o evento de submissão e
redireciona para a listagem se necessário.
    /// </remarks>
    public async Task OnValidSubmitAsync()
    {
        if (IsFormInvalid()) return;
        IsBusy = true;
        try
        {
            if (InputModel.ViewingDate.HasValue && ViewingTime.HasValue)
            {
                InputModel.ViewingDate = InputModel
                    .ViewingDate.Value.Date.Add(ViewingTime.Value)
                    .ToUniversalTime();
            }
```

```
            Response<Viewing?> result;
            if (Id == 0)
                result = await ViewingHandler.ScheduleAsync(InputModel);
            else
                result = await ViewingHandler.RescheduleAsync(InputModel);

            var resultMessage = result.Message ?? string.Empty;
            if (!result.IsSuccess)
            {
                Snackbar.Add(resultMessage, Severity.Error);
                return;
            }

            await OnSubmitButtonClickedAsync();
            Snackbar.Add(resultMessage, Severity.Success);
            if (RedirectToPageList)
                NavigationManager.NavigateTo("/visitas");
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    /// <summary>
    /// Valida se os campos obrigatórios do formulário foram preenchidos.
    /// </summary>
    /// <remarks>
    /// Exibe mensagens de erro via Snackbar caso cliente ou imóvel não estejam
informados.
    /// </remarks>
    /// <returns>True se o formulário for inválido, caso contrário false.</returns>
    private bool IsFormInvalid()
    {
        if (InputModel.Buyer is null && InputModel.Property is null)
        {
            Snackbar.Add("Informe o cliente e o imóvel", Severity.Error);
            return true;
        }
        if (InputModel.Buyer is null)
        {
            Snackbar.Add("Informe o cliente", Severity.Error);
            return true;
        }
        if (InputModel.Property is null)
        {
            Snackbar.Add("Informe o imóvel", Severity.Error);
            return true;
        }

        return false;
    }

    /// <summary>
    /// Carrega os dados da visita para edição a partir do ID informado.
    /// </summary>
    /// <remarks>
    /// Busca os dados da visita via handler, preenche o modelo de entrada e ajusta o
horário.
    /// Em caso de erro, exibe mensagem e redireciona para a listagem de visitas.
    /// </remarks>
    private async Task LoadViewingAsync()
    {
        GetViewingByIdRequest? request = null;
        try
        {
            request = new GetViewingByIdRequest { Id = Id };
        }
        catch
        {
            Snackbar.Add("Parâmetro inválido", Severity.Error);
        }
```

```
        if (request is null) return;

        var response = await ViewingHandler.GetByIdAsync(request);
        if (response is { IsSuccess: true, Data: not null })
        {
            InputModel.Id = response.Data.Id;
            InputModel.ViewingDate = response.Data.ViewingDate;
            InputModel.ViewingStatus = response.Data.ViewingStatus;
            InputModel.Buyer = response.Data.Buyer;
            InputModel.BuyerId = response.Data.BuyerId;
            InputModel.Property = response.Data.Property;
            InputModel.PropertyId = response.Data.PropertyId;
            InputModel.UserId = response.Data.UserId;
            ViewingTime = InputModel.ViewingDate!.Value.TimeOfDay;
        }
        else
        {
            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            NavigationManager.NavigateTo("/visitas");
        }
    }

    /// <summary>
    /// Marca a visita como realizada.
    /// </summary>
    /// <remarks>
    /// Chama o handler para finalizar a visita e atualiza o status no modelo.
    /// Exibe mensagem informativa via Snackbar.
    /// </remarks>
    public async Task OnClickDoneViewing()
    {
        var request = new DoneViewingRequest { Id = InputModel.Id };
        var result = await ViewingHandler.DoneAsync(request);
        var resultMessage = result.Message ?? string.Empty;
        if (result is { IsSuccess: true, Data: not null })
            InputModel.ViewingStatus = result.Data.ViewingStatus;

        Snackbar.Add(resultMessage, Severity.Info);
        StateHasChanged();
    }

    /// <summary>
    /// Cancela a visita.
    /// </summary>
    /// <remarks>
    /// Chama o handler para cancelar a visita e atualiza o status no modelo.
    /// Exibe mensagem informativa via Snackbar.
    /// </remarks>
    public async Task OnClickCancelViewing()
    {
        var request = new CancelViewingRequest { Id = InputModel.Id };
        var result = await ViewingHandler.CancelAsync(request);
        var resultMessage = result.Message ?? string.Empty;
        if (result is { IsSuccess: true, Data: not null })
            InputModel.ViewingStatus = result.Data.ViewingStatus;

        Snackbar.Add(resultMessage, Severity.Info);
        StateHasChanged();
    }

    /// <summary>
    /// Abre o diálogo para seleção do cliente.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo modal para seleção de um cliente. O diálogo só é aberto se <see
cref="LockCustomerSearch"/> for falso.
    /// </remarks>
    public async Task OpenCustomerDialog()
    {
        if (LockCustomerSearch) return;
        var parameters = new DialogParameters
        {
            { "OnCustomerSelected", EventCallback.Factory
                .Create<Customer>(this, SelectedCustomer) }
        };
```

```
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Large,
            FullWidth = true
        };
        var dialog = await DialogService
            .ShowAsync<CustomerDialog>("Informe o Cliente", parameters, options);
        var result = await dialog.Result;

        if (result is { Canceled: false, Data: Customer selectedCustomer })
            SelectedCustomer(selectedCustomer);
    }

    /// <summary>
    /// Manipula a seleção do cliente no diálogo, atualizando o modelo da visita.
    /// </summary>
    /// <param name="customer">Cliente selecionado.</param>
    private void SelectedCustomer(Customer customer)
    {
        InputModel.Buyer = customer;
        InputModel.BuyerId = customer.Id;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Abre o diálogo para seleção do imóvel.
    /// </summary>
    /// <remarks>
    /// Exibe um diálogo modal para seleção de um imóvel. O diálogo só é aberto se <see
cref="LockPropertySearch"/> for falso.
    /// </remarks>
    public async Task OpenPropertyDialog()
    {
        if (LockPropertySearch) return;
        var parameters = new DialogParameters
        {
            { "OnPropertySelected", EventCallback.Factory
                .Create<Property>(this, SelectedProperty) }
        };
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.ExtraLarge,
            FullWidth = true
        };
        var dialog = await DialogService
            .ShowAsync<PropertyDialog>("Informe o Imóvel", parameters, options);

        var result = await dialog.Result;

        if (result is { Canceled: false, Data: Property selectedProperty })
            SelectedProperty(selectedProperty);
    }

    /// <summary>
    /// Manipula a seleção do imóvel no diálogo, atualizando o modelo da visita.
    /// </summary>
    /// <param name="property">Imóvel selecionado.</param>
    private void SelectedProperty(Property property)
    {
        InputModel.Property = property;
        InputModel.PropertyId = property.Id;
        EditFormKey++;
        StateHasChanged();
    }

    /// <summary>
    /// Dispara o evento de clique no botão de submissão, se houver delegate.
    /// </summary>
    /// <remarks>
    /// Verifica se existe um delegate associado ao evento <see
cref="OnSubmitButtonClicked"/> e o executa de forma assíncrona.
    /// Permite que componentes pais sejam notificados após a submissão do formulário.
    /// </remarks>
```

```
    private async Task OnSubmitButtonClickedAsync()
    {
        if (OnSubmitButtonClicked.HasDelegate)
            await OnSubmitButtonClicked.InvokeAsync();
    }

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente, carregando os dados da visita para edição ou preparando
para agendamento.
    /// </summary>
    /// <remarks>
    /// Se <see cref="Id"/> for diferente de zero, busca os dados da visita para edição.
    /// Caso contrário, preenche os dados iniciais com o cliente e imóvel informados por
parâmetro.
    /// Exibe mensagens de erro via Snackbar em caso de falha.
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
        {
            if (Id != 0)
                await LoadViewingAsync();
            else
            {
                InputModel.Buyer = Customer;
                InputModel.BuyerId = Customer?.Id ?? 0;
                InputModel.Property = Property;
                InputModel.PropertyId = Property?.Id ?? 0;
            }
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Components\Viewings\ViewingStatus.razor

```
@using RealtyHub.Core.Extensions

@switch (Status)
{
    case EViewingStatus.Scheduled:
        <MudChip T="Enum" Color="Color.Primary">@Status.GetDisplayName()</MudChip>
        break;

    case EViewingStatus.Done:
        <MudChip T="Enum" Color="Color.Success">@Status.GetDisplayName()</MudChip>
        break;

    case EViewingStatus.Canceled:
        <MudChip T="Enum" Color="Color.Error">@Status.GetDisplayName()</MudChip>
        break;

}

@code
{
    [Parameter, EditorRequired]
    public EViewingStatus Status { get; set; }
}
```

.\RealtyHub.Web\Configuration.cs

```
using MudBlazor;

namespace RealtyHub.Web;

/// <summary>
/// Configurações globais da aplicação relativas ao backend, temas e logos.
/// </summary>
public static class Configuration
{
    /// <summary>
    /// Nome do cliente HTTP configurado para a aplicação.
    /// </summary>
    public const string HttpClientName = "realtyhub";

    /// <summary>
    /// URL do backend da aplicação.
    /// </summary>
    public static string BackendUrl { get; set; } = "http://localhost:5538";

    /// <summary>
    /// Caminho da logo atualmente utilizada na aplicação.
    /// </summary>
    public static string SrcLogo { get; set; } = SrcLogos.WhiteLogo;

    /// <summary>
    /// Nome do usuário fornecido após a autenticação.
    /// </summary>
    public static string GivenName { get; set; } = string.Empty;

    /// <summary>
    /// Tema da interface definido utilizando os componentes do MudBlazor.
    /// </summary>
    public static readonly MudTheme Theme = new()
    {
        Typography = new Typography
        {
            Default = new Default
            {
                FontFamily = ["Raleway", "sans-serif"]
            }
        },
        PaletteLight = new PaletteLight
        {
            TextPrimary = Colors.Shades.Black,
            DrawerText = Colors.Shades.Black
        },
        PaletteDark = new PaletteDark()
    };

    /// <summary>
    /// Define os caminhos para as logos disponíveis na aplicação.
    /// </summary>
    public static class SrcLogos
    {
        /// <summary>
        /// Caminho da logo branca sem fundo.
        /// </summary>
        public const string WhiteLogo = "/src/img/white-logo-nobg.png";

        /// <summary>
        /// Caminho da logo preta sem fundo.
        /// </summary>
        public const string BlackLogo = "/src/img/black-logo-nobg.png";
    }
}
```

.\RealtyHub.Web\Handlers\AccountHandler.cs

```
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models.Account;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Core.Responses;
using System.Net.Http.Json;
using System.Text;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável pelas operações de autenticação e gerenciamento de conta de
usuário na aplicação web.
/// </summary>
public class AccountHandler : IAccountHandler
{
    /// <summary>
    /// Cliente HTTP utilizado para realizar requisições à API.
    /// </summary>
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="AccountHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public AccountHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Realiza o login do usuário.
    /// </summary>
    /// <remarks>
    /// Envia a requisição de login para a API, utilizando cookies para autenticação.
    /// Retorna sucesso se o status HTTP for positivo, caso contrário retorna mensagem de
erro.
    /// </remarks>
    public async Task<Response<string>> LoginAsync(LoginRequest request)
    {
        var result = await _httpClient
            .PostAsJsonAsync("v1/identity/login?useCookies=true", request);

        return result.IsSuccessStatusCode
            ? new Response<string>()
            : new Response<string>(null, (int)result.StatusCode, "Não foi possível
realizar login");
    }

    /// <summary>
    /// Realiza o cadastro de um novo usuário.
    /// </summary>
    /// <remarks>
    /// Envia os dados de cadastro para a API. Retorna erro se o e-mail já estiver
cadastrado ou se a operação falhar.
    /// </remarks>
    public async Task<Response<string>> RegisterAsync(RegisterRequest request)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/identity/registeruser",
request);

        var content = await result.Content.ReadAsStringAsync();

        if (content.Contains("DuplicateUserName"))
            return new Response<string>(null, 400, "E-mail já cadastrado");

        if (!result.IsSuccessStatusCode)
            return new Response<string>(null, (int)result.StatusCode,
                "Não foi possível realizar o cadastro");
```

```
        var data = await result.Content.ReadFromJsonAsync<Response<string>>();

        return new Response<string>(null, 201, data?.Message);
    }

    /// <summary>
    /// Confirma o e-mail do usuário utilizando o token enviado por e-mail.
    /// </summary>
    /// <remarks>
    /// Valida o token de confirmação de e-mail recebido pela API. Retorna erro se o
token for inválido.
    /// </remarks>
    public async Task<Response<string>> ConfirmEmailAsync(ConfirmEmailRequest request)
    {
        var url =
$"v1/identity/confirm-email?userId={request.UserId}&token={request.Token}";
        var result = await _httpClient.GetAsync(url);

        var content = await result.Content.ReadAsStringAsync();

        if (content.Contains("InvalidToken"))
            return new Response<string>(null, 400, "Token inválido");

        var data = await result.Content.ReadFromJsonAsync<Response<string>>();

        return result.IsSuccessStatusCode
            ? new Response<string>(data?.Data, 200, data?.Message)
            : new Response<string>(data?.Data, (int)result.StatusCode, data?.Message);
    }

    /// <summary>
    /// Solicita redefinição de senha para o usuário.
    /// </summary>
    /// <remarks>
    /// Envia a requisição para a API para iniciar o processo de recuperação de senha.
Retorna erro se o usuário não for encontrado.
    /// </remarks>
    public async Task<Response<string>> ForgotPasswordAsync(ForgotPasswordRequest
request)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/identity/forgot-password",
request);
        var content = await result.Content.ReadAsStringAsync();
        if (content.Contains("Usuário não encontrado"))
            return new Response<string>(null, 404, "Usuário não encontrado");

        if (!result.IsSuccessStatusCode)
            return new Response<string>(null, (int)result.StatusCode,
                "Não foi possível redefinir a senha");

        var data = await result.Content.ReadFromJsonAsync<Response<string>>();
        return new Response<string>(null, 200, data?.Message);
    }

    /// <summary>
    /// Redefine a senha do usuário utilizando o token recebido por e-mail.
    /// </summary>
    /// <remarks>
    /// Envia a nova senha e o token para a API. Retorna erro se o token for inválido ou
se a operação falhar.
    /// </remarks>
    public async Task<Response<string>> ResetPasswordAsync(ResetPasswordRequest request)
    {
        var url =
$"v1/identity/reset-password?userId={request.UserId}&token={request.Token}";
        var result = await _httpClient.PostAsJsonAsync(url, request);
        var content = await result.Content.ReadAsStringAsync();

        if (content.Contains("InvalidToken"))
            return new Response<string>(null, 400, "Token inválido");

        var data = await result.Content.ReadFromJsonAsync<Response<string>>();

        return result.IsSuccessStatusCode
            ? new Response<string>(data?.Data, 200, data?.Message)
            : new Response<string>(data?.Data, (int)result.StatusCode, data?.Message);
```

```
    }

    /// <summary>
    /// Realiza o logout do usuário.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição para a API para encerrar a sessão do usuário autenticado.
    /// </remarks>
    public async Task LogoutAsync()
    {
        var emptyContent = new StringContent("{}", Encoding.UTF8, "application/json");
        await _httpClient.PostAsync("v1/identity/logout", emptyContent);
    }
}
```

.\RealtyHub.Web\Handlers\CondominiumHandler.cs

```
using System.Net.Http.Json;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Core.Responses;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável pelas operações de condomínio na aplicação web.
/// </summary>
public class CondominiumHandler : ICondominiumHandler
{
    /// <summary>
    /// Cliente HTTP utilizado para realizar requisições à API.
    /// </summary>
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="CondominiumHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public CondominiumHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Cria um novo condomínio.
    /// </summary>
    /// <remarks>
    /// Envia os dados do condomínio para a API e retorna o resultado da operação.
    /// </remarks>
    public async Task<Response<Condominium?>> CreateAsync(Condominium request)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/condominiums", request);

        return await result.Content.ReadFromJsonAsync<Response<Condominium?>>()
               ?? new Response<Condominium?>(null, 400, "Falha ao criar o condomínio");
    }

    /// <summary>
    /// Atualiza os dados de um condomínio existente.
    /// </summary>
    /// <remarks>
    /// Envia os dados atualizados do condomínio para a API e retorna o resultado da
operação.
    /// </remarks>
    public async Task<Response<Condominium?>> UpdateAsync(Condominium request)
    {
        var result = await _httpClient.PutAsJsonAsync($"v1/condominiums/{request.Id}",
request);

        return await result.Content.ReadFromJsonAsync<Response<Condominium?>>()
            ?? new Response<Condominium?>(null, 400, "Falha ao atualizar o condomínio");
    }

    /// <summary>
    /// Exclui um condomínio pelo ID.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição para a API para remover o condomínio especificado.
    /// </remarks>
    public async Task<Response<Condominium?>> DeleteAsync(DeleteCondominiumRequest
request)
    {
        var result = await _httpClient.DeleteAsync($"v1/condominiums/{request.Id}");

        return result.IsSuccessStatusCode
```

```
            ? new Response<Condominium?>(null, 200, "Condomínio excluído com sucesso")
            : new Response<Condominium?>(null, 400, "Falha ao excluir o condomínio");
    }

    /// <summary>
    /// Obtém os dados de um condomínio pelo ID.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar os dados do condomínio
especificado.
    /// </remarks>
    public async Task<Response<Condominium?>> GetByIdAsync(GetCondominiumByIdRequest
request)
    {
        var response = await _httpClient.GetAsync($"v1/condominiums/{request.Id}");

        return await response.Content.ReadFromJsonAsync<Response<Condominium?>>()
            ?? new Response<Condominium?>(null, 400, "Falha ao obter o condomínio");
    }

    /// <summary>
    /// Obtém uma lista paginada de condomínios, com suporte a filtros e busca.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar todos os condomínios, podendo
filtrar por termo de busca e outros critérios.
    /// </remarks>
    public async Task<PagedResponse<List<Condominium>?>>
GetAllAsync(GetAllCondominiumsRequest request)
    {
        var url =
$"v1/condominiums?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (!string.IsNullOrEmpty(request.FilterBy))
            url = $"{url}&filterBy={request.FilterBy}";

        if (!string.IsNullOrEmpty(request.SearchTerm))
            url = $"{url}&searchTerm={request.SearchTerm}";

        return await _httpClient.GetFromJsonAsync<PagedResponse<List<Condominium>?>>(url)
               ?? new PagedResponse<List<Condominium>?>(null, 400, "Falha ao obter os
condomínios");
    }
}
```

.\RealtyHub.Web\Handlers\ContractHandler.cs

```
using System.Net.Http.Json;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Core.Responses;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável pelas operações de contratos na aplicação web.
/// </summary>
public class ContractHandler : IContractHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ContractHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public ContractHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Cria um novo contrato.
    /// </summary>
    /// <remarks>
    /// Ajusta as datas do contrato para UTC, envia os dados para a API e retorna o
resultado da operação.
    /// </remarks>
    public async Task<Response<Contract?>> CreateAsync(Contract request)
    {
        request.IssueDate = request.IssueDate?.Date.ToUniversalTime();
        request.EffectiveDate = request.EffectiveDate?.Date.ToUniversalTime();
        request.TermEndDate = request.TermEndDate?.Date.ToUniversalTime();
        request.SignatureDate = request.SignatureDate?.Date.ToUniversalTime();

        var result = await _httpClient.PostAsJsonAsync("v1/contracts", request);

        return await result.Content.ReadFromJsonAsync<Response<Contract?>>()
               ?? new Response<Contract?>(null, 400, "Falha ao criar o contrato");
    }

    /// <summary>
    /// Atualiza os dados de um contrato existente.
    /// </summary>
    /// <remarks>
    /// Envia os dados atualizados do contrato para a API e retorna o resultado da
operação.
    /// </remarks>
    public async Task<Response<Contract?>> UpdateAsync(Contract request)
    {
        var result = await _httpClient.PutAsJsonAsync($"v1/contracts/{request.Id}",
request);

        return await result.Content.ReadFromJsonAsync<Response<Contract?>>()
            ?? new Response<Contract?>(null, 400, "Falha ao atualizar o contrato");
    }

    /// <summary>
    /// Exclui um contrato pelo ID.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição para a API para remover o contrato especificado.
    /// </remarks>
    public async Task<Response<Contract?>> DeleteAsync(DeleteContractRequest request)
    {
        var result = await _httpClient.DeleteAsync($"v1/contracts/{request.Id}");
```

```
        return result.IsSuccessStatusCode
            ? new Response<Contract?>(null, 200, "Contrato excluído com sucesso")
            : new Response<Contract?>(null, 400, "Falha ao excluir o contrato");
    }

    /// <summary>
    /// Obtém os dados de um contrato pelo ID.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar os dados do contrato especificado.
    /// </remarks>
    public async Task<Response<Contract?>> GetByIdAsync(GetContractByIdRequest request)
    {
        var result = await _httpClient.GetAsync($"v1/contracts/{request.Id}");

        return await result.Content.ReadFromJsonAsync<Response<Contract?>>()
               ?? new Response<Contract?>(null, 400, "Não foi possível obter o
contrato");
    }

    /// <summary>
    /// Obtém uma lista paginada de contratos.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar todos os contratos, com suporte a
paginação.
    /// </remarks>
    public async Task<PagedResponse<List<Contract>?>> GetAllAsync(GetAllContractsRequest
request)
    {
        var url =
$"v1/contracts?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        return await _httpClient.GetFromJsonAsync<PagedResponse<List<Contract>?>>(url)
               ?? new PagedResponse<List<Contract>?>(null, 400,
                   "Não foi possível obter os contratos");
    }
}
```

.\RealtyHub.Web\Handlers\ContractTemplateHandler.cs

```
using System.Net.Http.Json;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável pelas operações de modelos de contratos na aplicação web.
/// </summary>
public class ContractTemplateHandler : IContractTemplateHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ContractTemplateHandler"/> utilizando
a fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public ContractTemplateHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Obtém todos os modelos de contratos disponíveis.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar a lista de modelos de contratos.
    /// Retorna a lista ou uma mensagem de erro em caso de falha.
    /// </remarks>
    public async Task<Response<List<ContractTemplate>?>> GetAllAsync()
    {
        var response = await _httpClient.GetAsync("v1/contracts-templates");

        return await
response.Content.ReadFromJsonAsync<Response<List<ContractTemplate>?>>()
            ?? new Response<List<ContractTemplate>?>(null, 400, "Falha ao buscar os
contratos");
    }
}
```

.\RealtyHub.Web\Handlers\CustomerHandler.cs

```
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Core.Responses;
using System.Net.Http.Json;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável pelas operações de clientes na aplicação web.
/// </summary>
public class CustomerHandler : ICustomerHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="CustomerHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public CustomerHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Cria um novo cliente.
    /// </summary>
    /// <remarks>
    /// Envia os dados do cliente para a API e retorna o resultado da operação.
    /// </remarks>
    public async Task<Response<Customer?>> CreateAsync(Customer request)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/customers", request);

        return await result.Content.ReadFromJsonAsync<Response<Customer?>>()
               ?? new Response<Customer?>(null, 400, "Falha ao criar o cliente");
    }

    /// <summary>
    /// Atualiza os dados de um cliente existente.
    /// </summary>
    /// <remarks>
    /// Envia os dados atualizados do cliente para a API e retorna o resultado da
operação.
    /// </remarks>
    public async Task<Response<Customer?>> UpdateAsync(Customer request)
    {
        var result = await _httpClient.PutAsJsonAsync($"v1/customers/{request.Id}",
request);

        return await result.Content.ReadFromJsonAsync<Response<Customer?>>()
            ?? new Response<Customer?>(null, 400, "Falha ao atualizar o cliente");
    }

    /// <summary>
    /// Exclui um cliente pelo ID.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição para a API para remover o cliente especificado.
    /// </remarks>
    public async Task<Response<Customer?>> DeleteAsync(DeleteCustomerRequest request)
    {
        var result = await _httpClient.DeleteAsync($"v1/customers/{request.Id}");

        return result.IsSuccessStatusCode
            ? new Response<Customer?>(null, 200, "Cliente excluído com sucesso")
            : new Response<Customer?>(null, 400, "Falha ao excluir o cliente");
    }
```

```
    /// <summary>
    /// Obtém os dados de um cliente pelo ID.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar os dados do cliente especificado.
    /// </remarks>
    public async Task<Response<Customer?>> GetByIdAsync(GetCustomerByIdRequest request)
    {
        var response = await _httpClient.GetAsync($"v1/customers/{request.Id}");

        return await response.Content.ReadFromJsonAsync<Response<Customer?>>()
               ?? new Response<Customer?>(null, 400, "Falha ao obter o cliente");
    }

    /// <summary>
    /// Obtém uma lista paginada de clientes, com suporte a busca por termo.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição para a API para buscar todos os clientes, podendo filtrar
por termo de busca.
    /// </remarks>
    public async Task<PagedResponse<List<Customer>?>> GetAllAsync(GetAllCustomersRequest
request)
    {
        var url =
$"v1/customers?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (!string.IsNullOrEmpty(request.SearchTerm))
            url = $"{url}&searchTerm={request.SearchTerm}";

        return await _httpClient.GetFromJsonAsync<PagedResponse<List<Customer>?>>(url)
               ?? new PagedResponse<List<Customer>?>(null, 400, "Falha ao obter os
clientes");
    }
}
```

.\RealtyHub.Web\Handlers\OfferHandler.cs

```
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Core.Responses;
using System.Net.Http.Json;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável pelas operações relacionadas a propostas de compra de imóvel na
aplicação web.
/// </summary>
public class OfferHandler : IOfferHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="OfferHandler"/> utilizando a fábrica
de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public OfferHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Cria uma nova proposta.
    /// </summary>
    /// <remarks>
    /// Envia os dados da proposta para a API e retorna o resultado da operação.
    /// </remarks>
    public async Task<Response<Offer?>> CreateAsync(Offer request)
    {
        var result = await _httpClient
            .PostAsJsonAsync("v1/offers", request);

        return await result.Content.ReadFromJsonAsync<Response<Offer?>>()
               ?? new Response<Offer?>(null, 400, "Falha ao criar a proposta");
    }

    /// <summary>
    /// Atualiza os dados de uma proposta existente.
    /// </summary>
    /// <remarks>
    /// Envia os dados atualizados da proposta para a API e retorna o resultado da
operação.
    /// </remarks>
    public async Task<Response<Offer?>> UpdateAsync(Offer request)
    {
        var result = await _httpClient
            .PutAsJsonAsync($"v1/offers/{request.Id}", request);

        return await result.Content.ReadFromJsonAsync<Response<Offer?>>()
               ?? new Response<Offer?>(null, 400, "Falha ao atualizar a proposta");
    }

    /// <summary>
    /// Rejeita uma proposta.
    /// </summary>
    /// <remarks>
    /// Envia a requisição para rejeitar a proposta à API e retorna o resultado da
operação.
    /// </remarks>
    public async Task<Response<Offer?>> RejectAsync(RejectOfferRequest request)
    {
        var result = await _httpClient
            .PutAsJsonAsync($"v1/offers/{request.Id}/reject", request);

        return await result.Content.ReadFromJsonAsync<Response<Offer?>>()
```

```
               ?? new Response<Offer?>(null, 400, "Falha ao rejeitar a proposta");
    }

    /// <summary>
    /// Aceita uma proposta.
    /// </summary>
    /// <remarks>
    /// Envia a requisição para aceitar a proposta à API e retorna o resultado da
operação.
    /// </remarks>
    public async Task<Response<Offer?>> AcceptAsync(AcceptOfferRequest request)
    {
        var result = await _httpClient
            .PutAsJsonAsync($"v1/offers/{request.Id}/accept", request);

        return await result.Content.ReadFromJsonAsync<Response<Offer?>>()
               ?? new Response<Offer?>(null, 400, "Falha ao aceitar a proposta");
    }

    /// <summary>
    /// Obtém os dados de uma proposta pelo ID.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição GET à API para buscar os dados da proposta especificada.
    /// </remarks>
    public async Task<Response<Offer?>> GetByIdAsync(GetOfferByIdRequest request)
    {
        var response = await _httpClient.GetAsync($"v1/offers/{request.Id}");

        return await response.Content.ReadFromJsonAsync<Response<Offer?>>()
               ?? new Response<Offer?>(null, 400, "Falha ao obter a proposta");
    }

    /// <summary>
    /// Obtém a proposta aceita para um determinado imóvel.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição GET à API para buscar a proposta aceita associada a um
imóvel.
    /// </remarks>
    public async Task<Response<Offer?>> GetAcceptedByProperty(GetOfferAcceptedByProperty
request)
    {
        var url = $"v1/offers/property/{request.PropertyId}/accepted";
        var response = await _httpClient.GetAsync(url);

        return await response.Content.ReadFromJsonAsync<Response<Offer?>>()
               ?? new Response<Offer?>(null, 400, "Falha ao obter a proposta aceita");
    }

    /// <summary>
    /// Obtém uma lista paginada de propostas para um imóvel específico.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição GET à API com parâmetros de paginação e, opcionalmente,
intervalo de datas para filtrar as propostas.
    /// </remarks>
    public async Task<PagedResponse<List<Offer>?>> GetAllOffersByPropertyAsync(
        GetAllOffersByPropertyRequest request)
    {
        var url = $"v1/offers/property/{request.PropertyId}" +
                  $"?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (request.StartDate is not null & request.EndDate is not null)
            url = $"{url}&startDate={request.StartDate}&endDate={request.EndDate}";

        var response = await _httpClient.GetAsync(url);

        return await response.Content.ReadFromJsonAsync<PagedResponse<List<Offer>?>>()
               ?? new PagedResponse<List<Offer>?>(null, 400, "Falha ao obter as
propostas");
    }

    /// <summary>
    /// Obtém uma lista paginada de propostas associadas a um cliente.
    /// </summary>
```

```
    /// <remarks>
    /// Envia uma requisição GET à API com parâmetros de paginação e, opcionalmente,
intervalo de datas para filtrar as propostas do cliente.
    /// </remarks>
    public async Task<PagedResponse<List<Offer>?>> GetAllOffersByCustomerAsync(
        GetAllOffersByCustomerRequest request)
    {
        var url = $"v1/offers/customer/{request.CustomerId}?" +
                  $"pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (request.StartDate is not null & request.EndDate is not null)
            url = $"{url}&startDate={request.StartDate}&endDate={request.EndDate}";

        var response = await _httpClient.GetAsync(url);

        return await response.Content.ReadFromJsonAsync<PagedResponse<List<Offer>?>>()
               ?? new PagedResponse<List<Offer>?>(null, 400, "Falha ao obter as
propostas");
    }

    /// <summary>
    /// Obtém uma lista paginada de todas as propostas.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição GET à API para buscar todas as propostas com suporte a
paginação e, opcionalmente, filtro por intervalo de datas.
    /// </remarks>
    public async Task<PagedResponse<List<Offer>?>> GetAllAsync(GetAllOffersRequest
request)
    {
        var url =
$"v1/offers?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (request.StartDate is not null & request.EndDate is not null)
            url = $"{url}&startDate={request.StartDate}&endDate={request.EndDate}";

        var response = await _httpClient.GetAsync(url);

        return await response.Content.ReadFromJsonAsync<PagedResponse<List<Offer>?>>()
               ?? new PagedResponse<List<Offer>?>(null, 400, "Falha ao obter as
propostas");
    }
}
```

.\RealtyHub.Web\Handlers\PropertyHandler.cs

```
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Responses;
using System.Net.Http.Json;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável por gerenciar as operações de imóveis na aplicação web.
/// </summary>
public class PropertyHandler : IPropertyHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="PropertyHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public PropertyHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Cria um novo imóvel.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição POST para a API com os dados do imóvel e retorna o resultado
da operação.
    /// Caso a resposta não possa ser convertida, retorna um objeto de resposta com erro.
    /// </remarks>
    public async Task<Response<Property?>> CreateAsync(Property request)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/properties", request);

        return await result.Content.ReadFromJsonAsync<Response<Property?>>()
               ?? new Response<Property?>(null, 400, "Falha ao criar o imóvel");
    }

    /// <summary>
    /// Atualiza os dados de um imóvel existente.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição PUT para a API com os dados atualizados do imóvel
identificado por seu ID.
    /// Retorna o resultado da operação ou uma resposta de erro se a atualização falhar.
    /// </remarks>
    public async Task<Response<Property?>> UpdateAsync(Property request)
    {
        var result = await _httpClient.PutAsJsonAsync($"v1/properties/{request.Id}",
request);

        return await result.Content.ReadFromJsonAsync<Response<Property?>>()
            ?? new Response<Property?>(null, 400, "Falha ao atualizar o imóvel");
    }

    /// <summary>
    /// Exclui um imóvel pelo ID.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição DELETE para a API utilizando o ID do imóvel a ser excluído
    /// e retorna o resultado da operação.
    /// </remarks>
    public async Task<Response<Property?>> DeleteAsync(DeletePropertyRequest request)
    {
        var result = await _httpClient.DeleteAsync($"v1/properties/{request.Id}");

        return result.IsSuccessStatusCode
```

```
            ? new Response<Property?>(null, 200, "Imóvel excluído com sucesso")
            : new Response<Property?>(null, 400, "Falha ao excluir o imóvel");
    }

    /// <summary>
    /// Obtém os dados de um imóvel pelo ID.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição GET para a API com o ID do imóvel e retorna os detalhes do
imóvel.
    /// Se a operação falhar, retorna uma resposta de erro.
    /// </remarks>
    public async Task<Response<Property?>> GetByIdAsync(GetPropertyByIdRequest request)
    {
        var response = await _httpClient.GetAsync($"v1/properties/{request.Id}");

        return await response.Content.ReadFromJsonAsync<Response<Property?>>()
            ?? new Response<Property?>(null, 400, "Falha ao obter o imóvel");
    }

    /// <summary>
    /// Obtém uma lista paginada de imóveis.
    /// </summary>
    /// <remarks>
    /// Constrói a URL com parâmetros de paginação, filtro e termo de busca, e realiza
uma requisição GET à API.
    /// Retorna um <see cref="PagedResponse{List{Property}?}"/> contendo os imóveis ou
uma mensagem de erro em caso de falha.
    /// </remarks>
    public async Task<PagedResponse<List<Property>?>> GetAllAsync(GetAllPropertiesRequest
request)
    {
        var url =
$"v1/properties?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (!string.IsNullOrEmpty(request.FilterBy))
            url = $"{url}&filterBy={request.FilterBy}";

        if (!string.IsNullOrEmpty(request.SearchTerm))
            url = $"{url}&searchTerm={request.SearchTerm}";

        return await _httpClient.GetFromJsonAsync<PagedResponse<List<Property>?>>(url)
               ?? new PagedResponse<List<Property>?>(null, 400, "Falha ao obter os
imóveis");
    }

    /// <summary>
    /// Obtém uma lista paginada de visitas associadas a um imóvel.
    /// </summary>
    /// <remarks>
    /// Constrói a URL com parâmetros de paginação, termo de busca, filtro e intervalo de
datas,
    /// e realiza uma requisição GET à API para obter as visitas do imóvel especificado.
    /// Retorna um <see cref="PagedResponse{List{Viewing}?}"/> com as visitas ou uma
mensagem de erro se a operação falhar.
    /// </remarks>
    public async Task<PagedResponse<List<Viewing>?>>
GetAllViewingsAsync(GetAllViewingsByPropertyRequest request)
    {
        var url = $"v1/properties/{request.PropertyId}/viewings?" +
                  $"pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (!string.IsNullOrEmpty(request.SearchTerm))
            url = $"{url}&searchTerm={request.SearchTerm}";

        if (!string.IsNullOrEmpty(request.FilterBy))
            url = $"{url}&filterBy={request.FilterBy}";

        if (request.StartDate is not null & request.EndDate is not null)
            url = $"{url}&startDate={request.StartDate}&endDate={request.EndDate}";

        var response = await _httpClient.GetAsync(url);

        return await response.Content.ReadFromJsonAsync<PagedResponse<List<Viewing>?>>()
               ?? new PagedResponse<List<Viewing>?>(null, 400, "Falha ao obter as
visitas");
```

```
    }
}
```

.\RealtyHub.Web\Handlers\PropertyPhotosHandler.cs

```
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.PropertiesPhotos;
using RealtyHub.Core.Responses;
using System.Net.Http.Headers;
using System.Net.Http.Json;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável por gerenciar as operações relacionadas às fotos de propriedades
na aplicação web.
/// </summary>
public class PropertyPhotosHandler : IPropertyPhotosHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="PropertyPhotosHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public PropertyPhotosHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Cria novas fotos para uma propriedade.
    /// </summary>
    /// <remarks>
    /// Valida se há arquivos para upload. Em seguida, cria um conteúdo
multipart/form-data com os arquivos,
    /// incluindo cabeçalhos indicando se a foto é thumbnail e o seu identificador.
    /// Envia os dados para a API e retorna o resultado da operação.
    /// </remarks>
    public async Task<Response<PropertyPhoto?>> CreateAsync(CreatePropertyPhotosRequest
request)
    {
        if (request.FileBytes is null || request.FileBytes.Count == 0)
            return new Response<PropertyPhoto?>(null, 400, "Nenhum arquivo encontrado");

        using var content = new MultipartFormDataContent();
        foreach (var fileData in request.FileBytes)
        {
            var fileContent = new ByteArrayContent(fileData.Content);
            fileContent.Headers.ContentType = new
MediaTypeHeaderValue(fileData.ContentType);
            fileContent.Headers.Add("IsThumbnail", fileData.IsThumbnail.ToString());
            fileContent.Headers.Add("Id", $"{fileData.Id}");
            content.Add(fileContent, "photos", fileData.Name);
        }

        var url = $"/v1/properties/{request.PropertyId}/photos";
        var response = await _httpClient.PostAsync(url, content);

        return await response.Content.ReadFromJsonAsync<Response<PropertyPhoto?>>()
               ?? new Response<PropertyPhoto?>(null, 400, "Falha ao adicionar as fotos");
    }

    /// <summary>
    /// Atualiza as fotos de uma propriedade.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição PUT para a API com as informações das fotos a serem
atualizadas.
    /// Retorna a lista atualizada de fotos ou uma resposta de erro em caso de falha.
    /// </remarks>
    public async Task<Response<List<PropertyPhoto>?>>
UpdateAsync(UpdatePropertyPhotosRequest request)
```

```
    {
        var url = $"/v1/properties/{request.PropertyId}/photos";
        var result = await _httpClient.PutAsJsonAsync(url, request);

        return await result.Content.ReadFromJsonAsync<Response<List<PropertyPhoto>?>>()
               ?? new Response<List<PropertyPhoto>?>(null, 400, "Falha ao atualizar as
fotos");
    }

    /// <summary>
    /// Exclui uma foto de uma propriedade.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição DELETE para a API utilizando o ID da propriedade e o ID da
foto.
    /// Retorna uma resposta indicando sucesso ou falha na exclusão.
    /// </remarks>
    public async Task<Response<PropertyPhoto?>> DeleteAsync(DeletePropertyPhotoRequest
request)
    {
        var url = $"/v1/properties/{request.PropertyId}/photos/{request.Id}";
        var response = await _httpClient.DeleteAsync(url);

        return response.IsSuccessStatusCode
            ? new Response<PropertyPhoto?>(null, 204)
            : new Response<PropertyPhoto?>(null, 400, "Falha ao deletar a foto");
    }

    /// <summary>
    /// Obtém todas as fotos associadas a uma propriedade.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição GET para a API utilizando o ID da propriedade e retorna a
lista de fotos.
    /// Em caso de falha, retorna uma resposta de erro.
    /// </remarks>
    public async Task<Response<List<PropertyPhoto>?>> GetAllByPropertyAsync(
        GetAllPropertyPhotosByPropertyRequest request)
    {
        var url = $"/v1/properties/{request.PropertyId}/photos";
        var response = await _httpClient.GetAsync(url);

        return await response.Content.ReadFromJsonAsync<Response<List<PropertyPhoto>?>>()
               ?? new Response<List<PropertyPhoto>?>(null, 400, "Falha ao buscar as
fotos");
    }
}
```

.\RealtyHub.Web\Handlers\ViewingHandler.cs

```
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Core.Responses;
using System.Net.Http.Json;

namespace RealtyHub.Web.Handlers;

/// <summary>
/// Handler responsável por gerenciar as operações relacionadas a visitas na aplicação
web.
/// </summary>
public class ViewingHandler : IViewingHandler
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ViewingHandler"/> utilizando a
fábrica de HttpClient.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica de <see cref="IHttpClientFactory"/> para
criar o cliente HTTP.</param>
    public ViewingHandler(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Agenda uma nova visita.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição POST para a API para criar uma nova visita com os dados
informados.
    /// Retorna a resposta contendo os detalhes da visita ou uma mensagem de erro em caso
de falha.
    /// </remarks>
    public async Task<Response<Viewing?>> ScheduleAsync(Viewing request)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/viewings", request);
        return await result.Content.ReadFromJsonAsync<Response<Viewing?>>()
               ?? new Response<Viewing?>(null, 400, "Falha ao agendar a visita");
    }

    /// <summary>
    /// Reagenda uma visita existente.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição PUT para a API para atualizar os dados da visita e
reagendar.
    /// Retorna os detalhes da visita reagendada ou uma mensagem de erro em caso de
falha.
    /// </remarks>
    public async Task<Response<Viewing?>> RescheduleAsync(Viewing request)
    {
        var result = await
_httpClient.PutAsJsonAsync($"v1/viewings/{request.Id}/reschedule", request);
        return await result.Content.ReadFromJsonAsync<Response<Viewing?>>()
               ?? new Response<Viewing?>(null, 400, "Falha ao reagendar a visita");
    }

    /// <summary>
    /// Marca uma visita como finalizada.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição PUT para a API para atualizar o status da visita como
finalizada.
    /// Retorna os detalhes da visita finalizada ou uma mensagem de erro em caso de
falha.
    /// </remarks>
    public async Task<Response<Viewing?>> DoneAsync(DoneViewingRequest request)
    {
```

```
        var result = await _httpClient.PutAsJsonAsync($"v1/viewings/{request.Id}/done",
request);
        return await result.Content.ReadFromJsonAsync<Response<Viewing?>>()
               ?? new Response<Viewing?>(null, 400, "Falha ao finalizar a visita");
    }

    /// <summary>
    /// Cancela uma visita.
    /// </summary>
    /// <remarks>
    /// Envia uma requisição PUT para a API para atualizar o status da visita como
cancelada.
    /// Retorna os detalhes da visita cancelada ou uma mensagem de erro em caso de falha.
    /// </remarks>
    public async Task<Response<Viewing?>> CancelAsync(CancelViewingRequest request)
    {
        var result = await _httpClient.PutAsJsonAsync($"v1/viewings/{request.Id}/cancel",
request);
        return await result.Content.ReadFromJsonAsync<Response<Viewing?>>()
               ?? new Response<Viewing?>(null, 400, "Falha ao cancelar a visita");
    }

    /// <summary>
    /// Obtém os detalhes de uma visita pelo ID.
    /// </summary>
    /// <remarks>
    /// Realiza uma requisição GET para a API utilizando o ID da visita e retorna os
detalhes correspondentes.
    /// Se a operação falhar, retorna uma resposta de erro.
    /// </remarks>
    public async Task<Response<Viewing?>> GetByIdAsync(GetViewingByIdRequest request)
    {
        var response = await _httpClient.GetAsync($"v1/viewings/{request.Id}");
        return await response.Content.ReadFromJsonAsync<Response<Viewing?>>()
            ?? new Response<Viewing?>(null, 400, "Falha ao obter a visita");
    }

    /// <summary>
    /// Obtém uma lista paginada de visitas.
    /// </summary>
    /// <remarks>
    /// Constrói a URL com parâmetros de paginação, termo de busca e, opcionalmente,
intervalo de datas.
    /// Realiza uma requisição GET para a API e retorna a lista de visitas ou uma
mensagem de erro em caso de falha.
    /// </remarks>
    public async Task<PagedResponse<List<Viewing>?>> GetAllAsync(GetAllViewingsRequest
request)
    {
        var url =
$"v1/viewings?pageNumber={request.PageNumber}&pageSize={request.PageSize}";

        if (!string.IsNullOrEmpty(request.SearchTerm))
            url = $"{url}&searchTerm={request.SearchTerm}";

        if (request.StartDate is not null & request.EndDate is not null)
            url = $"{url}&startDate={request.StartDate}&endDate={request.EndDate}";

        var response = await _httpClient.GetAsync(url);
        return await response.Content.ReadFromJsonAsync<PagedResponse<List<Viewing>?>>()
               ?? new PagedResponse<List<Viewing>?>(null, 400, "Não foi possível obter as
visitas");
    }
}
```

.\RealtyHub.Web\Layout\HeadlessLayout.razor

```
@inherits LayoutComponentBase

<MudThemeProvider Theme="Configuration.Theme"/>
<MudSnackbarProvider />

<MudLayout>
    <CascadingValue Value="@Configuration.SrcLogo" Name="LogoCascading">
        <MudMainContent>
            <MudContainer>
                @Body
            </MudContainer>
        </MudMainContent>
    </CascadingValue>
</MudLayout>
```

.\RealtyHub.Web\Layout\MainLayout.razor

```
@using Blazored.LocalStorage
@inherits LayoutComponentBase
@inject ILocalStorageService LocalStorage

<MudThemeProvider @ref="_mudThemeProvider"
                  @bind-IsDarkMode="IsDarkMode"
                  Theme="Configuration.Theme" />
<MudSnackbarProvider />
<MudDialogProvider />
<MudPopoverProvider />

<AuthorizeView>
    <Authorized>
        <MudLayout>
            <CascadingValue Value="@Configuration.SrcLogo" Name="LogoCascading">
                <MudAppBar>
                    <MudTooltip Text="Abir/Fechar Menu de Navegação">
                        <MudIconButton Icon="@Icons.Material.Filled.Menu"
                                       Color="Color.Inherit"
                                       Edge="Edge.Start"
                                       OnClick="ToggleDrawer" />
                    </MudTooltip>
                    RealtyHub
                    <MudSpacer />
                    <MudTooltip Text="Mudar tema">
                        <MudSwitch Color="Color.Inherit"
                                   Value="IsDarkMode"
                                   ValueChanged="OnThemeToggled"
                                   T="bool"
                                   ThumbIcon="@Icons.Material.TwoTone.DarkMode"
                                   Class="ma-4"
                                   Disabled="DisableMudSwitch" />
                    </MudTooltip>

                    <MudTooltip Text="Logout">
                        <MudIconButton Icon="@Icons.Material.Filled.Logout"
                                       Color="Color.Inherit"
                                       Edge="Edge.Start"
                                       OnClick="LogoutAsync" />
                    </MudTooltip>
                </MudAppBar>
                <MudDrawer @bind-Open="@_isDrawerOpened">
                    <NavMenu @key="MudNavMenuKey" />
                </MudDrawer>
                <MudMainContent>
                    <MudContainer MaxWidth="MaxWidth.ExtraLarge" Class="mt-4">
                        @Body
                    </MudContainer>
                </MudMainContent>
            </CascadingValue>
        </MudLayout>
    </Authorized>
    <NotAuthorized>
        <HomeRedirect />
    </NotAuthorized>
</AuthorizeView>

@code
{
    public bool SaveState { get; set; }
    public bool IsDarkMode { get; set; }
    public bool DisableMudSwitch { get; set; }
    public bool _isDrawerOpened { get; set; } = true;
    public int MudNavMenuKey { get; set; }
    private MudThemeProvider _mudThemeProvider = null!;

    [Inject] public NavigationManager NavigationManager { get; set; } = null!;
    [Inject] public IDialogService DialogService { get; set; } = null!;

    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
```

```
        {
            var storedTheme = await LocalStorage.GetItemAsync<bool?>("themePreference");
            if (storedTheme.HasValue)
            {
                IsDarkMode = storedTheme.Value;
            }
            else
            {
                IsDarkMode = await _mudThemeProvider.GetSystemPreference();
            }
            await LocalStorage.SetItemAsync<bool?>("themePreference", IsDarkMode);
            StateHasChanged();
        }
        Configuration.SrcLogo = IsDarkMode
            ? Configuration.SrcLogos.BlackLogo
            : Configuration.SrcLogos.WhiteLogo;
        MudNavMenuKey++;
        SaveState = true;
    }

    private void ToggleDrawer()
    {
        _isDrawerOpened = !_isDrawerOpened;
        SaveState = true;
    }

    private async Task OnThemeToggled(bool newValue)
    {
        IsDarkMode = newValue;
        if (!SaveState) return;
        await LocalStorage.SetItemAsync("themePreference", IsDarkMode);
        StateHasChanged();
    }

    public void UpdateStatusDark(bool isDarkMode, bool disableMudSwtich)
    {
        IsDarkMode = isDarkMode;
        SaveState = false;
        DisableMudSwitch = disableMudSwtich;
        StateHasChanged();
    }

    public async Task LogoutAsync()
    {
        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja realmente sair do sistema?" },
            { "ButtonColor", Color.Error }
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        var result = await dialog.Result;

        if (result is { Canceled: true }) return;
        NavigationManager.NavigateTo("/sair");
    }
}
```

.\RealtyHub.Web\Layout\MainLayout.razor.css

```
.page {
    position: relative;
    display: flex;
    flex-direction: column;
}

main {
    flex: 1;
}

.sidebar {
    background-image: linear-gradient(180deg, rgb(5, 39, 103) 0%, #3a0647 70%);
}

.top-row {
    background-color: #f7f7f7;
    border-bottom: 1px solid #d6d5d5;
    justify-content: flex-end;
    height: 3.5rem;
    display: flex;
    align-items: center;
}

    .top-row ::deep a, .top-row .btn-link {
        white-space: nowrap;
        margin-left: 1.5rem;
    }

    .top-row a:first-child {
        overflow: hidden;
        text-overflow: ellipsis;
    }

@media (max-width: 640.98px) {
    .top-row:not(.auth) {
        display: none;
    }

    .top-row.auth {
        justify-content: space-between;
    }

    .top-row a, .top-row .btn-link {
        margin-left: 0;
    }
}

@media (min-width: 641px) {
    .page {
        flex-direction: row;
    }

    .sidebar {
        width: 250px;
        height: 100vh;
        position: sticky;
        top: 0;
    }

    .top-row {
        position: sticky;
        top: 0;
        z-index: 1;
    }

    .top-row, article {
        padding-left: 2rem !important;
        padding-right: 1.5rem !important;
    }
}
```

.\RealtyHub.Web\Layout\PublicLayout.razor

```
@using Blazored.LocalStorage
@inherits LayoutComponentBase
@inject ILocalStorageService LocalStorage

<MudThemeProvider @ref="_mudThemeProvider"
                  @bind-IsDarkMode="IsDarkMode"
                  Theme="Configuration.Theme" />
<MudSnackbarProvider />
<MudDialogProvider />
<MudPopoverProvider />

<MudLayout>
    <CascadingValue Value="@Configuration.SrcLogo" Name="LogoCascading">
        <MudAppBar>
            RealtyHub
            <MudButton Class="ml-4"
                       Variant="Variant.Text"
                       Style="color:white"
                       Href="/home">
                Home
            </MudButton>
            <MudButton Class="ml-4"
                       Variant="Variant.Text"
                       Style="color:white"
                       Href="/listar-imoveis">
                Imóveis
            </MudButton>
            <MudSpacer />
            <MudTooltip Text="Realizar login">
                <MudIconButton Class="mr-4"
                               Icon="@Icons.Material.Filled.Login"
                               Color="Color.Inherit"
                               Edge="Edge.Start"
                               Href="/login" />
            </MudTooltip>
            <MudTooltip Text="Registrar-se">
                <MudIconButton Class="mr-4"
                               Icon="@Icons.Material.Filled.PersonAdd"
                               Color="Color.Inherit"
                               Edge="Edge.Start"
                               Href="/comecar" />
            </MudTooltip>
            <MudTooltip Text="Mudar tema">
                <MudSwitch Color="Color.Inherit"
                           Value="IsDarkMode"
                           ValueChanged="OnThemeToggled"
                           T="bool"
                           ThumbIcon="@Icons.Material.TwoTone.DarkMode"
                           Class="ma-4"
                           Disabled="DisableMudSwitch" />
            </MudTooltip>
        </MudAppBar>
        <MudMainContent>
            <MudContainer MaxWidth="MaxWidth.ExtraLarge" Class="mt-4">
                @Body
            </MudContainer>
        </MudMainContent>
    </CascadingValue>
</MudLayout>

@code
{
    public bool SaveState { get; set; }
    public bool IsDarkMode { get; set; }
    public bool DisableMudSwitch { get; set; }

    private MudThemeProvider _mudThemeProvider = null!;

    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
```

```
            var storedTheme = await LocalStorage.GetItemAsync<bool?>("themePreference");
            if (storedTheme.HasValue)
            {
                IsDarkMode = storedTheme.Value;
            }
            else
            {
                IsDarkMode = await _mudThemeProvider.GetSystemPreference();
            }
            await LocalStorage.SetItemAsync<bool?>("themePreference", IsDarkMode);
            StateHasChanged();
        }
        Configuration.SrcLogo = IsDarkMode
            ? Configuration.SrcLogos.BlackLogo
            : Configuration.SrcLogos.WhiteLogo;
        SaveState = true;
    }

    private async Task OnThemeToggled(bool newValue)
    {
        IsDarkMode = newValue;
        if (!SaveState) return;
        await LocalStorage.SetItemAsync("themePreference", IsDarkMode);
        StateHasChanged();
    }

    public void UpdateStatusDark(bool isDarkMode, bool disableMudSwtich)
    {
        IsDarkMode = isDarkMode;
        SaveState = false;
        DisableMudSwitch = disableMudSwtich;
        StateHasChanged();
    }
}
```

.\RealtyHub.Web\Layout\PublicLayout.razor.css

```
.page {
    position: relative;
    display: flex;
    flex-direction: column;
}

main {
    flex: 1;
}

.sidebar {
    background-image: linear-gradient(180deg, rgb(5, 39, 103) 0%, #3a0647 70%);
}

.top-row {
    background-color: #f7f7f7;
    border-bottom: 1px solid #d6d5d5;
    justify-content: flex-end;
    height: 3.5rem;
    display: flex;
    align-items: center;
}

    .top-row ::deep a, .top-row .btn-link {
        white-space: nowrap;
        margin-left: 1.5rem;
    }

    .top-row a:first-child {
        overflow: hidden;
        text-overflow: ellipsis;
    }

@media (max-width: 640.98px) {
    .top-row:not(.auth) {
        display: none;
    }

    .top-row.auth {
        justify-content: space-between;
    }

    .top-row a, .top-row .btn-link {
        margin-left: 0;
    }
}

@media (min-width: 641px) {
    .page {
        flex-direction: row;
    }

    .sidebar {
        width: 250px;
        height: 100vh;
        position: sticky;
        top: 0;
    }

    .top-row {
        position: sticky;
        top: 0;
        z-index: 1;
    }

    .top-row, article {
        padding-left: 2rem !important;
        padding-right: 1.5rem !important;
    }
}
```

.\RealtyHub.Web\Pages\Condominiums\Create.razor

```
@page "/condominios/adicionar"
@inherits CondominiumFormComponent

<CondominiumForm />
```

.\RealtyHub.Web\Pages\Condominiums\Edit.razor

```
@page "/condominios/editar/{id:long}"
@inherits CondominiumFormComponent

<CondominiumForm Id="@Id" />
```

.\RealtyHub.Web\Pages\Condominiums\List.razor

```
@page "/condominios"
@using RealtyHub.Core.Models
@inherits ListCondominiumsPage

<MudText Typo="Typo.h3">Condomínios</MudText>

<PageTitle>Condomínios</PageTitle>

<div class="d-flex justify-end">
    <MudButton Variant="Variant.Filled"
               Color="Color.Primary"
               StartIcon="fa-solid fa-building-circle-arrow-right"
               Href="/condominios/adicionar">
        Novo Condomínio
    </MudButton>
</div>

<MudDataGrid T="Condominium"
             Class="mt-4"
             ServerData="LoadServerData"
             @ref="DataGrid"
             RowsPerPage="@Core.Configuration.DefaultPageSize"
             RowClick="@(e => SelectCondominium(e.Item))"
             RowStyle="@RowStyle">

    <ToolBarContent>
        <MudText Typo="Typo.h6">Imóveis</MudText>
        <MudSpacer />
        <MudSelect T="string"
                   Label="Selecione o Filtro"
                   Value="SelectedFilter"
                   ValueChanged="OnValueFilterChanged"
                   Class="me-2">
            @foreach (var option in FilterOptions)
            {
                <MudSelectItem
Value="@option.PropertyName">@option.DisplayName</MudSelectItem>
            }
        </MudSelect>
        <MudTextField @bind-Value="SearchTerm"
                      Placeholder="Filtrar..."
                      Adornment="Adornment.Start"
                      AdornmentIcon="@Icons.Material.Filled.Search"
                      Clearable="true"
                      OnClearButtonClick="OnClearSearchClick"
                      IconSize="Size.Medium"
                      Class="me-2">
        </MudTextField>
        <MudButton StartIcon="@Icons.Material.Filled.Search"
                   OnClick="OnButtonSearchClick"
                   ButtonType="ButtonType.Button"
                   Color="Color.Primary"
                   Variant="Variant.Filled">
            Pesquisar
        </MudButton>
    </ToolBarContent>
    <Columns>
        <PropertyColumn Property="x=>x.Id" Title="Id" />
        <PropertyColumn Property="x=>x.Name" Title="Nome" />
        <PropertyColumn Property="x=>x.Address.State" Title="Estado" />
        <PropertyColumn Property="x=>x.Address.City" Title="Cidade" />
        <PropertyColumn Property="x=>x.Address.Neighborhood" Title="Bairro" />
        <TemplateColumn Class="justify-end" Title="Ações">
            <CellTemplate>
                <MudStack Row>
                    <MudTooltip Text="Editar">
                        <MudIconButton Icon="@Icons.Material.Filled.Edit"
                                       Href="@($"/condominios/editar/{context.Item.Id}")"
                                       Color="Color.Primary">
                        </MudIconButton>
                    </MudTooltip>
                    <MudTooltip Text="Excluir">
```

```
                        <MudIconButton Icon="@Icons.Material.Filled.Delete"
                                       Color="Color.Error"
                                       aria-label="Excluir"
                                       OnClick="() =>
OnDeleteButtonClickedAsync(context.Item.Id, context.Item.Name)">
                        </MudIconButton>
                    </MudTooltip>
                </MudStack>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Condominium" />
    </PagerContent>
</MudDataGrid>
```

.\RealtyHub.Web\Pages\Condominiums\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Condominiums;
using RealtyHub.Web.Components.Common;

namespace RealtyHub.Web.Pages.Condominiums;

/// <summary>
/// Página responsável pela listagem e gerenciamento de condomínios na aplicação.
/// </summary>
public class ListCondominiumsPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Delegate para notificar a seleção de um condomínio.
    /// </summary>
    [Parameter]
    public EventCallback<Condominium> OnCondominiumSelected { get; set; }

    /// <summary>
    /// Estilo CSS aplicado às linhas da tabela.
    /// </summary>
    [Parameter]
    public string RowStyle { get; set; } = string.Empty;

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de mensagens de notificação.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    /// <summary>
    /// Handler para operações relacionadas aos condomínios.
    /// </summary>
    [Inject]
    public ICondominiumHandler Handler { get; set; } = null!;

    #endregion

    #region Properties

    /// <summary>
    /// Componente de grid para exibir a lista de condomínios.
    /// </summary>
    public MudDataGrid<Condominium> DataGrid { get; set; } = null!;

    /// <summary>
    /// Lista de condomínios carregados do servidor.
    /// </summary>
    public List<Condominium> Condominiums { get; set; } = [];

    /// <summary>
    /// Termo de busca utilizado para filtrar a lista de condomínios.
    /// </summary>
    public string SearchTerm { get; set; } = string.Empty;

    /// <summary>
    /// Filtro selecionado para a busca.
    /// </summary>
```

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

    /// <summary>
    /// Lista de opções de filtro disponíveis para a busca.
    /// </summary>
    public readonly List<FilterOption> FilterOptions =
    [
        new() { DisplayName = "Nome", PropertyName = "Name" },
        new() { DisplayName = "Bairro", PropertyName = "Address.Neighborhood" },
        new() { DisplayName = "Rua", PropertyName = "Address.Street" },
        new() { DisplayName = "Cidade", PropertyName = "Address.City" },
        new() { DisplayName = "Estado", PropertyName = "Address.State" },
        new() { DisplayName = "CEP", PropertyName = "Address.ZipCode" },
        new() { DisplayName = "País", PropertyName = "Address.Country" },
        new() { DisplayName = "Unidades", PropertyName = "Units" },
        new() { DisplayName = "Andares", PropertyName = "Floors" },
        new() { DisplayName = "Elevador", PropertyName = "HasElevator" },
        new() { DisplayName = "Piscina", PropertyName = "HasSwimmingPool" },
        new() { DisplayName = "Salão de Festas", PropertyName = "HasPartyRoom" },
        new() { DisplayName = "Playground", PropertyName = "HasPlayground" },
        new() { DisplayName = "Academia", PropertyName = "HasFitnessRoom" }
    ];

    #endregion

    #region Methods

    /// <summary>
    /// Método chamado quando o botão de exclusão é clicado.
    /// </summary>
    /// <param name="id">Identificador do condomínio a ser excluído.</param>
    /// <param name="name">Nome do condomínio (para exibição na mensagem de
confirmação).</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task OnDeleteButtonClickedAsync(long id, string name)
    {
        var parameters = new DialogParameters
        {
            { "ContentText", $"Ao prosseguir o condomínio {name} será excluido. " +
                             "Esta é uma ação irreversível! Deseja continuar?" },
            { "ButtonText", "Confirmar" },
            { "ButtonColor", Color.Error }
        };

        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Small
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Atenção", parameters,
options);
        var result = await dialog.Result;

        if (result is { Canceled: true }) return;

        await OnDeleteAsync(id, name);
        StateHasChanged();
    }

    /// <summary>
    /// Executa a operação de exclusão efetiva de um condomínio.
    /// </summary>
    /// <param name="id">Identificador do condomínio a ser excluído.</param>
    /// <param name="name">Nome do condomínio (utilizado para exibir a mensagem de
sucesso).</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    private async Task OnDeleteAsync(long id, string name)
    {
        try
        {
            await Handler.DeleteAsync(new DeleteCondominiumRequest { Id = id });
            Condominiums.RemoveAll(x => x.Id == id);
            await DataGrid.ReloadServerData();
            Snackbar.Add($"Condomínio {name} excluído", Severity.Success);
        }
```

```
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    /// <summary>
    /// Carrega os dados do servidor de forma paginada para exibição no grid.
    /// </summary>
    /// <param name="state">Estado atual do grid contendo paginação e filtros.</param>
    /// <returns>Um objeto <see cref="GridData{Condominium}"/> contendo a lista de
condomínios e a contagem total.</returns>
    public async Task<GridData<Condominium>> LoadServerData(GridState<Condominium> state)
    {
        try
        {
            var request = new GetAllCondominiumsRequest
            {
                PageNumber = state.Page + 1,
                PageSize = state.PageSize,
                SearchTerm = SearchTerm,
                FilterBy = SelectedFilter
            };

            var response = await Handler.GetAllAsync(request);
            if (response.IsSuccess)
                return new GridData<Condominium>
                {
                    Items = response.Data ?? [],
                    TotalItems = response.TotalCount
                };

            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<Condominium>();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Condominium>();
        }
    }

    /// <summary>
    /// Aciona a recarga dos dados do grid quando o botão de busca é clicado.
    /// </summary>
    public void OnButtonSearchClick() => DataGrid.ReloadServerData();

    /// <summary>
    /// Limpa o termo de busca e o filtro, e recarrega os dados do grid.
    /// </summary>
    public void OnClearSearchClick()
    {
        SearchTerm = string.Empty;
        SelectedFilter = string.Empty;
        DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Manipula a alteração do valor do filtro selecionado.
    /// </summary>
    /// <param name="newValue">Novo valor do filtro.</param>
    public void OnValueFilterChanged(string newValue)
    {
        SelectedFilter = newValue;
        StateHasChanged();
    }

    /// <summary>
    /// Notifica ao componente pai que um condomínio foi selecionado.
    /// </summary>
    /// <param name="condominium">O condomínio selecionado.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task SelectCondominium(Condominium condominium)
    {
        if (OnCondominiumSelected.HasDelegate)
            await OnCondominiumSelected.InvokeAsync(condominium);
```

```
    }
    #endregion
}
```

.\RealtyHub.Web\Pages\ContractsTemplates\List.razor

```
@page "/contratos/modelos"
@using RealtyHub.Core.Models
@inherits ListContractTemplatesPage

<MudText Typo="Typo.h3">Modelos de Contratos</MudText>

<MudDataGrid T="ContractTemplate"
             Class="mt-4"
             ServerData="LoadServerData"
             RowsPerPage="@Core.Configuration.DefaultPageSize">

    <ToolBarContent>
        <MudText Typo="Typo.h6">Modelos de Contratos</MudText>
        <MudSpacer />
    </ToolBarContent>
    <Columns>
        <PropertyColumn Property="c => c.Id" Title="Id"></PropertyColumn>
        <PropertyColumn Property="c => c.Name" Title="Nome"></PropertyColumn>
        <TemplateColumn Class="justify-end" Title="Exibir">
            <CellTemplate>
                <MudTooltip Text="Exibir em nova página">
                    <MudIconButton Icon="@Icons.Material.Filled.OpenInBrowser"
                                   Color="Color.Primary"
                                   OnClick="() => ShowInNewPage(context.Item)" />
                </MudTooltip>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
</MudDataGrid>
```

.\RealtyHub.Web\Pages\ContractsTemplates\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;

namespace RealtyHub.Web.Pages.ContractsTemplates;

/// <summary>
/// Página responsável por exibir e gerenciar a listagem dos modelos de contrato.
/// </summary>
public partial class ListContractTemplatesPage : ComponentBase
{
    #region Properties

    /// <summary>
    /// Lista de modelos de contrato a serem exibidos na página.
    /// </summary>
    public List<ContractTemplate> ContractTemplates { get; set; } = [];

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de mensagens de notificação.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações relacionadas aos modelos de contrato.
    /// </summary>
    [Inject]
    public IContractTemplateHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço JavaScript para interações com o navegador.
    /// </summary>
    [Inject]
    public IJSRuntime JsRuntime { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Carrega os dados do servidor de forma paginada para exibição no grid.
    /// </summary>
    /// <param name="state">Estado atual do grid contendo informações de
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{ContractTemplate}"/> contendo os modelos de
contrato filtrados e a contagem total.
    /// Caso a operação falhe, retorna um grid vazio e exibe uma mensagem de erro.
    /// </returns>
    public async Task<GridData<ContractTemplate>>
LoadServerData(GridState<ContractTemplate> state)
    {
        try
        {
            var response = await Handler.GetAllAsync();
            if (response.IsSuccess)
            {
                ContractTemplates = response.Data ?? [];
                return new GridData<ContractTemplate>
                {
                    Items = ContractTemplates.Where(c => c.ShowInPage),
                    TotalItems = ContractTemplates.Count
                };
            }
```

```
            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<ContractTemplate>();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<ContractTemplate>();
        }
    }

    /// <summary>
    /// Abre o modelo de contrato em PDF em uma nova aba do navegador.
    /// </summary>
    /// <param name="template">O modelo de contrato a ser visualizado.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task ShowInNewPage(ContractTemplate template)
    {
        await JsRuntime.InvokeVoidAsync("openContractPdfInNewTab", template.Path);
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Contracts\Create.razor

```
@page "/contratos/emitir"

<ContractForm />
```

.\RealtyHub.Web\Pages\Contracts\Edit.razor

```
@page "/contratos/editar/{contractId:long}"
@inherits ContractFormComponent

<ContractForm ContractId="ContractId" />
```

.\RealtyHub.Web\Pages\Contracts\List.razor

```
@page "/contratos"
@using RealtyHub.Core.Models
@inherits ListContractsPage

<MudText Typo="Typo.h3">Contratos</MudText>

<PageTitle>Contratos</PageTitle>

<MudDataGrid T="Contract"
             Class="mt-4"
             ServerData="LoadServerData"
             @ref="DataGrid"
             RowsPerPage="@Core.Configuration.DefaultPageSize">

    <ToolBarContent>
        <MudText Typo="Typo.h6">Contratos</MudText>
        <MudSpacer />
        <MudDateRangePicker PickerVariant="PickerVariant.Dialog"
                            Label="Escolha a data do contrato"
                            DateRange="DateRange"
                            DateRangeChanged="OnDateRangeChanged"
                            Clearable="true" />
    </ToolBarContent>
    <Columns>
        <PropertyColumn Property="c => c.Id" Title="Id"></PropertyColumn>
        <PropertyColumn Property="c => c.IssueDate" Title="Data de
emissão"></PropertyColumn>
        <PropertyColumn Property="c => c.OfferId" Title="Identificador da
proposta"></PropertyColumn>
        <PropertyColumn Property="c => c.Buyer!.Name" Title="Comprador"></PropertyColumn>
        <PropertyColumn Property="c => c.Seller!.Name" Title="Vendedor"></PropertyColumn>
        <PropertyColumn Property="c => c.Offer!.Property!.Title"
Title="Imóvel"></PropertyColumn>
        <TemplateColumn Title="Valor">
            <CellTemplate>
                @context.Item.Offer!.Amount.ToString("C")
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Class="justify-end" Title="Ações">
            <CellTemplate>
                <MudTooltip Text="Editar">
                    <MudIconButton Icon="@Icons.Material.Filled.Edit"
                                   Href="@($"/contratos/editar/{context.Item.Id}")"
                                   Color="Color.Primary"/>
                </MudTooltip>
                <MudTooltip Text="Deletar">
                    <MudIconButton Icon="@Icons.Material.Filled.Delete"
                                   Color="Color.Error"
                                   OnClick="() =>
OnDeleteButtonClickedAsync(context.Item.Id)"/>
                </MudTooltip>
                <MudTooltip Text="Abrir em nova página">
                    <MudIconButton Icon="@Icons.Material.Filled.OpenInBrowser"
                                   Color="Color.Primary"
                                   OnClick="() => ShowInNewPage(context.Item)" />
                </MudTooltip>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Contract"/>
    </PagerContent>
</MudDataGrid>
```

.\RealtyHub.Web\Pages\Contracts\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;
using MudBlazor;
using RealtyHub.Core.Extensions;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Contracts;
using RealtyHub.Web.Components.Common;

namespace RealtyHub.Web.Pages.Contracts;

/// <summary>
/// Página responsável pela listagem e gerenciamento de contratos na aplicação.
/// </summary>
public partial class ListContractsPage : ComponentBase
{
    #region Properties

    /// <summary>
    /// Componente de grid para exibir a lista de contratos.
    /// </summary>
    public MudDataGrid<Contract> DataGrid { get; set; } = null!;

    /// <summary>
    /// Lista de contratos a serem exibidos no grid.
    /// </summary>
    public List<Contract> Contracts { get; set; } = [];

    /// <summary>
    /// Intervalo de datas utilizado para filtrar os contratos.
    /// O intervalo é definido com a data inicial sendo o primeiro dia do mês atual
    /// e a data final sendo o último dia do mês atual.
    /// </summary>
    public DateRange DateRange { get; set; } = new(DateTime.Now.GetFirstDay(),
        DateTime.Now.GetLastDay());

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de mensagens de notificação.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações relacionadas a contratos.
    /// </summary>
    [Inject]
    public IContractHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    /// <summary>
    /// Serviço JavaScript para interações com o navegador (ex.: abrir nova aba).
    /// </summary>
    [Inject]
    public IJSRuntime JsRuntime { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Carrega os dados do servidor de forma paginada para exibição no grid.
    /// </summary>
    /// <param name="state">Estado atual do grid contendo informações sobre
```

```
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{Contract}"/> contendo a lista de contratos e a
contagem total.
    /// Em caso de falha, exibe uma mensagem de erro e retorna um grid vazio.
    /// </returns>
    public async Task<GridData<Contract>> LoadServerData(GridState<Contract> state)
    {
        try
        {
            var endDate = DateRange.End?.ToEndOfDay();
            var request = new GetAllContractsRequest
            {
                PageNumber = state.Page + 1,
                PageSize = state.PageSize,
                StartDate = DateRange.Start.ToUtcString(),
                EndDate = endDate.ToUtcString()
            };

            var response = await Handler.GetAllAsync(request);
            if (response.IsSuccess)
            {
                Contracts = response.Data ?? [];
                return new GridData<Contract>
                {
                    Items = Contracts.OrderByDescending(c => c.UpdatedAt),
                    TotalItems = response.TotalCount
                };
            }

            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<Contract>();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Contract>();
        }
    }

    /// <summary>
    /// Atualiza o intervalo de datas utilizado para filtrar os contratos e recarrega os
dados do grid.
    /// </summary>
    /// <param name="newDateRange">Novo intervalo de datas selecionado.</param>
    public void OnDateRangeChanged(DateRange newDateRange)
    {
        DateRange = newDateRange;
        DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Abre o arquivo PDF do contrato em uma nova aba do navegador.
    /// </summary>
    /// <param name="contract">Contrato cuja visualização em PDF será aberta.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task ShowInNewPage(Contract contract)
    {
        await JsRuntime.InvokeVoidAsync("openContractPdfInNewTab", contract.FilePath);
    }

    /// <summary>
    /// Abre um diálogo de confirmação para exclusão do contrato e, se confirmado,
executa a exclusão.
    /// </summary>
    /// <param name="id">Identificador do contrato a ser excluído.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task OnDeleteButtonClickedAsync(long id)
    {
        var parameters = new DialogParameters
        {
            { "ContentText", $"Ao prosseguir o contrato com identificador <b>{id}</b>
será excluido. " +
                             "Esta é uma ação irreversível! Deseja continuar?" },
            { "ButtonText", "Confirmar" },
            { "ButtonColor", Color.Error }
```

```
        };

        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Small
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Atenção", parameters,
options);
        var result = await dialog.Result;

        if (result is { Canceled: true }) return;

        await OnDeleteAsync(id);
        StateHasChanged();
    }

    /// <summary>
    /// Executa a exclusão efetiva do contrato.
    /// </summary>
    /// <param name="id">Identificador do contrato que será deletado.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    private async Task OnDeleteAsync(long id)
    {
        try
        {
            await Handler.DeleteAsync(new DeleteContractRequest { Id = id });
            Contracts.RemoveAll(x => x.Id == id);
            await DataGrid.ReloadServerData();
            Snackbar.Add($"Contrato {id} excluído", Severity.Success);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Customers\Create.razor

```
@page "/clientes/adicionar"
@inherits CustomerFormComponent

<CustomerForm/>
```

.\RealtyHub.Web\Pages\Customers\Edit.razor

```
@page "/clientes/editar/{id:long}"
@inherits CustomerFormComponent

<CustomerForm Id="@Id" />
```

.\RealtyHub.Web\Pages\Customers\List.razor

```
@page "/clientes"
@using RealtyHub.Core.Models
@inherits ListCustomersPage

<MudText Typo="Typo.h3">Clientes</MudText>

<PageTitle>Clientes</PageTitle>

<div class="d-flex justify-end">
    <MudButton Variant="Variant.Filled"
               Color="Color.Primary"
               StartIcon="@Icons.Material.Filled.PersonAdd"
               Href="/clientes/adicionar">
        Novo Cliente
    </MudButton>
</div>

<MudDataGrid T="Customer"
             Class="mt-4"
             ServerData="LoadServerData"
             @ref="DataGrid"
             RowsPerPage="@Core.Configuration.DefaultPageSize"
             RowClick="@(e => SelectCustomer(e.Item))"
             RowStyle="@RowStyle">
    <ToolBarContent>
        <MudText Typo="Typo.h6">Clientes</MudText>
        <MudSpacer />
        <MudTextField @bind-Value="SearchTerm"
                      Placeholder="Filtrar por Nome, Documento e Email"
                      Adornment="Adornment.Start"
                      AdornmentIcon="@Icons.Material.Filled.Search"
                      Clearable="true"
                      OnClearButtonClick="@(() => SearchTerm = "" )"
                      Immediate="true"
                      IconSize="Size.Medium"
                      Class="mt-0">
        </MudTextField>
    </ToolBarContent>
    <Columns>
        <PropertyColumn Property="x => x.Name" Title="Nome" />
        <PropertyColumn Property="x => x.DocumentNumber" Title="Documento" />
        <PropertyColumn Property="x => x.Email" Title="E-mail" />
        <TemplateColumn Title="Tipo">
            <CellTemplate>
                @if (context.Item.PersonType == EPersonType.Individual)
                {
                    <span>Pessoa Física</span>
                }
                else
                {
                    <span>Pessoa Jurídica</span>
                }
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Class="justify-end" Title="Ações">
            <CellTemplate>
                <MudStack Row>
                    <MudTooltip Text="Editar">
                        <MudIconButton Icon="@Icons.Material.Filled.Edit"
                                       Href="@($"/clientes/editar/{context.Item.Id}")"
                                       Color="Color.Primary">
                        </MudIconButton>
                    </MudTooltip>
                    <MudTooltip Text="Excluir">
                        <MudIconButton Icon="@Icons.Material.Filled.Delete"
                                       Color="Color.Error"
                                       OnClick="() => OnDeleteButtonClickedAsync
                                           (context.Item.Id, context.Item.Name)">
                        </MudIconButton>
                    </MudTooltip>
                </MudStack>
            </CellTemplate>
```

```
        </TemplateColumn>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Customer" />
    </PagerContent>
</MudDataGrid>
```

.\RealtyHub.Web\Pages\Customers\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Customers;
using RealtyHub.Web.Components.Common;

namespace RealtyHub.Web.Pages.Customers;

/// <summary>
/// Página responsável por exibir e gerenciar a listagem de clientes na aplicação.
/// </summary>
public partial class ListCustomersPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Evento chamado quando um cliente é selecionado.
    /// </summary>
    [Parameter]
    public EventCallback<Customer> OnCustomerSelected { get; set; }

    /// <summary>
    /// Estilo CSS aplicado às linhas da tabela.
    /// </summary>
    [Parameter]
    public string RowStyle { get; set; } = string.Empty;

    #endregion

    #region Properties

    /// <summary>
    /// Componente de grid do MudBlazor utilizado para exibir a lista de clientes.
    /// </summary>
    public MudDataGrid<Customer> DataGrid { get; set; } = null!;

    /// <summary>
    /// Lista de clientes a serem exibidos no grid.
    /// </summary>
    public List<Customer> Customers { get; set; } = [];

    private string _searchTerm = string.Empty;

    /// <summary>
    /// Termo de busca utilizado para filtrar os clientes.
    /// Ao alterar o valor, o grid é recarregado.
    /// </summary>
    public string SearchTerm
    {
        get => _searchTerm;
        set
        {
            if (_searchTerm == value) return;
            _searchTerm = value;
            _ = DataGrid.ReloadServerData();
        }
    }

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de notificações na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável por gerenciar as operações relacionadas aos clientes.
    /// </summary>
```

```
    [Inject]
    public ICustomerHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço utilizado para exibir diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Exibe um diálogo de confirmação e, se confirmado, executa a exclusão do cliente.
    /// </summary>
    /// <param name="id">Identificador do cliente a ser excluído.</param>
    /// <param name="name">Nome do cliente, para exibição na mensagem de
confirmação.</param>
    public async Task OnDeleteButtonClickedAsync(long id, string name)
    {
        var parameters = new DialogParameters
        {
            { "ContentText", $"Ao prosseguir o cliente {name} será excluido. " +
                             "Esta é uma ação irreversível! Deseja continuar?" },
            { "ButtonText", "Confirmar" },
            { "ButtonColor", Color.Error }
        };

        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Small
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Atenção", parameters,
options);
        var result = await dialog.Result;

        if (result is { Canceled: true }) return;

        await OnDeleteAsync(id, name);
        StateHasChanged();
    }

    /// <summary>
    /// Executa a exclusão efetiva do cliente.
    /// </summary>
    /// <param name="id">Identificador do cliente a ser excluído.</param>
    /// <param name="name">Nome do cliente, utilizado para exibir a notificação de
sucesso.</param>
    private async Task OnDeleteAsync(long id, string name)
    {
        try
        {
            await Handler.DeleteAsync(new DeleteCustomerRequest { Id = id });
            Customers.RemoveAll(x => x.Id == id);
            await DataGrid.ReloadServerData();
            Snackbar.Add($"Cliente {name} excluído", Severity.Success);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    /// <summary>
    /// Carrega os dados dos clientes do servidor de forma paginada para exibição no
grid.
    /// </summary>
    /// <param name="state">Estado atual do grid, contendo informações de
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{Customer}"/> contendo a lista de clientes e a
contagem total.
    /// Em caso de falha, retorna um grid vazio e exibe uma mensagem de erro.
```

```
    /// </returns>
    public async Task<GridData<Customer>> LoadServerData(GridState<Customer> state)
    {
        try
        {
            var request = new GetAllCustomersRequest
            {
                PageNumber = state.Page + 1,
                PageSize = state.PageSize,
                SearchTerm = SearchTerm
            };

            var response = await Handler.GetAllAsync(request);
            if (response.IsSuccess)
            {
                return new GridData<Customer>
                {
                    Items = response.Data ?? [],
                    TotalItems = response.TotalCount
                };
            }

            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<Customer>
            {
                Items = [],
                TotalItems = 0
            };
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Customer>
            {
                Items = [],
                TotalItems = 0
            };
        }
    }

    /// <summary>
    /// Notifica o componente pai que um cliente foi selecionado.
    /// </summary>
    /// <param name="customer">Cliente selecionado.</param>
    public async Task SelectCustomer(Customer customer)
    {
        if (OnCustomerSelected.HasDelegate)
            await OnCustomerSelected.InvokeAsync(customer);
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Home.razor

```
@page "/home"
@layout PublicLayout

<PageTitle>Home</PageTitle>

<MudPaper Class="pa-4">
    <MudPaper Class="p-4 d-flex flex-column align-items-center justify-center
mud-elevation-1">
        <MudText Typo="Typo.h3" Align="Align.Center" Class="mb-2">
            Bem-vindo ao RealtyHub
        </MudText>
        <MudText Align="Align.Center">
            Conectando corretores e clientes de forma simples e eficiente!
        </MudText>
        <MudButton Variant="Variant.Filled"
                   Color="Color.Primary"
                   Class="mt-3"
                   Href="/comecar">
            Começar agora
        </MudButton>
    </MudPaper>

    <MudGrid Class="mt-6">
        @foreach (var feature in Features)
        {
            <MudItem xs="12" sm="6" md="3">
                <MudCard Class="mb-4">
                    <MudCardHeader>
                        <MudText Typo="Typo.h6" Class="mb-0">@feature.Title</MudText>
                    </MudCardHeader>
                    <MudCardContent>
                        <MudText>
                            @feature.Description
                        </MudText>
                    </MudCardContent>
                </MudCard>
            </MudItem>
        }
    </MudGrid>

    <MudPaper Class="pa-4 p-4 mt-4 text-center mud-elevation-1">
        <MudText Class="mb-2" Typo="Typo.h5">Quer visualizar os imóveis disponíveis para
venda?</MudText>
        <MudText>
            Clique no botão abaixo.
        </MudText>
        <MudButton Color="Color.Primary"
                   Class="mt-2"
                   Href="/listar-imoveis">
            Imóveis
        </MudButton>
    </MudPaper>
</MudPaper>

@code {
    public class FeatureItem
    {
        public string Title { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
    }

    private List<FeatureItem> Features = new()
    {
        new FeatureItem
        {
            Title = "Maior Alcance",
            Description = "Liste imóveis e seja visto por mais potenciais clientes em
poucos cliques."
        },
        new FeatureItem
        {
            Title = "Facilidade de Uso",
```

```
            Description = "Interface intuitiva para gerenciar seus imóveis e acompanhar
negociações."
        },
        new FeatureItem
        {
            Title = "Melhor Custo-Benefício",
            Description = "Planos flexíveis e suporte especializado para alavancar suas
vendas."
        },
        new FeatureItem
        {
            Title = "Gerenciamento de Imóveis",
            Description = "Cadastre e edite informações completas dos imóveis em um só
lugar."
        },
        new FeatureItem
        {
            Title = "Gerenciamento de Clientes",
            Description = "Organize dados de compradores e vendedores para um atendimento
personalizado."
        },
        new FeatureItem
        {
            Title = "Gerenciamento de Visitas",
            Description = "Controle data e horários de cada visita realizada aos
imóveis."
        },
        new FeatureItem
        {
            Title = "Propostas",
            Description = "Permita que visitantes enviem propostas de compra sem precisar
criar conta."
        },
        new FeatureItem
        {
            Title = "Geração e Envio de Contratos",
            Description = "Otimize o fechamento de negócio com contratos e envio."
        }
    };
}
```

.\RealtyHub.Web\Pages\Identity\ConfirmEmail.razor

```
@page "/confirmar-email"
@inherits ConfirmEmailPage
@layout HeadlessLayout

<PageTitle>RealtyHub - Confirmar Email</PageTitle>

<MudGrid Justify="Justify.Center">
    <MudItem xs="12" sm="12" md="8" lg="6" xl="6" xxl="6">
        <MudPaper Class="mud-width-full pa-8">
            <Logo Width="200" />

            @if (IsBusy)
            {
                <MudProgressCircular Color="Color.Info" Indeterminate="true" />
            }
            else
            {
                <div class="flex-center">
                    <MudIcon Icon="@(Success
                                    ? Icons.Material.Filled.CheckCircle
                                    : Icons.Material.Filled.Error)">
                    </MudIcon>
                    <MudText Class="ml-2">@Status</MudText>
                </div>

                <div class="d-flex mt-8">
                    <MudButton ButtonType="ButtonType.Submit"
                               Href="/login"
                               Variant="Variant.Filled"
                               Color="Color.Primary"
                               StartIcon="@Icons.Material.Filled.Home">
                        Ir para página de login
                    </MudButton>
                </div>
            }
        </MudPaper>
    </MudItem>
</MudGrid>
```

.\RealtyHub.Web\Pages\Identity\ConfirmEmail.razor.cs

```
using Microsoft.AspNetCore.Components;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models.Account;

namespace RealtyHub.Web.Pages.Identity;

/// <summary>
/// Página responsável por confirmar o e-mail do usuário.
/// </summary>
public partial class ConfirmEmailPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do usuário. O valor é obtido a partir da query string usando o nome
"UserId".
    /// </summary>
    [Parameter]
    [SupplyParameterFromQuery(Name = "UserId")]
    public long UserId { get; set; }

    /// <summary>
    /// Token de confirmação de e-mail. O valor é obtido a partir da query string usando
o nome "Token".
    /// </summary>
    [Parameter]
    [SupplyParameterFromQuery(Name = "Token")]
    public string Token { get; set; } = string.Empty;

    #endregion

    #region Properties

    /// <summary>
    /// Indica se a página está realizando alguma operação (estado de carregamento).
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Mensagem de status a ser exibida após a tentativa de confirmação do e-mail.
    /// </summary>
    public string Status { get; set; } = string.Empty;

    /// <summary>
    /// Indica se a confirmação de e-mail foi realizada com sucesso.
    /// </summary>
    public bool Success { get; set; }

    #endregion

    #region Services

    /// <summary>
    /// Handler responsável pelas operações de conta, incluindo a confirmação de e-mail.
    /// </summary>
    [Inject]
    public IAccountHandler AccountHandler { get; set; } = null!;

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente e confirma o e-mail do usuário.
    /// </summary>
    /// <remarks>
    /// Durante a inicialização, o componente define o estado de carregamento, envia uma
solicitação de confirmação de e-mail
    /// utilizando as informações obtidas da query string, e atualiza o status e o
indicador de sucesso com base na resposta do handler.
    /// Caso ocorra um erro, uma mensagem de erro é definida. Ao final, o estado de
carregamento é desativado.
```

```
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        IsBusy = true;
        try
        {
            var request = new ConfirmEmailRequest
            {
                UserId = UserId,
                Token = Token
            };

            var result = await AccountHandler.ConfirmEmailAsync(request);
            Success = result.IsSuccess;
            Status = result.Message!;
        }
        catch (Exception e)
        {
            Status = $"Erro ao confirmar email!\n{e.Message}";
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Identity\Login.razor

```
@page "/login"
@layout HeadlessLayout
@inherits LoginPage

<PageTitle>RealtyHub - Entrar</PageTitle>

<MudGrid Justify="Justify.Center">
    <MudItem xs="12" sm="12" md="8" lg="6" xl="6" xxl="6">
        <MudPaper Class="mud-width-full pa-8">
            <Logo Width="200" />

            <EditForm Model="@InputModel" OnValidSubmit="OnValidSubmitAsync">
                <DataAnnotationsValidator />

                <MudText Class="flex-center justify-center mb-4"
                         Typo="Typo.h6">
                    Bem-vindo de volta!
                </MudText>

                <MudTextField T="string"
                              Label="E-mail"
                              InputType="InputType.Email"
                              For="@(() => InputModel.Email)"
                              @bind-Value="@InputModel.Email"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.Email"
                              Adornment="Adornment.Start" />

                <MudTextField T="string"
                              Label="Senha"
                              InputType="InputType.Password"
                              For="@(() => InputModel.Password)"
                              @bind-Value="@InputModel.Password"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.Password"
                              Adornment="Adornment.Start" />

                <a href="/recuperar-senha" style="color: blue">Esqueci a senha</a>

                <div class="d-flex mt-8">
                    @if (IsBusy)
                    {
                        <MudProgressCircular Color="Color.Info" Indeterminate="true" />
                    }
                    else
                    {
                        <MudButton ButtonType="ButtonType.Submit"
                                   Variant="Variant.Filled"
                                   Color="Color.Primary"
                                   StartIcon="@Icons.Material.Filled.Login">
                            Login
                        </MudButton>
                        <MudSpacer />
                        <MudButton Href="/comecar"
                                   Variant="Variant.Filled"
                                   Color="Color.Primary"
                                   StartIcon="@Icons.Material.Filled.AppRegistration">
                            Quero me cadastrar
                        </MudButton>
                    }
                </div>
            </EditForm>
        </MudPaper>
    </MudItem>
</MudGrid>
```

.\RealtyHub.Web\Pages\Identity\Login.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Web.Security;

namespace RealtyHub.Web.Pages.Identity;

/// <summary>
/// Página responsável por gerenciar o login do usuário na aplicação.
/// </summary>
public partial class LoginPage : ComponentBase
{
    #region Services

    /// <summary>
    /// Serviço para exibição de notificações na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de conta, como login.
    /// </summary>
    [Inject]
    public IAccountHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço para gerenciamento de navegação entre páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Provedor do estado de autenticação baseado em cookies.
    /// </summary>
    [Inject]
    public ICookieAuthenticationStateProvider AuthenticationStateProvider { get; set; } =
null!;

    #endregion

    #region Properties

    /// <summary>
    /// Modelo utilizado para vincular os dados de login.
    /// </summary>
    public LoginRequest InputModel { get; set; } = new();

    /// <summary>
    /// Indica se a página está ocupada realizando uma operação (ex: envio do
formulário).
    /// </summary>
    public bool IsBusy { get; set; }

    #endregion

    #region Overrides

    /// <summary>
    /// Método de inicialização do componente.
    /// </summary>
    /// <remarks>
    /// Verifica o estado atual de autenticação e, se o usuário já estiver autenticado,
redireciona para a página inicial.
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();
        var user = authState.User;

        if (user.Identity is { IsAuthenticated: true })
```

```
            NavigationManager.NavigateTo("/");
    }

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário de login.
    /// </summary>
    /// <remarks>
    /// Define o estado de carregamento, envia os dados de login para o handler e, em
caso de sucesso,
    /// atualiza o estado de autenticação e redireciona o usuário para a página de
listagem de imóveis.
    /// Em caso de falha, exibe uma mensagem de erro utilizando o serviço de Snackbar.
    /// </remarks>
    public async Task OnValidSubmitAsync()
    {
        IsBusy = true;
        try
        {
            var result = await Handler.LoginAsync(InputModel);
            if (result.IsSuccess)
            {
                await AuthenticationStateProvider.GetAuthenticationStateAsync();
                AuthenticationStateProvider.NotifyAuthenticationStateChanged();
                NavigationManager.NavigateTo("/listar-imoveis");
            }
            else
                Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Identity\Logout.razor

```
@page "/sair"
@inherits LogoutPage
@layout HeadlessLayout

<PageTitle>RealtyHub - Sair</PageTitle>

<MudText Typo="Typo.h3">Finalizando Sessão...</MudText>

<MudPaper Class="pa-8 mt-4">
    <Logo Width="200" />

    <MudAlert Severity="Severity.Info">
        <div>
            Sessão Finalizada...
            <MudButton Variant="Variant.Text"
                       Href="/login"
                       StartIcon="@Icons.Material.Filled.Login">
                Ir para a página de login
            </MudButton>
        </div>
    </MudAlert>
</MudPaper>
```

.\RealtyHub.Web\Pages\Identity\Logout.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Web.Security;

namespace RealtyHub.Web.Pages.Identity;

/// <summary>
/// Página responsável por gerenciar o logout do usuário na aplicação.
/// </summary>
public partial class LogoutPage : ComponentBase
{
    #region Services

    /// <summary>
    /// Serviço para exibição de notificações na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de conta, como logout.
    /// </summary>
    [Inject]
    public IAccountHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço para gerenciamento de navegação entre páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Provedor do estado de autenticação baseado em cookies.
    /// </summary>
    [Inject]
    public ICookieAuthenticationStateProvider AuthenticationStateProvider { get; set; } =
null!;

    #endregion

    #region Overrides

    /// <summary>
    /// Método chamado durante a inicialização do componente.
    /// </summary>
    /// <remarks>
    /// Verifica se o usuário está autenticado. Se sim, executa o logout
    /// e atualiza o estado de autenticação, notificando a aplicação da mudança.
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        if (await AuthenticationStateProvider.CheckAuthenticatedAsync())
        {
            await Handler.LogoutAsync();
            await AuthenticationStateProvider.GetAuthenticationStateAsync();
            AuthenticationStateProvider.NotifyAuthenticationStateChanged();
        }
        await base.OnInitializedAsync();
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Identity\Register.razor

```
@page "/comecar"
@layout HeadlessLayout
@inherits RegisterPage

<PageTitle>RealtyHub - Registrar</PageTitle>

<MudGrid Justify="Justify.Center">
    <MudItem xs="12" sm="12" md="8" lg="6" xl="6" xxl="6">
        <MudPaper Class="mud-width-full pa-8">
            <Logo Width="200" />

            <EditForm Model="@InputModel" OnValidSubmit="OnValidSubmitAsync">
                <DataAnnotationsValidator />

                <MudText Class="flex-center justify-center mb-4"
                         Typo="Typo.h6">
                    Crie sua conta para usar nossos serviços!
                </MudText>

                <MudTextField T="string"
                              Label="Nome"
                              InputType="InputType.Text"
                              For="@(() => InputModel.GivenName)"
                              @bind-Value="@InputModel.GivenName"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.Person"
                              Adornment="Adornment.Start" />

                <MudTextField T="string"
                              Label="Creci"
                              InputType="InputType.Text"
                              For="@(() => InputModel.Creci)"
                              @bind-Value="@InputModel.Creci"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.AppRegistration"
                              Adornment="Adornment.Start" />

                <MudTextField T="string"
                              Label="E-mail"
                              InputType="InputType.Email"
                              For="@(() => InputModel.Email)"
                              @bind-Value="@InputModel.Email"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.Email"
                              Adornment="Adornment.Start" />

                <MudTextField T="string"
                              Label="Senha"
                              InputType="InputType.Password"
                              For="@(() => InputModel.Password)"
                              @bind-Value="@InputModel.Password"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.Password"
                              Adornment="Adornment.Start" />

                <MudTextField T="string"
                              Label="Confirmar Senha"
                              InputType="InputType.Password"
                              For="@(() => InputModel.ConfirmPassword)"
                              @bind-Value="@InputModel.ConfirmPassword"
                              OnBlur="IsPasswordEqual"
                              Error="Error"
                              ErrorText="@ErrorText"
                              Class="mb-4"
                              AdornmentIcon="@Icons.Material.Filled.Password"
                              Adornment="Adornment.Start" />

                <div class="d-flex mt-8">
                    @if (IsBusy)
                    {
                        <MudProgressCircular Color="Color.Info" Indeterminate="true" />
                    }
```

```
                    else
                    {
                        <MudButton ButtonType="ButtonType.Submit"
                                   Variant="Variant.Filled"
                                   Color="Color.Primary"
                                   StartIcon="@Icons.Material.Filled.Start">
                            Começar
                        </MudButton>
                        <MudSpacer />
                        <MudButton Href="/login"
                                   Variant="Variant.Filled"
                                   Color="Color.Primary"
                                   StartIcon="@Icons.Material.Filled.Login">
                            Já tenho uma conta
                        </MudButton>
                    }

                </div>
            </EditForm>
        </MudPaper>
    </MudItem>
</MudGrid>
```

.\RealtyHub.Web\Pages\Identity\Register.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Requests.Account;
using RealtyHub.Web.Security;

namespace RealtyHub.Web.Pages.Identity;

/// <summary>
/// Página responsável pelo registro de novos usuários na aplicação.
/// </summary>
public partial class RegisterPage : ComponentBase
{
    #region Services

    /// <summary>
    /// Serviço para exibição de notificações na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de conta, como registro de novos usuários.
    /// </summary>
    [Inject]
    public IAccountHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço para gerenciamento de navegação entre páginas.
    /// </summary>
    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    /// <summary>
    /// Provedor do estado de autenticação baseado em cookies.
    /// </summary>
    [Inject]
    public ICookieAuthenticationStateProvider AuthenticationStateProvider { get; set; } =
null!;

    #endregion

    #region Properties

    /// <summary>
    /// Modelo utilizado para vincular os dados de registro.
    /// </summary>
    public RegisterRequest InputModel { get; set; } = new();

    /// <summary>
    /// Texto de erro a ser exibido quando as senhas não coincidem.
    /// </summary>
    public string? ErrorText { get; set; }

    /// <summary>
    /// Indica se há mensagem de erro relacionada ao registro.
    /// </summary>
    public bool Error => !string.IsNullOrEmpty(ErrorText);

    /// <summary>
    /// Indica se a página está ocupada realizando uma operação, como o envio do
formulário.
    /// </summary>
    public bool IsBusy { get; set; }

    #endregion

    #region Overrides

    /// <summary>
    /// Método de inicialização do componente.
    /// </summary>
```

```
    /// <remarks>
    /// Verifica o estado atual de autenticação. Caso o usuário já esteja autenticado,
redireciona para a página de dashboard.
    /// </remarks>
    protected override async Task OnInitializedAsync()
    {
        var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();
        var user = authState.User;
        if (user.Identity is { IsAuthenticated: true })
            NavigationManager.NavigateTo("/dashboard");
    }

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o envio válido do formulário de registro.
    /// </summary>
    /// <remarks>
    /// Define o estado de carregamento, envia os dados para o handler de registro e, em
caso de sucesso,
    /// exibe uma mensagem e redireciona o usuário para a página de login. Caso
contrário, exibe uma mensagem de erro.
    /// </remarks>
    public async Task OnValidSubmitAsync()
    {
        IsBusy = true;
        try
        {
            var result = await Handler.RegisterAsync(InputModel);
            var resultMessage = result.Message ?? string.Empty;
            if (result.IsSuccess)
            {
                Snackbar.Add(resultMessage, Severity.Success);
                NavigationManager.NavigateTo("/login");
            }
            else
            {
                Snackbar.Add(resultMessage, Severity.Error);
            }
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    /// <summary>
    /// Verifica se as senhas inseridas são iguais.
    /// </summary>
    /// <remarks>
    /// Se as senhas coincidirem, remove a mensagem de erro; caso contrário, define uma
mensagem de erro.
    /// </remarks>
    public void IsPasswordEqual()
    {
        if (InputModel.Password == InputModel.ConfirmPassword)
        {
            ErrorText = null;
            return;
        }
        ErrorText = "As senhas não coincidem";
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Identity\ResetPassword.razor

```
@page "/recuperar-senha"
@layout HeadlessLayout
@inherits ResetPasswordPage

<PageTitle>RealtyHub - Esqueci Senha</PageTitle>

<MudGrid Justify="Justify.Center">
    <MudItem xs="12" sm="12" md="8" lg="6" xl="6" xxl="6">
        <MudPaper Class="mud-width-full pa-8">
            <Logo Width="200" />

            @if (UserId == 0 && string.IsNullOrEmpty(Token))
            {
                <EditForm Model="ForgotPasswordModel"
OnValidSubmit="OnValidSubmitForgotFormAsync">
                    <DataAnnotationsValidator />

                    <MudText Class="mb-4">@HeaderText</MudText>

                    <MudTextField T="string"
                                  Label="E-mail"
                                  InputType="InputType.Email"
                                  Placeholder="Digite seu e-mail"
                                  For="@(() => ForgotPasswordModel.Email)"
                                  @bind-Value="@ForgotPasswordModel.Email"
                                  Class="mb-4"
                                  AdornmentIcon="@Icons.Material.Filled.Email"
                                  Adornment="Adornment.Start" />

                    <div class="d-flex mt-8">
                        @if (IsBusy)
                        {
                            <MudProgressCircular Color="Color.Info" Indeterminate="true"
/>
                        }
                        else
                        {
                            @if (IsSuccess)
                            {
                                <div>
                                    <div>
                                        <MudText Class="mb-4">Verifique seu
e-mail!</MudText>
                                    </div>
                                    <div>
                                        <MudButton ButtonType="ButtonType.Button"
                                                   Variant="Variant.Filled"
                                                   Color="Color.Primary"
                                                   Href="/login"

StartIcon="@Icons.Material.Filled.Login">
                                            Ir para página de login
                                        </MudButton>
                                    </div>
                                </div>
                            }
                            else
                            {
                                <MudButton ButtonType="ButtonType.Submit"
                                           Variant="Variant.Filled"
                                           Color="Color.Primary"
                                           StartIcon="@Icons.Material.Filled.Send">
                                    Enviar
                                </MudButton>
                            }
                        }
                    </div>
                </EditForm>
            }
            else
            {
                <EditForm Model="ResetPasswordModel"
```

```
OnValidSubmit="OnValidSubmitResetFormAsync">
                    <ObjectGraphDataAnnotationsValidator />

                    <PasswordInput
@bind-InputModel="ResetPasswordModel.PasswordResetModel" />

                    <div class="d-flex mt-8">
                        @if (IsBusy)
                        {
                            <MudProgressCircular Color="Color.Info" Indeterminate="true"
/>
                        }
                        else
                        {
                            <MudButton ButtonType="ButtonType.Submit"
                                       Variant="Variant.Filled"
                                       Color="Color.Primary"
                                       StartIcon="@Icons.Material.Filled.Send">
                                Resetar senha
                            </MudButton>
                        }
                    </div>
                </EditForm>
            }
        </MudPaper>
    </MudItem>
</MudGrid>
```

.\RealtyHub.Web\Pages\Identity\ResetPassword.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Requests.Account;

namespace RealtyHub.Web.Pages.Identity;

public class ResetPasswordPage : ComponentBase
{
    #region Parameters

    [Parameter]
    [SupplyParameterFromQuery(Name = "UserId")]
    public long UserId { get; set; }
    [Parameter]
    [SupplyParameterFromQuery(Name = "Token")]
    public string Token { get; set; } = string.Empty;

    [CascadingParameter(Name = "LogoCascading")]
    public string SrcLogo { get; set; } = string.Empty;

    #endregion

    #region Properties

    public string HeaderText { get; set; }
    = "Enviaremos um e-mail com um instruções de recuperação";

    public bool IsSuccess { get; set; }
    public bool IsBusy { get; set; }
    public ResetPasswordRequest ResetPasswordModel { get; set; } = new();
    public ForgotPasswordRequest ForgotPasswordModel { get; set; } = new();

    #endregion

    #region Services

    [Inject]
    public IAccountHandler AccountHandler { get; set; } = null!;

    [Inject]
    public NavigationManager NavigationManager { get; set; } = null!;

    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    #endregion

    #region Methods

    protected async Task OnValidSubmitForgotFormAsync()
    {
        IsBusy = true;
        try
        {
            var result = await AccountHandler.ForgotPasswordAsync(ForgotPasswordModel);
            IsSuccess = result.IsSuccess;
            if (IsSuccess)
                HeaderText = "O e-mail foi enviado com sucesso!";
            else
                Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }
    public async Task OnValidSubmitResetFormAsync()
```

```
    {
        IsBusy = true;
        try
        {
            ResetPasswordModel.UserId = UserId;
            ResetPasswordModel.Token = Token;
            var result = await AccountHandler.ResetPasswordAsync(ResetPasswordModel);
            if (result.IsSuccess)
            {
                Snackbar.Add(result.Message ?? string.Empty, Severity.Success);
                NavigationManager.NavigateTo("/login");
            }
            else
                Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
        finally
        {
            IsBusy = false;
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Offers\List.razor

```
@page "/propostas"
@using RealtyHub.Core.Models
@inherits ListOffersPage

<MudText Typo="Typo.h3">Propostas</MudText>

<PageTitle>Propostas</PageTitle>

<div class="d-flex justify-end">
    <MudButton Class="ml-4"
               Variant="Variant.Filled"
               Color="Color.Primary"
               StartIcon="@Icons.Material.Filled.FileOpen"
               OnClick="OnReportClickedAsync">
        Gerar Relatório
    </MudButton>
</div>

<MudDataGrid T="Offer"
             Class="mt-4"
             ServerData="LoadServerData"
             @ref="DataGrid"
             RowsPerPage="@Core.Configuration.DefaultPageSize"
             RowClick="@(e => SelectOffer(e.Item))"
             RowStyle="@RowStyle">

    <ToolBarContent>
        <MudText Typo="Typo.h6">Propostas</MudText>
        <MudSpacer />
        <MudDateRangePicker PickerVariant="PickerVariant.Dialog"
                            Label="Escolha a data da proposta"
                            DateRange="DateRange"
                            DateRangeChanged="OnDateRangeChanged"
                            Clearable="true" />
    </ToolBarContent>
    <Columns>
        <PropertyColumn Property="v => v.Id" Title="Id"></PropertyColumn>
        <PropertyColumn Property="v => v.SubmissionDate" Title="Data"></PropertyColumn>
        <PropertyColumn Property="v => v.Buyer!.Name" Title="Comprador"></PropertyColumn>
        <PropertyColumn Property="v => v.Property!.Seller!.Name"
Title="Vendedor"></PropertyColumn>
        <PropertyColumn Property="v => v.Property!.Title"
Title="Imóvel"></PropertyColumn>
        <TemplateColumn Title="Valor">
            <CellTemplate>
                @context.Item.Amount.ToString("C")
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Title="Status">
            <CellTemplate>
                <OfferStatus Status="@context.Item.OfferStatus" />
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Class="justify-end" Title="Ações">
            <CellTemplate>
                <MudTooltip Text="Visualizar">
                    <MudIconButton Icon="@Icons.Material.Filled.Visibility"
                                   Color="Color.Primary"
                                   OnClick="() => OpenOfferDialog(context.Item)" />
                </MudTooltip>
                <MudTooltip Text="Aceitar">
                    <MudIconButton Icon="@Icons.Material.Filled.Done"
                                   Color="Color.Success"
                                   OnClick="() => OnAcceptClickedAsync(context.Item)"
                                   Disabled="@(context.Item.OfferStatus !=
EOfferStatus.Analysis)"/>
                </MudTooltip>
                <MudTooltip Text="Rejeitar">
                    <MudIconButton Icon="@Icons.Material.Filled.Cancel"
                                   Color="Color.Error"
                                   OnClick="() => OnRejectClickedAsync(context.Item)"
                                   Disabled="@(context.Item.OfferStatus !=
```

```
EOfferStatus.Analysis)"/>
                </MudTooltip>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Offer" />
    </PagerContent>
</MudDataGrid>

<style>
    .detalhes-cell {
        white-space: normal;
        word-wrap: break-word;
        max-width: 200px;
    }
</style>
```

.\RealtyHub.Web\Pages\Offers\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;
using MudBlazor;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Extensions;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Offers;
using RealtyHub.Web.Components.Common;
using RealtyHub.Web.Components.Offers;
using RealtyHub.Web.Services;

namespace RealtyHub.Web.Pages.Offers;

/// <summary>
/// Página responsável por exibir e gerenciar a listagem de propostas de compra de imóvel
na aplicação.
/// </summary>
public partial class ListOffersPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Evento chamado quando uma proposta é selecionada.
    /// </summary>
    [Parameter]
    public EventCallback<Offer> OnOfferSelected { get; set; }

    /// <summary>
    /// Estilo CSS aplicado às linhas da tabela.
    /// </summary>
    [Parameter]
    public string RowStyle { get; set; } = string.Empty;

    /// <summary>
    /// Indica se somente as propostas aceitas devem ser exibidas.
    /// </summary>
    [Parameter]
    public bool OnlyAccepted { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Componente de grid do MudBlazor utilizado para exibir a lista de propostas.
    /// </summary>
    public MudDataGrid<Offer> DataGrid { get; set; } = null!;

    /// <summary>
    /// Intervalo de datas utilizado para filtrar as propostas.
    /// Inicia no primeiro dia do mês atual e termina no último dia do mês atual.
    /// </summary>
    public DateRange DateRange { get; set; } = new(DateTime.Now.GetFirstDay(),
DateTime.Now.GetLastDay());

    /// <summary>
    /// Lista de propostas a serem exibidas no grid.
    /// </summary>
    public List<Offer> Items { get; set; } = [];

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    /// <summary>
```

```
    /// Serviço para exibição de notificações na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações relacionadas a propostas.
    /// </summary>
    [Inject]
    public IOfferHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço responsável por gerar relatórios de propostas.
    /// </summary>
    [Inject]
    public OfferReport OfferReport { get; set; } = null!;

    /// <summary>
    /// Serviço JavaScript para interações com o navegador, como abrir um PDF em nova
aba.
    /// </summary>
    [Inject]
    public IJSRuntime JsRuntime { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o clique do botão de gerar relatório da proposta.
    /// Caso o relatório seja gerado com sucesso, abre o PDF em uma nova aba.
    /// </summary>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task OnReportClickedAsync()
    {
        var result = await OfferReport.GetOfferAsync();
        if (result.IsSuccess)
        {
            await JsRuntime.InvokeVoidAsync("openContractPdfInNewTab", result.Data?.Url);
            return;
        }
        Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
    }

    /// <summary>
    /// Carrega os dados das propostas do servidor de forma paginada para exibição no
grid.
    /// </summary>
    /// <param name="state">Estado atual do grid contendo informações de
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{Offer}"/> contendo a lista de propostas e a
contagem total.
    /// Caso ocorra uma falha, retorna um grid vazio e exibe uma mensagem de erro.
    /// </returns>
    public async Task<GridData<Offer>> LoadServerData(GridState<Offer> state)
    {
        try
        {
            var endDate = DateRange.End?.ToEndOfDay();
            var request = new GetAllOffersRequest
            {
                PageNumber = state.Page + 1,
                PageSize = state.PageSize,
                StartDate = DateRange.Start.ToUtcString(),
                EndDate = endDate.ToUtcString()
            };

            var response = await Handler.GetAllAsync(request);
            if (response.IsSuccess)
            {
                if (OnlyAccepted)
                    Items = response.Data?
                        .Where(o => o.OfferStatus == EOfferStatus.Accepted)
                        .ToList() ?? new List<Offer>();
                else
```

```
                    Items = response.Data ?? new List<Offer>();

                return new GridData<Offer>
                {
                    Items = Items.OrderByDescending(o => o.UpdatedAt),
                    TotalItems = response.TotalCount
                };
            }

            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<Offer>();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Offer>();
        }
    }

    /// <summary>
    /// Atualiza o intervalo de datas utilizado para filtrar as propostas e recarrega os
dados do grid.
    /// </summary>
    /// <param name="newDateRange">Novo intervalo de datas selecionado.</param>
    public void OnDateRangeChanged(DateRange newDateRange)
    {
        DateRange = newDateRange;
        DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Manipula o clique para aceitar uma proposta.
    /// Exibe um diálogo de confirmação e, se confirmado, envia a solicitação para
aceitar a proposta.
    /// Após a operação, exibe uma notificação e recarrega o grid.
    /// </summary>
    /// <param name="offer">Proposta a ser aceita.</param>
    public async Task OnAcceptClickedAsync(Offer offer)
    {
        if (IsOfferStatusInvalid(offer)) return;
        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja aceitar a proposta?<br>" +
                             "Após a alteração não será mais possível editar!" },
            { "ButtonColor", Color.Success }
        };
        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        if (await dialog.Result is { Canceled: true }) return;

        var request = new AcceptOfferRequest { Id = offer.Id };
        var result = await Handler.AcceptAsync(request);
        if (result is { IsSuccess: true, Data: not null })
        {
            Snackbar.Add("Proposta aceita", Severity.Success);
            await DataGrid.ReloadServerData();
        }
        else
        {
            Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
        }
    }

    /// <summary>
    /// Manipula o clique para rejeitar uma proposta.
    /// Exibe um diálogo de confirmação e, se confirmado, envia a solicitação para
rejeitar a proposta.
    /// Após a operação, exibe uma notificação e recarrega o grid.
    /// </summary>
    /// <param name="offer">Proposta a ser rejeitada.</param>
    public async Task OnRejectClickedAsync(Offer offer)
    {
        if (IsOfferStatusInvalid(offer)) return;
        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja recusar a proposta?<br>" +
```

```
                            "Após a alteração não será mais possível editar!" },
            { "ButtonColor", Color.Error }
        };
        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        if (await dialog.Result is { Canceled: true }) return;

        var request = new RejectOfferRequest { Id = offer.Id };
        var result = await Handler.RejectAsync(request);
        if (result is { IsSuccess: true, Data: not null })
        {
            Snackbar.Add("Proposta recusada", Severity.Success);
            await DataGrid.ReloadServerData();
        }
        else
        {
            Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
        }
    }

    /// <summary>
    /// Abre uma caixa de diálogo para visualizar os detalhes completos de uma proposta.
    /// </summary>
    /// <param name="offer">Proposta a ser visualizada.</param>
    public async Task OpenOfferDialog(Offer offer)
    {
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Medium,
            CloseOnEscapeKey = true,
            FullWidth = true
        };

        var parameters = new DialogParameters
        {
            { "OfferId", offer.Id },
            { "ReadyOnly", offer.OfferStatus != EOfferStatus.Analysis },
            { "ShowPayments", true }
        };

        await DialogService.ShowAsync<OfferDialog>("Visualizar proposta", parameters,
options);
    }

    /// <summary>
    /// Notifica o componente pai que uma proposta foi selecionada.
    /// </summary>
    /// <param name="offer">Proposta selecionada.</param>
    public async Task SelectOffer(Offer offer)
    {
        if (OnOfferSelected.HasDelegate)
            await OnOfferSelected.InvokeAsync(offer);
    }

    /// <summary>
    /// Verifica se o status da proposta é inválido para alteração.
    /// Se a proposta estiver em status "Accepted" ou "Rejected", exibe uma mensagem de
alerta.
    /// </summary>
    /// <param name="offer">Proposta a ser verificada.</param>
    /// <returns>
    /// True se o status da proposta for inválido para alteração; caso contrário, false.
    /// </returns>
    private bool IsOfferStatusInvalid(Offer offer)
    {
        switch (offer.OfferStatus)
        {
            case EOfferStatus.Accepted:
                Snackbar.Add("Proposta está aceita", Severity.Warning);
                return true;
            case EOfferStatus.Rejected:
                Snackbar.Add("Proposta está cancelada", Severity.Warning);
                return true;
            default:
                return false;
```

```
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Properties\Create.razor

```
@page "/imoveis/adicionar"
@inherits PropertyFormComponent

<PropertyForm/>
```

.\RealtyHub.Web\Pages\Properties\Edit.razor

```
@page "/imoveis/editar/{id:long}"
@inherits PropertyFormComponent

<PropertyForm Id="@Id"/>
```

.\RealtyHub.Web\Pages\Properties\List.razor

```
@page "/imoveis"
@using RealtyHub.Core.Extensions
@using RealtyHub.Core.Models
@inherits ListPropertiesPage

<MudText Typo="Typo.h3">Imóveis</MudText>

<PageTitle>Imóveis</PageTitle>

<div class="d-flex justify-end">
    <MudButton Variant="Variant.Filled"
               Color="Color.Primary"
               StartIcon="@Icons.Material.Filled.AddHome"
               Href="/imoveis/adicionar">
        Novo Imóvel
    </MudButton>
    <MudButton Class="ml-4"
               Variant="Variant.Filled"
               Color="Color.Primary"
               StartIcon="@Icons.Material.Filled.FileOpen"
               OnClick="OnReportClickedAsync">
        Gerar Relatório
    </MudButton>
</div>

<MudDataGrid T="Property"
             Class="mt-4"
             ServerData="LoadServerData"
             @ref="DataGrid"
             RowsPerPage="@Core.Configuration.DefaultPageSize"
             RowClick="@(e => SelectProperty(e.Item))"
             RowStyle="@RowStyle">

    <ToolBarContent>
        <MudText Typo="Typo.h6">Imóveis</MudText>
        <MudSpacer />
        <MudSelect T="string"
                   Label="Selecione o Filtro"
                   Value="SelectedFilter"
                   ValueChanged="OnValueFilterChanged"
                   Class="me-2">
            @foreach (var option in FilterOptions)
            {
                <MudSelectItem
Value="@option.PropertyName">@option.DisplayName</MudSelectItem>
            }
        </MudSelect>
        <MudTextField @bind-Value="SearchTerm"
                      Placeholder="Filtrar..."
                      Adornment="Adornment.Start"
                      AdornmentIcon="@Icons.Material.Filled.Search"
                      Clearable="true"
                      OnClearButtonClick="OnClearSearchClick"
                      IconSize="Size.Medium"
                      Class="me-2">
        </MudTextField>
        <MudButton StartIcon="@Icons.Material.Filled.Search"
                   OnClick="OnButtonSearchClick"
                   ButtonType="ButtonType.Button"
                   Color="Color.Primary"
                   Variant="Variant.Filled">
            Pesquisar
        </MudButton>
    </ToolBarContent>
    <Columns>
        <TemplateColumn Title="Foto principal">
            <CellTemplate>
                <MudImage Src="@GetSrcThumbnailPhoto(context.Item)"
                          Width="300"
                          Height="300"
                          ObjectFit="ObjectFit.Cover" />
            </CellTemplate>
```

```
        </TemplateColumn>
        <PropertyColumn Property="x => x.Title" Title="Título" />
        <PropertyColumn Property="x => x.Bedroom" Title="Quartos" />
        <PropertyColumn Property="x => x.Bathroom" Title="Banheiros" />
        <PropertyColumn Property="x => x.Garage" Title="Garagens" />
        <PropertyColumn Property="x => x.Area" Title="Área" />
        <TemplateColumn Title="Preço">
            <CellTemplate>
                @context.Item.Price.ToString("C")
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Title="Tipo">
            <CellTemplate>
                @context.Item.PropertyType.GetDisplayName()
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Class="justify-end" Title="Ações">
            <CellTemplate>
                <MudStack Row>
                    <MudTooltip Text="Editar">
                        <MudIconButton Icon="@Icons.Material.Filled.Edit"
                                       Href="@($"/imoveis/editar/{context.Item.Id}")"
                                       Color="Color.Primary">
                        </MudIconButton>
                    </MudTooltip>
                    <MudTooltip Text="Excluir">
                        <MudIconButton Icon="@Icons.Material.Filled.Delete"
                                       Color="Color.Error"
                                       aria-label="Excluir"
                                       OnClick="() =>
OnDeleteButtonClickedAsync(context.Item.Id, context.Item.Title)">
                        </MudIconButton>
                    </MudTooltip>
                </MudStack>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Property" />
    </PagerContent>
</MudDataGrid>
```

.\RealtyHub.Web\Pages\Properties\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Web.Components.Common;
using RealtyHub.Web.Services;

namespace RealtyHub.Web.Pages.Properties;

/// <summary>
/// Página responsável por exibir e gerenciar a listagem de imóveis na aplicação.
/// </summary>
public partial class ListPropertiesPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Evento chamado quando um imóvel é selecionado.
    /// </summary>
    [Parameter]
    public EventCallback<Property> OnPropertySelected { get; set; }

    /// <summary>
    /// Estilo CSS aplicado às linhas da tabela.
    /// </summary>
    [Parameter]
    public string RowStyle { get; set; } = string.Empty;

    #endregion

    #region Properties

    /// <summary>
    /// Componente de grid do MudBlazor utilizado para exibir os imóveis.
    /// </summary>
    public MudDataGrid<Property> DataGrid { get; set; } = null!;

    /// <summary>
    /// Lista de imóveis a serem exibidos no grid.
    /// </summary>
    public List<Property> Properties { get; set; } = [];

    /// <summary>
    /// Termo de busca para filtrar os imóveis.
    /// </summary>
    public string SearchTerm { get; set; } = string.Empty;

    /// <summary>
    /// Filtro selecionado para busca, aplicável às propriedades.
    /// </summary>
    public string SelectedFilter { get; set; } = string.Empty;

    /// <summary>
    /// Lista de opções de filtro para a busca de imóveis.
    /// </summary>
    public readonly List<FilterOption> FilterOptions = new()
    {
        new FilterOption { DisplayName = "Título", PropertyName = "Title" },
        new FilterOption { DisplayName = "Descrição", PropertyName = "Description" },
        new FilterOption { DisplayName = "Nome do condomínio", PropertyName =
"Condominium.Name" },
        new FilterOption { DisplayName = "Bairro", PropertyName = "Address.Neighborhood"
},
        new FilterOption { DisplayName = "Rua", PropertyName = "Address.Street" },
        new FilterOption { DisplayName = "Cidade", PropertyName = "Address.City" },
        new FilterOption { DisplayName = "Estado", PropertyName = "Address.State" },
        new FilterOption { DisplayName = "CEP", PropertyName = "Address.ZipCode" },
        new FilterOption { DisplayName = "País", PropertyName = "Address.Country" },
        new FilterOption { DisplayName = "Garagens", PropertyName = "Garage" },
        new FilterOption { DisplayName = "Quartos", PropertyName = "Bedroom" },
```

```
        new FilterOption { DisplayName = "Banheiros", PropertyName = "Bathroom" },
        new FilterOption { DisplayName = "Área", PropertyName = "Area" },
        new FilterOption { DisplayName = "Preço", PropertyName = "Price" },
    };

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de notificações (snackbars) na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações de imóveis, como recuperar e excluir
propriedades.
    /// </summary>
    [Inject]
    public IPropertyHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    /// <summary>
    /// Serviço JavaScript para interagir com o navegador (ex.: abrir novo aba).
    /// </summary>
    [Inject]
    public IJSRuntime JsRuntime { get; set; } = null!;

    /// <summary>
    /// Serviço responsável por gerar relatórios de propriedades.
    /// </summary>
    [Inject]
    public PropertyReport PropertyReport { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Manipula o clique do botão para gerar o relatório do imóvel.
    /// Caso o relatório seja gerado com sucesso, abre o PDF em uma nova aba.
    /// </summary>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task OnReportClickedAsync()
    {
        var result = await PropertyReport.GetPropertyAsync();
        if (result.IsSuccess)
        {
            await JsRuntime.InvokeVoidAsync("openContractPdfInNewTab", result.Data?.Url);
            return;
        }
        Snackbar.Add(result.Message ?? string.Empty, Severity.Error);
    }

    /// <summary>
    /// Abre um diálogo de confirmação para a exclusão de um imóvel e, se confirmado,
executa a exclusão.
    /// Após a exclusão, recarrega os dados no grid e exibe uma notificação.
    /// </summary>
    /// <param name="id">Identificador do imóvel a ser excluído.</param>
    /// <param name="title">Título do imóvel, usado na mensagem de confirmação.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task OnDeleteButtonClickedAsync(long id, string title)
    {
        var parameters = new DialogParameters
        {
            { "ContentText", $"Ao prosseguir o imóvel {title} será excluido. " +
                             "Esta é uma ação irreversível! Deseja continuar?" },
            { "ButtonText", "Confirmar" },
            { "ButtonColor", Color.Error }
```

```
        };

        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Small
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Atenção", parameters,
options);
        var result = await dialog.Result;

        if (result is { Canceled: true }) return;

        await OnDeleteAsync(id, title);
        StateHasChanged();
    }

    /// <summary>
    /// Executa a exclusão efetiva do imóvel.
    /// Remove o imóvel da lista local, recarrega os dados do grid e exibe uma
notificação.
    /// </summary>
    /// <param name="id">Identificador do imóvel a ser excluído.</param>
    /// <param name="name">Título do imóvel para a mensagem de sucesso.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    private async Task OnDeleteAsync(long id, string name)
    {
        try
        {
            await Handler.DeleteAsync(new DeletePropertyRequest { Id = id });
            Properties.RemoveAll(x => x.Id == id);
            await DataGrid.ReloadServerData();
            Snackbar.Add($"Imóvel {name} excluído", Severity.Success);
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    /// <summary>
    /// Carrega os dados dos imóveis do servidor de forma paginada para exibição no grid.
    /// </summary>
    /// <param name="state">Estado atual do grid contendo informações de
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{Property}"/> contendo a lista de imóveis e a
contagem total.
    /// Em caso de falha, retorna um grid vazio e exibe uma mensagem de erro.
    /// </returns>
    public async Task<GridData<Property>> LoadServerData(GridState<Property> state)
    {
        try
        {
            var request = new GetAllPropertiesRequest
            {
                PageNumber = state.Page + 1,
                PageSize = state.PageSize,
                SearchTerm = SearchTerm,
                FilterBy = SelectedFilter
            };

            var response = await Handler.GetAllAsync(request);
            if (response.IsSuccess)
                return new GridData<Property>
                {
                    Items = response.Data ?? [],
                    TotalItems = response.TotalCount
                };

            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<Property>();
        }
        catch (Exception e)
        {
```

```
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Property>();
        }
    }

    /// <summary>
    /// Aciona a recarga dos dados do grid com base no termo de busca atual.
    /// </summary>
    public void OnButtonSearchClick() => DataGrid.ReloadServerData();

    /// <summary>
    /// Limpa o termo de busca e recarrega o grid.
    /// </summary>
    public void OnClearSearchClick()
    {
        SearchTerm = string.Empty;
        DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Atualiza o filtro selecionado e solicita a atualização da interface.
    /// </summary>
    /// <param name="newValue">Novo valor do filtro.</param>
    public void OnValueFilterChanged(string newValue)
    {
        SelectedFilter = newValue;
        StateHasChanged();
    }

    /// <summary>
    /// Notifica o componente pai que um imóvel foi selecionado.
    /// </summary>
    /// <param name="property">Imóvel selecionado.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task SelectProperty(Property property)
    {
        if (OnPropertySelected.HasDelegate)
            await OnPropertySelected.InvokeAsync(property);
    }

    /// <summary>
    /// Obtém a URL da foto thumbnail do imóvel.
    /// Se uma foto marcada como thumbnail existir, utiliza-a; caso contrário, utiliza a
primeira foto disponível.
    /// </summary>
    /// <param name="property">Imóvel cujo thumbnail será obtido.</param>
    /// <returns>String com a URL da foto.</returns>
    public string GetSrcThumbnailPhoto(Property property)
    {
        var photo = property
                        .PropertyPhotos
                        .FirstOrDefault(p => p.IsThumbnail)
                    ?? property.PropertyPhotos.FirstOrDefault();

        return $"{Configuration.BackendUrl}/photos/{photo?.Id}{photo?.Extension}";
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\PropertyListHome.razor

```
@page "/listar-imoveis"
@using RealtyHub.Core.Models
@inherits PropertyListHomePage
@layout PublicLayout

<PageTitle>Imóveis</PageTitle>

@if (IsBusy)
{
    <div class="d-flex justify-center align-center" style="height: 50vh">
        <MudProgressCircular Size="Size.Large" Color="Color.Info" Indeterminate="true" />
    </div>
}
else
{
    <MudPaper Class="mb-4">
        <MudDataGrid T="Property"
                     Class="mt-4"
                     ServerData="LoadServerData"
                     @ref="DataGrid"
                     RowsPerPage="@Core.Configuration.DefaultPageSize">

            <ToolBarContent>
                <MudText Typo="Typo.h6">Imóveis</MudText>
                <MudSpacer />
                <MudSelect T="string"
                           Label="Selecione o Filtro"
                           Value="SelectedFilter"
                           ValueChanged="OnValueFilterChanged"
                           Class="me-2">
                    @foreach (var option in FilterOptions)
                    {
                        <MudSelectItem
Value="@option.PropertyName">@option.DisplayName</MudSelectItem>
                    }
                </MudSelect>
                <MudTextField @bind-Value="SearchTerm"
                              Placeholder="Filtrar..."
                              Adornment="Adornment.Start"
                              AdornmentIcon="@Icons.Material.Filled.Search"
                              Clearable="true"
                              OnClearButtonClick="OnClearSearchClick"
                              IconSize="Size.Medium"
                              Class="me-2">
                </MudTextField>
                <MudButton StartIcon="@Icons.Material.Filled.Search"
                           OnClick="OnButtonSearchClick"
                           ButtonType="ButtonType.Button"
                           Color="Color.Primary"
                           Variant="Variant.Filled">
                    Pesquisar
                </MudButton>
            </ToolBarContent>
            <Columns>
                <TemplateColumn Title="">
                    <CellTemplate>
                        <MudImage Src="@GetSrcThumbnailPhoto(context.Item)"
                                  Width="300"
                                  Height="300"
                                  ObjectFit="ObjectFit.Cover" />
                    </CellTemplate>
                </TemplateColumn>
                <TemplateColumn Title="">
                    <CellTemplate>
                        <MudText Typo="Typo.body2">@context.Item.Description</MudText>
<br />
                        <MudText Typo="Typo.body2">@context.Item.Bedroom
@(context.Item.Bedroom > 1 ? "Quartos" : "Quarto")</MudText> <br />
                        <MudText Typo="Typo.body2">@context.Item.Bathroom
@(context.Item.Bathroom > 1 ? "Banheiros" : "Banheiro")</MudText> <br />
                        <MudText Typo="Typo.body2">@context.Item.Garage
@(context.Item.Garage > 1 ? "Garagens" : "Garagem")</MudText> <br />
```

```
                        <MudText Typo="Typo.body2">Localização:
@context.Item.Address.Neighborhood</MudText> <br />
                        <MudText Typo="Typo.body2">@context.Item.Area M²: </MudText> <br
/>
                        <MudText
Typo="Typo.body2">@context.Item.Price.ToString("C")</MudText> <br />
                        <MudText
Typo="Typo.body2">@context.Item.TransactionsDetails</MudText> <br />
                    </CellTemplate>
                </TemplateColumn>
                <TemplateColumn Class="justify-end" Title="">
                    <CellTemplate>
                        <MudStack Row>
                            <MudTooltip Text="Enviar proposta">
                                <MudIconButton Icon="@Icons.Material.Filled.Send"
                                               OnClick="@(() =>
OnSendOfferClickedAsync(context.Item))"
                                               Color="Color.Primary">
                                </MudIconButton>
                            </MudTooltip>
                            <MudTooltip Text="Mais informações">
                                <MudIconButton Icon="@Icons.Material.Filled.Info"
                                               Color="Color.Primary"

Href="@($"/imoveis/detalhes/{context.Item.Id}")">
                                </MudIconButton>
                            </MudTooltip>
                        </MudStack>
                    </CellTemplate>
                </TemplateColumn>
            </Columns>
            <PagerContent>
                <MudDataGridPager T="Property" />
            </PagerContent>
        </MudDataGrid>
    </MudPaper>
}
```

.\RealtyHub.Web\Pages\PropertyListHome.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Web.Components.Offers;

namespace RealtyHub.Web.Pages;

/// <summary>
/// Página responsável por exibir e gerenciar a listagem de imóveis na página inicial da
aplicação.
/// </summary>
public partial class PropertyListHomePage : ComponentBase
{
    #region Properties

    /// <summary>
    /// Indica se a página está ocupada realizando alguma operação (ex.: carregamento de
dados).
    /// </summary>
    public bool IsBusy { get; set; }

    /// <summary>
    /// Componente de grid do MudBlazor utilizado para exibir a lista de imóveis.
    /// </summary>
    public MudDataGrid<Property> DataGrid { get; set; } = null!;

    /// <summary>
    /// Lista de imóveis a serem exibidos no grid.
    /// </summary>
    public List<Property> Properties { get; set; } = [];

    /// <summary>
    /// Termo de busca para filtrar os imóveis.
    /// </summary>
    public string SearchTerm { get; set; } = string.Empty;

    /// <summary>
    /// Filtro selecionado para refinar a busca.
    /// </summary>
    public string SelectedFilter { get; set; } = string.Empty;

    /// <summary>
    /// Lista de opções de filtro disponíveis para a busca de imóveis.
    /// </summary>
    public readonly List<FilterOption> FilterOptions = new()
    {
        new FilterOption { DisplayName = "Descrição", PropertyName = "Description" },
        new FilterOption { DisplayName = "Bairro", PropertyName = "Address.Neighborhood"
},
        new FilterOption { DisplayName = "Garagens", PropertyName = "Garage" },
        new FilterOption { DisplayName = "Quartos", PropertyName = "Bedroom" },
        new FilterOption { DisplayName = "Banheiros", PropertyName = "Bathroom" },
        new FilterOption { DisplayName = "Área", PropertyName = "Area" },
        new FilterOption { DisplayName = "Preço", PropertyName = "Price" },
    };

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de notificações (snackbars) na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações relacionadas a imóveis, como listar
propriedades.
    /// </summary>
```

```
    [Inject]
    public IPropertyHandler Handler { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Carrega os dados dos imóveis do servidor de forma paginada para exibição no grid.
    /// </summary>
    /// <param name="state">Estado atual do grid, contendo informações de
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{Property}"/> contendo a lista de imóveis e a
contagem total.
    /// Em caso de falha, retorna um grid vazio e exibe uma mensagem de erro.
    /// </returns>
    public async Task<GridData<Property>> LoadServerData(GridState<Property> state)
    {
        try
        {
            var request = new GetAllPropertiesRequest
            {
                PageNumber = state.Page + 1,
                PageSize = state.PageSize,
                SearchTerm = SearchTerm,
                FilterBy = SelectedFilter
            };

            var response = await Handler.GetAllAsync(request);
            if (response.IsSuccess)
            {
                return new GridData<Property>
                {
                    Items = response.Data ?? [],
                    TotalItems = response.TotalCount
                };
            }

            Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            return new GridData<Property>();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Property>();
        }
    }

    /// <summary>
    /// Aciona a recarga dos dados do grid com base no termo de busca e nos filtros
atuais.
    /// </summary>
    public void OnButtonSearchClick() => DataGrid.ReloadServerData();

    /// <summary>
    /// Limpa o termo de busca e recarrega os dados do grid.
    /// </summary>
    public void OnClearSearchClick()
    {
        SearchTerm = string.Empty;
        DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Atualiza o filtro selecionado e solicita a atualização da interface.
    /// </summary>
    /// <param name="newValue">Novo valor do filtro.</param>
    public void OnValueFilterChanged(string newValue)
    {
```

```
        SelectedFilter = newValue;
        StateHasChanged();
    }

    /// <summary>
    /// Obtém a URL da foto thumbnail do imóvel.
    /// Se houver uma foto marcada como thumbnail, essa URL é utilizada; caso contrário,
utiliza a primeira foto disponível.
    /// </summary>
    /// <param name="property">Imóvel cujo thumbnail será obtido.</param>
    /// <returns>String com a URL da foto.</returns>
    public string GetSrcThumbnailPhoto(Property property)
    {
        var photo = property
            .PropertyPhotos
            .FirstOrDefault(p => p.IsThumbnail)
            ?? property.PropertyPhotos.FirstOrDefault();

        return $"{Configuration.BackendUrl}/photos/{photo?.Id}{photo?.Extension}";
    }

    /// <summary>
    /// Abre um diálogo modal para o envio de uma proposta de compra para o imóvel
selecionado.
    /// </summary>
    /// <param name="property">Imóvel para o qual a proposta será enviada.</param>
    /// <returns>Task representando a operação assíncrona.</returns>
    public async Task OnSendOfferClickedAsync(Property property)
    {
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Medium,
            CloseOnEscapeKey = true,
            FullWidth = true
        };

        var parameters = new DialogParameters
        {
            { "PropertyId", property.Id }
        };

        await DialogService.ShowAsync<OfferDialog>("Enviar proposta", parameters,
options);
    }

    #endregion
}
```

.\RealtyHub.Web\Pages\Viewings\Create.razor

```
@page "/visitas/agendar"
@inherits ViewingFormComponent

<ViewingForm/>
```

.\RealtyHub.Web\Pages\Viewings\Edit.razor

```
@page "/visitas/reagendar/{id:long}"
@inherits ViewingFormComponent

<ViewingForm Id="@Id"/>
```

.\RealtyHub.Web\Pages\Viewings\List.razor

```
@page "/visitas/imoveis/{propertyId:long}"
@page "/visitas/imoveis"
@page "/visitas"
@using RealtyHub.Core.Models
@inherits ListViewingsPage

<PageTitle>Visitas</PageTitle>

<MudText Typo="Typo.h5">@HeaderTitle</MudText>

<div class="d-flex justify-end">
    <MudButton Variant="Variant.Filled"
               Color="Color.Primary"
               StartIcon="@Icons.Material.Filled.ScheduleSend"
               OnClick="OnScheduleButtonClickedAsync">
        Agendar nova visita
    </MudButton>
</div>

<MudDataGrid T="Viewing"
             Class="mt-4"
             ServerData="LoadServerData"
             @ref="DataGrid"
             RowsPerPage="@Core.Configuration.DefaultPageSize">
    <ToolBarContent>
        <MudText Typo="Typo.h6">Visitas</MudText>
        <MudSpacer />
        <MudDateRangePicker PickerVariant="PickerVariant.Dialog"
                            Label="Escolha a data da visita"
                            DateRange="DateRange"
                            DateRangeChanged="OnDateRangeChanged"
                            Clearable="true" />
    </ToolBarContent>
    <Columns>
        <PropertyColumn Property="v=>v.ViewingDate" Title="Data"></PropertyColumn>
        <PropertyColumn Property="v=>v.Buyer!.Name" Title="Cliente"></PropertyColumn>
        @if (Property is null)
        {
            <PropertyColumn Property="v=>v.Property!.Title"
Title="Imóvel"></PropertyColumn>
        }
        <TemplateColumn Title="Status">
            <CellTemplate>
                <ViewingStatus Status="@context.Item.ViewingStatus" />
            </CellTemplate>
        </TemplateColumn>
        <TemplateColumn Class="justify-end" Title="Ações">
            <CellTemplate>
                <MudStack Row>
                    <MudTooltip Text="Reagendar">
                        <MudIconButton Icon="@Icons.Material.Filled.Schedule"
                                       Color="Color.Primary"
                                       OnClick="() =>
OnRescheduleButtonClickedAsync(context.Item)"
                                       Disabled="@(context.Item.ViewingStatus !=
EViewingStatus.Scheduled)" />
                    </MudTooltip>
                    <MudTooltip Text="Finalizar">
                        <MudIconButton Icon="@Icons.Material.Filled.Done"
                                       Color="Color.Primary"
                                       OnClick="() =>
OnDoneButtonClickedAsync(context.Item)"
                                       Disabled="@(context.Item.ViewingStatus !=
EViewingStatus.Scheduled)" />
                    </MudTooltip>
                    <MudTooltip Text="Cancelar">
                        <MudIconButton Icon="@Icons.Material.Filled.Cancel"
                                       Color="Color.Error"
                                       OnClick="() =>
OnCancelButtonClickedAsync(context.Item)"
                                       Disabled="@(context.Item.ViewingStatus !=
EViewingStatus.Scheduled)" />
```

```
                    </MudTooltip>
                </MudStack>
            </CellTemplate>
        </TemplateColumn>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Viewing"/>
    </PagerContent>
</MudDataGrid>
```

.\RealtyHub.Web\Pages\Viewings\List.razor.cs

```
using Microsoft.AspNetCore.Components;
using MudBlazor;
using RealtyHub.Core.Enums;
using RealtyHub.Core.Extensions;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Models;
using RealtyHub.Core.Requests.Properties;
using RealtyHub.Core.Requests.Viewings;
using RealtyHub.Web.Components.Common;
using RealtyHub.Web.Components.Viewings;

namespace RealtyHub.Web.Pages.Viewings;

/// <summary>
/// Página responsável por exibir e gerenciar a listagem de visitas de um imóvel ou de
todas as visitas.
/// </summary>
public partial class ListViewingsPage : ComponentBase
{
    #region Parameters

    /// <summary>
    /// Identificador do imóvel para filtrar as visitas.
    /// Se 0, a listagem exibirá todas as visitas.
    /// </summary>
    [Parameter]
    public long PropertyId { get; set; }

    #endregion

    #region Properties

    /// <summary>
    /// Componente de grid do MudBlazor utilizado para exibir as visitas.
    /// </summary>
    public MudDataGrid<Viewing> DataGrid { get; set; } = null!;

    /// <summary>
    /// Intervalo de datas utilizado para filtrar as visitas.
    /// Inicializado do primeiro ao último dia do mês atual.
    /// </summary>
    public DateRange DateRange { get; set; } = new(DateTime.Now.GetFirstDay(),
DateTime.Now.GetLastDay());

    /// <summary>
    /// Lista de visitas a serem exibidas no grid.
    /// </summary>
    public List<Viewing> Items { get; set; } = [];

    /// <summary>
    /// Imóvel associado às visitas. Pode ser nulo se <see cref="PropertyId"/> for 0.
    /// </summary>
    public Property? Property { get; set; }

    /// <summary>
    /// Título do cabeçalho da página.
    /// Exibe "Visitas do imóvel {Property.Title}" se um imóvel estiver definido; caso
contrário, exibe "Todas as visitas".
    /// </summary>
    public string HeaderTitle => Property is null ? "Todas as visitas" : $"Visitas do
imóvel {Property.Title}";

    #endregion

    #region Services

    /// <summary>
    /// Serviço para exibição de notificações (snackbars) na tela.
    /// </summary>
    [Inject]
    public ISnackbar Snackbar { get; set; } = null!;
```

```
    /// <summary>
    /// Handler responsável pelas operações relacionadas a visitas.
    /// </summary>
    [Inject]
    public IViewingHandler ViewingHandler { get; set; } = null!;

    /// <summary>
    /// Handler responsável pelas operações relacionadas aos imóveis.
    /// Utilizado para recuperar informações do imóvel e suas visitas.
    /// </summary>
    [Inject]
    public IPropertyHandler PropertyHandler { get; set; } = null!;

    /// <summary>
    /// Serviço para exibição de diálogos modais.
    /// </summary>
    [Inject]
    public IDialogService DialogService { get; set; } = null!;

    #endregion

    #region Methods

    /// <summary>
    /// Carrega os dados das visitas do servidor de forma paginada para exibição no grid.
    /// </summary>
    /// <param name="state">Estado atual do grid, contendo informações de
paginação.</param>
    /// <returns>
    /// Um objeto <see cref="GridData{Viewing}"/> contendo a lista de visitas e a
contagem total.
    /// Caso ocorra erro, emite uma notificação e retorna um grid vazio.
    /// </returns>
    public async Task<GridData<Viewing>> LoadServerData(GridState<Viewing> state)
    {
        try
        {
            // Ajusta o final do dia para incluir todas as visitas do dia.
            var endDate = DateRange.End?.AddHours(23).AddMinutes(59).AddSeconds(59);
            string message;
            if (PropertyId != 0)
            {
                // Se um imóvel específico foi informado, busca as visitas associadas.
                var request = new GetAllViewingsByPropertyRequest
                {
                    PropertyId = PropertyId,
                    PageNumber = state.Page + 1,
                    PageSize = state.PageSize,
                    StartDate = DateRange.Start?.ToUniversalTime().ToString("o"),
                    EndDate = endDate?.ToUniversalTime().ToString("o")
                };

                var response = await PropertyHandler.GetAllViewingsAsync(request);
                if (response.IsSuccess)
                    return new GridData<Viewing>
                    {
                        Items = response.Data ?? [],
                        TotalItems = response.TotalCount
                    };
                message = response.Message ?? string.Empty;
            }
            else
            {
                // Caso contrário, busca todas as visitas.
                var request = new GetAllViewingsRequest
                {
                    PageNumber = state.Page + 1,
                    PageSize = state.PageSize,
                    StartDate = DateRange.Start?.ToUniversalTime().ToString("o"),
                    EndDate = endDate?.ToUniversalTime().ToString("o")
                };

                var response = await ViewingHandler.GetAllAsync(request);
                if (response.IsSuccess)
                    return new GridData<Viewing>
                    {
```

```
                        Items = response.Data ?? [],
                        TotalItems = response.TotalCount
                    };
                message = response.Message ?? string.Empty;
            }

            Snackbar.Add(message, Severity.Error);
            return new GridData<Viewing>();
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
            return new GridData<Viewing>();
        }
    }

    /// <summary>
    /// Dispara a atualização do grid de visitas.
    /// </summary>
    public async Task ReloadDataGridAsync()
        => await DataGrid.ReloadServerData();

    /// <summary>
    /// Abre o diálogo para agendamento de uma nova visita.
    /// Após o agendamento, o grid é recarregado para refletir as mudanças.
    /// </summary>
    public async Task OnScheduleButtonClickedAsync()
    {
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Medium,
            FullWidth = true
        };
        var parameters = new DialogParameters
        {
            { "RedirectToPageList", false }
        };
        var dialog = await DialogService.ShowAsync<ViewingDialog>("Agendar visita",
parameters, options);
        var result = await dialog.Result;
        if (result is { Canceled: false })
            await ReloadDataGridAsync();
    }

    /// <summary>
    /// Abre o diálogo para reagendamento de uma visita.
    /// </summary>
    /// <param name="viewing">A visita a ser reagendada.</param>
    public async Task OnRescheduleButtonClickedAsync(Viewing viewing)
    {
        if (IsViewingStatusInValid(viewing)) return;

        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Medium,
            FullWidth = true
        };
        var parameters = new DialogParameters
        {
            { "Property", Property },
            { "LockPropertySearch", true },
            { "RedirectToPageList", false },
            { "Id", viewing.Id }
        };
        await DialogService.ShowAsync<ViewingDialog>("Reagendar visita", parameters,
options);
        await DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Finaliza a visita, marcando-a como concluída.
    /// </summary>
    /// <param name="viewing">A visita a ser finalizada.</param>
    public async Task OnDoneButtonClickedAsync(Viewing viewing)
```

```
    {
        if (IsViewingStatusInValid(viewing)) return;

        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja finalizar a visita?" },
            { "ButtonColor", Color.Warning }
        };

        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        if (await dialog.Result is { Canceled: true }) return;

        var request = new DoneViewingRequest { Id = viewing.Id };
        var result = await ViewingHandler.DoneAsync(request);
        if (result is { IsSuccess: true, Data: not null })
        {
            var viewingExisting = Items.FirstOrDefault(x => x.Id == viewing.Id);
            if (viewingExisting is not null)
                viewingExisting.ViewingStatus = result.Data.ViewingStatus;
        }
        Snackbar.Add(result.Message ?? string.Empty, Severity.Info);
        await DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Cancela a visita.
    /// </summary>
    /// <param name="viewing">A visita a ser cancelada.</param>
    public async Task OnCancelButtonClickedAsync(Viewing viewing)
    {
        if (IsViewingStatusInValid(viewing)) return;
        var parameters = new DialogParameters
        {
            { "ContentText", "Deseja cancelar a visita?" },
            { "ButtonColor", Color.Error }
        };
        var dialog = await DialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters);
        if (await dialog.Result is { Canceled: true }) return;

        var request = new CancelViewingRequest { Id = viewing.Id };
        var result = await ViewingHandler.CancelAsync(request);
        if (result is { IsSuccess: true, Data: not null })
        {
            var viewingExisting = Items.FirstOrDefault(x => x.Id == viewing.Id);
            if (viewingExisting is not null)
                viewingExisting.ViewingStatus = result.Data.ViewingStatus;
        }
        Snackbar.Add(result.Message ?? string.Empty, Severity.Info);
        await DataGrid.ReloadServerData();
    }

    /// <summary>
    /// Verifica se o status da visita é inválido para alterações (finalização ou
cancelamento).
    /// Exibe uma notificação de alerta caso o status seja "Done" ou "Canceled".
    /// </summary>
    /// <param name="viewing">A visita a ser verificada.</param>
    /// <returns>True se a visita não puder ser alterada; caso contrário,
false.</returns>
    private bool IsViewingStatusInValid(Viewing viewing)
    {
        switch (viewing.ViewingStatus)
        {
            case EViewingStatus.Done:
                Snackbar.Add("Visita está finalizada", Severity.Warning);
                return true;
            case EViewingStatus.Canceled:
                Snackbar.Add("Visita está cancelada", Severity.Warning);
                return true;
            default:
                return false;
        }
    }
```

```
    /// <summary>
    /// Atualiza o intervalo de datas utilizado para filtrar as visitas e recarrega o
grid.
    /// </summary>
    /// <param name="newDateRange">Novo intervalo de datas selecionado.</param>
    public void OnDateRangeChanged(DateRange newDateRange)
    {
        DateRange = newDateRange;
        DataGrid.ReloadServerData();
    }

    #endregion

    #region Overrides

    /// <summary>
    /// Inicializa o componente.
    /// Se um <see cref="PropertyId"/> for informado, busca os detalhes do imóvel
correspondente.
    /// </summary>
    protected override async Task OnInitializedAsync()
    {
        try
        {
            if (PropertyId != 0)
            {
                var request = new GetPropertyByIdRequest { Id = PropertyId };
                var response = await PropertyHandler.GetByIdAsync(request);
                if (response is { IsSuccess: true, Data: not null })
                    Property = response.Data;
                else
                    Snackbar.Add(response.Message ?? string.Empty, Severity.Error);
            }
        }
        catch (Exception e)
        {
            Snackbar.Add(e.Message, Severity.Error);
        }
    }

    #endregion
}
```

.\RealtyHub.Web\Program.cs

```
using Blazored.LocalStorage;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using MudBlazor.Services;
using MudBlazor.Translations;
using RealtyHub.Core.Handlers;
using RealtyHub.Core.Services;
using RealtyHub.Web;
using RealtyHub.Web.Handlers;
using RealtyHub.Web.Security;
using RealtyHub.Web.Services;
using System.Globalization;

var builder = WebAssemblyHostBuilder.CreateDefault(args);

builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddScoped<CookieHandler>();

builder.Services.AddAuthorizationCore();

builder.Services.AddScoped<AuthenticationStateProvider,
CookieAuthenticationStateProvider>();
builder.Services.AddScoped(x =>

(ICookieAuthenticationStateProvider)x.GetRequiredService<AuthenticationStateProvider>());
builder.Services.AddMudServices();
builder.Services.AddMudTranslations();
builder.Services.AddBlazoredLocalStorage();

builder.Services
    .AddHttpClient(Configuration.HttpClientName, opt =>
    {
        opt.BaseAddress = new Uri(Configuration.BackendUrl);
    })
    .AddHttpMessageHandler<CookieHandler>();

builder.Services.AddTransient<IAccountHandler, AccountHandler>();
builder.Services.AddTransient<ICustomerHandler, CustomerHandler>();
builder.Services.AddTransient<IPropertyHandler, PropertyHandler>();
builder.Services.AddTransient<ICondominiumHandler, CondominiumHandler>();
builder.Services.AddTransient<IPropertyPhotosHandler, PropertyPhotosHandler>();
builder.Services.AddTransient<IViewingHandler, ViewingHandler>();
builder.Services.AddTransient<IOfferHandler, OfferHandler>();
builder.Services.AddTransient<IContractHandler, ContractHandler>();
builder.Services.AddTransient<IContractTemplateHandler, ContractTemplateHandler>();
builder.Services.AddTransient<IViaCepService, ViaCepService>();
builder.Services.AddTransient<IEmailService, EmailService>();
builder.Services.AddTransient<ShowDialogConfirm>();
builder.Services.AddTransient<DocumentValidator>();
builder.Services.AddTransient<PropertyReport>();
builder.Services.AddTransient<OfferReport>();

builder.Services.AddLocalization();
CultureInfo.DefaultThreadCurrentCulture = CultureInfo.CurrentCulture;
CultureInfo.DefaultThreadCurrentUICulture = CultureInfo.CurrentUICulture;

await builder.Build().RunAsync();
```

.\RealtyHub.Web\Properties\launchSettings.json

```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:34598",
      "sslPort": 44365
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "inspectUri":
"{wsProtocol}://{url.hostname}:{url.port}/_framework/debug/ws-proxy?browser={browserInspe
ctUri}",
      "applicationUrl": "http://localhost:5187",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "inspectUri":
"{wsProtocol}://{url.hostname}:{url.port}/_framework/debug/ws-proxy?browser={browserInspe
ctUri}",
      "applicationUrl": "https://localhost:7294;http://localhost:5187",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": false,
      "inspectUri":
"{wsProtocol}://{url.hostname}:{url.port}/_framework/debug/ws-proxy?browser={browserInspe
ctUri}",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

.\RealtyHub.Web\Security\CookieAuthenticationStateProvider.cs

```
using Microsoft.AspNetCore.Components.Authorization;
using RealtyHub.Core.Models.Account;
using System.Net.Http.Json;
using System.Security.Claims;

namespace RealtyHub.Web.Security;

/// <summary>
/// Provedor de estado de autenticação baseado em cookies. Este provedor obtém
/// informações do usuário e seus respectivos claims a partir de uma API backend,
/// atualizando o estado de autenticação da aplicação.
/// </summary>
public class CookieAuthenticationStateProvider :
AuthenticationStateProvider,
ICookieAuthenticationStateProvider
{
    private readonly HttpClient _httpClient;
    private bool _isAuthenticated;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="CookieAuthenticationStateProvider"/>
    /// utilizando a fábrica de <see cref="HttpClient"/>.
    /// </summary>
    /// <param name="clientFactory">Fábrica para criar instâncias do HttpClient
configuradas para o backend.</param>
    public CookieAuthenticationStateProvider(IHttpClientFactory clientFactory)
    {
        _httpClient = clientFactory.CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Verifica se o usuário está autenticado, atualizando o estado de autenticação.
    /// </summary>
    /// <returns>Task contendo um booleano indicando se o usuário está
autenticado.</returns>
    public async Task<bool> CheckAuthenticatedAsync()
    {
        await GetAuthenticationStateAsync();
        return _isAuthenticated;
    }

    /// <summary>
    /// Notifica a aplicação que o estado de autenticação foi alterado.
    /// </summary>
    public void NotifyAuthenticationStateChanged() =>
        NotifyAuthenticationStateChanged(GetAuthenticationStateAsync());

    /// <summary>
    /// Obtém o estado atual de autenticação do usuário. Este método busca as informações
do usuário
    /// a partir da API backend e, se bem-sucedido, constrói uma lista de claims para
definir o estado autenticado.
    /// </summary>
    /// <returns>
    /// Task contendo um objeto <see cref="AuthenticationState"/> representando o estado
do usuário.
    /// </returns>
    public override async Task<AuthenticationState> GetAuthenticationStateAsync()
    {
        _isAuthenticated = false;
        var user = new ClaimsPrincipal(new ClaimsIdentity());

        var userInfo = await GetUser();
        if (userInfo is null)
            return new AuthenticationState(user);

        var claims = await GetClaims(userInfo);

        var id = new ClaimsIdentity(claims, nameof(CookieAuthenticationStateProvider));
        user = new ClaimsPrincipal(id);

        _isAuthenticated = true;
```

```
        Configuration.GivenName = userInfo.GivenName;
        return new AuthenticationState(user);
    }

    /// <summary>
    /// Tenta obter as informações do usuário a partir da API backend.
    /// </summary>
    /// <returns>
    /// Task contendo o objeto <see cref="User"/> se obtido com sucesso; caso contrário,
null.
    /// </returns>
    private async Task<User?> GetUser()
    {
        try
        {
            return await _httpClient.GetFromJsonAsync<User?>("v1/identity/manageinfo");
        }
        catch
        {
            return null;
        }
    }

    /// <summary>
    /// Constrói uma lista de claims para o usuário com base nas informações obtidas e
nos roles retornados pela API.
    /// </summary>
    /// <param name="user">Objeto <see cref="User"/> contendo as informações do
usuário.</param>
    /// <returns>
    /// Task contendo uma lista de <see cref="Claim"/> representando as autorizações do
usuário.
    /// </returns>
    private async Task<List<Claim>> GetClaims(User user)
    {
        var claims = new List<Claim>
        {
            new (ClaimTypes.Name, user.Email),
            new (ClaimTypes.Email, user.Email),
            new ("Creci", user.Creci),
            new (ClaimTypes.GivenName, user.GivenName)
        };

        claims.AddRange(
            user.Claims
                .Where(x => x.Key != ClaimTypes.Name && x.Key != ClaimTypes.Email)
                .Select(x => new Claim(x.Key, x.Value))
        );

        RoleClaim[]? roles;
        try
        {
            roles = await
_httpClient.GetFromJsonAsync<RoleClaim[]>("v1/identity/manage/roles");
        }
        catch
        {
            return claims;
        }

        foreach (var role in roles ?? [])
            if (!string.IsNullOrEmpty(role.Type) && !string.IsNullOrEmpty(role.Value))
                claims.Add(new Claim(role.Type, role.Value,
                    role.ValueType, role.Issuer, role.OriginalIssuer));

        return claims;
    }
}
```

.\RealtyHub.Web\Security\CookieHandler.cs

```
using Microsoft.AspNetCore.Components.WebAssembly.Http;

namespace RealtyHub.Web.Security;

/// <summary>
/// Handler responsável por configurar as credenciais do navegador e adicionar cabeçalhos
específicos
/// para requisições HTTP, garantindo que os cookies sejam incluídos e o atributo
"X-Requested-With"
/// seja definido para "XMLHttpRequest".
/// </summary>
public class CookieHandler : DelegatingHandler
{
    /// <summary>
    /// Envia a requisição HTTP com as configurações de cookies e cabeçalhos adicionais.
    /// </summary>
    /// <param name="request">A requisição HTTP a ser enviada.</param>
    /// <param name="cancellationToken">Token para cancelamento assíncrono da
operação.</param>
    /// <returns>Task que representa a operação assíncrona e contém a resposta
HTTP.</returns>
    protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
        CancellationToken cancellationToken)
    {
        request.SetBrowserRequestCredentials(BrowserRequestCredentials.Include);
        request.Headers.Add("X-Requested-With", "XMLHttpRequest");
        return base.SendAsync(request, cancellationToken);
    }
}
```

.\RealtyHub.Web\Security\ICookieAuthenticationStateProvider.cs

```
using Microsoft.AspNetCore.Components.Authorization;

namespace RealtyHub.Web.Security;

/// <summary>
/// Interface que define o contrato para um provedor de estado de autenticação baseado em
cookies.
/// </summary>
public interface ICookieAuthenticationStateProvider
{
    /// <summary>
    /// Verifica se o usuário está autenticado.
    /// </summary>
    /// <returns>Task contendo um booleano indicando se o usuário está
autenticado.</returns>
    Task<bool> CheckAuthenticatedAsync();

    /// <summary>
    /// Obtém o estado atual de autenticação do usuário.
    /// </summary>
    /// <returns>Task contendo o objeto <see cref="AuthenticationState"/> com o estado
atual.</returns>
    Task<AuthenticationState> GetAuthenticationStateAsync();

    /// <summary>
    /// Notifica a aplicação que houve alteração no estado de autenticação.
    /// </summary>
    void NotifyAuthenticationStateChanged();
}
```

.\RealtyHub.Web\Services\DocumentValidator.cs

```
namespace RealtyHub.Web.Services;

public class DocumentValidator
{
    /// <summary>
    /// Verifica se o CPF informado é válido.
    /// </summary>
    /// <param name="cpf">O CPF a ser validado.</param>
    /// <returns>True se o CPF for válido; caso contrário, false.</returns>
    public bool IsValidCpf(string cpf)
    {
        if (string.IsNullOrWhiteSpace(cpf))
            return false;

        cpf = cpf.Trim().Replace(".", "").Replace("-", "");

        if (cpf.Length != 11)
            return false;

        if (new string(cpf[0], cpf.Length) == cpf)
            return false;

        var soma = 0;
        for (var i = 0; i < 9; i++)
            soma += (cpf[i] - '0') * (10 - i);

        var resto = soma % 11;
        var digitoVerificador1 = resto < 2 ? 0 : 11 - resto;

        soma = 0;
        for (var i = 0; i < 10; i++)
            soma += (cpf[i] - '0') * (11 - i);

        resto = soma % 11;
        var digitoVerificador2 = (resto < 2) ? 0 : 11 - resto;

        return cpf[9] == digitoVerificador1 + '0' && cpf[10] == digitoVerificador2 + '0';
    }

    /// <summary>
    /// Verifica se o CNPJ informado é válido.
    /// </summary>
    /// <param name="cnpj">O CNPJ a ser validado.</param>
    /// <returns>True se o CNPJ for válido; caso contrário, false.</returns>
    public bool IsValidCnpj(string cnpj)
    {
        if (string.IsNullOrWhiteSpace(cnpj))
            return false;

        cnpj = cnpj.Trim().Replace(".", "").Replace("-", "").Replace("/", "");

        if (cnpj.Length != 14)
            return false;

        if (new string(cnpj[0], cnpj.Length) == cnpj)
            return false;

        int[] peso1 = [5, 4, 3, 2, 9, 8, 7, 6, 5, 4, 3, 2];
        int[] peso2 = [6, 5, 4, 3, 2, 9, 8, 7, 6, 5, 4, 3, 2];

        var soma = 0;
        for (var i = 0; i < 12; i++)
        {
            soma += (cnpj[i] - '0') * peso1[i];
        }
        var resto = soma % 11;
        var digitoVerificador1 = resto < 2 ? 0 : 11 - resto;

        soma = 0;
        for (var i = 0; i < 13; i++)
            soma += (cnpj[i] - '0') * peso2[i];
```

```
        resto = soma % 11;
        var digitoVerificador2 = resto < 2 ? 0 : 11 - resto;

        return (cnpj[12] - '0') == digitoVerificador1 &&
               (cnpj[13] - '0') == digitoVerificador2;
    }
}
```

.\RealtyHub.Web\Services\EmailService.cs

```
using RealtyHub.Core.Requests.Emails;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;
using System.Net.Http.Json;

namespace RealtyHub.Web.Services;

/// <summary>
/// Serviço responsável por enviar e-mails relacionados à confirmação de conta,
/// recuperação de senha e envio de contratos.
/// </summary>
public class EmailService : IEmailService
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="EmailService"/> utilizando a fábrica
de <see cref="HttpClient"/>.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica para criar instâncias do HttpClient
configuradas para a comunicação com o backend.</param>
    public EmailService(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Envia um link de confirmação de e-mail para o usuário.
    /// </summary>
    /// <param name="message">Objeto contendo as informações necessárias para o envio do
e-mail de confirmação.</param>
    /// <returns>Task contendo um <see cref="Response{T}"/> indicando se o envio foi
bem-sucedido.</returns>
    public Task<Response<bool>> SendConfirmationLinkAsync(ConfirmEmailMessage message)
    {
        throw new NotImplementedException();
    }

    /// <summary>
    /// Envia um link para resetar a senha do usuário.
    /// </summary>
    /// <param name="message">Objeto contendo as informações necessárias para o envio do
e-mail de recuperação de senha.</param>
    /// <returns>Task contendo um <see cref="Response{T}"/> indicando se o envio foi
bem-sucedido.</returns>
    public Task<Response<bool>> SendResetPasswordLinkAsync(ResetPasswordMessage message)
    {
        throw new NotImplementedException();
    }

    /// <summary>
    /// Envia um contrato por e-mail como anexo.
    /// </summary>
    /// <param name="message">Objeto contendo as informações do contrato e do
destinatário.</param>
    /// <returns>
    /// Task contendo um <see cref="Response{T}"/> que indica se o envio do contrato foi
realizado com sucesso.
    /// </returns>
    public async Task<Response<bool>> SendContractAsync(AttachmentMessage message)
    {
        var result = await _httpClient.PostAsJsonAsync("v1/emails/contract", message);

        return result.IsSuccessStatusCode
            ? new Response<bool>(true)
            : new Response<bool>(false, (int)result.StatusCode, "Não foi possível enviar
o contrato");
    }
}
```

.\RealtyHub.Web\Services\OfferReport.cs

```
using System.Net.Http.Json;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.Web.Services;

/// <summary>
/// Serviço responsável por obter o relatório de ofertas do backend.
/// </summary>
public class OfferReport
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="OfferReport"/> utilizando a fábrica
de <see cref="HttpClient"/>.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica para criar instâncias do HttpClient
configuradas para a comunicação com o backend.</param>
    public OfferReport(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Obtém o relatório de ofertas do backend.
    /// </summary>
    /// <returns>
    /// Task contendo um <see cref="Response{Report}"/> que indica se o relatório foi
obtido com sucesso.
    /// Em caso de falha, retorna uma resposta com código 400 e mensagem de erro.
    /// </returns>
    public async Task<Response<Report>> GetOfferAsync()
    {
        var result = await _httpClient.GetAsync("v1/reports/offer");

        return await result.Content.ReadFromJsonAsync<Response<Report>>()
               ?? new Response<Report>(null, 400, "Não foi possível obter o relatório de
ofertas");
    }
}
```

.\RealtyHub.Web\Services\PropertyReport.cs

```
using System.Net.Http.Json;
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;

namespace RealtyHub.Web.Services;

/// <summary>
/// Serviço responsável por obter o relatório de imóveis do backend.
/// </summary>
public class PropertyReport
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="PropertyReport"/> utilizando a
fábrica de <see cref="HttpClient"/>.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica para criar instâncias do HttpClient
configuradas para a comunicação com o backend.</param>
    public PropertyReport(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory
            .CreateClient(Configuration.HttpClientName);
    }

    /// <summary>
    /// Obtém o relatório de imóveis do backend.
    /// </summary>
    /// <returns>
    /// Task contendo um <see cref="Response{Report}"/> representando o relatório de
imóveis.
    /// Em caso de falha, retorna uma resposta com código 400 e uma mensagem de erro.
    /// </returns>
    public async Task<Response<Report>> GetPropertyAsync()
    {
        var result = await _httpClient.GetAsync("v1/reports/property");

        return await result.Content.ReadFromJsonAsync<Response<Report>>()
               ?? new Response<Report>(null, 400, "Não foi possível obter o relatório de
imóveis");
    }
}
```

.\RealtyHub.Web\Services\ShowDialogConfirm.cs

```
using MudBlazor;
using RealtyHub.Web.Components.Common;

namespace RealtyHub.Web.Services;

/// <summary>
/// Serviço responsável por exibir um diálogo de confirmação utilizando os componentes do
MudBlazor.
/// </summary>
public class ShowDialogConfirm
{
    private readonly IDialogService _dialogService;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ShowDialogConfirm"/>.
    /// </summary>
    /// <param name="dialogService">Serviço de diálogo utilizado para exibir o componente
de confirmação.</param>
    public ShowDialogConfirm(IDialogService dialogService)
    {
        _dialogService = dialogService;
    }

    /// <summary>
    /// Exibe um diálogo de confirmação com as configurações especificadas.
    /// </summary>
    /// <param name="parameters">Parâmetros do diálogo que determinam o conteúdo e as
ações disponíveis.</param>
    /// <returns>
    /// Task contendo o resultado do diálogo (<see cref="DialogResult"/>) ou null caso a
operação seja cancelada.
    /// </returns>
    public async Task<DialogResult?> ShowDialogAsync(DialogParameters parameters)
    {
        var options = new DialogOptions
        {
            CloseButton = true,
            MaxWidth = MaxWidth.Small
        };

        var dialog = await _dialogService.ShowAsync<DialogConfirm>("Confirmação",
parameters, options);
        return await dialog.Result;
    }
}
```

.\RealtyHub.Web\Services\ViaCepService.cs

```
using RealtyHub.Core.Models;
using RealtyHub.Core.Responses;
using RealtyHub.Core.Services;
using System.Text.Json;

namespace RealtyHub.Web.Services;

/// <summary>
/// Serviço responsável por buscar informações de endereço a partir do serviço ViaCep.
/// </summary>
public class ViaCepService : IViaCepService
{
    private readonly HttpClient _httpClient;

    /// <summary>
    /// Inicializa uma nova instância de <see cref="ViaCepService"/> utilizando a fábrica
de <see cref="HttpClient"/>.
    /// </summary>
    /// <param name="httpClientFactory">Fábrica para criar instâncias do
HttpClient.</param>
    public ViaCepService(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory.CreateClient();
    }

    /// <summary>
    /// Busca as informações de endereço utilizando o CEP informado através da API
ViaCep.
    /// </summary>
    /// <param name="cep">CEP a ser pesquisado.</param>
    /// <returns>
    /// Task contendo um <see cref="Response{ViaCepResponse?}"/> com as informações
retornadas pela API ou mensagem de erro.
    /// </returns>
    public async Task<Response<ViaCepResponse?>> SearchAddressAsync(string cep)
    {
        try
        {
            var url = $"https://viacep.com.br/ws/{cep}/json/";

            var response = await _httpClient.GetAsync(url);

            if (!response.IsSuccessStatusCode)
            {
                var statusMessage = await response.Content.ReadAsStringAsync();
                return new Response<ViaCepResponse?>(null, (int)response.StatusCode,
statusMessage);
            }

            var content = await response.Content.ReadAsStringAsync();
            var viaCepResponse = JsonSerializer.Deserialize<ViaCepResponse>(content);

            return new Response<ViaCepResponse?>(viaCepResponse);
        }
        catch (Exception ex)
        {
            return new Response<ViaCepResponse?>(null, 500, ex.Message);
        }
    }

    /// <summary>
    /// Retorna um objeto <see cref="Address"/> com base no CEP informado, realizando a
busca através do serviço ViaCep.
    /// </summary>
    /// <param name="cep">CEP a ser pesquisado.</param>
    /// <returns>
    /// Task contendo um <see cref="Response{Address?}"/> com as informações de endereço
ou mensagem de erro.
    /// </returns>
    public async Task<Response<Address?>> GetAddressAsync(string cep)
    {
        try
```

```
        {
            var searchAddressAsync = await SearchAddressAsync(cep);
            if (searchAddressAsync is { IsSuccess: false, Data: null })
                return new Response<Address?>(null, 500, searchAddressAsync.Message);

            var address = new Address
            {
                ZipCode = cep,
                Street = searchAddressAsync.Data!.Street,
                Neighborhood = searchAddressAsync.Data.Neighborhood,
                City = searchAddressAsync.Data.City,
                State = searchAddressAsync.Data.State,
                Complement = searchAddressAsync.Data.Complement
            };

            return new Response<Address?>(address);
        }
        catch (Exception e)
        {
            return new Response<Address?>(null, 500, e.Message);
        }
    }
}
```

.\RealtyHub.Web\Utility.cs

```
using MudBlazor;
using System.Text.RegularExpressions;

namespace RealtyHub.Web;

/// <summary>
/// Classe utilitária que contém máscaras de entrada e validações comuns utilizadas na
aplicação.
/// </summary>
public static class Utility
{
    /// <summary>
    /// Propriedade que fornece as máscaras de entrada configuradas.
    /// </summary>
    public static Mask Masks { get; } = new();

    /// <summary>
    /// Classe que contém definições de máscaras de entrada para diversos formatos.
    /// </summary>
    public class Mask
    {
        /// <summary>
        /// Máscara para números de telefone no formato (##) #########.
        /// </summary>
        public readonly PatternMask Phone = new("(##) #########")
        {
            MaskChars = [new MaskChar('#', @"[0-9]")]
        };

        /// <summary>
        /// Máscara para CEP no formato #####-###.
        /// </summary>
        public readonly PatternMask ZipCode = new("#####-###")
        {
            MaskChars = [new MaskChar('#', @"[0-9]")]
        };

        /// <summary>
        /// Máscara para CPF no formato ###.###.###-##.
        /// </summary>
        public readonly PatternMask Cpf = new("###.###.###-##")
        {
            MaskChars = [new MaskChar('#', @"[0-9]")]
        };

        /// <summary>
        /// Máscara para CNPJ no formato ##.###.###/####-##.
        /// </summary>
        public readonly PatternMask Cnpj = new("##.###.###/####-##")
        {
            MaskChars = [new MaskChar('#', @"[0-9]")]
        };

        /// <summary>
        /// Máscara para números simples no formato ###.
        /// </summary>
        public readonly PatternMask Number = new("###")
        {
            MaskChars = [new MaskChar('#', @"[0-9]")]
        };
    }

    /// <summary>
    /// Classe que contém expressões regulares para validações de entrada.
    /// </summary>
    public static class Validations
    {
        /// <summary>
        /// Expressão regular para validar o formato de horas (HH:mm).
        /// Aceita valores de 00:00 até 23:59.
        /// </summary>
        public static readonly Regex TimeRegex = new(@"^(?:[01]\d|2[0-3]):[0-5]\d$",
```

```
RegexOptions.Compiled);
    }

}
```

.\RealtyHub.Web_Imports.razor

```
@using System.Net.Http
@using System.Net.Http.Json
@using System.ComponentModel.DataAnnotations

@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop

@using MudBlazor

@using RealtyHub.Web
@using RealtyHub.Web.Components
@using RealtyHub.Web.Layout
@using RealtyHub.Web.Pages
@using RealtyHub.Web.Services
@using RealtyHub.Web.Security
@using RealtyHub.Web.Handlers
@using RealtyHub.Web.Components.Identity
@using RealtyHub.Web.Components.Customers
@using RealtyHub.Web.Components.Properties
@using RealtyHub.Web.Components.Viewings
@using RealtyHub.Web.Components.Offers
@using RealtyHub.Web.Components.Home
@using RealtyHub.Web.Components.Common
@using RealtyHub.Web.Components.Contracts
@using RealtyHub.Web.Components.Email
@using RealtyHub.Web.Components.Condominiums
@using RealtyHub.Core.Enums
```

.\RealtyHub.Web\appsettings.Development.json

```
{
  "DetailedErrors": true,
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

.\RealtyHub.Web\appsettings.json

```
{
  "BackendUrl": "http://localhost:5538",
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

.\RealtyHub.Web\wwwroot\css\app.css

```
.valid.modified:not([type=checkbox]) {
    outline: 1px solid #26b050;
}

.invalid {
    outline: 1px solid red;
}

.validation-message {
    color: red;
}

#blazor-error-ui {
    background: lightyellow;
    bottom: 0;
    box-shadow: 0 -1px 2px rgba(0, 0, 0, 0.2);
    display: none;
    left: 0;
    padding: 0.6rem 1.25rem 0.7rem 1.25rem;
    position: fixed;
    width: 100%;
    z-index: 1000;
}

    #blazor-error-ui .dismiss {
        cursor: pointer;
        position: absolute;
        right: 0.75rem;
        top: 0.5rem;
    }

.blazor-error-boundary {
    background:
url(
3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbms9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbmsiIG92ZXJm
bG93PSJoaWRkZW4iPjxkZWZzPjxjbGlwUGF0aCBpZD0iY2xpcDAiPjxyZWN0IHg9IjIzNSIgeT0iNTEiIHdpZHRoP
SI1NiIgaGVpZ2h0PSI0OSIvPjwvY2xpcFBhdGg+PC9kZWZzPjxnIGNsaXAtcGF0aD0idXJsKCNjbGlwMCkiIHRyYW
5zZm9ybT0idHJhbnNsYXRlKC0yMzUgLTUxKSI+PHBhdGggZD0iTTI2My41MDYgNTFDMjY0LjcxNyA1MSAyNjUuODE
zIDUxLjQ4MzcgMjY2LjYwNiA1Mi4yNjU4TDI2Ny4wNTIgNTIuNzk4NyAyNjcuNTM5IDUzLjYyODMgMjkwLjE4NSA5
Mi4xODMxIDI5MC41NDUgOTIuNzk1IDI5MC42NTYgOTIuOTk2QzI5MC44NzcgOTMuNTEzIDI5MSA5NC4wODE1IDI5M
SA5NC42NzgyIDI5MSA5Ny4wNjUxIDI4OS4wMzggOTkgMjg2LjYxNyA5OUwyNDAuMzgzIDk5QzIzNy45NjMgOTkgMj
M2IDk3LjA2NTEgMjM2IDk0LjY3ODIgMjM2IDk0LjM3OTkgMjM2LjAzMSA5NC4wODg2IDIzNi4wODkgOTMuODA3Mkw
yMzYuMzM4IDkzLjAxNjIgMjM2Ljg1OCA5Mi4xMzE0IDI1OS40NzMgNTMuNjI5NCAyNTkuOTYxIDUyLjc5ODUgMjYw
LjQwNyA1Mi4yNjU4QzI2MS4yIDUxLjQ4MzcgMjYyLjI5NiA1MSAyNjMuNTA2IDUxWk0yNjMuNTg2IDY2LjAxODNDM
jYwLjczNyA2Ni4wMTgzIDI1OS4zMTMgNjcuNTkyIDI1OS4zMTMgNzAuMDAwMSAyNjMuMzM5IDY5LjMzNyAyNTkuMzEzIDY5LjYxMDIgMjU5Lj
MzMiA2OS44NjA4IDI1OS4zNzEgNzAuMDg4N0wyNjEuNzk1IDg0LjAxNjEgMjY1LjM4IDg0LjAxNjEgMjY3LjgyMSA
2OS43NDc1QzI2Ny44NiA2OS43MzA5IDI2Ny44NzkgNjkuNTg3NyAyNjcuODc5IDY5LjMxNzkgMjY3Ljg3OSA2Ny4x
MTgyIDI2Ni40NDggNjYuMDE4MyAyNjMuNTg2IDY2LjAxODNaTTI2My41NzYgODYuMDU0N0MyNjEuMDQ5IDg2LjA1N
DcgMjU5Ljc4NiA4Ny4zMDA1IDI1OS43ODYgODkuNzkyMSAyNTkuNzg2IDkyLjI4MzcgMjYxLjA0OSA5My41Mjk1ID
I2My41NzYgOTMuNTI5NSAyNjYuMTE2IDkzLjUyOTUgMjY3LjM4NyA5Mi4yODM3IDI2Ny4zODcgODkuNzkyMSAyNjc
uMzg3IDg3LjMwMDUgMjY2LjExNiA4Ni4wNTQ3IDI2My41NzYgODYuMDU0N1oiIGZpbGw9IiNGRkU1MDAiIGZpbGwt
cnVsZT0iZXZlbm9kZCIvPjwvZz48L3N2Zz4=) no-repeat 1rem/1.8rem, #b32121;
    padding: 1rem 1rem 1rem 3.7rem;
    color: white;
}

    .blazor-error-boundary::after {
        content: "An error has occurred."
    }

.loading-progress {
    position: relative;
    display: block;
    width: 8rem;
    height: 8rem;
    margin: 20vh auto 1rem auto;
}

    .loading-progress circle {
        fill: none;
        stroke: #e0e0e0;
        stroke-width: 0.6rem;
```

```
        transform-origin: 50% 50%;
        transform: rotate(-90deg);
    }

        .loading-progress circle:last-child {
            stroke: #1b6ec2;
            stroke-dasharray: calc(3.141 * var(--blazor-load-percentage, 0%) * 0.8),
500%;
            transition: stroke-dasharray 0.05s ease-in-out;
        }

.loading-progress-text {
    position: absolute;
    text-align: center;
    font-weight: bold;
    inset: calc(20vh + 3.25rem) 0 auto 0.2rem;
}

    .loading-progress-text:after {
        content: var(--blazor-load-percentage-text, "Loading");
    }

code {
    color: #c02d76;
}

.flex-center {
    display: flex;
    align-items: center;
}
```

.\RealtyHub.Web\wwwroot\index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>RealtyHub</title>
    <base href="/" />
    <link rel="stylesheet" href="css/app.css" />
    <!-- If you add any scoped CSS files, uncomment the following to load them
    <link href="RealtyHub.Frontend.styles.css" rel="stylesheet" /> -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Raleway:ital,wght@0,100..900;1,100..900&di
splay=swap" rel="stylesheet">
    <link href="https://use.fontawesome.com/releases/v5.14.0/css/all.css"
rel="stylesheet">
    <link href="_content/MudBlazor/MudBlazor.min.css" rel="stylesheet" />

    <link href="manifest.webmanifest" rel="manifest" />
    <link rel="apple-touch-icon" sizes="512x512" href="icon-512.png" />
    <link rel="apple-touch-icon" sizes="192x192" href="icon-192.png" />
</head>

<body>
    <div id="app">
        <svg class="loading-progress">
            <circle r="40%" cx="50%" cy="50%" />
            <circle r="40%" cx="50%" cy="50%" />
        </svg>
        <div class="loading-progress-text"></div>
    </div>

    <div id="blazor-error-ui">
        An unhandled error has occurred.
        <a href="" class="reload">Reload</a>
        <a class="dismiss">?</a>
    </div>
    <script src="https://kit.fontawesome.com/6a5e7192fc.js"
crossorigin="anonymous"></script>
    <script src="_framework/blazor.webassembly.js"></script>
    <script src="_content/MudBlazor/MudBlazor.min.js"></script>
    <script>
        function openContractPdfInNewTab(path) {
            window.open(path);
        }
    </script>
    <script>navigator.serviceWorker.register('service-worker.js');</script>
</body>

</html>
```

.\RealtyHub.Web\wwwroot\service-worker.js

```
// In development, always fetch from the network and do not enable offline support.
// This is because caching would make development more difficult (changes would not
// be reflected on the first load after each change).
self.addEventListener('fetch', () => { });
```

.\RealtyHub.Web\wwwroot\service-worker.published.js

```
// Caution! Be sure you understand the caveats before publishing an application with
// offline support. See https://aka.ms/blazor-offline-considerations

self.importScripts('./service-worker-assets.js');
self.addEventListener('install', event => event.waitUntil(onInstall(event)));
self.addEventListener('activate', event => event.waitUntil(onActivate(event)));
self.addEventListener('fetch', event => event.respondWith(onFetch(event)));

const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const offlineAssetsInclude = [ /\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/,
/\.json$/, /\.css$/, /\.woff$/, /\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/, /\.blat$/,
/\.dat$/ ];
const offlineAssetsExclude = [ /^service-worker\.js$/ ];

// Replace with your base path if you are hosting on a subfolder. Ensure there is a
trailing '/'.
const base = "/";
const baseUrl = new URL(base, self.origin);
const manifestUrlList = self.assetsManifest.assets.map(asset => new URL(asset.url,
baseUrl).href);

async function onInstall(event) {
    console.info('Service worker: Install');

    // Fetch and cache all matching items from the assets manifest
    const assetsRequests = self.assetsManifest.assets
        .filter(asset => offlineAssetsInclude.some(pattern => pattern.test(asset.url)))
        .filter(asset => !offlineAssetsExclude.some(pattern => pattern.test(asset.url)))
        .map(asset => new Request(asset.url, { integrity: asset.hash, cache: 'no-cache'
}));
    await caches.open(cacheName).then(cache => cache.addAll(assetsRequests));
}

async function onActivate(event) {
    console.info('Service worker: Activate');

    // Delete unused caches
    const cacheKeys = await caches.keys();
    await Promise.all(cacheKeys
        .filter(key => key.startsWith(cacheNamePrefix) && key !== cacheName)
        .map(key => caches.delete(key)));
}

async function onFetch(event) {
    let cachedResponse = null;
    if (event.request.method === 'GET') {
        // For all navigation requests, try to serve index.html from cache,
        // unless that request is for an offline resource.
        // If you need some URLs to be server-rendered, edit the following check to
exclude those URLs
        const shouldServeIndexHtml = event.request.mode === 'navigate'
            && !manifestUrlList.some(url => url === event.request.url);

        const request = shouldServeIndexHtml ? 'index.html' : event.request;
        const cache = await caches.open(cacheName);
        cachedResponse = await cache.match(request);
    }

    return cachedResponse || fetch(event.request);
}
```