

---

# AGENDA 10

---

**UTILIZANDO  
MICROSOFT MAUI  
PARA  
DESENVOLVIMENTO  
MOBILE**

GEEaD - Grupo de Estudos de Educação a Distância

Centro de Educação Tecnológica Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO  
EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO CURSO TÉCNICO  
EM DESENVOLVIMENTO DE SISTEMAS PROGRAMAÇÃO MOBILE I

**Expediente**

Autores:

*GUILHERME HENRIQUE GIROLI*

*TIAGO ANTONIO DA SILVA*

*Revisão Técnica:*

*Kelly Cristiane de Oliveira Dal Pozzo*

São Paulo – SP, 2024

O C# é uma linguagem de programação moderna e de alto nível, desenvolvida pela Microsoft, amplamente utilizada no desenvolvimento de uma variedade de aplicativos, desde Desktop até dispositivos móveis. No contexto do desenvolvimento Cross-Platform, o C# desempenha um papel essencial, especialmente com a chegada de frameworks como .NET Core e MAUI.

Veja algumas das principais maneiras pelas quais C# é utilizada no desenvolvimento Cross-Platform:

- ✓ **.NET Core e .NET 8:** O .NET Core é um Framework de código aberto e multiplataforma, introduzido pela Microsoft para o desenvolvimento de aplicativos. Com ele, é possível escrever aplicativos em C# que podem ser executados em várias plataformas, como Windows, MacOS e Linux. A partir do .NET 5 e com o .NET 6, a Microsoft unificou o .NET Core, o .NET Framework e o Xamarin em um único ecossistema, facilitando ainda mais o desenvolvimento Cross-Platform com C#.
- ✓ **Xamarin:** O Xamarin é uma plataforma que permite o desenvolvimento de aplicativos móveis nativos para iOS, Android e Windows usando C# e o Framework .NET. Com Xamarin, os desenvolvedores podem compartilhar uma parte significativa do código entre diferentes plataformas, reduzindo tempo e custos de desenvolvimento. Além disso, o Xamarin.Forms permite a criação de interfaces de usuário compartilhadas, simplificando o processo de desenvolvimento cross-platform ainda mais.
- ✓ **MAUI (Multi-platform App UI):** O MAUI é uma evolução do Xamarin.Forms, também anunciado pela Microsoft. Ele permite o desenvolvimento de aplicativos nativos para iOS, Android, Windows e macOS a partir de uma única base de código escrita em C#. Com o MAUI, os desenvolvedores podem criar interfaces de usuário flexíveis e adaptáveis, que se ajustam automaticamente a diferentes dispositivos e plataformas. O MAUI está totalmente integrado ao .NET 8, fornecendo acesso a todas as funcionalidades e APIs do .NET, tornando o desenvolvimento cross-platform mais robusto e eficiente.
- ✓ **ASP.NET Core:** Além de aplicativos móveis, C# também é amplamente utilizado no desenvolvimento de aplicativos web. O ASP.NET Core permite criar aplicativos web modernos e escaláveis usando C# e o Framework .NET. Esses aplicativos podem ser implantados em diversos servidores e plataformas, fazendo do ASP.NET Core uma escolha robusta para o desenvolvimento Cross-Platform na web.

No geral, C# é uma linguagem de programação poderosa e versátil, altamente utilizada no desenvolvimento Cross-Platform. Com o suporte de Frameworks como .NET Core, Xamarin e MAUI, os desenvolvedores podem criar aplicativos nativos para diversas plataformas, reduzindo o esforço de desenvolvimento e maximizando a reutilização de código.

## Layouts, Entrada, Processamento e Saída de Dados

Para que você consiga digitar um texto e enviar para seu amigo por meio de um APP de troca de mensagens, vários **padrões** e **programações** foram estabelecidas para que o resultado seja alcançado por você, usuário, e seu amigo receba o texto digitado!

O aplicativo é constituído, basicamente, de uma tela e de uma classe responsável pelas ações realizadas por ela. Nesta agenda, vamos aprender como estabelecer e criar o layout do aplicativo, desenvolvendo as telas e seus componentes. Vamos trabalhar com cores e definir os padrões de desenvolvimento através do layouts.



MERGULHANDO NO  
TEMA

## Layout e XAML

O layout de uma aplicação é algo muito importante, é ele o responsável pela não satisfação do usuário com um determinado aplicativo. Não adianta o aplicativo conter as melhores codificações e desempenho, se o usuário não aprovar o seu layout e usabilidade.

O XAML (Extensible Application Markup Language) é uma linguagem de marcação declarativa usada para criar interfaces de usuário no .NET MAUI. Com o XAML, é possível definir de maneira intuitiva a estrutura e o comportamento dos elementos da interface, separando a lógica de negócios da apresentação visual.



No contexto do .NET MAUI, o XAML é essencial para a criação de layouts de aplicativos que se adaptam a diferentes tamanhos de tela e dispositivos. Ele oferece uma forma poderosa e flexível de definir a aparência e o comportamento dos controles e elementos da interface do usuário, mantendo a simplicidade e legibilidade do código. Com uma sintaxe familiar e suporte integrado para vinculação de dados (data binding), o XAML no .NET MAUI acelera o desenvolvimento de aplicativos multiplataforma.

Para trabalhar com o Layout é necessário compreender como ele funciona. Onde a interface, ou seja, a aparência do seu aplicativo é um fator determinante para o seu sucesso. Então é aconselhável planejar o design antes de elaborar toda a programação. Isto não é perda de tempo e sim um investimento futuro.

## Criando o Primeiro Projeto

Para praticar a criação de layouts utilizando o .NET MAUI, vamos desenvolver um aplicativo simples para calcular o consumo de combustível. Esse aplicativo permitirá que o usuário insira informações como a distância percorrida

e o consumo médio do veículo, fornecendo rapidamente o valor estimado de combustível necessário. Esse projeto será uma excelente oportunidade para aplicar conceitos de design de interface e lógica de negócios, utilizando layouts como o VerticalStackLayout para organizar os elementos da tela de forma clara e funcional.

Após iniciar VisualStudio 2022, selecione na tela inicial a opção **Criar um Projeto**:

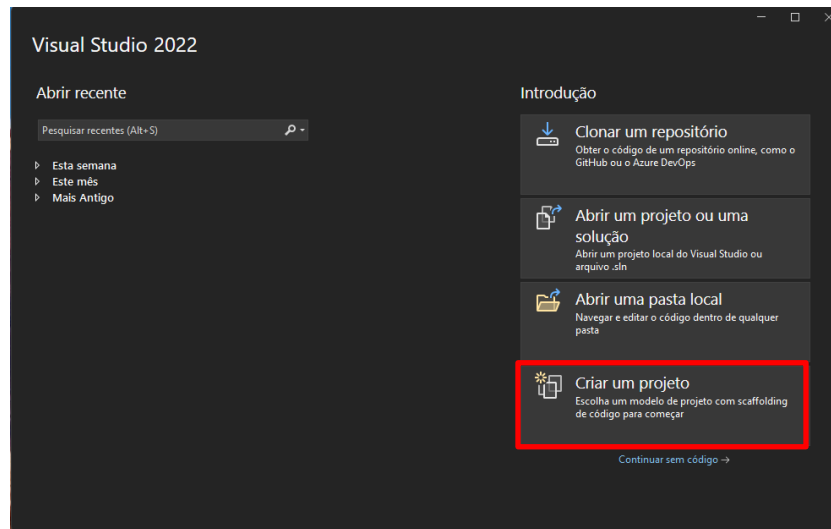


Figura 2 – Tela Inicial VisualStudio 2022.

Em seguida, busque a opção Aplicativo .NET MAUI e a selecione, para prosseguir clique em Próximo:

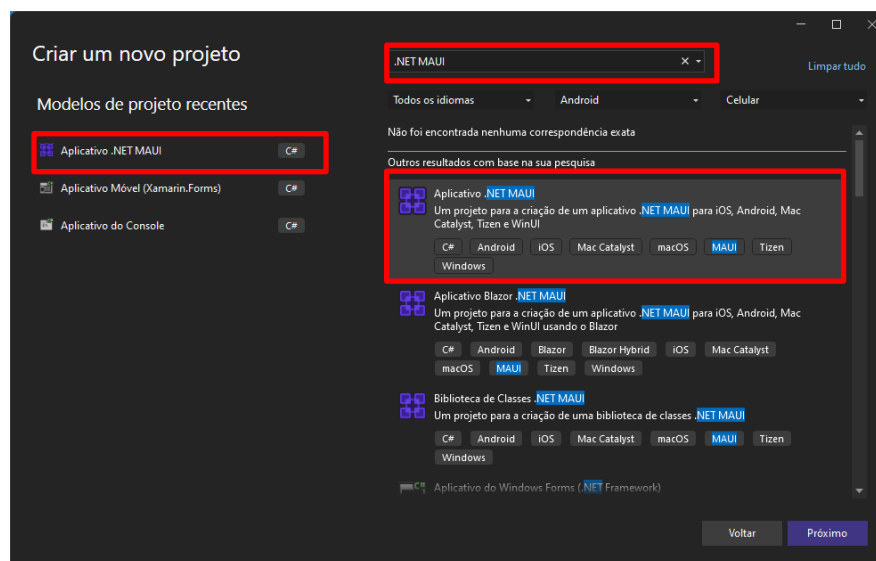
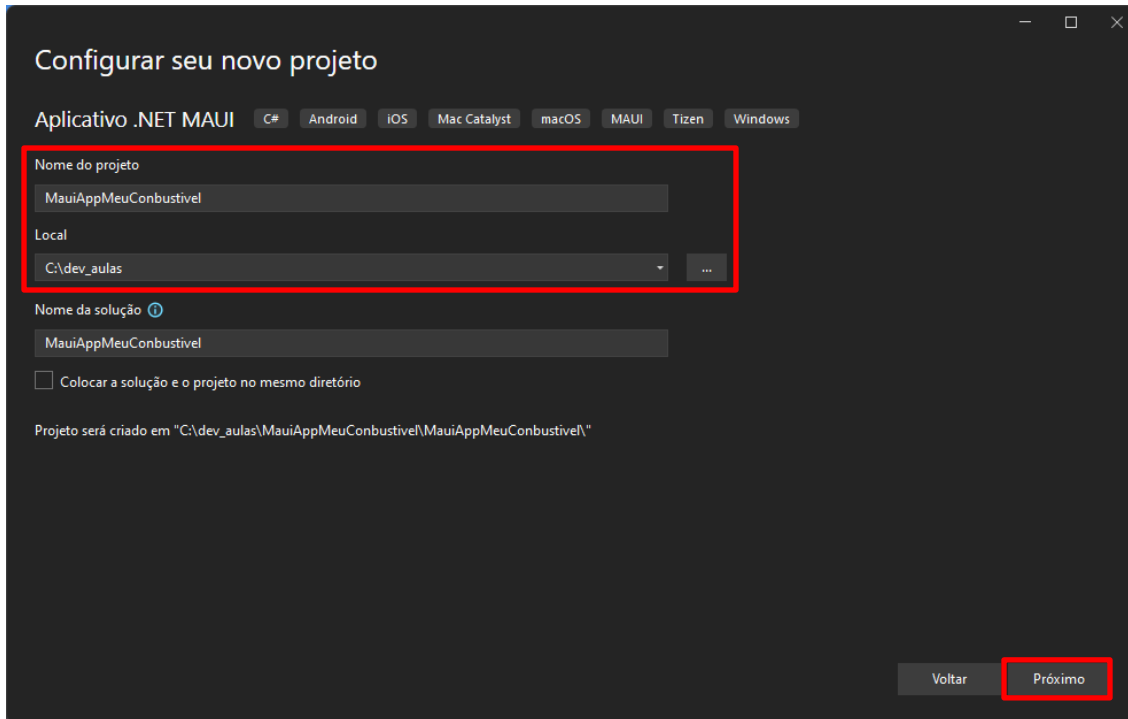


Figura 3 – Tela Criar um novo projeto, VisualStudio 2022.

Adicione um nome para o projeto, determine um local para armazenamento, clique em Próximo:



Configurar seu novo projeto

Aplicativo .NET MAUI C# Android iOS Mac Catalyst macOS MAUI Tizen Windows

Nome do projeto  
MauiAppMeuCombustivel

Local  
C:\dev\_aulas

Nome da solução ⓘ  
MauiAppMeuCombustivel

☐ Colocar a solução e o projeto no mesmo diretório

Projeto será criado em "C:\dev\_aulas\MauiAppMeuCombustivel\MauiAppMeuCombustivel\"

Voltar Próximo

Figura 4 – Configurar novo projeto, VisualStudio 2022.



**Não esqueça das regras para os nomes dos projetos. Os projetos não devem conter caracteres especiais, espaçamento em branco ou iniciar com números.**

Deixe selecionada a opção .NET 8.0, clique em Criar:

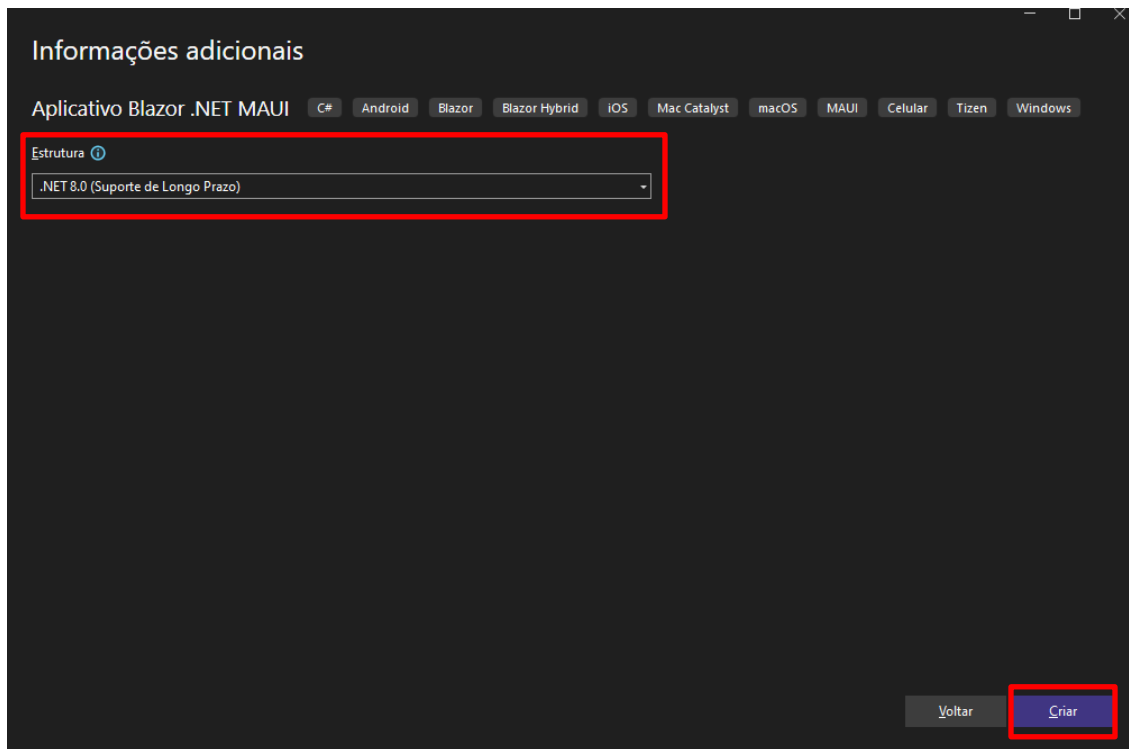


Figura 5 – Informações adicionais, VisualStudio 2022.

Do lado direito da tela inicial é possível visualizar os arquivos que vamos editar no projeto.

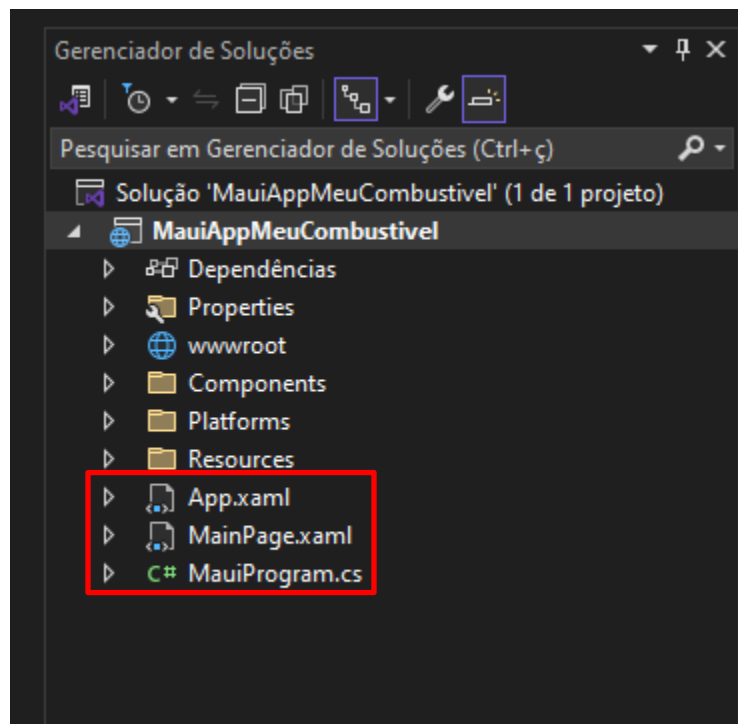


Figura 6 – Área Gerenciador de Soluções, VisualStudio 2022.

Assista o vídeo que apresenta a criação do primeiro aplicativo:

[https://youtu.be/gpQ9pioNHOC?si=4DDV0vP05jrt\\_XBa](https://youtu.be/gpQ9pioNHOC?si=4DDV0vP05jrt_XBa)

Agora que já criamos o projeto vamos conhecer um pouco mais sobre os códigos de configuração.

## Composição de Layout com VerticalStackLayout

O VerticalStackLayout é um layout usado no .NET MAUI para organizar elementos da interface do usuário de forma vertical. Ele dispõe os controles ou outros elementos em uma pilha vertical, um abaixo do outro, garantindo que cada elemento seja exibido na ordem em que foi adicionado.

Esse tipo de layout é ideal quando se deseja organizar componentes como botões, rótulos, campos de entrada de texto ou imagens em uma sequência vertical.

Ao combinar o .NET MAUI com o VerticalStackLayout, é possível criar aplicativos elegantes e funcionais de maneira eficiente. A capacidade de compartilhar tanto a lógica de negócios quanto a interface do usuário entre diferentes plataformas permite reduzir o tempo de desenvolvimento, ao mesmo tempo em que se alcança uma ampla base de usuários. Isso garante coesão visual e funcional em todas as plataformas, representando um avanço significativo no desenvolvimento de aplicativos multiplataforma, tornando-o mais eficaz e poderoso.

Você pode configurar um VerticalStackLayout de várias maneiras para controlar o comportamento e a aparência dos elementos na pilha vertical. Algumas opções de configuração incluem:

1. **Spacing (Espaçamento):** Você pode definir o espaçamento entre os elementos filhos da pilha vertical usando a propriedade Spacing. Isso permite ajustar a distância entre os elementos para criar uma aparência visualmente agradável.
2. **HorizontalOptions e VerticalOptions (Opções Horizontais e Verticais):** Essas propriedades permitem controlar como o VerticalStackLayout se ajusta em relação ao espaço disponível na tela, tanto horizontal quanto verticalmente. Por exemplo, você pode definir **HorizontalOptions** como **LayoutOptions.FillAndExpand** para que a pilha preencha o espaço horizontalmente.
3. **HorizontalContentAlignment e VerticalContentAlignment (Alinhamento Horizontal e Vertical):** Essas propriedades determinam como o conteúdo de cada elemento filho é alinhado dentro do espaço atribuído a ele na pilha. Você pode definir essas propriedades para **LayoutOptions.Center**, **LayoutOptions.Start**, **LayoutOptions.End** ou **LayoutOptions.Fill**.
4. **Margin (Margem):** A propriedade Margin permite definir o espaço ao redor do VerticalStackLayout como um todo. Você pode atribuir valores diferentes para a margem em cada lado (superior, inferior, esquerdo e



direito) usando a classe Thickness. Isso é útil para ajustar o espaçamento entre o VerticalStackLayout e os elementos vizinhos na interface do usuário.

5. **Padding (Preenchimento):** A propriedade Padding determina o espaço interno do VerticalStackLayout ao redor de seus elementos filhos. Assim como a propriedade Margin, você pode especificar valores para o preenchimento em cada lado usando a classe Thickness. Isso é útil para controlar o espaço interno entre os elementos na pilha vertical, permitindo criar margens uniformes ao redor do conteúdo.

Ao usar essas opções em conjunto, você pode criar layouts flexíveis e visualmente agradáveis em um VerticalStackLayout no .NET MAUI. Ajustar a margem e o preenchimento adequadamente pode garantir que seu conteúdo seja exibido de forma coesa e bem organizada, independentemente do tamanho da tela ou da plataforma em que o aplicativo está sendo executado.

No projeto criado é possível visualizar esses elementos no arquivo **MainPage.xaml**.

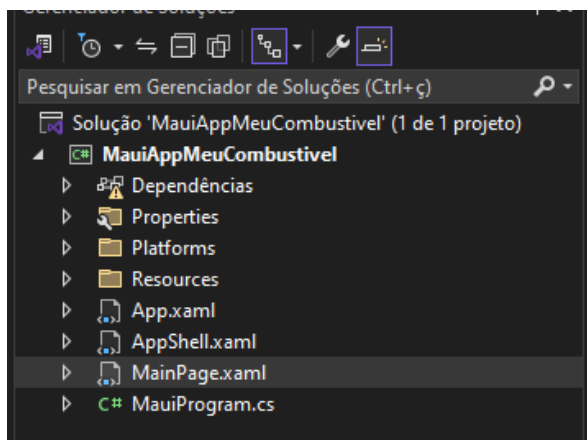


Figura 6 – Área Gerenciador de Soluções, arquivo MainPage.xaml

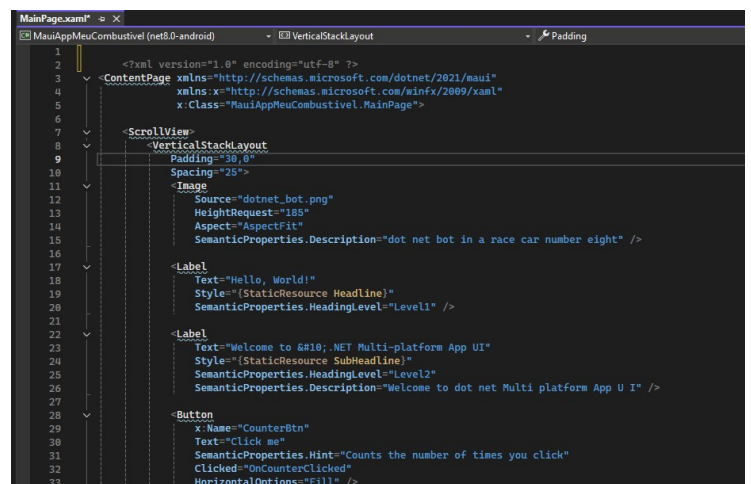


Figura 7 – Arquivo MainPage.xaml

**Agora vamos conhecer os elementos que compõem o layout.**

### Elemento Label

O elemento Label é amplamente utilizado para exibir texto estático na interface do usuário em aplicativos .NET MAUI. Ele oferece diversas propriedades de configuração que permitem personalizar sua aparência e comportamento. Algumas das principais propriedades de configuração incluem:

- ✓ **Text (Texto):** A propriedade Text define o conteúdo textual exibido pelo Label. Você pode atribuir uma string a essa propriedade para exibir o texto desejado.
- ✓ **TextColor (Cor do Texto):** propriedade define a cor do texto exibido pelo Label. Você pode especificar uma

cor pelo nome (como "Red" ou "Blue") ou por código hexadecimal (como "#FF0000" para vermelho).

- ✓ **FontFamily (Família da Fonte):** permite especificar a família da fonte a ser usada para exibir o texto. Podemos atribuir o nome de uma fonte específica para personalizar a aparência do texto.
- ✓ **FontSize (Tamanho da Fonte):** Esta propriedade define o tamanho da fonte utilizada para exibir o texto. Podendo ser em unidades de pixels ou em tamanhos relativos, como "Small", "Medium" ou "Large".
- ✓ **FontAttributes (Atributos da Fonte):** Esta propriedade permite aplicar atributos ao texto, como negrito, itálico ou sublinhado.
- ✓ **HorizontalOptions e VerticalOptions (Opções Horizontais e Verticais):** Essas propriedades determinam como o Label se comporta em relação ao layout pai. Você pode especificar como o Label se alinha horizontal e verticalmente usando as enumerações LayoutOptions.

## Elemento Entry

O elemento Entry é usado para permitir que os usuários insiram texto em um campo de entrada em um aplicativo. Ele oferece diversas propriedades de configuração para personalizar sua aparência e comportamento. Algumas das principais propriedades de configuração incluem:

- ✓ **Text (Texto):** A propriedade Text define o conteúdo textual inserido pelo usuário no campo de entrada.
- ✓ **Placeholder (Marcador de Posição):** Esta propriedade define o texto exibido quando o campo de entrada está vazio. Geralmente é usado para fornecer uma sugestão sobre o tipo de entrada esperada.
- ✓ **TextColor (Cor do Texto):** Esta propriedade define a cor do texto exibido no campo de entrada. Podemos especificar uma cor pelo nome ou por código hexadecimal, da mesma forma que no elemento Label.
- ✓ **PlaceholderColor (Cor do Marcador de Posição):** Esta propriedade define a cor do texto do marcador de posição exibido quando o campo de entrada está vazio.
- ✓ **IsPassword (É Senha):** Esta propriedade determina se o texto inserido no campo de entrada será exibido como caracteres ocultos (como pontos ou asteriscos), geralmente usado para entradas de senha.
- ✓ **Keyboard (Teclado):** define o tipo de teclado exibido ao abrir o campo de entrada.
- ✓ **MaxLength (Comprimento Máximo):** define o número máximo de caracteres permitidos no campo de entrada. Se o usuário tentar digitar mais caracteres do que o limite especificado, o texto será cortado automaticamente.
- ✓ **HorizontalOptions e VerticalOptions (Opções Horizontais e Verticais):** Assim como no elemento Label, essas propriedades determinam como o Entry se comporta em relação ao layout pai, permitindo especificar seu alinhamento horizontal e vertical.

Além dessas propriedades principais, também temos o atributo **x:Name**, que é usado em XAML para nomear elementos da interface do usuário, permitindo identificá-los e acessá-los facilmente no código-behind. A

importância do `x` é destacada por:

- ✓ **Identificação de Elementos:** Ao atribuir um **`x.Name`** a um Entry no XAML, criamos um identificador exclusivo que pode ser usado para referenciá-lo no código-behind. Isso facilita o acesso e a manipulação do campo de entrada.
- ✓ **Manipulação de Propriedades e Eventos:** Com uma referência ao Entry no código-behind, é possível manipular suas propriedades e associar eventos. Por exemplo, é possível alterar o texto, modificar o estilo ou associar manipuladores de eventos, como o `TextChanged`, conforme necessário.

## Elemento Button

O elemento Button é usado para criar botões clicáveis em um aplicativo, permitindo que os usuários interajam com a interface do usuário. Algumas de suas principais propriedades de configuração incluem:

- ✓ **Text (Texto):** define o texto exibido dentro do botão. Você pode atribuir uma string a essa propriedade para indicar a ação que será executada quando o botão for clicado.
- ✓ **TextColor (Cor do Texto):** define a cor do texto exibido no botão.
- ✓ **BackgroundColor (Cor de Fundo):** define a cor de fundo do botão. Assim como outras propriedades de cor, você pode especificar a cor pelo nome ou por código hexadecimal, personalizando a aparência do botão.
- ✓ **BorderColor (Cor da Borda):** Esta propriedade define a cor da borda do botão. Você pode especificar uma cor pelo nome ou por código hexadecimal para adicionar um contorno ao redor do botão, destacando-o visualmente.
- ✓ **BorderWidth (Largura da Borda):** A propriedade `BorderWidth` define a largura da borda do botão em pixels. Ajustando essa propriedade, é possível controlar a espessura da borda.
- ✓ **CornerRadius (Raio do Canto):** define o raio dos cantos do botão, arredondando suas bordas.
- ✓ **Clicked (Evento de Clique):** O evento `Clicked` é acionado quando o botão é clicado. Você pode manipular esse evento no código para definir ações a serem executadas em resposta ao clique do usuário.

## Primeiro Layout

Vamos descrever o passo a passo, para a construção do primeiro layout. Agora que já conhecemos a base para codificação xaml, vamos modificar o arquivo **MainPage.xaml**, vamos iniciar apagando os elementos existentes deixando apenas a imagem do carrinho.

```

<?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="MauiAppMeuCombustivel.MainPage">

  <ScrollView>
    <VerticalStackLayout
      Padding="30,0"
      Spacing="25">
      <Image
        Source="dotnet_bot.png"
        HeightRequest="185"
        Aspect="AspectFit"
        SemanticProperties.Description="dot net bot in a race car number eight" />

    </VerticalStackLayout>
  </ScrollView>

</ContentPage>

```

Agora vamos adicionar os elementos Entry, permitindo a inserção de dados pelo usuário.

```

<Entry x:Name="txt_etanol" Placeholder="R$ " />

<Entry x:Name="txt_gasolina" Placeholder="R$ " />

```

Os componentes Label trazem a informação do que deve ser preenchido:

```

<Label Text="Preço do Etanol:" />
<Entry x:Name="txt_etanol" Placeholder="R$ " />

<Label Text="Preço da Gasolina:" />
<Entry x:Name="txt_gasolina" Placeholder="R$ " />

```

Vamos adicionar o botão para trazer interação a tela:

```

<Button Text="Qual compensa mais?" Clicked="Button_Clicked" />

```

Veja o código completo:

```

<?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="MauiAppMeuCombustivel.MainPage">

  <ScrollView>
    <VerticalStackLayout

```

```

        Padding="30,0"
        Spacing="25">
        <Image
            Source="dotnet_bot.png"
            HeightRequest="185"
            Aspect="AspectFit"
            SemanticProperties.Description="dot net bot in a race car number eight" />

        <Label Text="Preço do Etanol:" />
        <Entry x:Name="txt_etanol" Placeholder="R$ " />

        <Label Text="Preço da Gasolina:" />
        <Entry x:Name="txt_gasolina" Placeholder="R$ " />

        <Button Text="Qual compensa mais?" Clicked="Button_Clicked" />

    </VerticalStackLayout>
</ScrollView>

</ContentPage>

```

Assista o trecho do vídeo que explica sobre os códigos relacionados ao layout:

<https://www.youtube.com/watch?v=gpQ9pioNH0c&t=721s>

## Manipulador de Eventos

No contexto do .NET MAUI, o evento **Clicked** é acionado quando um elemento clicável, como um botão clicado pelo usuário. Especificamente no caso do elemento Button, o evento Clicked ocorre ao ser pressionado, permitindo que os usuários interajam com a interface e atuem ações ou funcionalidades específicas dentro do aplicativo. Para lidar com o evento **Clicked**, geralmente é definido um manipulador de eventos, que é uma função ou método responsável por executar a lógica desejada quando o evento é acionado, tal ação é feita utilizando a linguagem C#. Abaixo está um exemplo simples de como você pode definir um manipulador de eventos para o evento Clicked de um botão em .NET MAUI:

```

private void OnButtonClicked(object sender, EventArgs e)
{
    DisplayAlert("Botão Clicado", "O botão foi clicado!", "OK");
}

```

No exemplo apresentado, foi criado um botão e definido um manipulador de eventos para o evento Clicked. Quando o botão é clicado, o método OnButtonClicked é chamado, e dentro dele, estamos exibindo um alerta para o usuário.

O manipulador de eventos é uma maneira poderosa de controlar o comportamento do aplicativo em resposta às

## Tratamento de Exceções no C#

interações do usuário, permitindo que você crie experiências interativas e dinâmicas em seu aplicativo .NET MAUI.

O uso da estrutura try-catch é uma prática recomendada para tratar exceções de forma controlada e evitar que erros inesperados interrompam a execução do aplicativo. Isso ajuda a garantir a robustez e confiabilidade do seu código, permitindo que ele responda adequadamente a situações excepcionais.

É recomendável o uso da estrutura **try-catch** dentro dos manipuladores de eventos. O try-catch é uma construção fundamental na programação para lidar com exceções, permitindo que você monitore e trate erros que possam ocorrer durante a execução de um bloco específico de código. Vamos compreender como funciona:

- ✓ **Bloco Try:** O código que pode gerar uma exceção é envolvido dentro de um bloco try. Este é o local onde você espera que uma exceção possa ocorrer. Se uma exceção for gerada dentro deste bloco, o controle do programa é imediatamente transferido para o bloco catch.
- ✓ **Bloco Catch:** No bloco catch, você pode capturar a exceção, especificando o tipo de erro que deseja tratar. Quando uma exceção correspondente ocorre no bloco try, o controle do programa é transferido para o catch. Neste ponto, você pode tomar medidas, como exibir uma mensagem de erro para o usuário, registrar o erro ou executar ações de recuperação.
- ✓ **Bloco Finally (Opcional):** O bloco finally é opcional e contém o código que será executado independentemente de uma exceção ocorrer ou não no bloco try. Ele é útil para liberar recursos, como fechar arquivos ou conexões de banco de dados, garantindo que essas operações ocorram mesmo em caso de exceção.

## DISPLAYALERT

No .NET MAUI, o método DisplayAlert é uma maneira conveniente de exibir alertas modais para os usuários. Ele permite mostrar mensagens informativas, de confirmação ou de aviso em uma janela sobreposta à interface principal do aplicativo.

### Um resumo do DisplayAlert:

- ✓ **Exibição Modal:** O DisplayAlert cria uma janela modal que interrompe temporariamente a interação do usuário com o restante da interface enquanto o alerta é exibido. Isso garante que a atenção do usuário seja direcionada para a mensagem.
- ✓ **Mensagem e Botões:** Você pode personalizar o conteúdo do alerta, incluindo um título, uma mensagem de texto e botões para interação do usuário. Normalmente, são usados botões como

"OK" ou "Cancelar", dependendo do propósito do alerta.

- ✓ **Retorno de Tarefa Assíncrona:** O método `DisplayAlert` retorna uma tarefa assíncrona, permitindo que você aguarde a resposta do usuário antes de continuar a execução do código. Isso é útil se for necessário tomar decisões ou realizar ações com base na escolha do usuário.
- ✓ **Sintaxe Simples:** A sintaxe para usar o `DisplayAlert` é clara e simples. Abaixo está um exemplo básico de como utilizá-lo:

```
await DisplayAlert("Título", "Mensagem de alerta", "OK");
```

Este código exibirá um alerta modal com um título "Título", uma mensagem "Mensagem de alerta" e um botão "OK" para o usuário clicar. O método `DisplayAlert` retorna um valor booleano indicando se o botão de ação foi pressionado (`true` para o botão "OK" neste caso).

Veja o código completo em C# do arquivo **MainPage.xaml.cs**:

```
namespace MauiAppMeuCombustivel
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void Button_Clicked(object sender, EventArgs e)
        {
            {
                try
                {
                    double etanol = Convert.ToDouble(txt_etanol.Text);
                    double gasolina = Convert.ToDouble(txt_gasolina.Text);

                    string msg = "";

                    if (etanol <= (gasolina * 0.7))
                    {
                        msg = "O etanol está compensando.";
                    }
                    else
                    {
                        msg = "A gasolina está compensando.";
                    }

                    DisplayAlert("Calculado", msg, "OK");
                }
            }
        }
    }
}
```

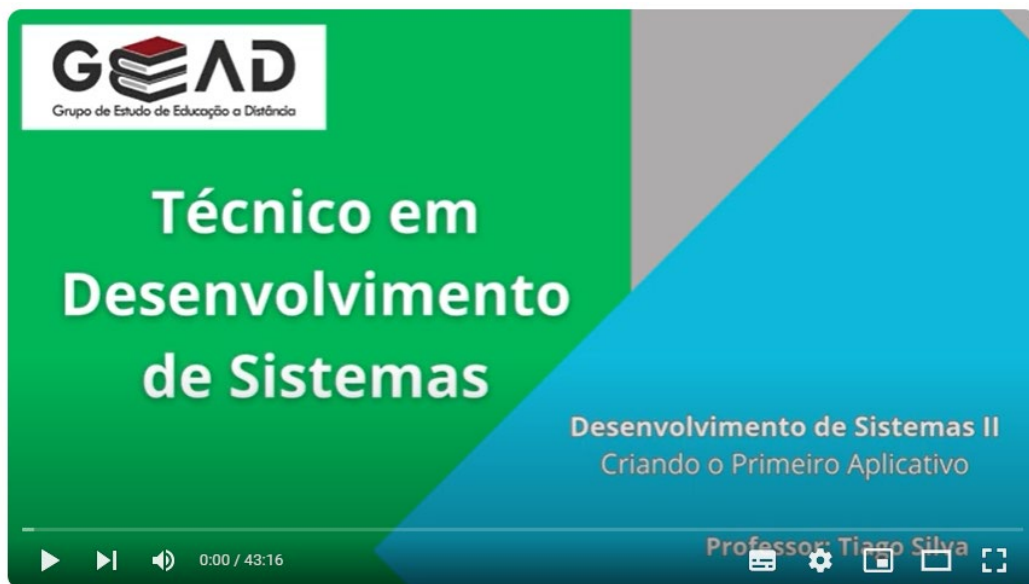
```

    }
    catch (Exception ex)
    {
        DisplayAlert("Ops", ex.Message, "Fechar");
    }

    } // Fecha método
  } //Fecha Class
} // Fecha namespace
}

```

Para que as aulas se torne mais interativas, estamos disponibilizando um vídeo referente a construção completa do projeto apresentado, então assista ao vídeo:



Disponível em: <https://www.youtube.com/watch?v=gpQ9pioNH0c>

Acesse o código fonte completo: <https://github.com/tiagotas/MauiAppMeuCombustivel>