AGENDA 15

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:

TIAGO ANTONIO DA SILVA

KELLY CRISTIANE DE OLIVEIRA DAL POZZO

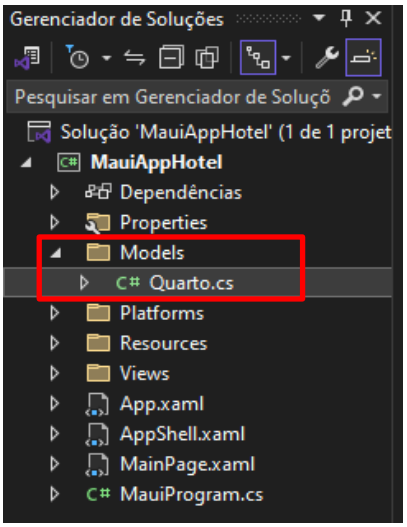
São Paulo – SP, 2024

MERGULHANDO NO TEMA

Estamos nos aproximando da finalização do projeto AppHotel, no qual foi possível aplicar diversos conceitos de programação em C# e design de interfaces com XAML. Nesta etapa, vamos aprofundar a integração entre essas duas tecnologias. Em C#, um model é uma classe usada para representar dados e a estrutura lógica de informações em um sistema, geralmente representa uma entidade ou uma estrutura de dados. Na agenda anterior foi criado a classe model denominada Quarto, agora vamos adicionar mais uma classe na pasta Model denominada Hospedagem.

Acesse o Gerenciador de Soluções e encontre o diretório Models, criado na agenda anterior.

Em seguida clique com o botão direito sobre o diretório, selecione a opção Adicionar, Novo Item, Classe C#, adicione o no Hospedagem. Modifique a classe de internal para public.




```
public class Hospedagem
{}
```

A estrutura da classe deve ser a seguinte:

```
namespace MauiAppHotel.Models
{
    public class Hospedagem
    {
        public Quarto QuartoSelecionado { get; set; }
        public int QntAdultos { get; set; }
        public int QntCriancas { get; set; }
        public DateTime DataCheckIn { get; set; }
        public DateTime DataCheckOut { get; set; }
        public int Estadia
        {
            get => DataCheckOut.Subtract(DataCheckIn).Days;
        }
        public double ValorTotal
        {
            get
            {
                double valor_adultos = QntAdultos * QuartoSelecionado.ValorDiariaAdulto;
                double valor_criancas = QntCriancas * QuartoSelecionado.ValorDiariaCrianca;

                double total = (valor_adultos + valor_criancas) * Estadia;
                return total;
            }
        }
    }
}
```

Vamos compreender a propriedade ValorTotal:

O método ValorTotal é uma propriedade que calcula e retorna o valor total de uma estadia em um quarto de hotel com base no número de adultos e crianças, as diárias correspondentes e a duração da estadia. Essa propriedade é uma forma de encapsular a lógica de cálculo do custo total da estadia. Toda vez que ValorTotal for acessada, o cálculo será feito com base nos valores atuais de QntAdultos, QntCriancas, QuartoSelecionado e Estadia.

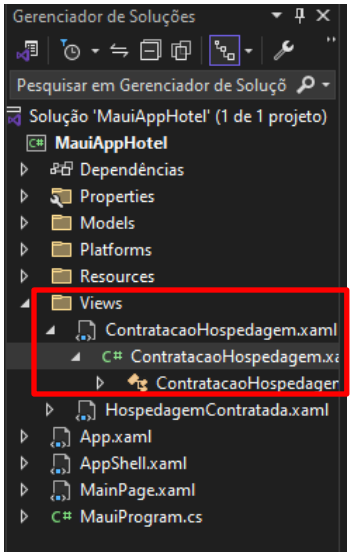
Métodos Assíncronos

Métodos assíncronos em C# são funções que permitem a execução de tarefas demoradas sem travar a aplicação. Eles são indicados pela palavra-chave **async**, que informa ao compilador que aquele método pode ser executado de forma simultaneamente sem bloquear outras tarefas.

Por que isso é importante?
Imagine uma aplicação que precisa buscar dados de uma API. Se o método de busca for executado de forma síncrona, a interface do usuário pode congelar enquanto aguarda a resposta. Com um método assíncrono, a aplicação continua a funcionar e a responder, e a tarefa de busca é concluída em segundo plano.

A palavra-chave **await** é usada dentro de métodos async e serve para pausar a execução do método até que a tarefa seja concluída. Durante essa pausa, a execução da aplicação não fica parada; ela continua a responder a outras solicitações, como cliques do usuário ou outras operações em andamento.

Para melhor compreensão vamos alterar o método do botão contido no arquivo ContratacaoHospedagem.xaml.cs disponível no diretório View:

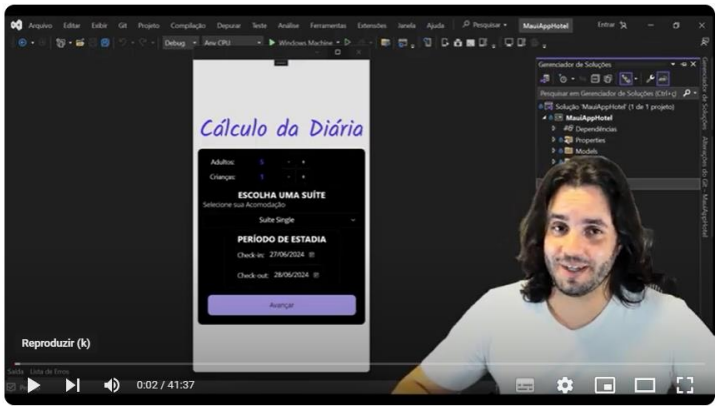


```
private async void Button_Clicked(object sender, EventArgs e)
{
    try
    {
        Hospedagem h = new Hospedagem
        {
            QuartoSelecionado = (Quarto)pck_quarto.SelectedItem,
            QntAdultos = Convert.ToInt32(stp_adultos.Value),
            QntCriancas = Convert.ToInt32(stp_criancas.Value),
            DataCheckIn = dtpck_checkin.Date,
            DataCheckOut = dtpck_checkout.Date,
        };

        await Navigation.PushAsync(new HospedagemContratada()
        {
            BindingContext = h
        });

    }
    catch (Exception ex)
    {
        await DisplayAlert("Ops", ex.Message, "OK");
    }
}
```

Assista a vídeo aula explicativa sobre o projeto:



Disponível em: https://www.youtube.com/watch?v=6NhQcOjJyjw.

Vamos agora compreender alguns conceitos apresentados neste projeto.

Transportar dados entre páginas XAML com BidingContext

A propriedade BindingContext de uma ContentPage no XAML é fundamental para a vinculação de dados entre a interface de usuário e o código do aplicativo. Ela permite associar um objeto de dados (geralmente uma model) à página, o que facilita a exibição de dados na interface e a reação a mudanças nos dados automaticamente. Quando definimos o BindingContext de uma ContentPage, todos os controles na página podem se vincular a propriedades do objeto definido como contexto de dados. Essa associação é feita por

meio de bindings, que ligam diretamente as propriedades dos elementos da interface gráfica às propriedades do contexto de dados, proporcionando uma comunicação eficiente e simplificada.
É comum definir o BindingContext no arquivo de código back-end da página (arquivo .xaml.cs). Isso permite flexibilidade para instanciar e configurar o contexto de dados de forma programática. Veja um exemplo:

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

        // Define um contexto de dados com um objeto simples
        BindingContext = new Pessoa
        {
            Nome = "Maria",
            Idade = 25
        };
    }
}

public class Pessoa
{
    public string Nome { get; set; }
    public int Idade { get; set; }
}
```

No arquivo XAML, podemos referir às propriedades de Pessoa:

```
<Label Text="{Binding Nome}" />
<Label Text="{Binding Idade}" />
```

BidingConterxt e StringFormat no XAML

Conforme já vimos anteriormente, no XAML o Binding (vinculação) é uma maneira de conectar as propriedades da interface de usuário a dados no BindingContext.

A funcionalidade de StringFormat é um recurso poderoso para formatação de Bindings, permitindo que exiba os dados em um formato específico na interface, sem precisar modificar os dados em si.

Quando fazemos um Binding, indica para o XAML para buscar o valor de PropertyName no BindingContext da página ou do elemento em questão. Veja um exemplo onde temos uma propriedade Name em uma model e deseja exibir em um Label:

```
<Label Text="{Binding Name}" />
```

No exemplo apresentado, o Label exibirá o valor de Name, desde que a model tenha uma propriedade

chamada Name, caso contrário gerará um erro de associação do XAML.

O StringFormat permite a personalização de como os dados são exibidos. Isso é particularmente útil para formatação de números, datas ou strings, adicionando prefixos, sufixos ou organizando o texto em um formato específico.

A sintaxe do StringFormat usa um marcador {0}, onde o valor do binding será inserido:

```
<Label Text="{Binding Age, StringFormat='Idade: {0} anos'}" />
```

Neste exemplo, se Age for 25, o Label exibirá Idade: 25 anos. Agora, caso queira exibir um preço formatado como moeda:

```
<Label Text="{Binding Price, StringFormat='Preço: {0:C}'}" />
```

O {0:C} formatará o valor como moeda, respeitando as configurações de cultura do dispositivo. Se Price for 15.5, isso exibirá Preço: R$ 15,50.
Para datas, o StringFormat permite a utilização de diversos padrões de formatação, como por exemplo:

```
<Label Text="{Binding Date, StringFormat='Data: {0:dd/MM/yyyy}'}" />
```

Esse código exibirá a data no formato "dia/mês/ano". Se Date for 2024-01-01, o Label exibirá Data: 01/01/2024.

Para a formatação correta, deve-se levar em consideração:

- ✓ **Cultura Padrão:** O StringFormat respeita a cultura padrão do dispositivo, o que significa que formatações de datas, moedas e números serão exibidas de acordo com as configurações regionais do dispositivo.
- ✓ **Personalização com Culture:** Se precisar definir uma cultura específica para o formato, isso deve ser feito programaticamente.
- ✓ **Versatilidade:** O StringFormat é particularmente útil em casos onde é preciso formatação personalizada sem escrever código adicional ou converter manualmente os valores no code-behind.

Calcular datas em C#

Em C#, a manipulação de datas pode ser feita com a classe **DateTime** e com a estrutura **TimeSpan**. Essas estruturas permitem realizar cálculos como adicionar, subtrair datas e calcular períodos de dias ou outras unidades de tempo entre duas datas.

O **DateTime** possui métodos específicos para adicionar dias, meses, anos, horas, minutos, segundos, e milissegundos a uma data. Esses métodos são úteis para operações de agendamento ou para calcular uma data futura, no AppHotel foi utilizado para o cálculo da estadia. Para adicionar dias a uma data, podemos utilizar o método **AddDays**:

```
DateTime hoje = DateTime.Now;
DateTime daquiCincoDias = hoje.AddDays(5);

Console.WriteLine($"Hoje: {hoje}");
Console.WriteLine($"Daqui a 5 dias: {daquiCincoDias}");
```

Outros métodos de adição incluem:

- ✓ AddMonths(int meses): Adiciona meses à data.
- ✓ AddYears(int anos): Adiciona anos à data.
- ✓ AddHours(int horas): Adiciona horas à data.
- ✓ AddMinutes(int minutos): Adiciona minutos à data.
- ✓ AddSeconds(int segundos): Adiciona segundos à data.
- ✓ AddMilliseconds(int milissegundos): Adiciona milissegundos à data.

No caso do AppHotel, é necessário saber a diferença entre duas datas, a operação de subtração entre objetos **DateTime** resulta em um objeto **TimeSpan**. O **TimeSpan** representa o intervalo de tempo e pode ser convertido em dias, horas, minutos, entre outros.

```
DateTime dataInicio = new DateTime(2024, 11, 1);
DateTime dataFim = new DateTime(2024, 11, 10);

TimeSpan diferenca = dataFim - dataInicio;

Console.WriteLine($"Dias de diferença: {diferenca.Days}");
Console.WriteLine($"Horas de diferença: {diferenca.TotalHours}");
Console.WriteLine($"Minutos de diferença: {diferenca.TotalMinutes}");
```

Para obter especificamente a diferença em dias entre duas datas, podemos usar a propriedade **Days** de **TimeSpan**, como mostrado anteriormente.

LINK DO CÓDIGO-FONTE REALIZADO NESTA AGENDA

https://github.com/tiagotas/MauiAppHotel/tree/f0601d6f4064eae8dd0934259d374e3a49fa784c