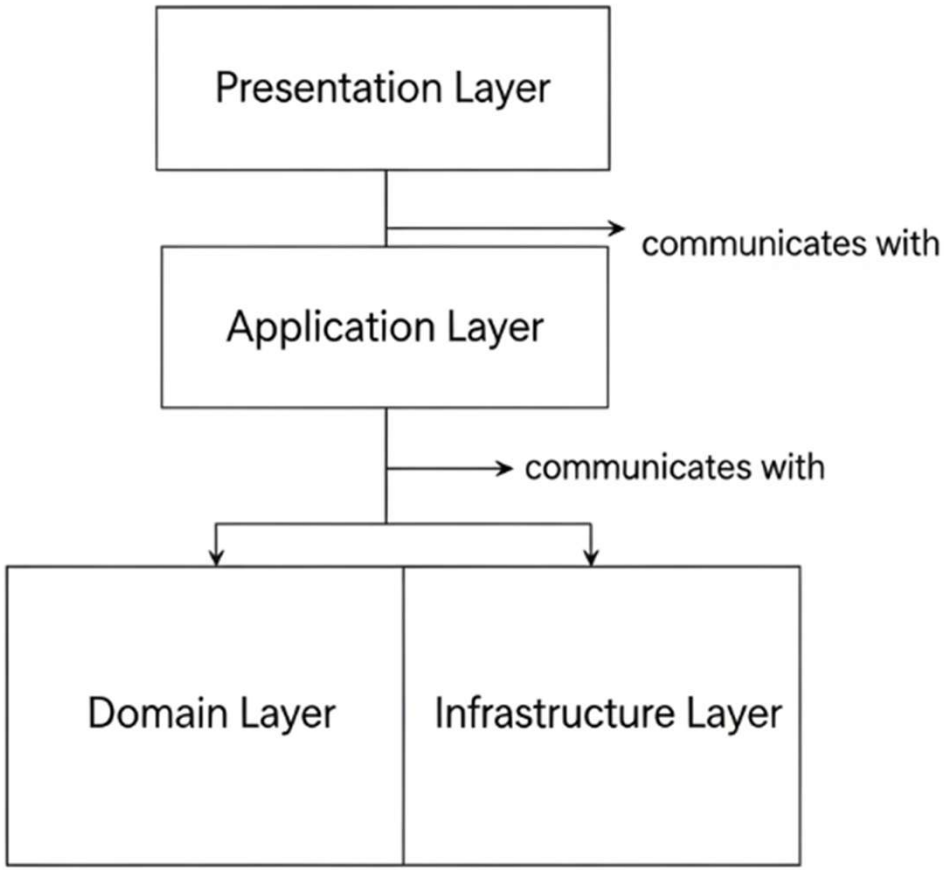
09 de 20


FRONTEND

- **Domain - OK**
  - É o coração do sistema, onde a lógica de negócios e o conhecimento específico do domínio são implementados.
  - Esta camada deve ser isolada e livre de dependências externas, focando na modelagem do problema de negócio.
  - Contém as **entidades**, **value objects**, **agregados** e as **interfaces dos repositórios**.
- **Application - OK**
  - Essa camada coordena as ações e fluxos de trabalho do sistema, atuando como mediadora entre a camada de interface do usuário e a camada de domínio.
  - Ela executa casos de uso e orquestra o fluxo de informações entre as camadas superiores e inferiores.
- **Infrastructure - OK**
  - Essa camada lida com detalhes técnicos e de suporte, como acesso a banco de dados, comunicação com serviços externos, logging, etc.
  - É a camada de menor nível de abstração e onde as tecnologias são implementadas.
- **Presentation**
  - É a camada responsável por interagir com o usuário, recebendo comandos e exibindo resultados.
  - Ela não deve conter lógica de negócios, apenas a apresentação das informações e o tratamento de interações do usuário.
  - Por exemplo, Console App, ASP.NET Core, Windows Forms, WPF, API REST etc.


Presentation Layer
communicates with
Application Layer
communicates with
Domain Layer
Infrastructure Layer


- Me chamo **Zé** e sou sócio proprietário da **Academia do Zé**.
- Somos uma academia de musculação e precisamos, de um programa que nos permita manter um **controle das nossas matrículas**, algo que seja simples e que facilite nosso processo, tornando-o ágil e confiável.

- Na sequência estou detalhando os requisitos funcionais para o sistema de gerenciamento da Academia do Zé, visando otimizar o controle de alunos, planos, atividades e relatórios.

- **Cadastro de alunos:**
    - O sistema deve permitir o registro dos alunos, incluindo:
        - *nome completo, *cpf, *data de nascimento, *telefone, e-mail, *senha, foto, *logradouro {cep, pais, estado, cidade, bairro, nome logradouro}, *número e complemento.
    - Regras:
        - Cpf: único.

- **Cadastro de colaboradores:**
    - Deve ser possível registrar os colaboradores com:
        - *nome completo, *cpf, *data de nascimento, *telefone, *e-mail, *senha, foto, *logradouro {cep, pais, estado, cidade, bairro, nome logradouro}, *número e complemento, *data admissão, *tipo {administrador, atendente, instrutor}, *vinculo {clt, estágio}.
    - Regras
        - Cpf: único.

- **Cadastro da matrícula**
    - O sistema deve permitir o registro da matrícula:
        - *aluno, *plano {mensal, trimestral, semestral ou anual}, *data de início, *data final, *objetivo, restrições {ex: diabetes, pressão alta, labirintite, alergias, problemas respiratórios, uso de remédios contínuos, etc.}, observações sobre as restrições, laudo médico.
    - **Regras:**
        - Não será realizado controle financeiro.
        - Não permitir nova matrícula se ainda tiver matrícula ativa.
        - **Requisitos por idade:**
            - Alunos de 12 a 16 anos, devem obrigatoriamente apresentar um laudo médico que os autorize a praticar atividades físicas.
            - O laudo deve ser salvo na matrícula.
        - **Observações de saúde:**
            - Deve ter uma seção para registrar observações importantes sobre a saúde do aluno (ex: diabetes, pressão alta, labirintite, alergias, problemas respiratórios, uso de remédios contínuos, etc.).
            - Alunos com restrições de saúde registradas devem, obrigatoriamente, apresentar um parecer médico autorizando a realização de atividades físicas, e o mesmo deve ficar salvo na matrícula.

- ~~**Entrada/saída:**~~
  - ~~Para cada ida à academia, deve ser registrada:~~
    - ~~aluno, data e hora da chegada e saída.~~
    - ~~colaborador, data e hora da chegada e saída.~~
  - ~~Regras:~~
    - ~~Aluno:~~
      - ~~Validar se possui matrícula ativa.~~
      - ~~Na entrada, mostrar quanto tempo ainda tem de plano.~~
      - ~~Na saída, mostrar o tempo que permaneceu na academia.~~
    - ~~Colaborador~~
      - ~~Validar se já não ultrapassa o limite de: 8 horas se for ctl, 6 horas se for estagio.~~
      - ~~Na saída, mostrar o tempo que permaneceu na academia, devendo ser somado todos os registros do dia.~~

- **Permissões:**
  - Administrador
    - Acesso total.
  - Atendente
    - Cadastro de alunos, matrícula~~, registro de entrada/saída de aluno.~~
    - ~~Realizar seu registro de entrada e saída.~~
  - Instrutor
    - ~~Registro de entrada/saída de aluno.~~
    - ~~Realizar seu registro de entrada e saída.~~
  - Aluno
    - ~~Consultar/visualizar seus dados estatísticos, por exemplo: tempo de permanência, tempo de contrato.~~
    - ~~Realizar seu registro de entrada e saída.~~
    - Troca da sua senha.

- **Relatórios e estatísticas**
    - O sistema deve ser capaz de gerar relatórios estatísticos para análise de gestão, incluindo, por exemplo:
        - ~~Horários de maior procura.~~
        - ~~Permanência média dos alunos na academia.~~
        - ~~Tendências de evasão e retenção.~~
        - ~~No caso de colaboradores, horas trabalhadas por dia.~~

- **Suporte a idiomas:**
    - Considerando a expansão para franquias, o sistema deve ter suporte para os idiomas português, inglês e espanhol.

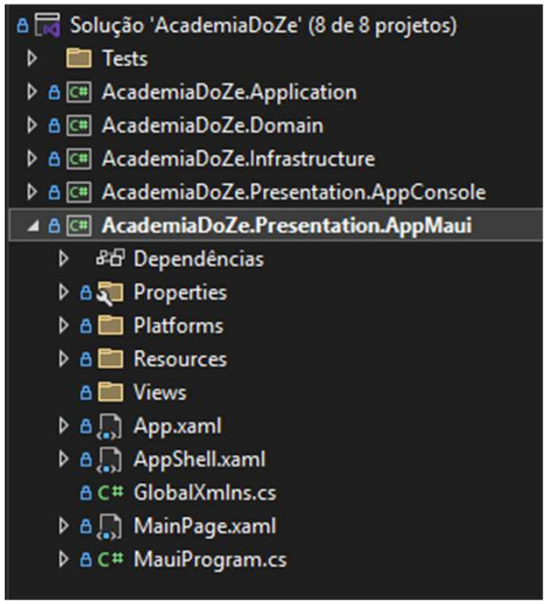

ACADEMIA DO ZÉ
De teus pulos

# Apresentação – Aplicação .NET MAUI

- Na sua solução **AcademiaDoZe** no Visual Studio, clique com o botão direito na Solução, e selecione Adicionar -> Novo Projeto....

- Na barra de busca de templates, digite **MAUI**, selecionando o template **.NET MAUI App**.

- Avance, e nomeie o projeto seguindo uma convenção padrão:
  - **AcademiaDoZe.Presentation.AppMaui**



# Configurando as referências

- No projeto **AcademiaDoZe.Presentation.AppMaui**, clique com o botão direito em **Adicionar -> Referência de Projeto**, e marque a caixa de seleção para:

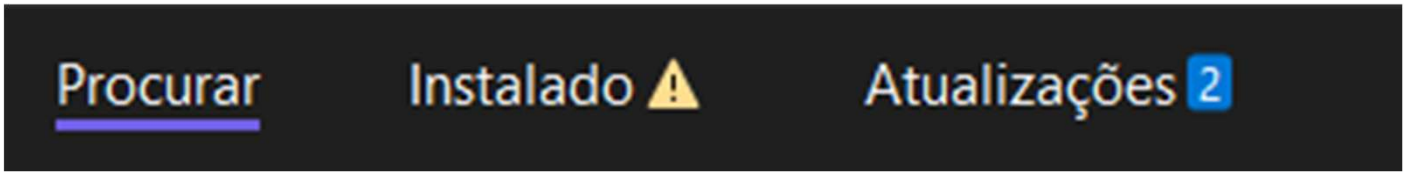
  - **AcademiaDoZe. Application**

# Instalar pacotes via gerenciador NuGet

- No projeto **AcademiaDoZe.Presentation.AppMaui**, acesse o gerenciador de pacotes **NuGet** e instale os pacotes abaixo:

  - **Microsoft.Extensions.DependencyInjection**
  - **CommunityToolkit.Mvvm**

- Se o **NuGet** apresentar **avisos** ou **atualizações**, já aproveite e resolva antes de continuar.



- A biblioteca **CommunityToolkit.Mvvm** é amplamente utilizada em projetos .NET MAUI, e WPF/UWP, para facilitar a implementação do padrão MVVM, fornecendo recursos como:
  - ObservableObject e INotifyPropertyChanged automáticos
  - [RelayCommand] para comandos sem boilerplate
  - [ObservableProperty] para propriedades automáticas
  - Mensageria (WeakReferenceMessenger)
  - Injeção de dependência simplificada

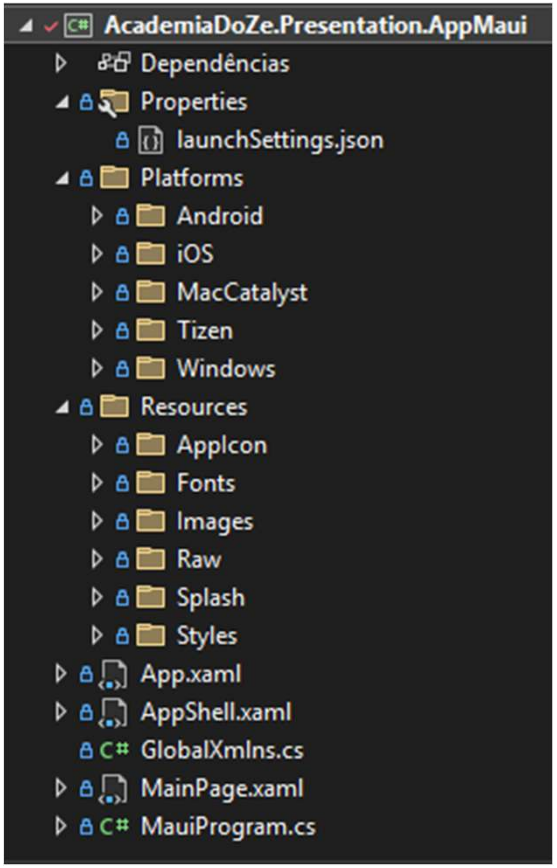
# Definir como projeto de inicialização

- Clique com o botão direito do mouse no projeto **AcademiaDoZe.Presentation.AppMaui**, e marque a opção:
  - **Definir como Projeto de Inicialização**

- Quando criamos o projeto **.NET MAUI App** no Visual Studio, o template gerou uma estrutura de arquivos e diretórios padrão para organizar seu código e recursos.

- Essa estrutura foi projetada para facilitar o desenvolvimento multiplataforma, mantendo os arquivos comuns e os específicos de cada plataforma bem separados.

- A seguir mostro um detalhamento da estrutura padrão.



- **App.xaml** e **App.xaml.cs**:
  - **App.xaml**: Contém o arquivo de marcação XAML para a sua aplicação. É aqui que você pode definir recursos em nível de aplicativo, como **cores**, **estilos** e **modelos de dados** que serão **compartilhados em todas as páginas**.
  - **App.xaml.cs**: É o arquivo **code-behind** do App.xaml. Ele contém a **lógica inicial da aplicação**, incluindo a criação da janela principal, no caso a chamada para **AppShell()**.

- **AppShell.xaml** e **AppShell.xaml.cs**:
  - **AppShell** é a estrutura que define a **navegação** e a **hierarquia** do seu aplicativo. Ele pode incluir a navegação de menu lateral (**flyout**) e a navegação de abas (**tabs**). É aqui que você define as **rotas para as suas páginas**, o que facilita o roteamento entre telas. Inicialmente, temos somente uma rota criada para a página de exemplo do template, MainPage.

- **MainPage.xaml** e **MainPage.xaml.cs**:
  - **MainPage.xaml**: É a primeira página da sua aplicação. É aqui que você começa a construir a interface de usuário principal, usando XAML.
  - **MainPage.xaml.cs**: Contém a lógica de programação para a MainPage. É onde você manipula eventos e interage com os dados.
  - MainPage não é um arquivo de sistema, ou seja, posteriormente se o fluxo da aplicação mudar, pode ocorrer dele não ser mais necessário, podendo ser removido, se necessário.

- **MauiProgram.cs**:
  - Este é o **ponto de entrada da sua aplicação**. Ele é responsável por configurar e inicializar o aplicativo. É aqui que você registra fontes, serviços, dependências e outras configurações que serão usadas em todo o projeto.

- O diretório **Resources** é onde todos os arquivos compartilhados, como imagens, fontes e estilos, são armazenados.

- O .NET MAUI se encarrega de otimizá-los para cada plataforma.
  - **Fonts**: Onde você coloca arquivos de fontes personalizadas para serem usadas no aplicativo.
  - **Images**: Onde você armazena arquivos de imagem, como ícones e logotipos. O .NET MAUI os redimensiona automaticamente para cada plataforma.
  - **Raw**: Para arquivos de áudio, vídeo ou outros tipos de dados brutos que precisam ser incluídos no projeto.
  - **Splash**: Contém o arquivo de imagem para a tela de abertura (splash screen) do aplicativo.
  - **Styles**: Onde você armazena arquivos XAML de estilo, como dicionários de recursos, que podem ser usados para manter a consistência visual.

- **Platforms** possui um diretório específico por plataforma, sendo um dos recursos mais poderosos do MAUI.

- Ele contém subdiretórios para cada plataforma, onde você pode colocar código e recursos específicos, caso precise.
  - **Android**: Para arquivos Android específicos, como o AndroidManifest.xml ou classes que usam APIs nativas.
  - **iOS**: Para arquivos iOS específicos, como o Info.plist ou classes que usam APIs da Apple.
  - **MacCatalyst**: Para configurações do macOS.
  - **Tizen**: .
  - **Windows**: Para arquivos específicos do Windows, como a imagem do aplicativo e a lógica para a interface do Windows.


**Temos um problema!**

- O **namespace** da nossa camada de aplicação, AcademiaDoZe.**Application**, conflita com o nome de uma **classe** utilizada pelo MAUI, especificamente no arquivo **App.xaml.cs**.

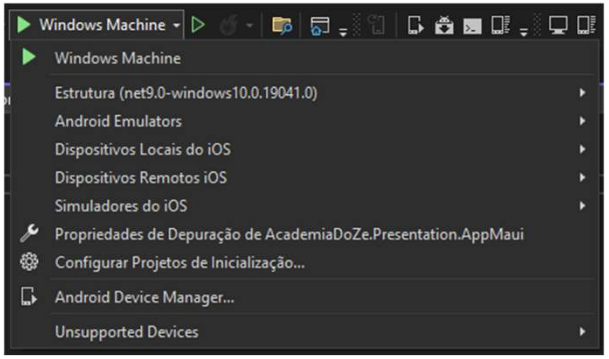
- Para resolver, precisamos editar o arquivo **App.xaml.cs**, incluindo o namespace correto da classe **Application** , herdada por App, bastando incluir **Microsoft.Maui.Controls** na chamada de **Application**, indicando desta o uso da classe correta.

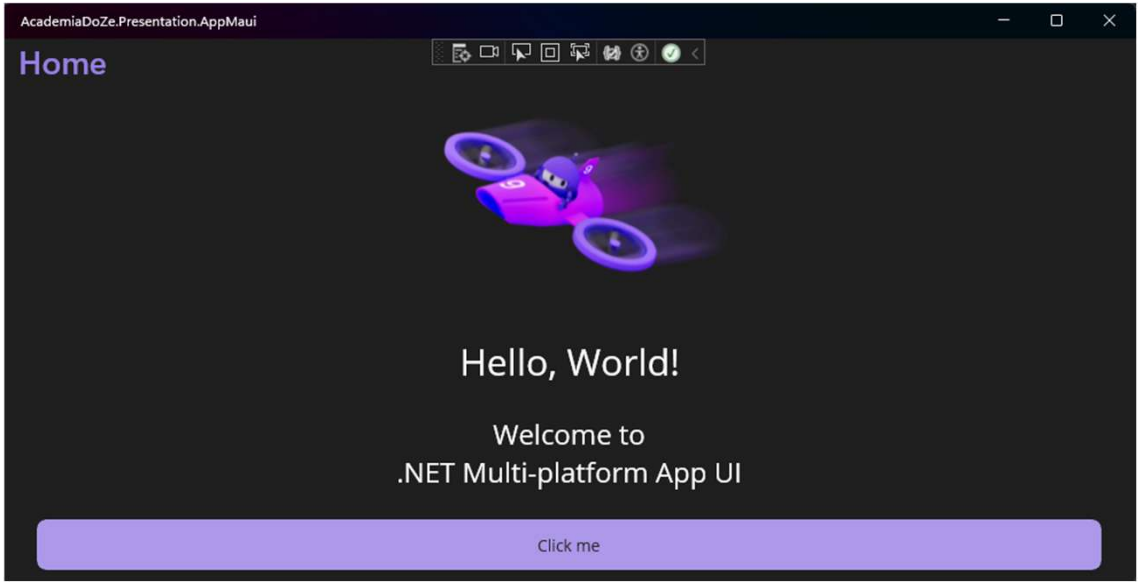
**Felizmente fácil de resolver!**

AcademiaDoZe.Presentation.AppMaui**App.xaml.cs**

```
namespace AcademiaDoZe.Presentation.AppMaui
{
    // Application conflita com o nome da nossa camada de aplicação
    // Incluir o namespace completo, Microsoft.Maui.Controls.Application, para evitar conflito
    // Direcionando para a classe Application do MAUI
    public partial class App : Microsoft.Maui.Controls.Application
    {
        public App()
        {
            InitializeComponent();
        }

        protected override Window CreateWindow(IActivationState? activationState)
        {
            return new Window(new AppShell());
        }
    }
}
```

- Após tudo entendido, e ajustado, só precisamos definir a **plataforma de destino** e executar nosso frontend.
- Como plataforma de destino, defina **Windows Machine**.
- A tela de exemplo, criada pelo template em **MainPage**, já deve abrir normalmente.



- Hora de trabalhar! Agora vamos efetivamente iniciar a implementação da nossa **camada de apresentação**, criando nossa aplicação **MAUI**.

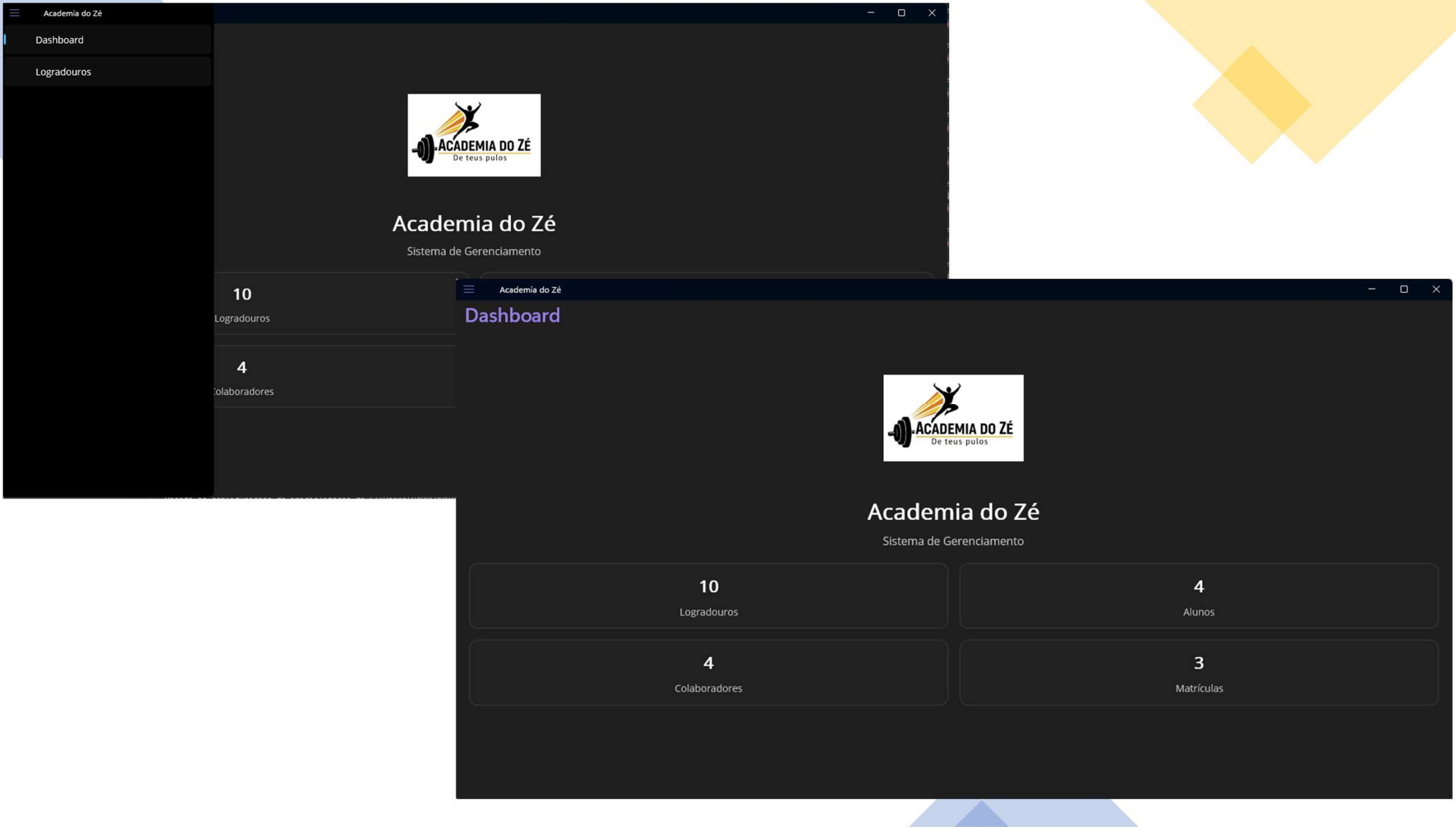
- Como base de exemplificação, vou organizar nossa aplicação seguindo o seguinte padrão:
  - **Página principal** contendo as opções de acesso para as funcionalidades: **Dashboard, Logradouros, Alunos, Colaboradores e Matrículas**.

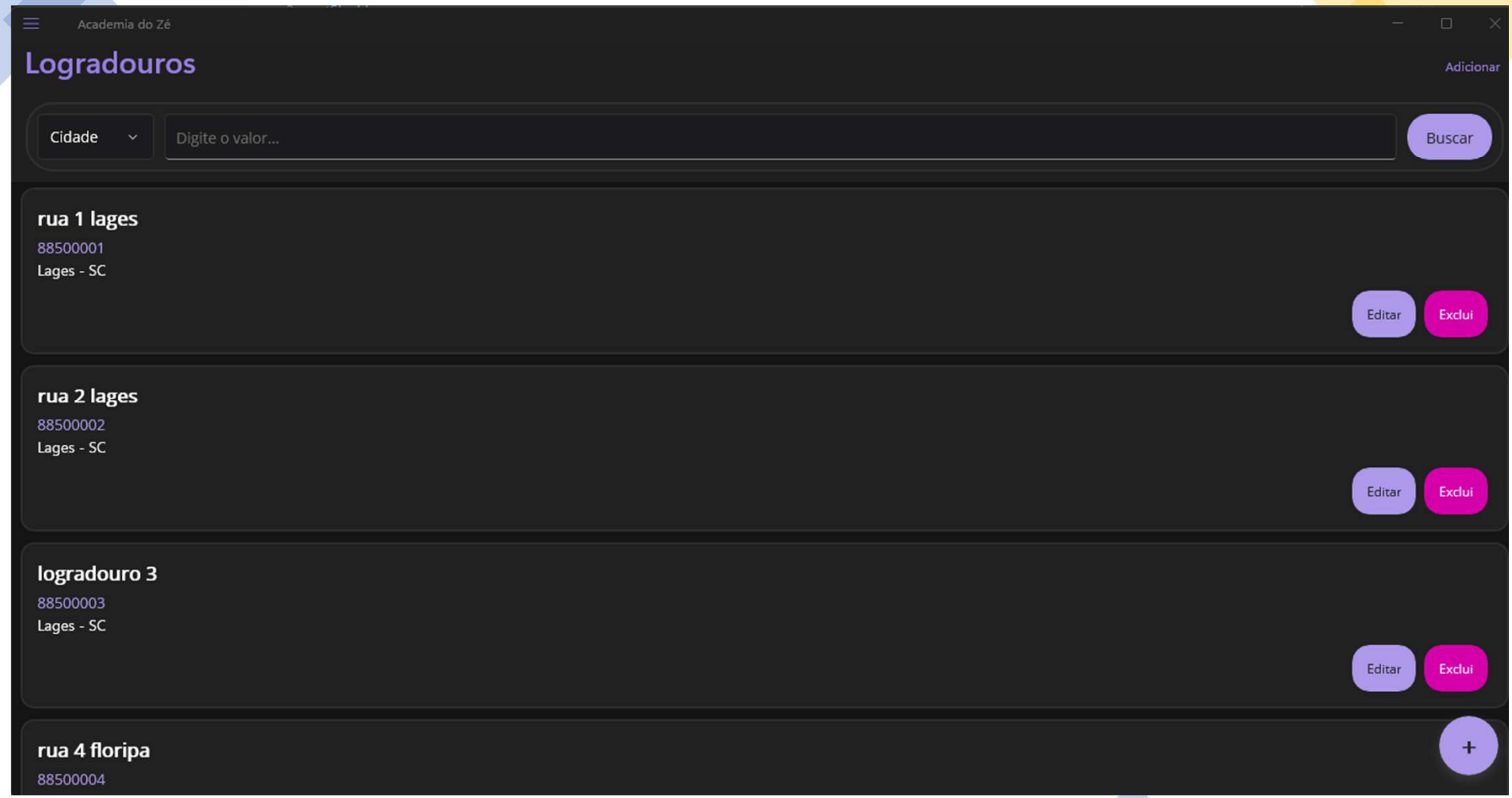
  - **Dashboard**, aberto através da página principal, contendo **cards interativos** por funcionalidade, mostrando o total de itens, e possibilitando a navegação para as respectivas seções.

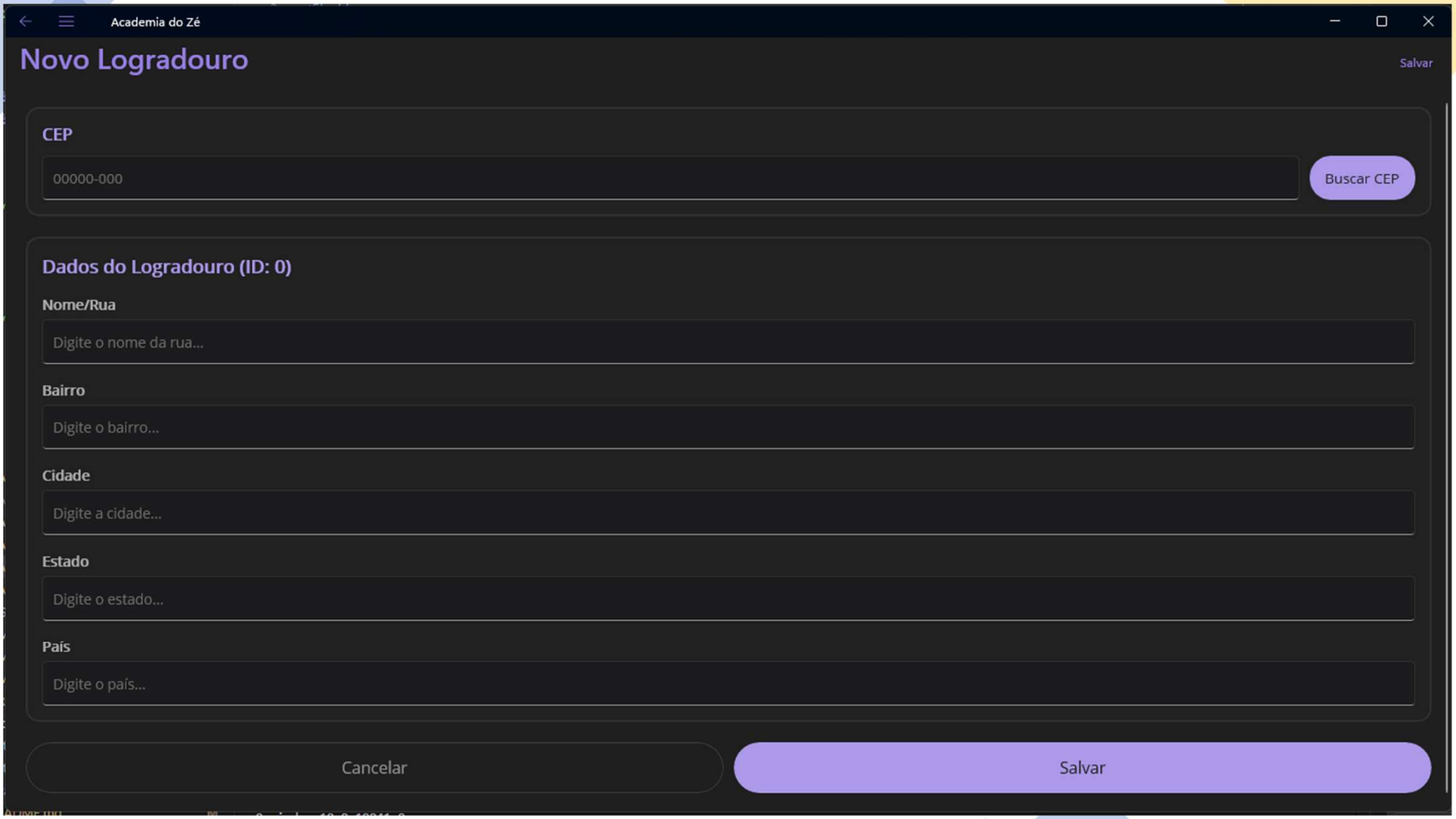
  - Para cada funcionalidade, uma página, aberta através da página principal, **listando todos os itens** cadastrados, com as opções: **Buscar**, **Editar**, **Excluir** e **Adicionar**.

  - Tela de **cadastro** e **edição** dos dados, aberta através da página de listar.

- Reforço que o código das minhas telas são somente um exemplo, o ideal é que cada um explore os recursos e realize a sua própria implementação!

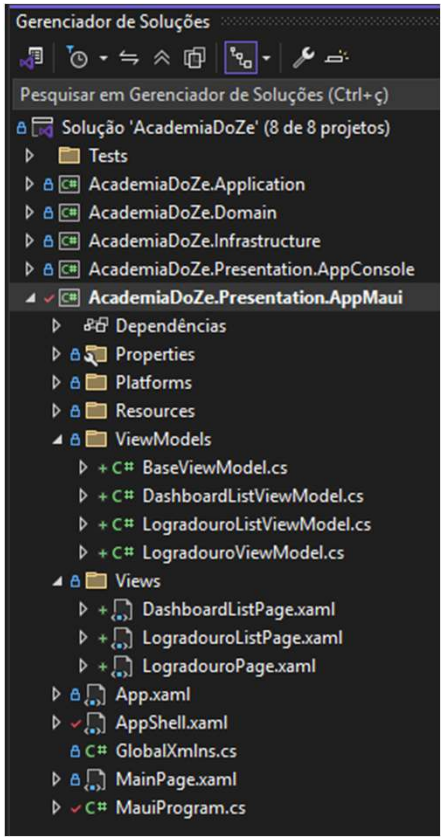

Academia do Zé
Dashboard
Logradouros
ACADEMIA DO ZÉ
De teus pulos
Academia do Zé
Sistema de Gerenciamento
10
Logradouros
4
Colaboradores
Academia do Zé
Dashboard
ACADEMIA DO ZÉ
De teus pulos
Academia do Zé
Sistema de Gerenciamento
10
Logradouros
4
Alunos
4
Colaboradores
3
Matrículas


Academia do Zé
Logradouros
Adicionar
Cidade
Digite o valor...
Buscar
rua 1 lages
88500001
Lages - SC
Editar
Exclui
rua 2 lages
88500002
Lages - SC
Editar
Exclui
logradouro 3
88500003
Lages - SC
Editar
Exclui
rua 4 floripa
88500004
+


Academia do Zé
Novo Logradouro
Salvar
CEP
00000-000
Buscar CEP
Dados do Logradouro (ID: 0)
Nome/Rua
Digite o nome da rua...
Bairro
Digite o bairro...
Cidade
Digite a cidade...
Estado
Digite o estado...
País
Digite o país...
Cancelar
Salvar

# Estrutura de camadas MVVM

- O padrão **MVVM**, trabalha com as seguintes pastas:
  - **Models**
    - Contém os dados e a lógica de negócios.
    - Classes de dados, serviços de acesso a banco de dados.
    - Em nosso projeto, **NÃO precisamos criar esta pasta**, pois tudo que iria nela já possuímos em nossa **camada de aplicação**.
  - **Views**
    - Contém todas as interfaces gráficas do usuário.
    - Aqui criaremos todos os arquivos XAML das páginas, e seus respectivos code-behind, código de apoio.
  - **ViewModels**
    - Contém a lógica de apresentação e a comunicação entre o Model e a View.
    - Propriedades, comandos, notificação de mudanças.



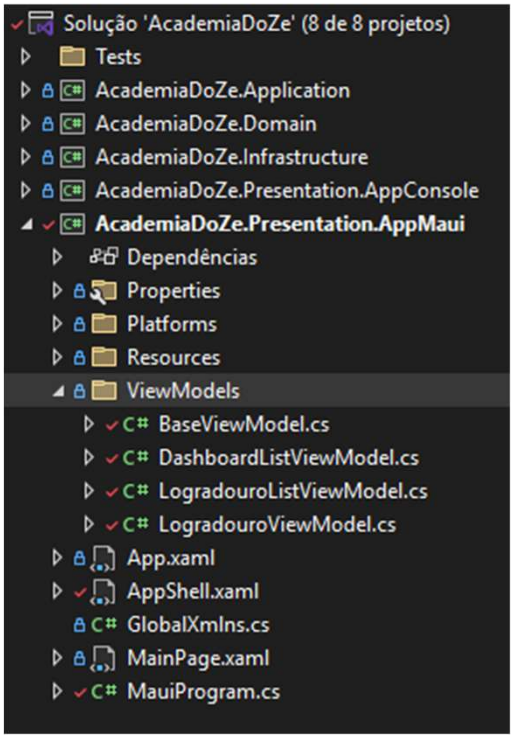
- Criar a pasta **ViewModels**, e nela criar as classes:

  - **BaseViewModel.cs**
    - Classe base para todas as ViewModels do projeto.
    - Centraliza propriedades e funcionalidades comuns, como notificação de mudança de propriedades, **INotifyPropertyChanged**, controle de estado IsBusy e IsRefreshing, e métodos utilitários para facilitar a implementação do padrão MVVM nas ViewModels derivadas.

  - **DashboardListViewModel.cs**
    - ViewModel para o dashboard do app.
    - Centraliza a lógica para buscar e exibir os totais de logradouros, alunos, colaboradores e matrículas, além de fornecer comandos para navegação rápida para cada uma dessas telas.
    - É responsável por expor dados agregados e comandos de navegação para a interface do dashboard.

  - **LogradouroListViewModel.cs**
    - ViewModel para a tela de listagem de logradouros.
    - Gerencia a busca, filtragem, atualização e exclusão de logradouros, além de expor comandos para navegação e manipulação desses dados.
    - Centraliza a lógica de apresentação e interação da lista de logradouros para a interface do usuário.

  - **LogradouroViewModel.cs**
    - ViewModel para a tela de cadastro/edição de logradouros.
    - Gerencia os dados de um logradouro individual, controla a lógica de salvar, atualizar ou excluir, e expõe comandos para essas ações, além de lidar com navegação e validação.
    - Centraliza toda a lógica de apresentação e manipulação de um logradouro na interface.




- Os códigos seguinte são cheios de detalhes, então é extremamente importe vocês analisarem com muita atenção, cada etapa que está sendo implementada, e que também realizem a leitura dos comentários e explicações.
- No cabeçalho de cada página que contém código fonte, está o caminho completo do arquivo e seu nome, tenha cuidado para salvar/editar nos locais e nomes corretos.
- Inclui comentários no código, em locais especiais.
- E, após cada código fonte, inclui uma nova página no material contendo a explicação do que foi feito, e dos pontos mais importantes.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**BaseViewModel.cs**

```
using CommunityToolkit.Mvvm.ComponentModel;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    public partial class BaseViewModel : ObservableObject
    {
        private bool isBusy; // Indica se uma operação está em andamento, útil para mostrar indicadores de carregamento na UI.
        public bool IsBusy
        {
            get => isBusy;
            set => SetProperty(ref isBusy, value);
        }

        private string title = string.Empty; // Título da ViewModel, pode ser usado para definir o título da página na UI.
        public string Title
        {
            get => title;
            set => SetProperty(ref title, value);
        }

        private bool isRefreshing; // Indica se a ViewModel está em estado de atualização, útil para pull-to-refresh na UI.
        public bool IsRefreshing
        {
            get => isRefreshing;
            set => SetProperty(ref isRefreshing, value);
        }
    }

    // ObservableObject é uma classe base que implementa a interface INotifyPropertyChanged.
    // Permitindo que as propriedades notifiquem automaticamente quando seus valores mudem.
    // Isso é útil para atualizar a interface do usuário em resposta a mudanças nos dados.
    // O atributo [ObservableProperty] é um recurso do CommunityToolkit.Mvvm que simplifica a criação de propriedades observáveis.
}
```

- **ObservableObject** é uma classe base do MVVM Toolkit que implementa a interface **INotifyPropertyChanged**.
- Ela facilita a criação de **ViewModels** e **Models** que **notificam automaticamente** a interface do usuário sobre mudanças em suas **propriedades**, permitindo a atualização reativa dos dados na UI.
- Isso permite que propriedades contidas na ViewModel, como TotalLogradouros, TotalAlunos etc, sejam usadas em **bindings** no XAML.
- **Data binding** no XAML é o mecanismo que conecta propriedades da ViewModel, ou Model, à interface do usuário de forma declarativa.
- Com ele, controles como Label, Entry ou Button exibem e atualizam dados automaticamente conforme o valor das propriedades na ViewModel muda, e vice-versa.
- Isso é possível porque a ViewModel implementa INotifyPropertyChanged, permitindo que a UI reaja a mudanças sem código extra, promovendo desacoplamento e atualização reativa da interface.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**DashboardListViewModel.cs**

```
using AcademiaDoZe.Application.Interfaces; using CommunityToolkit.Mvvm.Input;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    public partial class DashboardListViewModel : BaseViewModel
    {
        private int _totalLogradouros;
        public int TotalLogradouros {get => _totalLogradouros; set => SetProperty(ref _totalLogradouros, value); }

        private int _totalAlunos;
        public int TotalAlunos {get => _totalAlunos; set => SetProperty(ref _totalAlunos, value); }

        private int _totalColaboradores;
        public int TotalColaboradores {get => _totalColaboradores; set => SetProperty(ref _totalColaboradores, value); }

        private int _totalMatriculas;
        public int TotalMatriculas {get => _totalMatriculas; set => SetProperty(ref _totalMatriculas, value); }

        [RelayCommand]
        private async Task LoadDashboardDataAsync()
        {
            if (IsBusy)
                return;

            try
            {
                IsBusy = true;

                TotalLogradouros = 0;
                TotalAlunos = 0;
                TotalColaboradores = 0;
                TotalMatriculas = 0;
            }
            finally
            {
                IsBusy = false;
            }
        }

        [RelayCommand]
        private async Task NavigateToLogradourosAsync() => await Shell.Current.GoToAsync("//logradouros");

        [RelayCommand]
        private async Task NavigateToAlunosAsync() => await Shell.Current.GoToAsync("//alunos");

        [RelayCommand]
        private async Task NavigateToColaboradoresAsync() => await Shell.Current.GoToAsync("//colaboradores");

        [RelayCommand]
        private async Task NavigateToMatriculasAsync() => await Shell.Current.GoToAsync("//matriculas");
    }
}
```

- Reparem que herdamos de BaseViewModel, sendo assim já utilizamos **INotifyPropertyChanged,** [ObservableProperty].

- O atributo **[RelayCommand]** do MVVM Toolkit transforma **métodos** em **comandos** prontos para **data binding** no XAML.

- Ele gera automaticamente uma propriedade do tipo ICommand na ViewModel, permitindo que botões e outros controles de interface executem métodos assíncronos ou síncronos da ViewModel sem necessidade de criar manualmente classes de comando.

- Isso simplifica a ligação entre ações da interface e a lógica da ViewModel no padrão MVVM.

## AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**LogradouroListViewModel.cs**

```
using AcademiaDoZe.Application.DTOs;
using AcademiaDoZe.Application.Interfaces;
using CommunityToolkit.Mvvm.Input;
using System.Collections.ObjectModel;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    public partial class LogradouroListViewModel : BaseViewModel
    {
        public ObservableCollection<string> FilterTypes { get; } = new() { "Cidade", "Id", "Cep" };

        private string _searchText = string.Empty;
        public string SearchText
        {
            get => _searchText;
            set => SetProperty(ref _searchText, value);
        }

        private string _selectedFilterType = "Cidade"; // Cidade, Id, Cep
        public string SelectedFilterType
        {
            get => _selectedFilterType;
            set => SetProperty(ref _selectedFilterType, value);
        }

        // inicialmente só vamos incluir aqui o comando para navegar para a tela de cadastro


        [RelayCommand]
        private async Task AddLogradouroAsync()
        {
            try
            {
                // GoToAsync é usado para navegar entre páginas no MAUI Shell.
                // logradouro é o nome da rota registrada no AppShell.xaml.cs
                await Shell.Current.GoToAsync("logradouro");
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao navegar para tela de cadastro: {ex.Message}", "OK");
            }
        }

    }
}
```

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**LogradouroViewModel.cs**

```
using CommunityToolkit.Mvvm.Input;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    public partial class LogradouroViewModel : BaseViewModel
    {

        // inicialmente só estamos incluindo o comando de cancelar


        [RelayCommand]
        private async Task CancelAsync()
        {
            await Shell.Current.GoToAsync("..");
        }

    }
}
```

- O arquivo **Styles.xaml** é usado para definir estilos, cores, fontes e recursos visuais reutilizáveis em toda a aplicação XAML.

- Ele centraliza configurações de aparência, como estilos de botões, textos, layouts e temas, permitindo padronização visual e fácil manutenção do design da interface.

- Esses estilos podem ser aplicados globalmente ou em controles específicos via XAML.

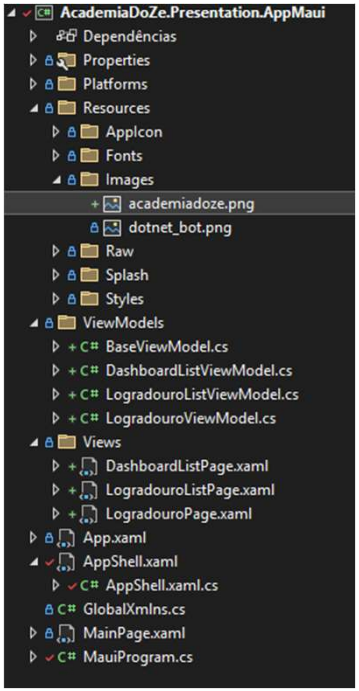
- Vamos criar nossas páginas utilizando **estilos globais** aplicados para Label, Button, Entry etc, conforme padrão **DRY**.
  - O padrão **DRY, Don't Repeat Yourself**, é um princípio de desenvolvimento de software que recomenda evitar duplicação de código ou lógica.

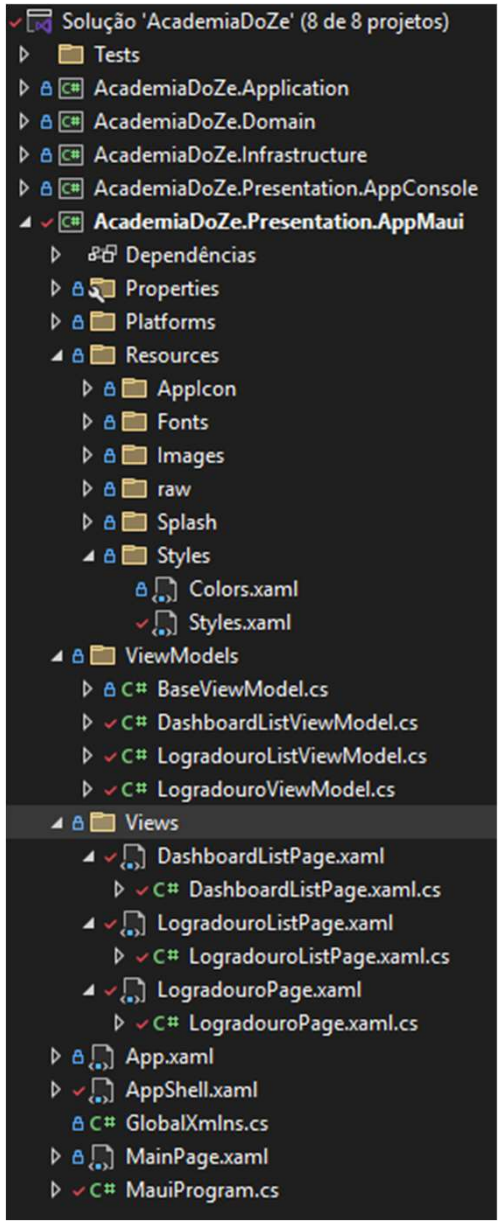
- Como base inicial, utilizaremos o padrão de cores e estilos já disponibilizados no arquivo, somente incluindo em seu final, algumas personalizações para Border e Botões.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Resources\Styles**Styles.xaml**

```
<!-- Estilos nomeados para Border -->
<Style TargetType="Border" x:Key="CardBorder">
    <Setter Property="BackgroundColor" Value="{AppThemeBinding Light={StaticResource Secondary}, Dark={StaticResource Gray900}}"/>
    <Setter Property="Stroke" Value="{AppThemeBinding Light={StaticResource Gray200}, Dark={StaticResource Gray600}}"/>
    <Setter Property="StrokeThickness" Value="1"/>
    <Setter Property="StrokeShape" Value="RoundRectangle 12"/>
    <Setter Property="Padding" Value="15"/>
</Style>

<!-- Estilos nomeados para Button -->
<Style TargetType="Button" x:Key="ButtonPrimary">
    <Setter Property="BackgroundColor" Value="{AppThemeBinding Light={StaticResource Primary}, Dark={StaticResource PrimaryDark}}"/>
    <Setter Property="TextColor" Value="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource PrimaryDarkText}}"/>
    <Setter Property="FontSize" Value="16"/>
    <Setter Property="CornerRadius" Value="25"/>
    <Setter Property="HeightRequest" Value="50"/>
</Style>

<Style TargetType="Button" x:Key="ButtonSecondary">
    <Setter Property="BackgroundColor" Value="Transparent"/>
    <Setter Property="TextColor" Value="{AppThemeBinding Light={StaticResource Gray600}, Dark={StaticResource Gray400}}"/>
    <Setter Property="BorderColor" Value="{AppThemeBinding Light={StaticResource Gray300}, Dark={StaticResource Gray600}}"/>
    <Setter Property="BorderWidth" Value="1"/>
    <Setter Property="FontSize" Value="16"/>
    <Setter Property="CornerRadius" Value="25"/>
    <Setter Property="HeightRequest" Value="50"/>
</Style>

<Style TargetType="Button" x:Key="ButtonSmall">
    <Setter Property="FontSize" Value="12"/>
    <Setter Property="CornerRadius" Value="16"/>
    <Setter Property="HeightRequest" Value="32"/>
    <Setter Property="WidthRequest" Value="60"/>
</Style>

<Style TargetType="Button" x:Key="ButtonDanger">
    <Setter Property="BackgroundColor" Value="{AppThemeBinding Light={StaticResource Magenta}, Dark={StaticResource Magenta}}"/>
    <Setter Property="TextColor" Value="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource White}}"/>
    <Setter Property="FontSize" Value="12"/>
    <Setter Property="CornerRadius" Value="16"/>
    <Setter Property="HeightRequest" Value="32"/>
    <Setter Property="WidthRequest" Value="60"/>
</Style>
```

- Gere uma imagem que simbolize sua aplicação.
- **Obrigatoriamente a imagem deve conter seu nome**.
- Renomeie como **academiadoze.png**.
- Salve a imagem no diretório Resources\Images**academiadoze.png**




- Criar a pasta **Views**, e nela adicionar novos itens do tipo **.NET MAUI ContentPage (XAML)**
  - **DashboardListPage.xaml**
    - Define a interface da página inicial do sistema, exibindo um dashboard com cards para Logradouros, Alunos, Colaboradores e Matrículas.
    - Cada card mostra o total respectivo e permite navegação ao tocar.
    - O layout inclui um cabeçalho com logo e título, utiliza estilos centralizados e exibe um indicador de carregamento, ActivityIndicator, quando a ViewModel está ocupada.
    - Grid responsivo e com **bindings** diretos à **DashboardListViewModel**.
  - **LogradouroListPage.xaml**
    - Define a interface da tela de listagem de logradouros.
    - Possui uma barra de busca com filtro, uma lista, CollectionView, que exibe os logradouros com botões de editar e excluir, e um botão flutuante para adicionar novos itens.
    - Usa **bindings** para comandos e propriedades da LogradouroListViewModel, exibe mensagens quando a lista está vazia e mostra um indicador de carregamento, ActivityIndicator, durante operações assíncronas.
    - Layout responsivo e utiliza estilos centralizados para manter a padronização visual.
  - **LogradouroPage.xaml**
    - Define a interface para cadastro e edição de logradouros.
    - Campos vinculados à ViewModel.
    - O layout é organizado em cartões (Border) e utiliza estilos centralizados, proporcionando uma experiência de edição simples e padronizada para o usuário.
- Observação: Ao realizar a criação do arquivo XAML, automaticamente já é criado seu code-behind, ou seja, seu arquivo .cs.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**DashboardListPage.xaml**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AcademiaDoZe.Presentation.AppMaui.Views.DashboardListPage"
             xmlns:vm="clr-namespace:AcademiaDoZe.Presentation.AppMaui.ViewModels"
             x:DataType="vm:DashboardListViewModel"
             Title="{Binding Title}">

    <ScrollView>
        <Grid RowDefinitions="Auto,*">

            <!-- Header -->
            <StackLayout Spacing="10" Margin="0,0,0,20">
                <Image Source="academiadoze.png" HorizontalOptions="Center" VerticalOptions="Start" HeightRequest="200" WidthRequest="200"/>
                <Label Text="Academia do Zé" Style="{StaticResource Headline}"/>
                <Label Text="Sistema de Gerenciamento" Style="{StaticResource SubHeadline}"/>
            </StackLayout>

            <!-- Cards Container -->
            <Grid Grid.Row="1" RowDefinitions="Auto,Auto" ColumnDefinitions="*,*" RowSpacing="15" ColumnSpacing="15">


                <!-- Card Logradouros -->

                <!-- Card Alunos -->

                <!-- Card Colaboradores -->

                <!-- Card Matrículas -->

            </Grid>

            <!-- Loading Indicator -->
            <ActivityIndicator Grid.Row="0" Grid.RowSpan="2" IsVisible="{Binding IsBusy}" IsRunning="{Binding IsBusy}" VerticalOptions="Center" HorizontalOptions="Center" />

        </Grid>
    </ScrollView>

</ContentPage>
```

```xml
<!-- Card Logradouros -->
<Border Grid.Row="0" Grid.Column="0" Style="{StaticResource CardBorder}" Margin="15,10">
    <Border.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding NavigateToLogradourosCommand}"/>
    </Border.GestureRecognizers>

    <StackLayout Spacing="10">
        <Label Text="{Binding TotalLogradouros}" Style="{StaticResource SubHeadline}"/>
        <Label Text="Logradouros" HorizontalTextAlignment="Center"/>
    </StackLayout>
</Border>

<!-- Card Alunos -->
<Border Grid.Row="0" Grid.Column="1" Style="{StaticResource CardBorder}" Margin="15,10">
    <Border.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding NavigateToAlunosCommand}"/>
    </Border.GestureRecognizers>

    <StackLayout Spacing="10">
        <Label Text="{Binding TotalAlunos}" Style="{StaticResource SubHeadline}"/>
        <Label Text="Alunos" HorizontalTextAlignment="Center"/>
    </StackLayout>
</Border>

<!-- Card Colaboradores -->
<Border Grid.Row="1" Grid.Column="0" Style="{StaticResource CardBorder}" Margin="15,10">
    <Border.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding NavigateToColaboradoresCommand}"/>
    </Border.GestureRecognizers>

    <StackLayout Spacing="10">
        <Label Text="{Binding TotalColaboradores}" Style="{StaticResource SubHeadline}"/>
        <Label Text="Colaboradores" HorizontalTextAlignment="Center"/>
    </StackLayout>
</Border>

<!-- Card Matrículas -->
<Border Grid.Row="1" Grid.Column="1" Style="{StaticResource CardBorder}" Margin="15,10">
    <Border.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding NavigateToMatriculasCommand}"/>
    </Border.GestureRecognizers>

    <StackLayout Spacing="10">
        <Label Text="{Binding TotalMatriculas}" Style="{StaticResource SubHeadline}"/>
        <Label Text="Matrículas" HorizontalTextAlignment="Center"/>
    </StackLayout>
</Border>
```

- **x:Class="AcademiaDoZe.Presentation.AppMaui.Views.DashboardListPage"**
  - Define a classe C# que representa essa página, permitindo o code-behind e a ligação entre XAML e C#.

- **xmlns:vm="clr-namespace:AcademiaDoZe.Presentation.AppMaui.ViewModels"**
  - Cria um prefixo **vm** para referenciar tipos do namespace das ViewModels no XAML.

- **x:DataType="vm:DashboardListViewModel"**
  - Especifica o tipo de ViewModel usado para binding forte, IntelliSense e checagem de tipos, no XAML, melhorando a experiência de desenvolvimento e evitando erros de binding.

- Reparem que em vários locais, utilizamos **Binding** no XAML, **conectando propriedades da interface diretamente com as propriedades ou comandos da ViewModel**. Como já vimos, isso permite que a UI reflita automaticamente mudanças nos dados e acione lógicas da ViewModel sem código-behind. O binding é a base do padrão MVVM, promovendo desacoplamento, reatividade e manutenção facilitada da interface. Veja os exemplos abaixo:
  - **Command="{Binding NavigateToLogradourosCommand}"/>**
  - **Text="{Binding TotalLogradouros}"**

- Também utilizamos aqui, estilos nomeados. Veja o exemplo abaixo:
  - **Style="{StaticResource CardBorder}"**

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**DashboardListPage.xaml.cs**

```
using AcademiaDoZe.Presentation.AppMaui.ViewModels;

namespace AcademiaDoZe.Presentation.AppMaui.Views;

public partial class DashboardListPage : ContentPage
{
    public DashboardListPage(DashboardListViewModel viewModel)
    {
        InitializeComponent();
        BindingContext = viewModel;
    }

    protected override async void OnAppearing()
    {
        base.OnAppearing();

        if (BindingContext is DashboardListViewModel viewModel)
        {
            await viewModel.LoadDashboardDataCommand.ExecuteAsync(null);
        }
    }
}
```

- O **código-behind** é o arquivo C# associado a um arquivo XAML, por exemplo, **DashboardListPage.xaml.cs**, vinculado ao arquivo **DashboardListPage.xaml**.
- Ele serve para implementar lógicas específicas da interface que não são facilmente resolvidas apenas com bindings, como manipulação de eventos, inicialização de componentes, navegação ou integração com APIs de plataforma.
- No padrão MVVM, **seu uso deve ser mínimo**, priorizando a lógica na ViewModel, mas ele ainda é útil para casos onde o XAML puro não é suficiente.

- No código-behind, inicialmente ajustamos o construtor para receber uma **instância** da **DashboardListViewModel**, injetada por DI, inicializando os componentes da interface e definindo o **BindingContext** da página para essa ViewModel, permitindo que todos os bindings do XAML funcionem corretamente. O BindingContext é a propriedade que define a fonte de dados para os bindings de uma página ou controle no XAML.

- Na sequencia, criamos o método **OnAppearing** no código-behind, e ele é chamado automaticamente toda vez que a página aparece na tela. Ele serve para garantir que, ao abrir a página, o comando **LoadDashboardDataCommand** da ViewModel seja executado, atualizando os dados do dashboard sempre que o usuário acessar essa tela.

## AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**LogradouroListPage.xaml**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AcademiaDoZe.Presentation.AppMaui.Views.LogradouroListPage"
             xmlns:vm="clr-namespace:AcademiaDoZe.Presentation.AppMaui.ViewModels"
             x:DataType="vm:LogradouroListViewModel"
             xmlns:dto="clr-namespace:AcademiaDoZe.Application.DTOs;assembly=AcademiaDoZe.Application"
             Title="{Binding Title}">


    <Grid RowDefinitions="Auto,*">

        <!-- Search Bar com Picker de filtro -->
        <Border Grid.Row="0" Style="{StaticResource CardBorder}" Margin="15,10">
            <Grid ColumnDefinitions="Auto,10,*,10,Auto" VerticalOptions="Center">

                <!-- Picker de filtro -->
                <Picker Grid.Column="0" Title="" ItemsSource="{Binding FilterTypes}" SelectedItem="{Binding SelectedFilterType}" WidthRequest="110"
HorizontalOptions="Start" HeightRequest="40" VerticalOptions="Center"/>

                <!-- Entry de busca -->
                <Entry Grid.Column="2" Text="{Binding SearchText}" Placeholder="Digite o valor..." HeightRequest="40" VerticalOptions="Center" Margin="0,0,0,0"/>

                <!-- Botão de busca -->
                <Button Grid.Column="4" Text="Buscar" Style="{StaticResource ButtonPrimary}" FontSize="14" WidthRequest="80" HeightRequest="40" Margin="0,0,0,0"
Command="{Binding SearchLogradourosCommand}"/>

            </Grid>
        </Border>


        <!-- Lista de Logradouros com FAB  - "Floating Action Button" - Botão de Ação Flutuante -->



        <!-- Loading Indicator -->
        <ActivityIndicator Grid.Row="0" Grid.RowSpan="2" IsVisible="{Binding IsBusy}" IsRunning="{Binding IsBusy}" VerticalOptions="Center" HorizontalOptions="Center"/>

    </Grid>

</ContentPage>
```

```xml
        <!-- Lista de Logradouros com FAB  - "Floating Action Button" - Botão de Ação Flutuante -->
        <Grid Grid.Row="1">
            <RefreshView IsRefreshing="{Binding IsRefreshing}" Command="{Binding RefreshCommand}">

                <CollectionView ItemsSource="{Binding Logradouros}" SelectionMode="None">

                    <CollectionView.EmptyView>
                        <StackLayout VerticalOptions="Center">
                            <Label Text="Nenhum logradouro encontrado" HorizontalOptions="Center" Style="{StaticResource SubHeadline}"/>
                            <Button Text="Adicionar Primeiro Logradouro" Command="{Binding AddLogradouroCommand}" Style="{StaticResource ButtonPrimary}" HeightRequest="50" Margin="0,20,0,0"/>
                        </StackLayout>
                    </CollectionView.EmptyView>

                    <CollectionView.ItemTemplate>
                        <DataTemplate x:DataType="dto:LogradouroDTO">
                            <Grid Padding="1" Margin="5">
                                <Border Style="{StaticResource CardBorder}" Margin="10">

                                    <Grid RowDefinitions="Auto,Auto,Auto,Auto,Auto" ColumnDefinitions="*,Auto,Auto">

                                        <Label Grid.Row="0" Grid.Column="0" Text="{Binding Cep}" FontAttributes="Bold" Margin="0,0,0,5"/>
                                        <Label Grid.Row="1" Grid.Column="0" Text="{Binding Nome}" Margin="0,0,0,5"/>
                                        <Label Grid.Row="2" Grid.Column="0" Text="{Binding Bairro}" Margin="0,0,0,5"/>
                                        <Label Grid.Row="3" Grid.Column="0" Margin="0,0,0,5">
                                            <Label.FormattedText>
                                                <FormattedString>
                                                    <Span Text="{Binding Cidade}"/> <Span Text=" - "/> <Span Text="{Binding Estado}"/>
                                                </FormattedString>
                                            </Label.FormattedText>
                                        </Label>
                                        <Label Grid.Row="4" Grid.Column="0" Text="{Binding Pais}" Margin="0,0,0,5"/>

                                        <!-- Botões de Ação -->
                                        <Button Grid.Row="0" Grid.Column="1" Text="Editar" Style="{StaticResource ButtonSmall}" Margin="4,0" Clicked="OnEditButtonClicked" />
                                        <Button Grid.Row="0" Grid.Column="2" Text="Excluir" Style="{StaticResource ButtonDanger}" Margin="4,0" Clicked="OnDeleteButtonClicked" />
                                    </Grid>
                                </Border>
                            </Grid>
                        </DataTemplate>
                    </CollectionView.ItemTemplate>
                </CollectionView>
            </RefreshView>

            <!-- Floating Action Button -->
            <Button Text="+" Command="{Binding AddLogradouroCommand}" Style="{StaticResource ButtonPrimary}"
            WidthRequest="56" HeightRequest="56" HorizontalOptions="End" VerticalOptions="End" Margin="20,20,20,20">
                <Button.Shadow>
                    <Shadow Brush="{AppThemeBinding Light=Black, Dark=Gray}" Offset="0,4" Radius="8" Opacity="0.3"/>
                </Button.Shadow>
            </Button>

        </Grid>
```

- **xmlns:dto="clr-namespace:AcademiaDoZe.Application.DTOs;assembly=AcademiaDoZe.Application"**
  - Cria o prefixo **dto** no XAML, permitindo referenciar tipos do namespace AcademiaDoZe.Application.DTOs que estão definidos no assembly **AcademiaDoZe.Application**.
  - Assim, você pode usar, por exemplo, **<DataTemplate x:DataType="dto:LogradouroDTO">**, que embora não seja obrigatório, permite ter binding forte e IntelliSense para esse tipo no XAML.

- **<CollectionView ItemsSource="{Binding Logradouros}" SelectionMode="None">**
  - Define que o **CollectionView** exibirá uma lista de itens baseada na **propriedade Logradouros** da **ViewModel**.
  - Ou seja, para cada item presente em Logradouros, será gerado um elemento visual conforme o ItemTemplate.
  - O SelectionMode="None" indica que os itens não podem ser selecionados pelo usuário, servindo apenas para exibição.

- No XAML anterior podemos ver duas formas de chamar a execução de métodos:
  - **Command="{Binding SearchLogradourosCommand}**
    - Executa métodos marcados com [RelayCommand] no ViewModel.
  - **Clicked="OnEditButtonClicked"**
    - Para tratar o evento, executa métodos implementados no código-behind.

- <Picker **ItemsSource="{Binding FilterTypes}" SelectedItem="{Binding SelectedFilterType}"** />
  - Com base em propriedades criadas na ViewModel, populamos os itens do campo, e retornamos seu valor selecionado.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**LogradouroListPage.xaml.cs**

```
using AcademiaDoZe.Application.DTOs;
using AcademiaDoZe.Presentation.AppMaui.ViewModels;

namespace AcademiaDoZe.Presentation.AppMaui.Views;

public partial class LogradouroListPage : ContentPage
{
    public LogradouroListPage(LogradouroListViewModel viewModel)
    {
        InitializeComponent();
        BindingContext = viewModel;
    }

    protected override async void OnAppearing()
    {
        /* implementar depois */
    }

    private async void OnEditButtonClicked(object sender, EventArgs e)
    {
        /* implementar depois */
    }

    private async void OnDeleteButtonClicked(object sender, EventArgs e)
    {
        /* implementar depois */
    }
}
```

- Aqui, no código-behind, deixamos a espera para a implementação de três métodos.
  - **protected override async void OnAppearing()**
    - Como já vimos, é utilizado para atualizar os dados sempre que o usuário acessar a tela.
  - **private async void OnEditButtonClicked(object sender, EventArgs e)**
    - Implementaremos para chamar o comando da ViewModel que realiza a abertura da pagina para edição dos dados.
  - **private async void OnDeleteButtonClicked(object sender, EventArgs e)**
    - Implementaremos para chamar o comando da ViewModel que realiza a exclusão dos dados.

## AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**LogradouroPage.xaml**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AcademiaDoZe.Presentation.AppMaui.Views.LogradouroPage"
             xmlns:vm="clr-namespace:AcademiaDoZe.Presentation.AppMaui.ViewModels"
             x:DataType="vm:LogradouroViewModel"
             Title="{Binding Title}">


    <ScrollView>
        <StackLayout Spacing="20">

            <!-- CEP Section -->
            <Border Style="{StaticResource CardBorder}" Margin="15,10">

                <StackLayout Spacing="5">
                    <Label Text="CEP"/>

                    <Grid ColumnDefinitions="*,Auto">
                        <Entry Grid.Column="0" Text="{Binding Logradouro.Cep}" Placeholder="00000-000" Keyboard="Numeric" Margin="0,0,10,0"/>
                        <Button Grid.Column="1" Text="Buscar CEP" Command="{Binding SearchByCepCommand}" Style="{StaticResource ButtonPrimary}" HeightRequest="40" Padding="15,0"/>
                    </Grid>
                </StackLayout>
            </Border>

            <!-- Dados do Logradouro -->



            <!-- Botões de Ação -->
            <Grid ColumnDefinitions="*,*" ColumnSpacing="10" Margin="15,10">
                <Button Grid.Column="0" Text="Cancelar" Command="{Binding CancelCommand}" Style="{StaticResource ButtonSecondary}" HeightRequest="50"/>
                <Button Grid.Column="1" Text="Salvar" Command="{Binding SaveLogradouroCommand}" Style="{StaticResource ButtonPrimary}" HeightRequest="50"/>
            </Grid>

            <!-- Loading Indicator -->
            <ActivityIndicator IsVisible="{Binding IsBusy}" IsRunning="{Binding IsBusy}" VerticalOptions="Center" HorizontalOptions="Center"/>

        </StackLayout>
    </ScrollView>

</ContentPage>
```

```
        <!-- Dados do Logradouro -->
        <Border Style="{StaticResource CardBorder}" Margin="15,10">

            <StackLayout Spacing="15">
                <Label Text="{Binding Logradouro.Id, StringFormat='Logradouro ID: {0}'}" Style="{StaticResource SubHeadline}"/>

                <!-- Nome -->
                <StackLayout Spacing="5">
                    <Label Text="Nome/Rua"/>
                    <Entry Text="{Binding Logradouro.Nome}" Placeholder="Digite o nome da rua..."/>
                </StackLayout>

                <!-- Bairro -->
                <StackLayout Spacing="5">
                    <Label Text="Bairro"/>
                    <Entry Text="{Binding Logradouro.Bairro}" Placeholder="Digite o bairro..."/>
                </StackLayout>

                <!-- Cidade -->
                <StackLayout Spacing="5">
                    <Label Text="Cidade"/>
                    <Entry Text="{Binding Logradouro.Cidade}" Placeholder="Digite a cidade..."/>
                </StackLayout>

                <!-- Estado -->
                <StackLayout Spacing="5">
                    <Label Text="Estado"/>
                    <Entry Text="{Binding Logradouro.Estado}" Placeholder="Digite o estado..."/>
                </StackLayout>

                <!-- País -->
                <StackLayout Spacing="5">
                    <Label Text="País"/>
                    <Entry Text="{Binding Logradouro.Pais}" Placeholder="Digite o país..."/>
                </StackLayout>

            </StackLayout>
        </Border>
```

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**LogradouroPage.xaml.cs**

```
using AcademiaDoZe.Presentation.AppMaui.ViewModels;

namespace AcademiaDoZe.Presentation.AppMaui.Views;

public partial class LogradouroPage : ContentPage
{
    public LogradouroPage(LogradouroViewModel viewModel)
    {
        InitializeComponent();
        BindingContext = viewModel;
    }

    protected override async void OnAppearing()
    {
        /* implementar depois */
    }
}
```

- Versão inicial das **ViewModel** e **Páginas** criadas.
- Na sequencia só precisamos editar alguns arquivos do projeto para viabilizar a navegação pelas páginas criadas.

- AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**AppShell.xaml**
  - Contem as **rotas de acesso**.
  - Incialmente possui uma rota chamada **MainPage**, apontando para a view **MainPage**.
  - Vamos apagar essa rota, e incluir rotas que permitam acessar nossas páginas.
  - Realizaremos isso através da criação de um **menu de acesso**.
  - No exemplo mostro como fazer utilizando **FlyoutItem** e **TabBar**.

- AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**AppShell.xaml.cs**
  - No **código-behind** de **AppShell**, realizamos o **Routing.RegisterRoute** para que o Shell do MAUI reconheça e permita a navegação para páginas que não serão acessadas diretamente através do TabBar ou Flyout, como páginas de detalhe, edição ou cadastro etc.

- AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**MauiProgram.cs**
  - É o responsável por configurar e inicializar a aplicação MAUI.
  - Aqui definimos serviços de **injeção de dependência**, configurações globais, recursos, handlers e registra as **ViewModels** e **páginas**.
  - Ele retorna o objeto MauiApp, que representa a aplicação pronta para ser executada.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**AppShell.xaml**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="AcademiaDoZe.Presentation.AppMaui.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:views="clr-namespace:AcademiaDoZe.Presentation.AppMaui.Views"
    Title="Academia do Zé">

    <!--
    <ShellContentTitle="Home" ContentTemplate="{DataTemplate views:MainPage}" Route="MainPage" />
    -->

    <FlyoutItem Title="Dashboard">
        <ShellContent ContentTemplate="{DataTemplate views:DashboardListPage}" Route="dashboard" />
    </FlyoutItem>

    <FlyoutItem Title="Logradouros">
        <ShellContent ContentTemplate="{DataTemplate views:LogradouroListPage}" Route="logradouros" />
    </FlyoutItem>


    <!--
    <TabBar>
        <ShellContent Title="Dashboard" ContentTemplate="{DataTemplate views:DashboardListPage}" Route="dashboard" />
        <ShellContent Title="Logradouros" ContentTemplate="{DataTemplate views:LogradouroListPage}" Route="logradouros" />
    </TabBar>
    -->


</Shell>
```

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**AppShell.xaml.cs**

```
using AcademiaDoZe.Presentation.AppMaui.Views;

namespace AcademiaDoZe.Presentation.AppMaui
{
    public partial class AppShell : Shell
    {
        public AppShell()
        {
            InitializeComponent();
            RegisterRoutes();
        }

        // O Routing.RegisterRoute é necessário para que o Shell do MAUI reconheça e permita a navegação
        // para páginas que não estão diretamente no TabBar ou Flyout,
        // como páginas de detalhe, edição ou cadastro.
        private static void RegisterRoutes()
        {
            Routing.RegisterRoute("logradouro", typeof(LogradouroPage));
        }

    }
}
```

## AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**MauiProgram.cs**

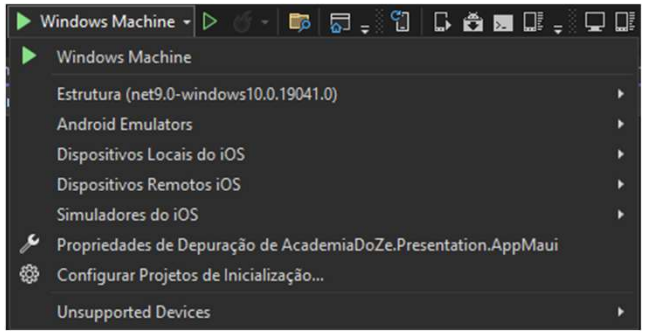
```
using AcademiaDoZe.Presentation.AppMaui.ViewModels;
using AcademiaDoZe.Presentation.AppMaui.Views;
using Microsoft.Extensions.Logging;

namespace AcademiaDoZe.Presentation.AppMaui
{
    public static class MauiProgram
    {
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
                });


            // Configurar serviços da aplicação e repositórios
            //ConfigurationHelper.ConfigureServices(builder.Services);

            // Registrar ViewModels
            builder.Services.AddTransient<DashboardListViewModel>();
            builder.Services.AddTransient<LogradouroListViewModel>();
            builder.Services.AddTransient<LogradouroViewModel>();

            // Registrar Views
            builder.Services.AddTransient<DashboardListPage>();
            builder.Services.AddTransient<LogradouroListPage>();
            builder.Services.AddTransient<LogradouroPage>();


#if DEBUG
            builder.Logging.AddDebug();
#endif

            return builder.Build();
        }
    }
}
```

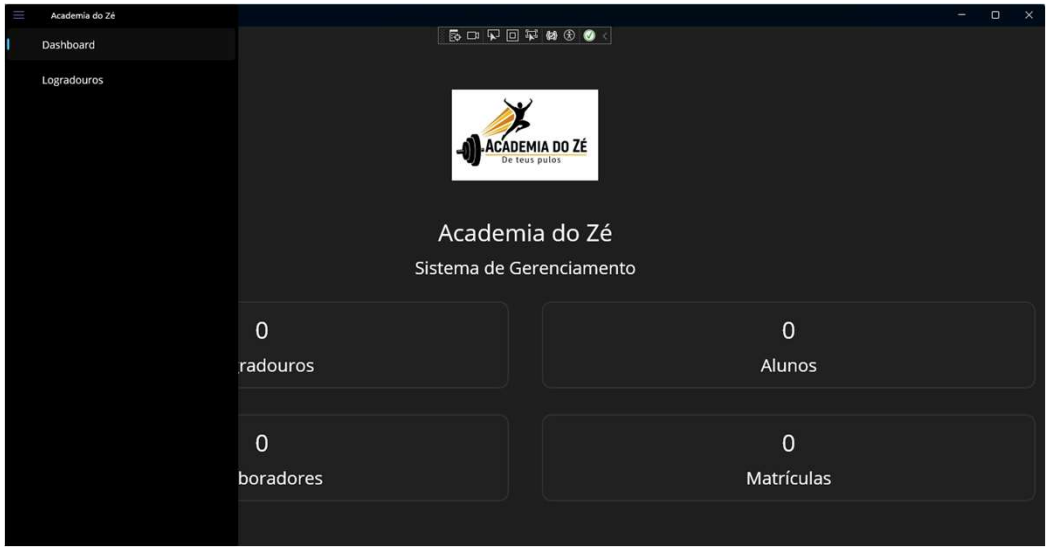
- A **View** de **Dashboard** e **Logradouro** já está **completa**, com todo o vínculo necessário para **receber dados diretamente da ViewModel**.

- Na sequência, o foco principal será **concluir a ViewModel**, integrando-a com a **camada de aplicação** para que a lógica de negócio seja implementada.

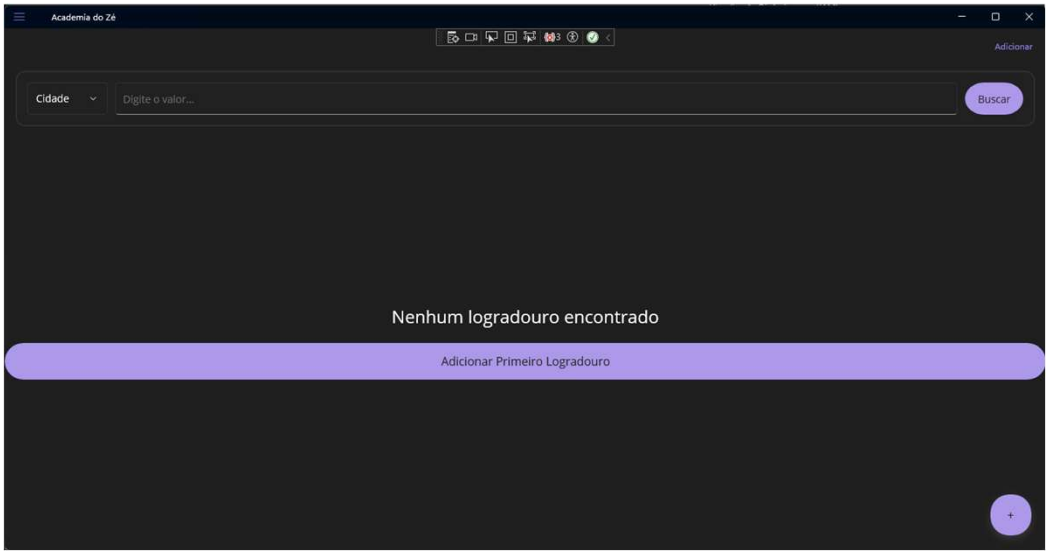
- Apesar da **ViewModel** ainda não estar finalizada, já é possível executar e testar como está ficando nossa aplicação.

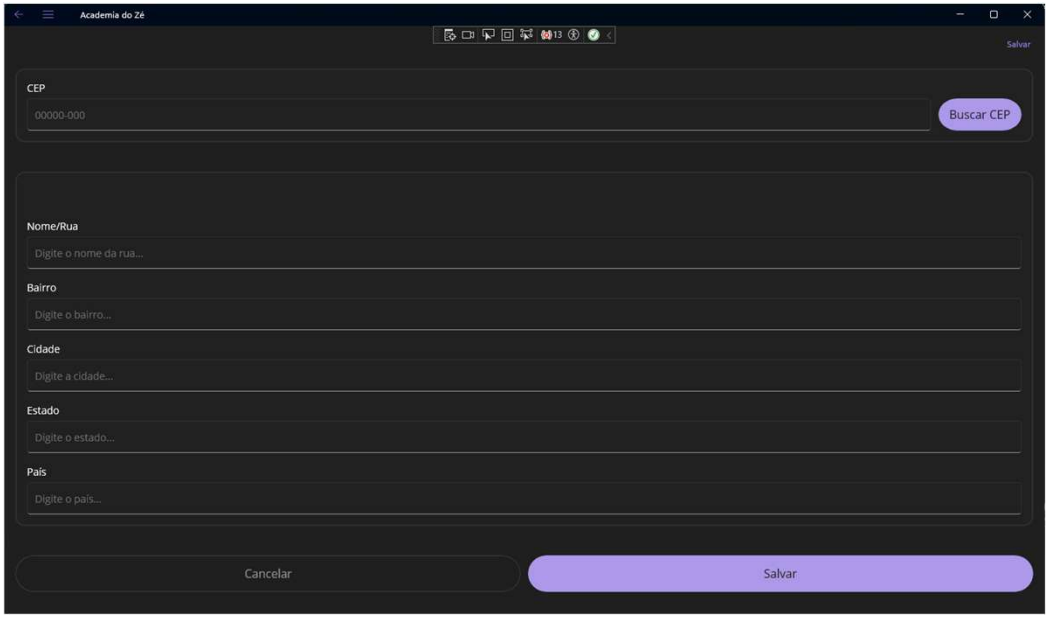
- Isso nos permite validar o fluxo de navegação e fazer os ajustes necessários nas telas criadas, garantindo que a experiência do usuário seja a melhor possível.

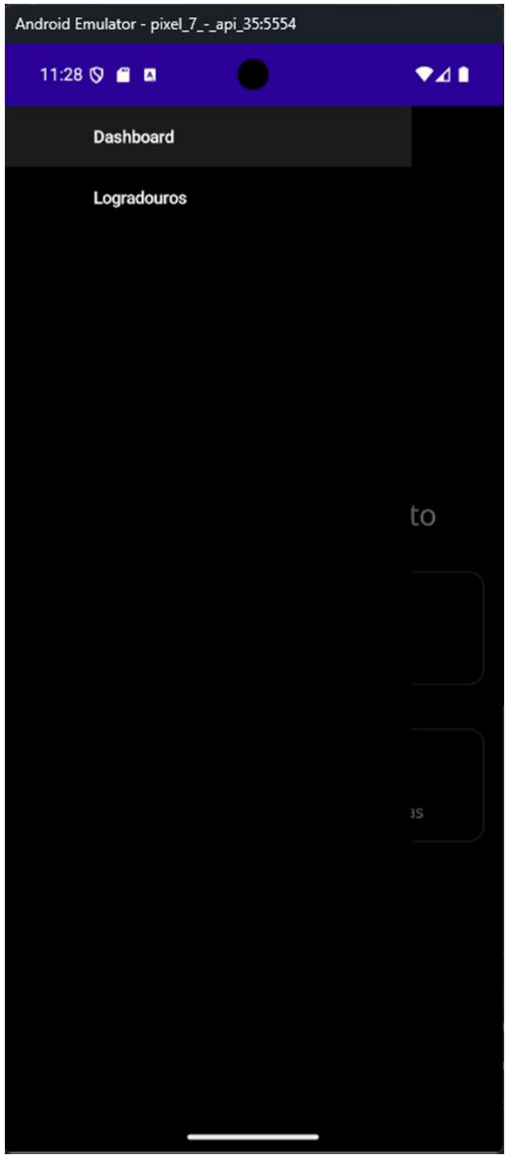



Academia do Zé
Dashboard
Logradouros
ACADEMIA DO ZÉ
De teus pulos
Academia do Zé
Sistema de Gerenciamento
0
radouros
0
Alunos
0
boradores
0
Matrículas

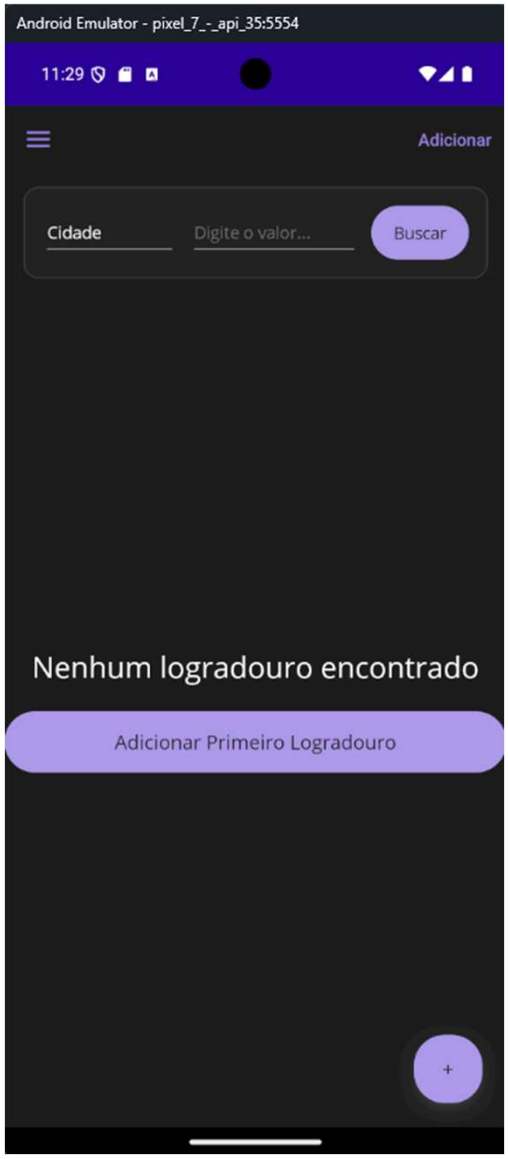

Academia do Zé
ACADEMIA DO ZÉ
De teus pulos
Academia do Zé
Sistema de Gerenciamento
0
Logradouros
0
Alunos
0
Colaboradores
0
Matrículas


Academia do Zé
Adicionar
Cidade
Digite o valor...
Buscar
Nenhum logradouro encontrado
Adicionar Primeiro Logradouro


Academia do Zé
Salvar
CEP
00000-000
Buscar CEP
Nome/Rua
Digite o nome da rua...
Bairro
Digite o bairro...
Cidade
Digite a cidade...
Estado
Digite o estado...
País
Digite o país...
Cancelar
Salvar


Android Emulator - pixel_7_-_api_35:5554
11:28
Dashboard
Logradouros


Android Emulator - pixel_7_-_api_35:5554
11:29
ACADEMIA DO ZÉ
De teus pulos
Academia do Zé
Sistema de Gerenciamento
0
Logradouros
0
Alunos
0
Colaboradores
0
Matrículas


Android Emulator - pixel_7_-_api_35:5554
11:29
Adicionar
Cidade
Digite o valor...
Buscar
Nenhum logradouro encontrado
Adicionar Primeiro Logradouro


Android Emulator - pixel_7_-_api_35:5554
11:30
Salvar
CEP
00000-000
Buscar CEP
Nome/Rua
Digite o nome da rua...
Bairro
Digite o bairro...
Cidade
Digite a cidade...
Estado
Digite o estado...
País
Digite o país...
Cancelar
Salvar

- Criação da estrutura inicial de dashboard e logradouro realizada.
- Navegação entre as páginas funcionando e com o **binding** configurado nos controles.

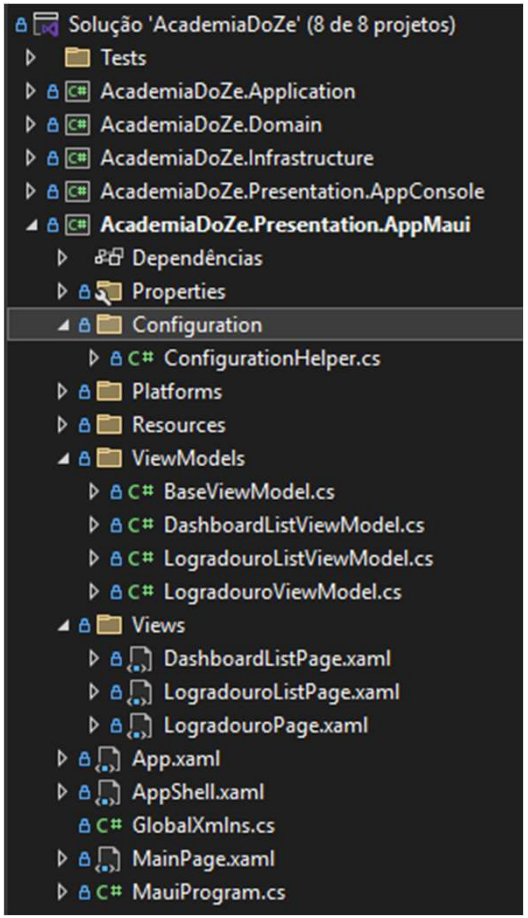
- Próximas etapas:
  - Criar a classe para realizar a **injeção de dependência** com os dados do repositório.
  - Incluir os dados para injeção de dependência na inicialização da aplicação.
  - Codificar a ViewModel de cada página para realizar a comunicação com o repositório e vincular os dados do repositório com os binding criados.
  - Concluir a implementação na **código-behind**.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Configuration**ConfigurationHelper.cs**

```
using AcademiaDoZe.Application.DependencyInjection;
using AcademiaDoZe.Application.Enums;

namespace AcademiaDoZe.Presentation.AppMaui.Configuration
{
    public static class ConfigurationHelper
    {
        public static void ConfigureServices(IServiceCollection services)
        {
            // dados conexão
            const string dbServer = "172.24.32.1";
            const string dbDatabase = "db_academia_do_ze";
            const string dbUser = "sa";
            const string dbPassword = "abcBolinhas12345";
            const string dbComplemento = "TrustServerCertificate=True;Encrypt=True;";

            // se for necessário indicar a porta, incluir junto em dbComplemento

            // Configurações de conexão
            const string connectionString = $"Server={dbServer};Database={dbDatabase};User Id={dbUser};Password={dbPassword};{dbComplemento}";
            const EAppDatabaseType databaseType = EAppDatabaseType.SqlServer;

            // Configura a fábrica de repositórios com a string de conexão e tipo de banco
            services.AddSingleton(new RepositoryConfig
            {
                ConnectionString = connectionString,
                DatabaseType = databaseType
            });

            // configura os serviços da camada de aplicação
            services.AddApplicationServices();
        }
    }
}
```

# AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui**MauiProgram.cs**

Incluir a configuração da injeção de dependência na inicialização da aplicação.




```
using AcademiaDoZe.Presentation.AppMaui.ViewModels;
using AcademiaDoZe.Presentation.AppMaui.Views;
using Microsoft.Extensions.Logging;
using AcademiaDoZe.Presentation.AppMaui.Configuration;

namespace AcademiaDoZe.Presentation.AppMaui
{
    public static class MauiProgram
    {
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
                });


            // Configurar serviços da aplicação e repositórios
            ConfigurationHelper.ConfigureServices(builder.Services);

            // Registrar ViewModels
            builder.Services.AddTransient<DashboardListViewModel>();
            builder.Services.AddTransient<LogradouroListViewModel>();
            builder.Services.AddTransient<LogradouroViewModel>();

            // Registrar Views
            builder.Services.AddTransient<DashboardListPage>();
            builder.Services.AddTransient<LogradouroListPage>();
            builder.Services.AddTransient<LogradouroPage>();


#if DEBUG
            builder.Logging.AddDebug();
#endif

            return builder.Build();
        }
    }
}
```

- Agora que já temos acesso ao repositório, precisamos **retornar** e concluir a implementação da **ViewModel** e **código-behind**, realizando a comunicação com a camada de aplicação.

- **Na sequência editaremos arquivos que já criamos e editamos antes de Dashboard e Logradouro.**
- **Estou passando a seguir os arquivos completos, então avalie onde é interessante trocar o arquivo todo, ou somente incluir as alterações.**

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**DashboardListViewModel.cs**

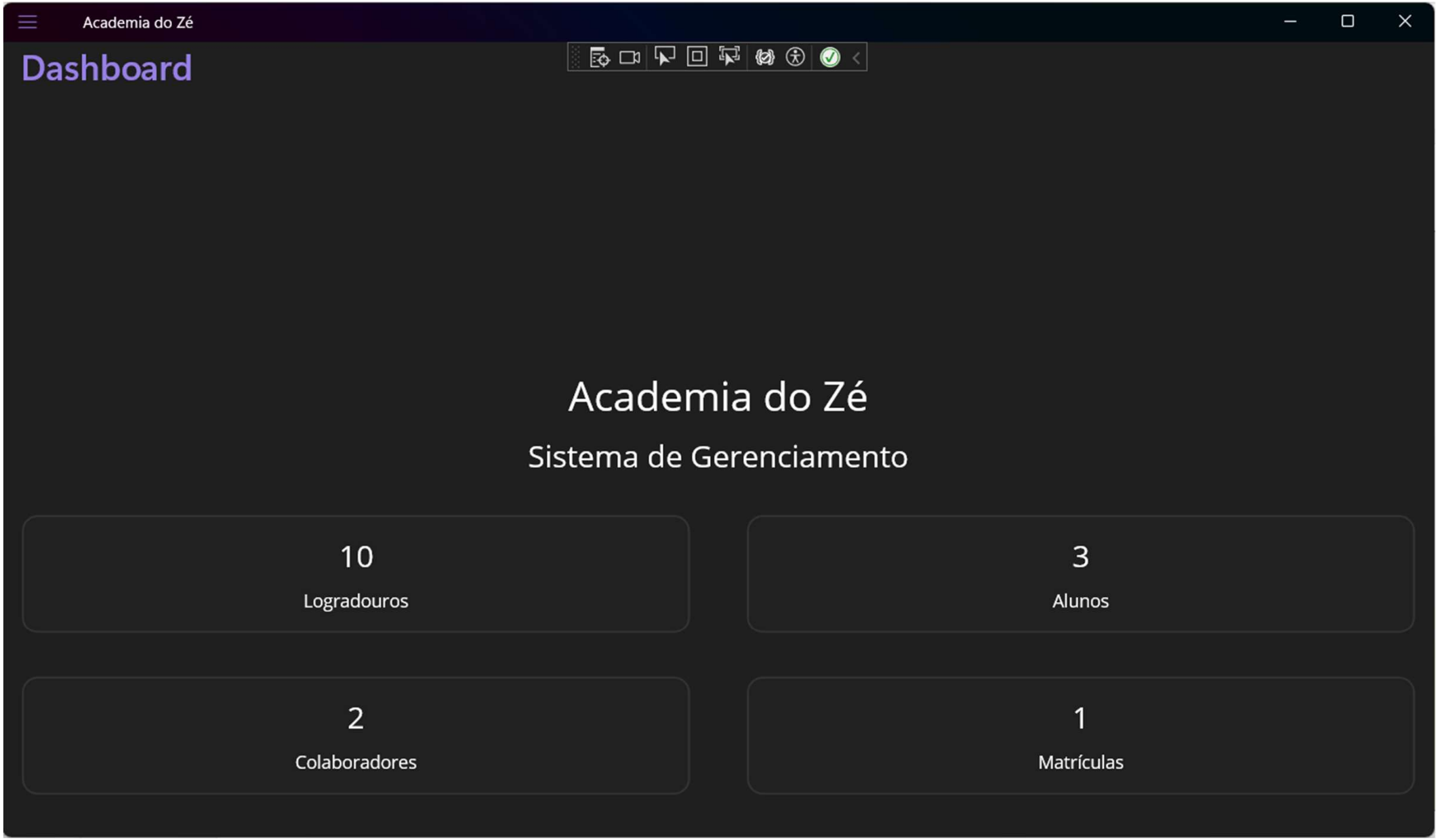
```
using AcademiaDoZe.Application.Interfaces; using CommunityToolkit.Mvvm.Input;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    public partial class DashboardListViewModel : BaseViewModel
    {
        private readonly ILogradouroService _logradouroService;
        private readonly IAlunoService _alunoService;
        private readonly IColaboradorService _colaboradorService;
        private readonly IMatriculaService _matriculaService;

        private int _totalLogradouros;
        public int TotalLogradouros {get => _totalLogradouros; set => SetProperty(ref _totalLogradouros, value); }

        private int _totalAlunos;
        public int TotalAlunos {get => _totalAlunos; set => SetProperty(ref _totalAlunos, value); }

        private int _totalColaboradores;
        public int TotalColaboradores {get => _totalColaboradores; set => SetProperty(ref _totalColaboradores, value); }

        private int _totalMatriculas;
        public int TotalMatriculas {get => _totalMatriculas; set => SetProperty(ref _totalMatriculas, value); }

        public DashboardListViewModel(ILogradouroService logradouroService, IAlunoService alunoService, IColaboradorService colaboradorService, IMatriculaService matriculaService)
        {
            _logradouroService = logradouroService;
            _alunoService = alunoService;
            _colaboradorService = colaboradorService;
            _matriculaService = matriculaService;
            Title = "Dashboard";
        }

        [RelayCommand]
        private async Task LoadDashboardDataAsync()
        {

        }

        [RelayCommand]
        private async Task NavigateToLogradourosAsync() => await Shell.Current.GoToAsync("//logradouros");

        [RelayCommand]
        private async Task NavigateToAlunosAsync() => await Shell.Current.GoToAsync("//alunos");

        [RelayCommand]
        private async Task NavigateToColaboradoresAsync() => await Shell.Current.GoToAsync("//colaboradores");

        [RelayCommand]
        private async Task NavigateToMatriculasAsync() => await Shell.Current.GoToAsync("//matriculas");
    }
}
```

```
        [RelayCommand]
        private async Task LoadDashboardDataAsync()
        {
            if (IsBusy)
                return;

            try
            {
                IsBusy = true;

                var logradourosTask = _logradouroService.ObterTodosAsync();
                var logradouros = new List<object>();
                try { logradouros = (await logradourosTask).ToList<object>(); }
                catch (Exception ex) { await Shell.Current.DisplayAlert("Erro", $"Erro ao carregar logradouros: {ex.Message}", "OK"); }
                TotalLogradouros = logradouros.Count;

                var alunosTask = _alunoService.ObterTodosAsync();
                var alunos = new List<object>();
                try { alunos = (await alunosTask).ToList<object>(); }
                catch (Exception ex) { await Shell.Current.DisplayAlert("Erro", $"Erro ao carregar alunos: {ex.Message}", "OK"); }
                TotalAlunos = alunos.Count;

                var colaboradoresTask = _colaboradorService.ObterTodosAsync();
                var colaboradores = new List<object>();
                try { colaboradores = (await colaboradoresTask).ToList<object>(); }
                catch (Exception ex) { await Shell.Current.DisplayAlert("Erro", $"Erro ao carregar colaboradores: {ex.Message}", "OK"); }
                TotalColaboradores = colaboradores.Count;

                var matriculasTask = _matriculaService.ObterTodasAsync();
                var matriculas = new List<object>();
                try { matriculas = (await matriculasTask).ToList<object>(); }
                catch (Exception ex) { await Shell.Current.DisplayAlert("Erro", $"Erro ao carregar matrículas: {ex.Message}", "OK"); }
                TotalMatriculas = matriculas.Count;
            }
            finally
            {
                IsBusy = false;
            }
        }
```

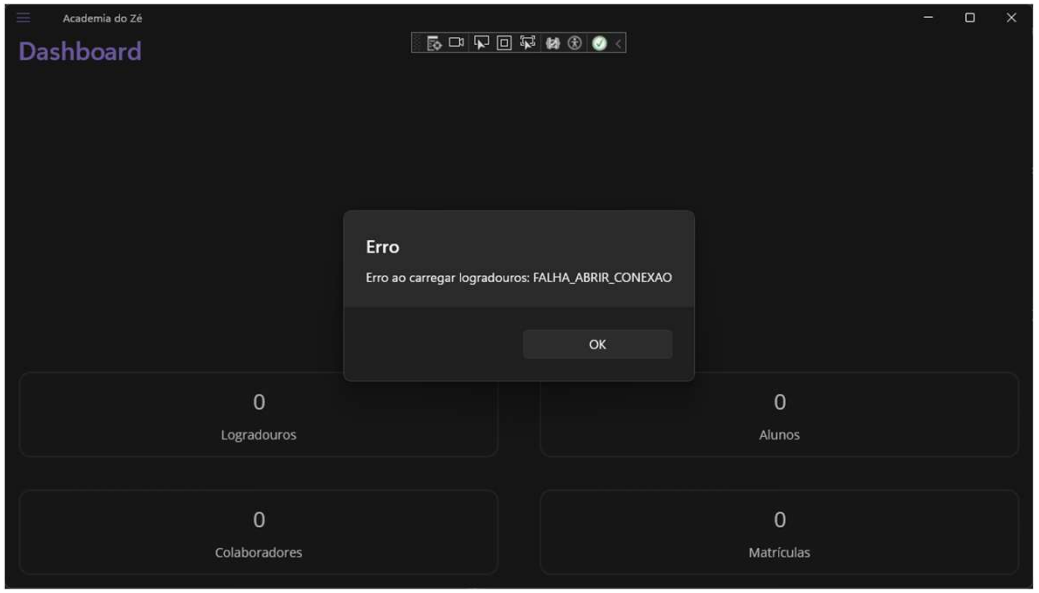
- Como a injeção de dependência já foi realizada, ou seja, nossa aplicação MAUI já sabe quem é o repositório, aqui basicamente só precisamos realizar a declaração dos objetos que vão referenciar os serviços, e realizar o acesso do serviço desejado, no nosso caso, ObterTodosAsync().

- Como já deixamos a configuração do Binding na View, agora bastou verificar a quantidade de cada item, e alimentar o valor nos atributos TotalLogradouros, TotalAlunos, TotalColaboradores e TotalMatriculas.

- Realize a execução da aplicação, e veja que a quantidade de cada item já está sendo mostrada na dashboard.


Academia do Zé
Dashboard
Academia do Zé
Sistema de Gerenciamento
10
Logradouros
3
Alunos
2
Colaboradores
1
Matrículas

- Se ocorrer erro, verifique:
  - Se a injeção de dependência foi realizada corretamente em MauiProgram.cs.
  - Se suas **credenciais do banco de dados** estão corretas.
  - No tratamento de exceção de DashboardListViewModel, estamos exibindo somente a propriedade **Message**, isso retorna a mensagem de erro que configuramos nas outras camadas.
  - Caso precise de mais detalhes, pode mandar mostrar o objeto da exceção.



- Aplicação executando e comunicação com a camada de aplicação funcionando.
- Na sequência vamos nos concentrar na implementação do que falta para que as quatro operações do CRUD de Logradouro funcionem.

## AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**LogradouroListViewModel.cs**

```
using AcademiaDoZe.Application.DTOs; using AcademiaDoZe.Application.Interfaces;
using CommunityToolkit.Mvvm.Input; using System.Collections.ObjectModel;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    public partial class LogradouroListViewModel : BaseViewModel
    {
        public ObservableCollection<string> FilterTypes { get; } = new() { "Cidade", "Id", "Cep" };

        private readonly ILogradouroService _logradouroService;

        private string _searchText = string.Empty;
        public string SearchText {get => _searchText; set => SetProperty(ref _searchText, value); }

        private string _selectedFilterType = "Cidade"; // Cidade, Id, Cep
        public string SelectedFilterType {get => _selectedFilterType; set => SetProperty(ref _selectedFilterType, value);
        }

        private ObservableCollection<LogradouroDTO> _logradouros = new();
        public ObservableCollection<LogradouroDTO> Logradouros {get => _logradouros; set => SetProperty(ref _logradouros, value); }

        private LogradouroDTO? _selectedLogradouro;
        public LogradouroDTO? SelectedLogradouro {get => _selectedLogradouro; set => SetProperty(ref _selectedLogradouro, value); }


        public LogradouroListViewModel(ILogradouroService logradouroService)
        {
            _logradouroService = logradouroService;
            Title = "Logradouros";
        }


       // Incluir os Métodos com [RelayCommand]



    }
}
```

```
        [RelayCommand]
        private async Task AddLogradouroAsync()
        {
            try
            {
                // GoToAsync é usado para navegar entre páginas no MAUI Shell.
                // logradouro é o nome da rota registrada no AppShell.xaml.cs
                await Shell.Current.GoToAsync("logradouro");
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao navegar para tela de cadastro: {ex.Message}", "OK");
            }
        }



        [RelayCommand]
        private async Task EditLogradouroAsync(LogradouroDTO logradouro)
        {
            try
            {
                if (logradouro == null)
                    return;

                await Shell.Current.GoToAsync($"logradouro?Id={logradouro.Id}");
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao navegar para tela de edição: {ex.Message}", "OK");
            }
        }



        [RelayCommand]
        private async Task RefreshAsync()
        {
            IsRefreshing = true;
            await LoadLogradourosAsync();
        }
```

```
        [RelayCommand]
        private async Task SearchLogradourosAsync()
        {
            if (IsBusy)
                return;

            try
            {
                IsBusy = true;

                // Limpa a lista atual
                await MainThread.InvokeOnMainThreadAsync(() =>
                {
                    Logradouros.Clear();
                });

                IEnumerable<LogradouroDTO> resultados = Enumerable.Empty<LogradouroDTO>();

                // Busca os logradouros de acordo com o filtro
                if (string.IsNullOrWhiteSpace(SearchText))
                {
                    resultados = await _logradouroService.ObterTodosAsync() ?? Enumerable.Empty<LogradouroDTO>();
                }
                else if (SelectedFilterType == "Cidade")
                {
                    resultados = await _logradouroService.ObterPorCidadeAsync(SearchText) ?? Enumerable.Empty<LogradouroDTO>();
                }
                else if (SelectedFilterType == "Id" && int.TryParse(SearchText, out int id))
                {
                    var logradouro = await _logradouroService.ObterPorIdAsync(id);
                    if (logradouro != null)
                        resultados = new[] { logradouro };
                }
                else if (SelectedFilterType == "Cep")
                {
                    var logradouro = await _logradouroService.ObterPorCepAsync(SearchText);
                    if (logradouro != null)
                        resultados = new[] { logradouro };
                }

                // Atualiza a coleção na thread principal
                await MainThread.InvokeOnMainThreadAsync(() =>
                {
                    foreach (var item in resultados)
                    {
                        Logradouros.Add(item);
                    }
                    OnPropertyChanged(nameof(Logradouros));
                });

            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao buscar logradouros: {ex.Message}", "OK");
            }
            finally
            {
                IsBusy = false;
            }
        }
```

```
        [RelayCommand]
        private async Task LoadLogradourosAsync()
        {
            if (IsBusy)
                return;

            try
            {
                IsBusy = true;

                // Limpa a lista atual antes de carregar novos dados
                await MainThread.InvokeOnMainThreadAsync(() =>
                {
                    Logradouros.Clear();
                    OnPropertyChanged(nameof(Logradouros));
                });

                var logradourosList = await _logradouroService.ObterTodosAsync();


                if (logradourosList != null)
                {
                    // Garantir que a atualização da UI aconteça na thread principal
                    await MainThread.InvokeOnMainThreadAsync(() =>
                    {
                        foreach (var logradouro in logradourosList)
                        {
                            Logradouros.Add(logradouro);
                        }

                        OnPropertyChanged(nameof(Logradouros));
                    });
                }
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao carregar logradouros: {ex.Message}", "OK");
            }
            finally
            {
                IsBusy = false;
                IsRefreshing = false;
            }
        }
```

```
        [RelayCommand]
        private async Task DeleteLogradouroAsync(LogradouroDTO logradouro)
        {
            if (logradouro == null)
                return;

            bool confirm = await Shell.Current.DisplayAlert(
                "Confirmar Exclusão",
                $"Deseja realmente excluir o logradouro {logradouro.Nome}?",
                "Sim", "Não");

            if (!confirm)
                return;

            try
            {
                IsBusy = true;
                bool success = await _logradouroService.RemoverAsync(logradouro.Id);

                if (success)
                {
                    Logradouros.Remove(logradouro);
                    await Shell.Current.DisplayAlert("Sucesso", "Logradouro excluído com sucesso!", "OK");
                }
                else
                {
                    await Shell.Current.DisplayAlert("Erro", "Não foi possível excluir o logradouro.", "OK");
                }
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao excluir logradouro: {ex.Message}", "OK");
            }
            finally
            {
                IsBusy = false;
            }
        }
```

## AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**LogradouroListPage.xaml.cs**

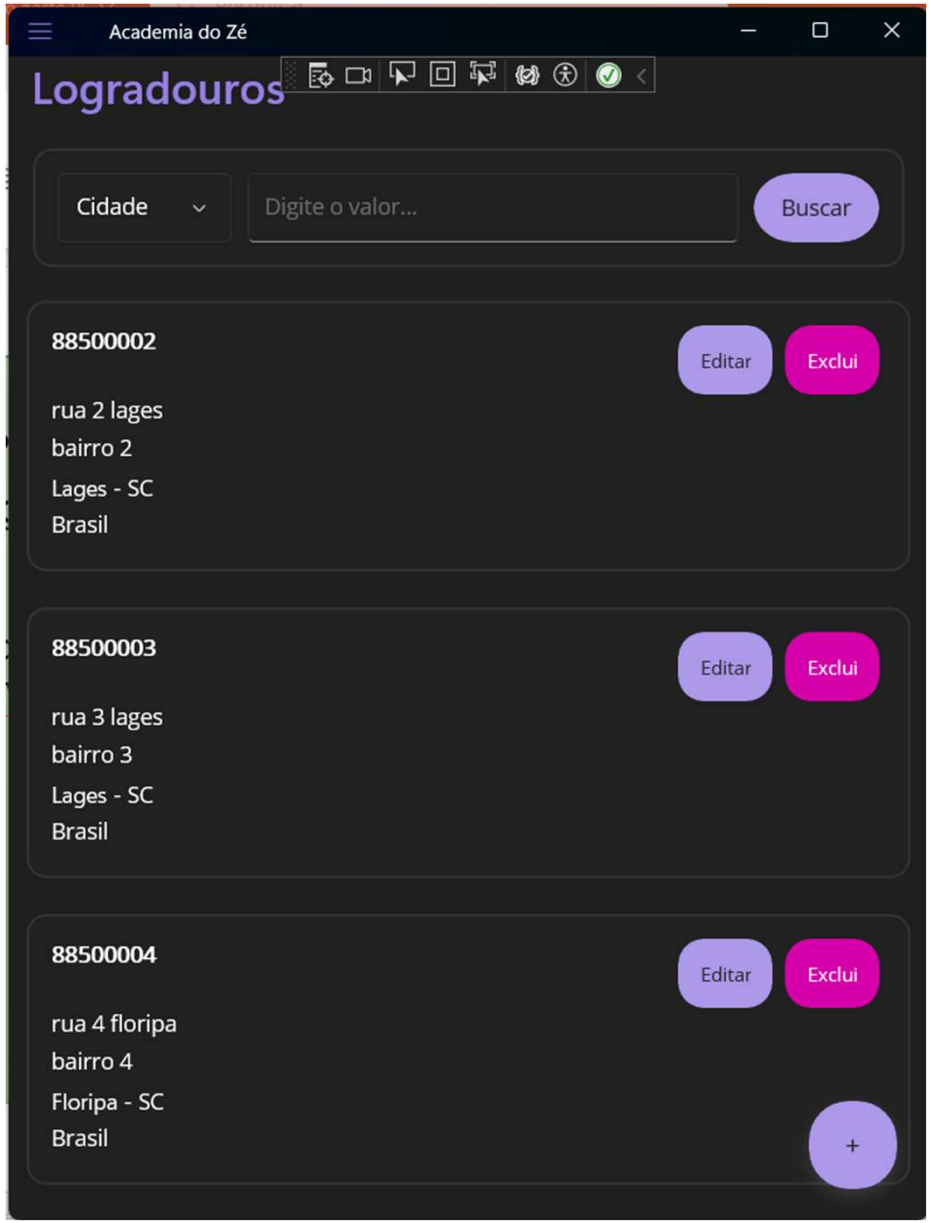
```
using AcademiaDoZe.Application.DTOs;
using AcademiaDoZe.Presentation.AppMaui.ViewModels;

namespace AcademiaDoZe.Presentation.AppMaui.Views;

public partial class LogradouroListPage : ContentPage
{
    public LogradouroListPage(LogradouroListViewModel viewModel)
    {
        InitializeComponent();
        BindingContext = viewModel;
    }

    protected override async void OnAppearing()
    {
        base.OnAppearing();

        if (BindingContext is LogradouroListViewModel viewModel)
        {
            await viewModel.LoadLogradourosCommand.ExecuteAsync(null);
        }
    }

    private async void OnEditButtonClicked(object sender, EventArgs e)
    {
        try
        {
            if (sender is Button button && button.BindingContext is LogradouroDTO logradouro && BindingContext is LogradouroListViewModel viewModel)
            {
                await viewModel.EditLogradouroCommand.ExecuteAsync(logradouro);
            }
        } catch (Exception ex) {await DisplayAlert("Erro", $"Erro ao editar logradouro: {ex.Message}", "OK"); }
    }

    private async void OnDeleteButtonClicked(object sender, EventArgs e)
    {

        try
        {
            if (sender is Button button && button.BindingContext is LogradouroDTO logradouro && BindingContext is LogradouroListViewModel viewModel)
            {
                await viewModel.DeleteLogradouroCommand.ExecuteAsync(logradouro);
            }
        } catch (Exception ex) {await DisplayAlert("Erro", $"Erro ao excluir logradouro: {ex.Message}", "OK"); }

    }
}
```

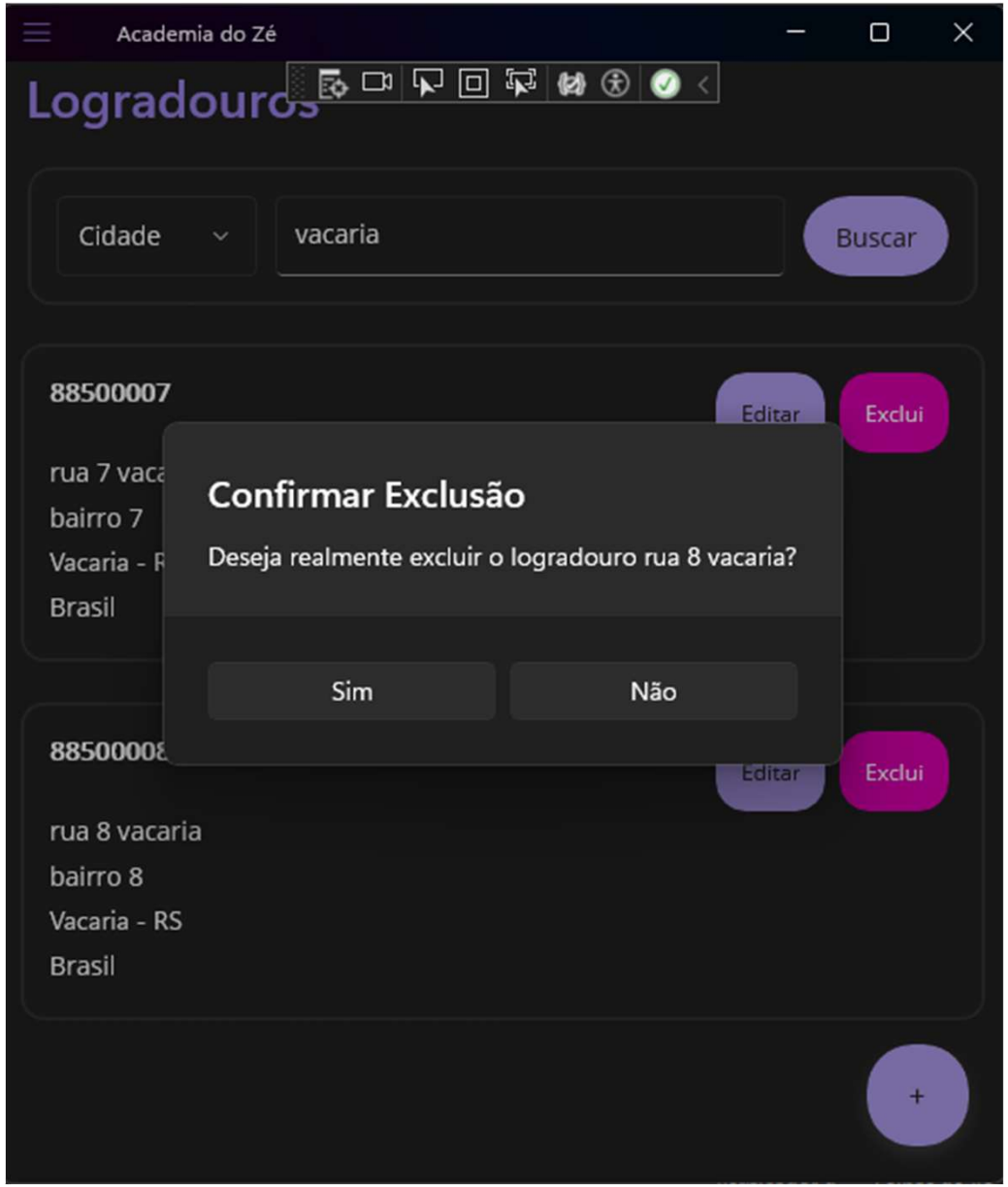
- No code-behind, concluímos a implementação dos métodos iniciados anteriormente.

- OnAppearing
  - Listar os dados ao abrir

- Botão de editar
  - Carregar página para edição

- Botão de excluir
  - Chamar operação para excluir


Academia do Zé
Logradouros
Cidade
Digite o valor...
Buscar
88500002
Editar
Exclui
rua 2 lages
bairro 2
Lages - SC
Brasil
88500003
Editar
Exclui
rua 3 lages
bairro 3
Lages - SC
Brasil
88500004
Editar
Exclui
rua 4 floripa
bairro 4
Floripa - SC
Brasil
+


Academia do Zé
Logradouros
Cidade
vacaria
Buscar
88500007
Editar
Exclui
rua 7 vaca
bairro 7
Vacaria - R
Brasil
Confirmar Exclusão
Deseja realmente excluir o logradouro rua 8 vacaria?
Sim
Não
88500008
Editar
Exclui
rua 8 vacaria
bairro 8
Vacaria - RS
Brasil
+

- Ao executar, já conseguimos acessar a página para **listar** os **Logradouros** existentes.
- Nela, as operações de **Buscar** e **Excluir** já estão funcionais.
- As operações de **Editar** e **Novo**, ainda dependem da conclusão da implementação de LogradouroViewModel.

AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\ViewModels**LogradouroViewModel.cs**

```
using AcademiaDoZe.Application.DTOs; using AcademiaDoZe.Application.Interfaces; using CommunityToolkit.Mvvm.Input;

namespace AcademiaDoZe.Presentation.AppMaui.ViewModels
{
    [QueryProperty(nameof(LogradouroId), "Id")]
    public partial class LogradouroViewModel : BaseViewModel
    {
        private readonly ILogradouroService _logradouroService;

        private LogradouroDTO _logradouro = new()
        {
            Cep = string.Empty, Nome = string.Empty, Bairro = string.Empty, Cidade = string.Empty, Estado = string.Empty, Pais = string.Empty
        };

        public LogradouroDTO Logradouro
        {
            get => _logradouro;
            set => SetProperty(ref _logradouro, value);

        }

        private int _logradouroId;
        public int LogradouroId
        {
            get => _logradouroId;
            set
            {
                if (SetProperty(ref _logradouroId, value))
                {
                    // Quando o LogradouroId é alterado, inicializa os dados
                    Task.Run(InitializeAsync);
                }
            }
        }

        private bool _isEditMode;
        public bool IsEditMode
        {
            get => _isEditMode;
            set => SetProperty(ref _isEditMode, value);
        }

        public LogradouroViewModel(ILogradouroService logradouroService)
        {
            _logradouroService = logradouroService;
            Title = "Detalhes do Logradouro";
        }


        // Incluir os Métodos aqui


    }
}
```

```
        public async Task InitializeAsync()
        {
            if (LogradouroId > 0)
            {
                IsEditMode = true;
                Title = "Editar Logradouro";
                await LoadLogradouroAsync();
            }
            else
            {
                IsEditMode = false;
                Title = "Novo Logradouro";
            }
        }


        [RelayCommand]
        private async Task CancelAsync()
        {
            await Shell.Current.GoToAsync("..");
        }

        [RelayCommand]
        private async Task LoadLogradouroAsync()
        {
            if (LogradouroId <= 0)
                return;

            try
            {
                IsBusy = true;
                var logradouroData = await _logradouroService.ObterPorIdAsync(LogradouroId);
                if (logradouroData != null)
                {
                    Logradouro = logradouroData;
                }
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao carregar logradouro: {ex.Message}", "OK");
            }
            finally
            {
                IsBusy = false;
            }
        }
```

```
        [RelayCommand]
        private async Task SearchByCepAsync()
        {
            if (string.IsNullOrWhiteSpace(Logradouro.Cep))
                return;

            try
            {
                IsBusy = true;
                var logradouroData = await _logradouroService.ObterPorCepAsync(Logradouro.Cep);
                if (logradouroData != null)
                {
                    Logradouro = logradouroData;
                    IsEditMode = true;
                    await Shell.Current.DisplayAlert("Aviso", "CEP já cadastrado! Dados carregados para edição.", "OK");
                }
                else
                {
                    await Shell.Current.DisplayAlert("Aviso", "CEP não encontrado.", "OK");
                }
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao buscar CEP: {ex.Message}", "OK");
            }
            finally
            {
                IsBusy = false;
            }
        }
```

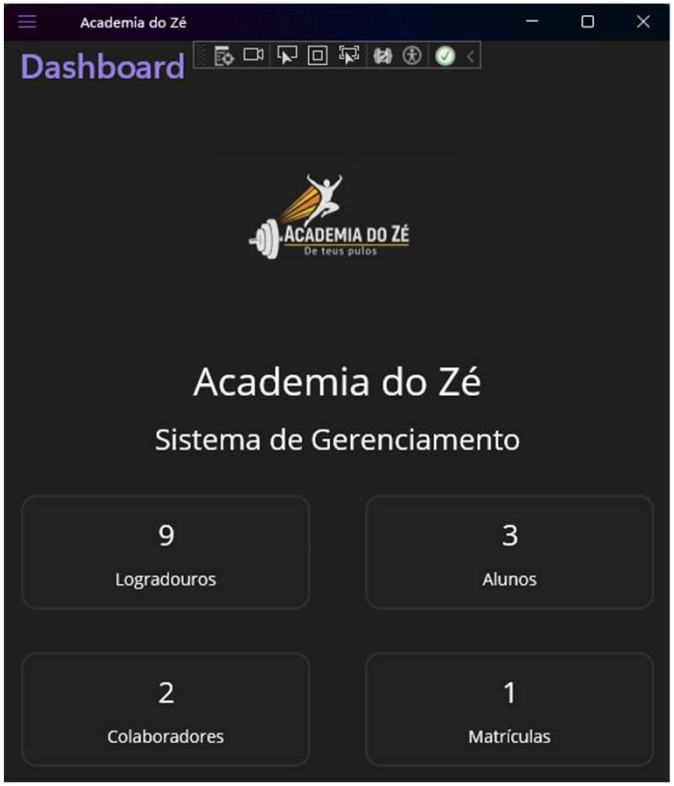
```
        [RelayCommand]
        private async Task SaveLogradouroAsync()
        {
            if (IsBusy)
                return;

            if (!ValidateLogradouro(Logradouro))
                return;

            try
            {
                IsBusy = true;

                if (IsEditMode)
                {
                    await _logradouroService.AtualizarAsync(Logradouro);
                    await Shell.Current.DisplayAlert("Sucesso", "Logradouro atualizado com sucesso!", "OK");
                }
                else
                {
                    await _logradouroService.AdicionarAsync(Logradouro);
                    await Shell.Current.DisplayAlert("Sucesso", "Logradouro criado com sucesso!", "OK");
                }

                await Shell.Current.GoToAsync("..");
            }
            catch (Exception ex)
            {
                await Shell.Current.DisplayAlert("Erro", $"Erro ao salvar logradouro: {ex.Message}", "OK");
            }
            finally
            {
                IsBusy = false;
            }
        }
```

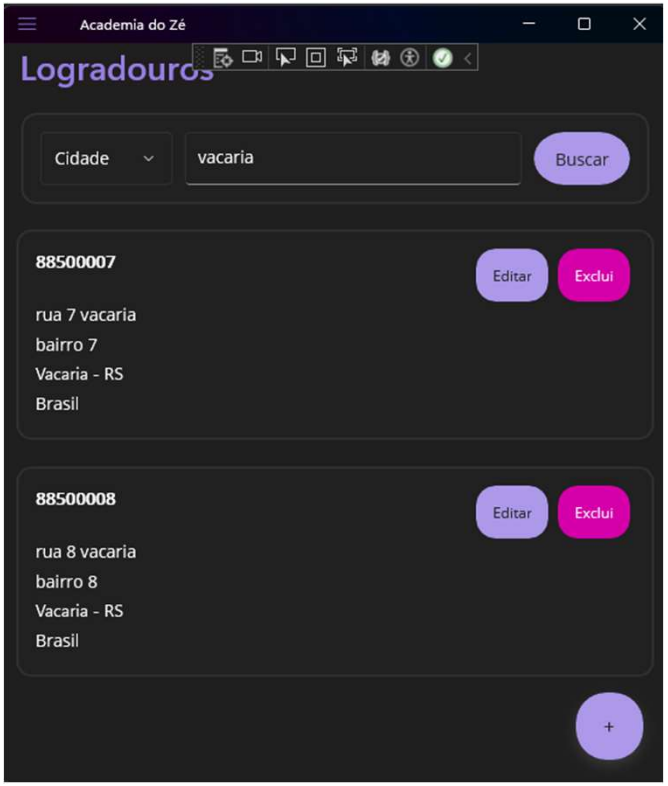
```
        private static bool ValidateLogradouro(LogradouroDTO logradouro)
        {
            const string validationTitle = "Validação";

            if (string.IsNullOrWhiteSpace(logradouro.Cep))
            {
                Shell.Current.DisplayAlert(validationTitle, "CEP é obrigatório.", "OK");
                return false;
            }

            if (string.IsNullOrWhiteSpace(logradouro.Nome))
            {
                Shell.Current.DisplayAlert(validationTitle, "Nome é obrigatório.", "OK");
                return false;
            }

            if (string.IsNullOrWhiteSpace(logradouro.Bairro))
            {
                Shell.Current.DisplayAlert(validationTitle, "Bairro é obrigatório.", "OK");
                return false;
            }

            if (string.IsNullOrWhiteSpace(logradouro.Cidade))
            {
                Shell.Current.DisplayAlert(validationTitle, "Cidade é obrigatória.", "OK");
                return false;
            }

            if (string.IsNullOrWhiteSpace(logradouro.Estado))
            {
                Shell.Current.DisplayAlert(validationTitle, "Estado é obrigatório.", "OK");
                return false;
            }

            if (string.IsNullOrWhiteSpace(logradouro.Pais))
            {
                Shell.Current.DisplayAlert(validationTitle, "País é obrigatório.", "OK");
                return false;
            }

            return true;
        }
```

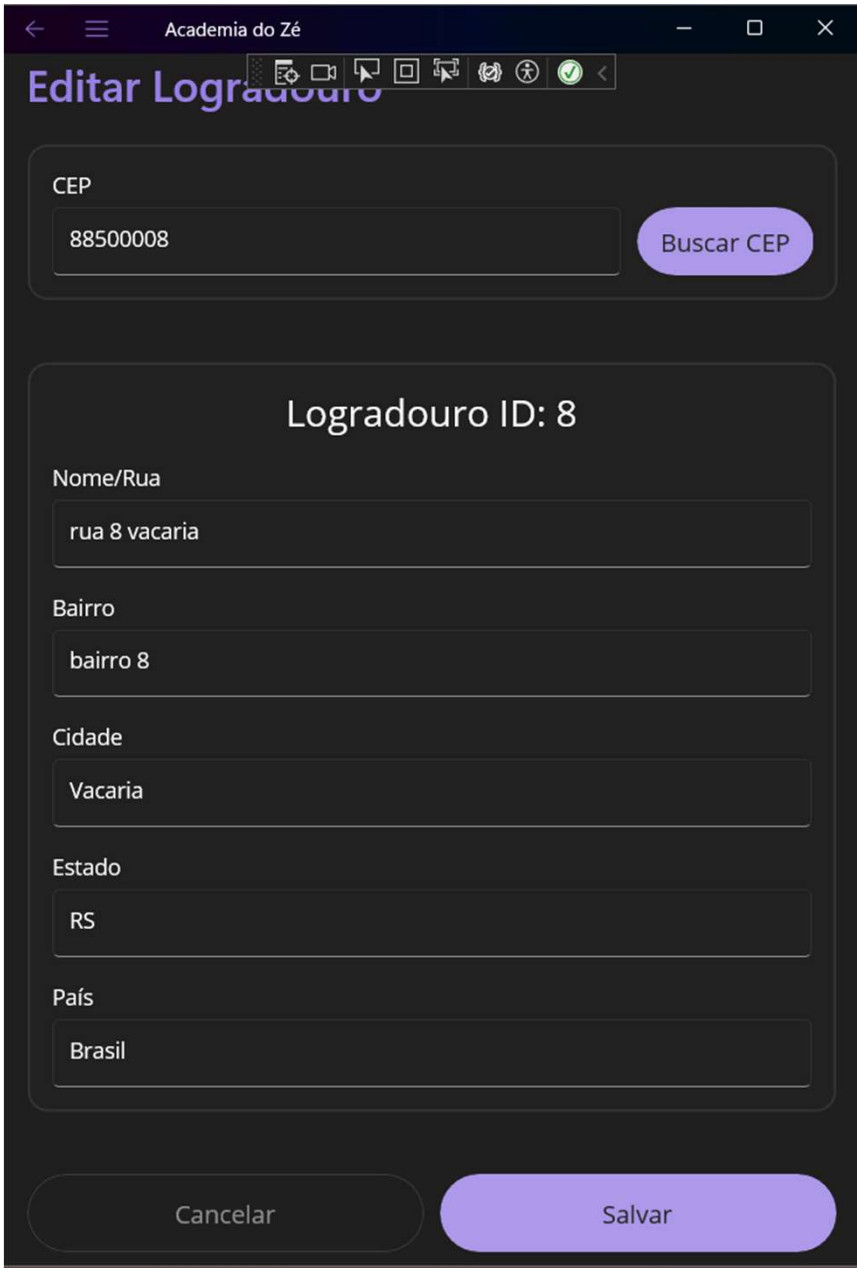
AcademiaDoZe\AcademiaDoZe.Presentation.AppMaui\Views**LogradouroPage.xaml.cs**

```
using AcademiaDoZe.Presentation.AppMaui.ViewModels;

namespace AcademiaDoZe.Presentation.AppMaui.Views;

public partial class LogradouroPage : ContentPage
{
    public LogradouroPage(LogradouroViewModel viewModel)
    {
        InitializeComponent();
        BindingContext = viewModel;
    }

    protected override async void OnAppearing()
    {
        base.OnAppearing();

        if (BindingContext is LogradouroViewModel viewModel)
        {
            await viewModel.InitializeAsync();
        }
    }
}
```

- Funcionalidades **Dashboard** e **Logradouro** implementadas e funcionais.
- Execute a aplicação e valide se tudo está funcionando conforme o esperado.
- O padrão gráfico e o estilo aplicado são somente base de exemplificação, com toda certeza vocês conseguem implementar algo muito melhor do que montei e apresentei no exemplo, então inicialmente foquem em deixar tudo funcional, e na sequência usem suas habilidades para deixar tudo mais bonito e funcional.
- Aproveitem e realizem testes no **Windows**, **Android** e quem tiver dispositivo Apple, aproveita e testa também.


Academia do Zé
Dashboard
ACADEMIA DO ZÉ
De teus pulos
Academia do Zé
Sistema de Gerenciamento
9
Logradouros
3
Alunos
2
Colaboradores
1
Matrículas


Academia do Zé
Logradouros
Cidade
vacaria
Buscar
88500007
Editar
Exclui
rua 7 vacaria
bairro 7
Vacaria - RS
Brasil
88500008
Editar
Exclui
rua 8 vacaria
bairro 8
Vacaria - RS
Brasil
+


Academia do Zé
Editar Logradouro
CEP
88500008
Buscar CEP
Logradouro ID: 8
Nome/Rua
rua 8 vacaria
Bairro
bairro 8
Cidade
Vacaria
Estado
RS
País
Brasil
Cancelar
Salvar

- Temas **Light** e **Dark**
  - O .NET MAUI oferece **suporte nativo a temas claro e escuro**, permitindo que sua aplicação adapte automaticamente cores, imagens e estilos conforme a preferência do sistema operacional do usuário.
  - Ao criar ou editar seus estilos, é importante ter cuidado com as configurações de **Light** e **Dark**, garantindo desta forma que o resultado esperado será alcançado nos dois temas.
  - O MAUI **detecta automaticamente** se o sistema está usando tema claro ou escuro e aplica os recursos correspondentes, sem necessidade de código adicional, tudo feito com base nas configurações dos seus arquivos de estilo.
  - Caso opte por não utilizar o recurso automaticamente, o que não faz muito sentido, terá que forçar o seu tema escolhido, indicando ele no arquivo **App.xaml.cs**:
    - `UserAppTheme = AppTheme.Light; // ou AppTheme.Dark`

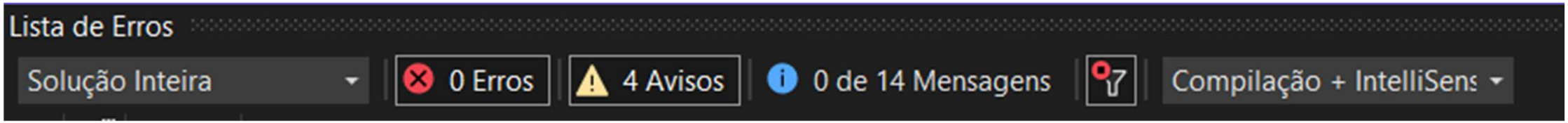
- **AppThemeBinding**
  - Para alternar recursos (cores, imagens, etc.) conforme o tema, utilize o AppThemeBinding no XAML.
  - No exemplo abaixo, estou alternando a imagem que incluímos na Dashboard conforme o tema escolhido.

```
<Image HeightRequest="200" WidthRequest="200" HorizontalOptions="Center" VerticalOptions="Start">
    <Image.Source>
        <AppThemeBinding Light="academiadoze_claro.png" Dark="academiadoze_escuro.png" />
    </Image.Source>
</Image>
```


Alterne a configuração de tema do sistema operacional onde sua aplicação está executando, e veja o resultado.

# Erros / Avisos / Mensagens

- Analise, avalie as dicas da IDE, e resolva todos os:
  - Erros
  - Avisos
  - Mensagens




- Acho que para iniciar já está bom rs.
- **Dashboard** e **Logradouro** implementados e funcionais.
- Em um momento oportuno, retornaremos para melhorar nossa injeção de dependência, ou seja, criaremos uma tela de configuração e também uma validação na inicialização da aplicação, mas isso é coisa futura, no momento os dados fixos no código já nos atendem.

MAIU

De teus pulos

Agora é com você

# Laboratório prático

## Parcial de nota avaliação 02

5 parciais de nota

- Com base na teorização e exemplificação, crie seu projeto .NET MAUI, e nele implemente a camada de apresentação da nossa solução.

- Implemente as seguintes funcionalidades:
  - **Dashboard**
  - **CRUD completo de Logradouro**

- Está tudo cuidadosamente exemplificado e detalhado no material de apoio.
- Tenha o cuidado de aplicar a sua identidade visual no projeto, ou seja, visualmente não espero aplicações idênticas ao que foi aplicado nos exemplos e teorizações!


- Toda sua solução deve estar em um **repositório GIT** de sua propriedade.
- Na **Dashboard**, obrigatoriamente deve ter uma imagem com a logo da sua aplicação, e **na imagem obrigatoriamente deve conter seu nome**.
- O não cumprimento dos requisitos, automaticamente ZERAM o parcial!

- **Data de entrega**
  - **Turma 01: 08/09**
  - **Turma 02: 11/09**


- **Data de entrega da recuperação**
  - **Turma 01: 15/09 18:40**
  - **Turma 02: 18/09 18:40**

- Deve ser postado na atividade correspondente do Classroom o seguinte:
  - Comentário, ou arquivo texto, com o **endereço do GIT**.
  - Arquivo zip com a **release criada no GitHub** até este ponto.
  - Print da estrutura de diretórios e arquivos do seu projeto.
  - Breve vídeo demonstrando a execução do projeto, devendo aparecer:
    - Funcionamento das opções de menu.
    - Dashboard contendo a **imagem com seu nome**, o total de itens, e o acesso a funcionalidade Logradouro.
    - Logradouro, devendo demostrar:
      - Cadastro de um novo Logradouro, devendo obrigatoriamente aparecer seu nome completo no nome do logradouro.
      - Pagina de listar com dados armazenados no banco, e filtrando o logradouro cadastrado.
      - Editar o logradouro cadastrado, mudando o bairro para Abc Bolinhas.
      - Excluir o logradouro cadastrado.

  - Use alguma ferramenta para gravar a tela de sue computador, NÃO SERÁ ACEITO VÍDEO COM QUALIDADE RUIM.
  - Resumindo, deve ser postado no Classroom um total de 4 arquivos, sendo um zip do projeto todo, um arquivo de vídeo, uma imagem e um texto.