MAUI do Zero ao Pro

Apps Multiplataforma com VS Code

Uma imagem com texto, computador, Dispositivo eletrónico, aparelho

Os conteúdos gerados por IA poderão estar incorretos.

Luís Simões da Cunha, 2025

[Uma imagem com símbolo, Tipo de letra, Gráficos, captura de ecrã

Os conteúdos gerados por IA poderão estar incorretos.](https://creativecommons.org/licenses/by-nc/4.0/)

Índice

[👋 Bem-vindo ao Tutorial de .NET MAUI 1](#_Toc193421969)

[🧭 Cria apps modernas, bonitas e multiplataforma — passo a passo e sem complicações. 1](#_Toc193421970)

[🎯 O que vais aprender 1](#_Toc193421971)

[🧰 O que precisas para começar 1](#_Toc193421972)

[📦 Soluções dos exercícios 2](#_Toc193421973)

[🚀 Vamos começar? 2](#_Toc193421974)

[🌺 Parte 1 - Cria a tua primeira app com .NET MAUI – “Aloha, World!” (VS Code Edition) 3](#_Toc193421975)

[🧠 Objetivo de aprendizagem 3](#_Toc193421976)

[🛠️ Etapa 1: Preparar o ambiente 3](#_Toc193421977)

[O que precisas: 3](#_Toc193421978)

[📁 Etapa 2: Criar o projeto “AlohaWorld” 3](#_Toc193421979)

[🧭 Etapa 3: Explorar os ficheiros principais 3](#_Toc193421980)

[▶️ Etapa 4: Executar a app 4](#_Toc193421981)

[🔄 Etapa 5: Modificar o texto com Hot Reload (versão VS Code não suporta!) 4](#_Toc193421982)

[🧠 Etapa 6: Entender como tudo se liga (sem dor de cabeça) 5](#_Toc193421983)

[✨ Etapa extra: Personaliza com o teu bot 5](#_Toc193421984)

[✅ Recapitulando 5](#_Toc193421985)

[📲 Parte 2 – Interação e persistência: contador com base de dados local (SQLite) 6](#_Toc193421986)

[🎯 Objetivo de aprendizagem 6](#_Toc193421987)

[📦 Etapa 1: Adicionar SQLite ao projeto 6](#_Toc193421988)

[🧱 Etapa 2: Criar o modelo de dados 6](#_Toc193421989)

[🛠️ Etapa 3: Criar o serviço de base de dados 7](#_Toc193421990)

[🔌 Etapa 4: Ligar o serviço à tua aplicação 8](#_Toc193421991)

[🎬 Etapa 5: Atualizar o MainPage.xaml.cs 8](#_Toc193421992)

[▶️ Etapa 6: Executar e testar 9](#_Toc193421993)

[🧠 Dica de Psicologia Cognitiva 9](#_Toc193421994)

[✅ Conclusão desta parte 9](#_Toc193421995)

[📱 Parte 3 – Apps interativas com .NET MAUI no VS Code 11](#_Toc193421996)

[🎯 Objetivo 11](#_Toc193421997)

[🧭 Projeto 1 – **FindMe!**: Partilha a tua localização 11](#_Toc193421998)

[🛠️ Etapa 1: Criar a app 11](#_Toc193421999)

[🔒 Etapa 2: Pedir permissões de localização 11](#_Toc193422000)

[✏️ Etapa 3: Atualizar o layout (MainPage.xaml) 12](#_Toc193422001)

[💻 Etapa 4: Escrever o código (MainPage.xaml.cs) 12](#_Toc193422002)

[▶️ Etapa 5: Correr a app 14](#_Toc193422003)

[✅ O que aprendeste até agora 14](#_Toc193422004)

[📋 Projeto 2 – **MauiTodo**: A tua lista de tarefas segura 14](#_Toc193422005)

[🛠️ Etapa 1: Criar a app 14](#_Toc193422006)

[📦 Etapa 2: Instalar as bibliotecas necessárias 14](#_Toc193422007)

[🧱 Etapa 3: Modelo de dados 14](#_Toc193422008)

[🔐 Etapa 4: Serviço de base de dados 15](#_Toc193422009)

[🖼️ Etapa 5: Interface (MainPage.xaml) 15](#_Toc193422010)

[🔗 Etapa 6: Código e ligação (MainPage.xaml.cs) 16](#_Toc193422011)

[🧠 Dicas de aprendizagem 17](#_Toc193422012)

[🎓 Conclusão 17](#_Toc193422013)

[🧩 Parte 4 – Controlo Total: Criar interfaces modernas com .NET MAUI 18](#_Toc193422014)

[📌 1. O que são “Views” e “Controls”? 18](#_Toc193422015)

[✅ Tipos de “Views”: 18](#_Toc193422016)

[🖼️ 2. Controles visuais (só mostram, não interagem) 18](#_Toc193422017)

[⌨️ 3. Controles interativos (recebem input do utilizador) 18](#_Toc193422018)

[🖱️ 4. Controles de ação (executam comandos) 19](#_Toc193422019)

[🎨 5. Mostrar imagens e gráficos 19](#_Toc193422020)

[📚 6. Listas e coleções dinâmicas 19](#_Toc193422021)

[a) CollectionView – o mais usado! 19](#_Toc193422022)

[b) CarouselView – tipo "carrossel" 20](#_Toc193422023)

[c) ListView – versão antiga, menos flexível 20](#_Toc193422024)

[d) TableView – ideal para **definições e formulários** 20](#_Toc193422025)

[🧰 7. Personalizações com “modificadores” 20](#_Toc193422026)

[✂️ Clip (recorte de forma) 20](#_Toc193422027)

[⬛ Border (borda) 20](#_Toc193422028)

[☁️ Shadow (sombra) 20](#_Toc193422029)

[✋ GestureRecognizers (toques e gestos) 21](#_Toc193422030)

[🔄 RefreshView (puxar para atualizar) 21](#_Toc193422031)

[👈 SwipeView (deslizar para ações) 21](#_Toc193422032)

[🧠 Psicologia aplicada ao design deste tutorial: 21](#_Toc193422033)

[✅ O que aprendeste 21](#_Toc193422034)

[🧱 Parte 5 – Layouts em .NET MAUI: organiza o teu mundo visual 22](#_Toc193422035)

[🧠 Antes de mais… o que é um “Layout”? 22](#_Toc193422036)

[📐 Tipos principais de layout em .NET MAUI 22](#_Toc193422037)

[🧩 1. Grid – o mais poderoso 22](#_Toc193422038)

[🛠️ Exemplo prático: Mini-Calculadora 22](#_Toc193422039)

[📏 2. StackLayouts – empilha com simplicidade 23](#_Toc193422040)

[a) VerticalStackLayout 23](#_Toc193422041)

[b) HorizontalStackLayout 23](#_Toc193422042)

[📜 3. ScrollView – quando há conteúdo a mais 23](#_Toc193422043)

[🎯 Dica visual: Combina layouts! 24](#_Toc193422044)

[🧠 Psicologia cognitiva aplicada neste tutorial: 24](#_Toc193422045)

[✅ O que aprendeste nesta parte 24](#_Toc193422046)

[🚀 Desafio prático para ti\* 25](#_Toc193422047)

[🎯 Parte 6 – Layouts Avançados em .NET MAUI: flexibilidade máxima na tua app 26](#_Toc193422048)

[🧠 Objetivo 26](#_Toc193422049)

[🧩 1. Pensar como um Grid 26](#_Toc193422050)

[Exemplo prático: ecrã de perfil 26](#_Toc193422051)

[🔄 2. BindableLayout – listas sem CollectionView 26](#_Toc193422052)

[Exemplo com FlexLayout – app de filmes 26](#_Toc193422053)

[📍 3. AbsoluteLayout – quando queres um botão “flutuante” (FAB) 27](#_Toc193422054)

[Exemplo – botão no canto inferior direito: 27](#_Toc193422055)

[🧑‍🎨 4. Exemplo completo: App de filmes interativa (MauiMovies) 27](#_Toc193422056)

[Funcionalidades: 27](#_Toc193422057)

[Ferramentas usadas: 28](#_Toc193422058)

[🧠 Psicologia aplicada: 28](#_Toc193422059)

[✅ O que levas deste tutorial 28](#_Toc193422060)

[🧭 Parte 7 – Páginas e Navegação em .NET MAUI (com VS Code) 29](#_Toc193422061)

[🧠 1. Por que precisamos de várias páginas? 29](#_Toc193422062)

[📄 2. ContentPage – a página base 29](#_Toc193422063)

[Exemplo simples: 29](#_Toc193422064)

[🧭 3. Tipos de navegação em .NET MAUI 29](#_Toc193422065)

[🛠️ 4. Começar com Shell 30](#_Toc193422066)

[a) Cria um novo projeto com Shell (no terminal): 30](#_Toc193422067)

[📑 5. Criar várias páginas 30](#_Toc193422068)

[🧭 6. Navegação com Shell: TabBar + Flyout 30](#_Toc193422069)

[a) AppShell.xaml (estrutura da app) 30](#_Toc193422070)

[🔄 7. Navegação programática (GoToAsync) 31](#_Toc193422071)

[a) Para ir da InputPage para uma nova página (ex: detalhes): 31](#_Toc193422072)

[b) Registar a página de destino (ex: ProductPage) 31](#_Toc193422073)

[📦 8. Passar dados entre páginas com Shell 31](#_Toc193422074)

[a) Decorar a página destino com [QueryProperty] 31](#_Toc193422075)

[🔁 9. Ciclo de vida das páginas 32](#_Toc193422076)

[Exemplo: 32](#_Toc193422077)

[📌 10. Resumo (Resumo = retenção 💪) 32](#_Toc193422078)

[🎯 Desafio para fixar\* 32](#_Toc193422079)

[🏢 Parte 8 – Desenvolver Apps Empresariais com .NET MAUI 33](#_Toc193422080)

[🎯 Objetivo 33](#_Toc193422081)

[🧠 1. Por que separar a lógica da interface? 33](#_Toc193422082)

[🛠️ 2. Exemplo: Página de Login com serviço de autenticação 33](#_Toc193422083)

[a) Criar interface IAuthService 33](#_Toc193422084)

[b) Criar implementação mock: MockAuthService 33](#_Toc193422085)

[c) Ligar à UI (LoginPage.xaml.cs) 34](#_Toc193422086)

[🔐 3. Fazer login real com IdentityServer 34](#_Toc193422087)

[a) Fluxo OAuth2 simplificado: 34](#_Toc193422088)

[b) Criar AuthService com OidcClient 34](#_Toc193422089)

[Implementar: 34](#_Toc193422090)

[🌐 4. Ligar a APIs seguras com tokens 35](#_Toc193422091)

[a) AuthHandler.cs 35](#_Toc193422092)

[💉 5. Usar Injeção de Dependência (DI) com o Generic Host Builder 35](#_Toc193422093)

[a) No MauiProgram.cs 35](#_Toc193422094)

[🧠 6. Padrões de Arquitetura Profissionais 36](#_Toc193422095)

[Clean Architecture (CA) – Como o projeto está estruturado: 36](#_Toc193422096)

[🌍 7. Visão “Enterprise” — reutilização máxima 36](#_Toc193422097)

[✅ Resumo Cognitivo 36](#_Toc193422098)

[🚀 Próximo desafio 37](#_Toc193422099)

[🎯 Parte 9 – MVVM em .NET MAUI: apps organizadas, robustas e fáceis de manter 38](#_Toc193422100)

[🧠 O que é MVVM? 38](#_Toc193422101)

[Porquê usar MVVM? 38](#_Toc193422102)

[🔧 Refatorar a app MauiTodo com MVVM 38](#_Toc193422103)

[1. Criar a ViewModel 38](#_Toc193422104)

[2. Propriedades e comandos 38](#_Toc193422105)

[3. Métodos principais 39](#_Toc193422106)

[4. Construtor da ViewModel 39](#_Toc193422107)

[🖼️ 5. Atualizar a UI (MainPage.xaml) 39](#_Toc193422108)

[💡 Resolver a limitação do CheckBox 40](#_Toc193422109)

[✅ Solução: usar **EventToCommandBehavior** da CommunityToolkit 40](#_Toc193422110)

[🧼 Organizar melhor: BaseViewModel 40](#_Toc193422111)

[🧪 Exemplo avançado: MauiStockTake com MVVM 41](#_Toc193422112)

[🎓 Vantagens do MVVM 41](#_Toc193422113)

[🚀 Desafio para ti (Solução no final do documento 😉) 41](#_Toc193422114)

[🎨 Parte 10 – Estilos, Temas e Layouts Multi-Plataforma em .NET MAUI 42](#_Toc193422115)

[🎯 Objetivo 42](#_Toc193422116)

[🧠 Princípio-chave: **Consistência visual = Experiência de Utilizador intuitiva** 42](#_Toc193422117)

[🛠️ 1. Criar estilos (e evitar repetir código) 42](#_Toc193422118)

[a) Onde ficam os estilos? 42](#_Toc193422119)

[b) Exemplo de estilo para botões: 42](#_Toc193422120)

[🖼️ 2. Estilizar páginas com gradientes 43](#_Toc193422121)

[a) Define no Colors.xaml: 43](#_Toc193422122)

[b) Usa um Brush no Styles.xaml: 43](#_Toc193422123)

[🌙 3. Suporte a modo claro/escuro (automático) 43](#_Toc193422124)

[a) Define as cores para o modo escuro: 43](#_Toc193422125)

[b) Usa AppThemeBinding: 43](#_Toc193422126)

[🧢 4. Criar temas personalizados (ex: Tema "Praia") 44](#_Toc193422127)

[a) Alternar o tema em runtime: 44](#_Toc193422128)

[🚦 5. Triggers: reagir a estados 44](#_Toc193422129)

[🎭 6. Visual State Manager: diferentes visuais por estado 44](#_Toc193422130)

[🖥️ 7. Layouts multi-plataforma 45](#_Toc193422131)

[Exemplo com OnIdiom: 45](#_Toc193422132)

[🪟 8. Funcionalidades de desktop: menus e janelas 45](#_Toc193422133)

[a) Menu na barra superior (apenas visível no desktop): 45](#_Toc193422134)

[b) Abrir uma nova janela com o conteúdo da página “Sobre”: 45](#_Toc193422135)

[✅ O que levas deste tutorial 46](#_Toc193422136)

[🎛️ Parte 11 – Controlo Total: Criação de Controles Personalizados em .NET MAUI 47](#_Toc193422137)

[🎯 Objetivo 47](#_Toc193422138)

[🧠 Porquê criar controlos personalizados? 47](#_Toc193422139)

[🧱 1. Criar um controlo personalizado com ContentView 47](#_Toc193422140)

[a) Estrutura do layout (MildredStepper.xaml) 47](#_Toc193422141)

[b) Código por trás (MildredStepper.xaml.cs) 48](#_Toc193422142)

[🔗 2. Ligar o controlo à ViewModel com Bindable Properties 49](#_Toc193422143)

[a) Criar propriedade vinculável (bindable) 49](#_Toc193422144)

[🧩 3. Adicionar propriedades extra (ex: IsEnabled com DataTrigger) 49](#_Toc193422145)

[🧪 4. Melhorar a experiência do utilizador 50](#_Toc193422146)

[🧙 5. Personalizar o controlo por plataforma com **Handlers** 50](#_Toc193422147)

[a) Criar uma subclasse BorderlessEntry: 50](#_Toc193422148)

[b) Usar BorderlessEntry no MildredStepper: 51](#_Toc193422149)

[🪄 6. Criar bibliotecas de controlos 51](#_Toc193422150)

[a) Cria um projeto tipo: 51](#_Toc193422151)

[b) Move o controlo para essa biblioteca 51](#_Toc193422152)

[✅ Resumo Cognitivo 51](#_Toc193422153)

[🚀 Desafio prático\* 51](#_Toc193422154)

[🚀 Parte 12 – Publicar Apps com GitHub Actions: do VS Code à App Store 52](#_Toc193422155)

[🎯 Objetivo 52](#_Toc193422156)

[📱 1. Dar personalidade à tua app 52](#_Toc193422157)

[1.1 Substituir o ícone da app 52](#_Toc193422158)

[1.2 Personalizar o splash screen 52](#_Toc193422159)

[🏷️ 2. Nome, identificador e versão 52](#_Toc193422160)

[🤖 3. Introdução ao GitHub Actions (CI/CD) 53](#_Toc193422161)

[O que é? 53](#_Toc193422162)

[📁 4. Criar o ficheiro do workflow 53](#_Toc193422163)

[⚙️ 5. Primeiro job: Build Android 53](#_Toc193422164)

[🍏 6. Segundo job: Build iOS 54](#_Toc193422165)

[📦 7. Último job: Deploy para as lojas 54](#_Toc193422166)

[Exemplo Android: 55](#_Toc193422167)

[🔐 8. Como guardar os segredos? 55](#_Toc193422168)

[✅ Resumo final 55](#_Toc193422169)

[🚀 Próximo passo sugerido\* 55](#_Toc193422170)

[📂 Consulta as soluções no GitHub 56](#_Toc193422171)

# 👋 Bem-vindo ao Tutorial de .NET MAUI

## 🧭 Cria apps modernas, bonitas e multiplataforma — passo a passo e sem complicações.

Este tutorial foi pensado especialmente para quem está a dar os primeiros passos no desenvolvimento de aplicações com **.NET MAUI**. Aqui vais aprender a construir **apps reais**, desde os conceitos básicos até funcionalidades mais avançadas como **armazenamento local, navegação, estilos, MVVM e publicação automática**.

Tudo isto com uma abordagem:

* 📘 **Didática** – inspirada em princípios de **psicologia cognitiva** e boas práticas de ensino
* 🧱 **Progressiva** – começamos com o simples e vamos construindo conhecimento sólido
* 💡 **Prática** – cada conceito é acompanhado de **exercícios com soluções**
* 🌍 **Multiplataforma** – desenvolves para Android, iOS, Windows e macOS, com o mesmo código

## 🎯 O que vais aprender

* Como estruturar interfaces modernas com XAML
* Criar layouts adaptáveis e componentes reutilizáveis
* Guardar dados com SQLite e Preferences
* Separar lógica com o padrão MVVM
* Ligar a APIs externas e proteger dados
* Usar GitHub Actions para publicação automática

## 🧰 O que precisas para começar

Ter o **.NET MAUI instalado** ([Guia oficial](https://learn.microsoft.com/pt-pt/dotnet/maui/get-started/installation))

Usar o **Visual Studio Code**

Vontade de aprender e explorar!

## 📦 Soluções dos exercícios

Todas as soluções estão disponíveis aqui:  
🔗 [github.com/luiscunhacsc/GitHub-MAUI](https://github.com/luiscunhacsc/GitHub-MAUI)

Podes consultar, correr e adaptar cada exercício ao teu ritmo.

## 🚀 Vamos começar?

Este tutorial é feito para ti — seja para aprender, ensinar ou criar apps reais.  
Sê bem-vindo e… **vamos criar apps incríveis!** 💪📱

# 🌺 Parte 1 - Cria a tua primeira app com .NET MAUI – “Aloha, World!” (VS Code Edition)

### 🧠 Objetivo de aprendizagem

No final deste tutorial, vais saber como:

* Criar uma app com .NET MAUI no VS Code (usando o terminal/CLI).
* Executar e ver a app a funcionar.
* Entender a estrutura base de uma app .NET MAUI.
* Fazer alterações simples e ver o resultado.

## 🛠️ Etapa 1: Preparar o ambiente

⏳ Duração: 5 minutos

### O que precisas:

✅ Visual Studio Code instalado  
✅ SDK do .NET 6 ou superior  
✅ Workload do .NET MAUI instalado  
✅ Um terminal (PowerShell, Terminal do macOS, etc.)

Se ainda não tens isto pronto, consulta o [Guia de Instalação .NET MAUI](https://learn.microsoft.com/en-us/dotnet/maui/get-started/installation) — é super visual e rápido de seguir.

## 📁 Etapa 2: Criar o projeto “AlohaWorld”

⏳ Duração: 2 minutos

Abre o terminal e escreve:

mkdir AlohaWorld  
cd AlohaWorld  
dotnet new blankmaui

🎉 **Boom!** Criaste uma app .NET MAUI com a estrutura base pronta.

## 🧭 Etapa 3: Explorar os ficheiros principais

⏳ Duração: 3 minutos

Pensa numa app como uma peça de teatro:

* 🎭 **MainPage.xaml** – o palco principal (interface)
* 🎬 **App.xaml.cs** – onde ligamos o palco à app
* 🎓 **MauiProgram.cs** – é o encenador, inicia tudo
* 🧩 **Platforms/** – ficheiros específicos para Windows, macOS, Android...

🔎 Vai para o ficheiro MainPage.xaml. Aqui tens:

<ContentPage>  
 <ScrollView>  
 <VerticalStackLayout>  
 <Label Text="Hello, World!" />  
 <Button Text="Click me" Clicked="OnCounterClicked" />  
 </VerticalStackLayout>  
 </ScrollView>  
</ContentPage>

Simples, certo? Já lá está o botão “Click me” e o texto de boas-vindas.

## ▶️ Etapa 4: Executar a app

⏳ Duração: 3 minutos

Agora vem a magia: correr a app. No terminal, escreve:

dotnet build -t:Run -f:net6.0-[plataforma]

Exemplo:

* Para macOS: dotnet build -t:Run -f:net6.0-maccatalyst
* Para Android: dotnet build -t:Run -f:net6.0-android

⚠️ A plataforma tem de ser suportada pelo teu sistema operativo!

## 🔄 Etapa 5: Modificar o texto com Hot Reload (versão VS Code não suporta!)

**Nota:** O Hot Reload só funciona com Visual Studio (não VS Code), mas podes fazer alterações e correr novamente o build para ver as mudanças.

Exemplo:

1. Abre MainPage.xaml
2. Muda:

Text="Hello, World!"

para:

Text="Aloha, World!"

1. Guarda, volta ao terminal, e corre novamente:

dotnet build -t:Run -f:net6.0-[plataforma]

## 🧠 Etapa 6: Entender como tudo se liga (sem dor de cabeça)

Pensa assim:

🧱 **MauiProgram.cs** cria a app →  
🚪 **App.xaml.cs** define a página inicial (MainPage) →  
📄 **MainPage.xaml** define o que vês na janela →  
🖱️ Clicas num botão → contador aumenta → 🎉 sucesso!

## ✨ Etapa extra: Personaliza com o teu bot

1. Vai a <https://mod-dotnet-bot.net/> e cria o teu bot.
2. Faz download da imagem.
3. Mete no diretório Resources/Images.
4. No XAML, muda:

Source="dotnet\_bot.png"

para o nome do teu bot.

## ✅ Recapitulando

* Criaste um projeto com .NET CLI
* Exploraste a estrutura do código
* Executaste a app no teu sistema
* Alteraste o layout (e viste o resultado!)
* Percebeste como o .NET MAUI estrutura apps com clareza

# 📲 Parte 2 – Interação e persistência: contador com base de dados local (SQLite)

## 🎯 Objetivo de aprendizagem

No final desta parte, vais saber como:

* Ligar a tua app .NET MAUI a uma base de dados local.
* Guardar e ler dados de forma persistente com SQLite.
* Atualizar a interface para mostrar e guardar o número de cliques num botão (mesmo depois de fechar a app!).

## 📦 Etapa 1: Adicionar SQLite ao projeto

⏳ Duração: 3 minutos

Abre o terminal na pasta do teu projeto e executa:

dotnet add package sqlite-net-pcl

Este pacote vai permitir comunicar com uma base de dados SQLite local, de forma super simples, como se estivesses a usar classes C# normais.

## 🧱 Etapa 2: Criar o modelo de dados

🧠 Ponto de aprendizagem: criar uma "classe" que representa a tabela da base de dados.

Cria uma nova pasta chamada Models. Dentro dela, cria um ficheiro chamado Counter.cs com o seguinte conteúdo:

using SQLite;  
  
namespace AlohaWorld.Models  
{  
 public class Counter  
 {  
 [PrimaryKey, AutoIncrement]  
 public int Id { get; set; }  
  
 public int Value { get; set; }  
 }  
}

✍️ Esta classe é como uma folha de Excel com duas colunas: Id e Value.

## 🛠️ Etapa 3: Criar o serviço de base de dados

Cria uma nova pasta chamada Services. Dentro dela, cria um ficheiro chamado DatabaseService.cs:

using SQLite;  
using AlohaWorld.Models;  
  
namespace AlohaWorld.Services  
{  
 public class DatabaseService  
 {  
 private readonly SQLiteAsyncConnection \_db;  
  
 public DatabaseService(string dbPath)  
 {  
 \_db = new SQLiteAsyncConnection(dbPath);  
 \_db.CreateTableAsync<Counter>().Wait();  
 }  
  
 public async Task<int> GetCounterValueAsync()  
 {  
 var counter = await \_db.Table<Counter>().FirstOrDefaultAsync();  
 return counter?.Value ?? 0;  
 }  
  
 public async Task SaveCounterValueAsync(int value)  
 {  
 var counter = await \_db.Table<Counter>().FirstOrDefaultAsync();  
 if (counter == null)  
 await \_db.InsertAsync(new Counter { Value = value });  
 else  
 {  
 counter.Value = value;  
 await \_db.UpdateAsync(counter);  
 }  
 }  
 }  
}

🧠 Este serviço faz 2 coisas: **lê** o valor do contador e **guarda** o novo valor.

## 🔌 Etapa 4: Ligar o serviço à tua aplicação

Abre o ficheiro MauiProgram.cs e altera-o para injetar o serviço no arranque da app:

using AlohaWorld.Services;  
  
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");  
 });  
  
 string dbPath = Path.Combine(FileSystem.AppDataDirectory, "app.db");  
 builder.Services.AddSingleton(new DatabaseService(dbPath));  
  
 return builder.Build();  
 }  
}

🧠 Aqui estamos a dizer: "Este é o caminho para o ficheiro da base de dados. Guarda lá os dados e fornece este serviço à app toda."

## 🎬 Etapa 5: Atualizar o MainPage.xaml.cs

Agora, vamos ligar o botão ao serviço para que o valor fique guardado.

Abre o ficheiro MainPage.xaml.cs e muda-o para isto:

using AlohaWorld.Services;  
  
namespace AlohaWorld;  
  
public partial class MainPage : ContentPage  
{  
 private int count = 0;  
 private readonly DatabaseService \_db;  
  
 public MainPage(DatabaseService db)  
 {  
 InitializeComponent();  
 \_db = db;  
 LoadCounter();  
 }  
  
 private async void LoadCounter()  
 {  
 count = await \_db.GetCounterValueAsync();  
 CounterLabel.Text = $"Current count: {count}";  
 }  
  
 private async void OnCounterClicked(object sender, EventArgs e)  
 {  
 count++;  
 CounterLabel.Text = $"Current count: {count}";  
 await \_db.SaveCounterValueAsync(count);  
 }  
}

🧠 Usamos o método LoadCounter ao arrancar para **mostrar o último valor guardado**. Ao clicar no botão, guardamos o novo valor.

## ▶️ Etapa 6: Executar e testar

1. Compila e executa novamente a app:

dotnet build -t:Run -f:net6.0-[plataforma]

1. Clica no botão algumas vezes e **fecha a app**.
2. Reabre e… voilá! 🎉 O contador continua do mesmo valor!

## 🧠 Dica de Psicologia Cognitiva

Este tutorial está organizado para respeitar os princípios de:

* **Segmentação**: pequenas etapas para facilitar a assimilação.
* **Aprendizagem ativa**: escreves código em vez de apenas ler.
* **Ligação com conhecimento anterior**: expandimos o botão do "Hello World!" para algo mais útil.

## ✅ Conclusão desta parte

* Aprendeste a guardar dados localmente com SQLite.
* A tua app agora **lembra-se** do utilizador!
* Ganhaste uma nova competência que podes aplicar em qualquer app: **persistência de dados**.

# 📱 Parte 3 – Apps interativas com .NET MAUI no VS Code

## 🎯 Objetivo

Neste tutorial vais aprender a:

* Aceder à **localização do utilizador**
* Partilhar dados com outras apps (como redes sociais ou e-mail)
* **Guardar dados localmente**, com ou sem encriptação
* Criar **listas de tarefas** com base de dados
* Usar **data binding** para ligar a interface ao código

Tudo isto de forma simples e prática. Vamos a isso? 😉

## 🧭 Projeto 1 – **FindMe!**: Partilha a tua localização

### 🛠️ Etapa 1: Criar a app

No terminal do VS Code, escreve:

dotnet new blankmaui -n FindMe  
cd FindMe

### 🔒 Etapa 2: Pedir permissões de localização

Em apps móveis, tens sempre de pedir permissão ao utilizador para aceder à localização.

#### 🟢 Android

Abre o ficheiro: Platforms/Android/AndroidManifest.xml  
Adiciona estas linhas **antes de </manifest>**:

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
<uses-feature android:name="android.hardware.location" android:required="false" />

#### 🍏 iOS/macOS

Abre: Platforms/iOS/Info.plist e Platforms/MacCatalyst/Info.plist  
Adiciona isto dentro de <dict>:

<key>NSLocationWhenInUseUsageDescription</key>  
<string>Precisamos da tua localização para poderes partilhá-la com outras pessoas.</string>

#### 🪟 Windows

Abre: Platforms/Windows/Package.appxmanifest  
Adiciona dentro de <Capabilities>:

<DeviceCapability Name="location" />

### ✏️ Etapa 3: Atualizar o layout (MainPage.xaml)

Substitui o conteúdo de <ScrollView> por este:

<Entry x:Name="UsernameEntry"  
 Placeholder="Escreve o teu nome"  
 SemanticProperties.Hint="Nome a partilhar com a localização"  
 HorizontalOptions="Center" />  
  
<Button Text="Encontra-me!"  
 Clicked="OnFindMeClicked"  
 SemanticProperties.Hint="Partilhar localização"  
 HorizontalOptions="Center" />  
  
<Image Source="location\_pin.png"  
 WidthRequest="100"  
 HeightRequest="100"  
 HorizontalOptions="Center" />

⚠️ Não te esqueças de colocar uma imagem chamada location\_pin.png na pasta Resources/Images.

### 💻 Etapa 4: Escrever o código (MainPage.xaml.cs)

Substitui o conteúdo por este:

using Microsoft.Maui.ApplicationModel;  
using Microsoft.Maui.Controls;  
using Microsoft.Maui.Essentials;  
  
namespace FindMe;  
  
public partial class MainPage : ContentPage  
{  
 string baseUrl = "https://bing.com/maps/default.aspx?cp=";  
 public string UserName { get; set; }  
  
 public MainPage()  
 {  
 InitializeComponent();  
 }  
  
 private async void OnFindMeClicked(object sender, EventArgs e)  
 {  
 var permission = await Permissions.CheckStatusAsync<Permissions.LocationWhenInUse>();  
  
 if (permission != PermissionStatus.Granted)  
 permission = await Permissions.RequestAsync<Permissions.LocationWhenInUse>();  
  
 if (permission == PermissionStatus.Granted)  
 {  
 await ShareLocation();  
 }  
 else  
 {  
 await DisplayAlert("Permissões", "Precisamos da tua localização para partilhar!", "OK");  
 }  
 }  
  
 private async Task ShareLocation()  
 {  
 UserName = UsernameEntry.Text;  
  
 var locationRequest = new GeolocationRequest(GeolocationAccuracy.Best);  
 var location = await Geolocation.GetLocationAsync(locationRequest);  
  
 await Share.RequestAsync(new ShareTextRequest  
 {  
 Title = "Encontra-me!",  
 Text = $"{UserName} está aqui!",  
 Uri = $"{baseUrl}{location.Latitude}~{location.Longitude}"  
 });  
 }  
}

### ▶️ Etapa 5: Correr a app

dotnet build -t:Run -f:net6.0-[plataforma]

Substitui [plataforma] por android, maccatalyst, etc.

✅ Pronto! Agora podes partilhar a tua localização com qualquer app no dispositivo.

## ✅ O que aprendeste até agora

* A criar um formulário básico
* A pedir **permissões de localização**
* A usar a **API de localização e partilha**
* A trabalhar com **várias plataformas** usando o mesmo código

## 📋 Projeto 2 – **MauiTodo**: A tua lista de tarefas segura

Este projeto vai ensinar-te:

* A **guardar dados** com base de dados local (SQLite)
* A usar **data binding**
* A criar **listas dinâmicas e seguras**

### 🛠️ Etapa 1: Criar a app

dotnet new blankmaui -n MauiTodo  
cd MauiTodo

### 📦 Etapa 2: Instalar as bibliotecas necessárias

dotnet add package sqlite-net-pcl  
dotnet add package sqlite-net-sqlcipher

### 🧱 Etapa 3: Modelo de dados

Cria a pasta Models, com o ficheiro TodoItem.cs:

namespace MauiTodo.Models;  
  
public class TodoItem  
{  
 public int Id { get; set; }  
 public string Title { get; set; }  
 public DateTime Due { get; set; }  
 public bool Done { get; set; } = false;  
}

### 🔐 Etapa 4: Serviço de base de dados

Na pasta Data, cria Database.cs:

using SQLite;  
using MauiTodo.Models;  
  
namespace MauiTodo.Data;  
  
public class Database  
{  
 private readonly SQLiteAsyncConnection \_connection;  
  
 public Database()  
 {  
 var dbPath = Path.Combine(FileSystem.AppDataDirectory, "MauiTodo.db");  
 var key = SecureStorage.GetAsync("dbKey").Result ?? Guid.NewGuid().ToString();  
 SecureStorage.SetAsync("dbKey", key);  
 var dbOptions = new SQLiteConnectionString(dbPath, true, key: key);  
 \_connection = new SQLiteAsyncConnection(dbOptions);  
 \_ = Init();  
 }  
  
 private async Task Init() =>  
 await \_connection.CreateTableAsync<TodoItem>();  
  
 public Task<List<TodoItem>> GetTodos() =>  
 \_connection.Table<TodoItem>().ToListAsync();  
  
 public Task<int> AddTodo(TodoItem item) =>  
 \_connection.InsertAsync(item);  
}

### 🖼️ Etapa 5: Interface (MainPage.xaml)

Substitui tudo por:

<Grid Padding="20" RowDefinitions="Auto, Auto, Auto, \*, \*">  
 <Entry x:Name="TitleEntry" Placeholder="Nova tarefa" Grid.Row="0" />  
 <DatePicker x:Name="DuePicker" Grid.Row="1" />  
 <Button Text="Adicionar" Clicked="OnAddClicked" Grid.Row="2" />  
 <CollectionView x:Name="TodosView" Grid.Row="3">  
 <CollectionView.ItemTemplate>  
 <DataTemplate>  
 <StackLayout>  
 <Label Text="{Binding Title}" FontAttributes="Bold"/>  
 <Label Text="{Binding Due, StringFormat='Data: {0:dd/MM/yyyy}'}"/>  
 </StackLayout>  
 </DataTemplate>  
 </CollectionView.ItemTemplate>  
 </CollectionView>  
</Grid>

### 🔗 Etapa 6: Código e ligação (MainPage.xaml.cs)

using MauiTodo.Data;  
using MauiTodo.Models;  
using System.Collections.ObjectModel;  
  
namespace MauiTodo;  
  
public partial class MainPage : ContentPage  
{  
 readonly Database \_db = new();  
 public ObservableCollection<TodoItem> Todos { get; set; } = new();  
  
 public MainPage()  
 {  
 InitializeComponent();  
 TodosView.ItemsSource = Todos;  
 \_ = LoadTodos();  
 }  
  
 private async Task LoadTodos()  
 {  
 var items = await \_db.GetTodos();  
 foreach (var item in items)  
 Todos.Add(item);  
 }  
  
 private async void OnAddClicked(object sender, EventArgs e)  
 {  
 var todo = new TodoItem  
 {  
 Title = TitleEntry.Text,  
 Due = DuePicker.Date  
 };  
  
 if (await \_db.AddTodo(todo) != 0)  
 {  
 Todos.Add(todo);  
 TitleEntry.Text = string.Empty;  
 DuePicker.Date = DateTime.Today;  
 }  
 }  
}

## 🧠 Dicas de aprendizagem

* **Repetição com variação**: Ao fazermos dois projetos com partes semelhantes (botões, campos de texto), consolidamos os conceitos com novas aplicações.
* **Carga cognitiva reduzida**: Cada projeto tem blocos pequenos, com código e UI separados, explicados passo a passo.
* **Associação visual**: Usamos interfaces com labels, entradas e listas para facilitar a associação visual com o comportamento esperado.

## 🎓 Conclusão

🚀 Com estes dois projetos aprendeste:

* A **ligar a tua app a APIs do sistema** (localização e partilha)
* A **guardar dados localmente**, incluindo com encriptação
* A construir UIs interativas com **data binding**
* A organizar código em ficheiros e pastas de forma clara

# 🧩 Parte 4 – Controlo Total: Criar interfaces modernas com .NET MAUI

🎯 Objetivo: aprender a construir interfaces visuais com **controles reutilizáveis**, **listas dinâmicas**, **modificadores visuais** e **gestos** (como "swipe" ou "tap") — tudo com base em **VS Code** e com estilo!

## 📌 1. O que são “Views” e “Controls”?

🧠 Analogia: imagina que a tua app é uma casa.

* **Page** = divisão (sala, cozinha…)
* **Layout** = móveis (definem como os objetos estão organizados)
* **Controls** = objetos da casa (botões, imagens, caixas de texto...)

### ✅ Tipos de “Views”:

| Tipo | Exemplo |
| --- | --- |
| Page | MainPage, LoginPage |
| Layout | VerticalStackLayout, Grid |
| Control | Button, Label, Entry |

## 🖼️ 2. Controles visuais (só mostram, não interagem)

Estes são perfeitos para **apresentar informação** ao utilizador:

| Controlo | Para quê? |
| --- | --- |
| Label | Mostrar texto |
| ProgressBar | Mostrar progresso numérico (0 a 1) |
| ActivityIndicator | Mostrar que algo está a acontecer |

<Label Text="Olá, Afonso!" />  
<ProgressBar Progress="0.5" />  
<ActivityIndicator IsRunning="true" />

## ⌨️ 3. Controles interativos (recebem input do utilizador)

🧠 Regra de ouro: se o utilizador **toca ou escreve**, é input!

| Controlo | Usado para... |
| --- | --- |
| Entry | Texto simples (1 linha) |
| Editor | Texto longo (várias linhas) |
| CheckBox | Verdadeiro/falso |
| DatePicker | Escolher data |
| Slider | Escolher valor com deslize |
| Stepper | Incrementar/decrementar |
| TimePicker | Escolher hora |
| Picker | Selecionar de uma lista |
| RadioButton | Escolher uma opção |

<Entry Placeholder="Escreve o teu nome" />  
<CheckBox IsChecked="true" />  
<DatePicker />

## 🖱️ 4. Controles de ação (executam comandos)

Estes não pedem valores — apenas **disparam ações**.

| Controlo | O que faz? |
| --- | --- |
| Button | Ação com texto |
| ImageButton | Ação com imagem |
| SearchBar | Caixa de busca |

<Button Text="Enviar" Clicked="OnEnviarClicked" />  
<ImageButton Source="enviar.png" Clicked="OnEnviarClicked" />

## 🎨 5. Mostrar imagens e gráficos

| Controlo | Para quê? |
| --- | --- |
| Image | Mostrar imagens |
| Shapes | Desenhos simples (ex: círculo) |
| GraphicsView | Desenhos avançados |

<Image Source="logo.png" WidthRequest="100" />

## 📚 6. Listas e coleções dinâmicas

### a) CollectionView – o mais usado!

Lista de itens com layout personalizado.

<CollectionView ItemsSource="{Binding Produtos}">  
 <CollectionView.ItemTemplate>  
 <DataTemplate>  
 <Label Text="{Binding Nome}" />  
 </DataTemplate>  
 </CollectionView.ItemTemplate>  
</CollectionView>

### b) CarouselView – tipo "carrossel"

<CarouselView ItemsSource="{Binding Imagens}">  
 <CarouselView.ItemTemplate>  
 <DataTemplate>  
 <Image Source="{Binding Url}" />  
 </DataTemplate>  
 </CarouselView.ItemTemplate>  
</CarouselView>

### c) ListView – versão antiga, menos flexível

### d) TableView – ideal para **definições e formulários**

## 🧰 7. Personalizações com “modificadores”

🧠 Pensar em camadas: podes **envolver** um controlo para o modificar.

### ✂️ Clip (recorte de forma)

<Image.Clip>  
 <EllipseGeometry Center="150,200" RadiusX="150" RadiusY="200" />  
</Image.Clip>

🎯 Resultado: a imagem aparece dentro de um círculo.

### ⬛ Border (borda)

<Border Stroke="Black" StrokeThickness="3">  
 <Label Text="Com borda" />  
</Border>

### ☁️ Shadow (sombra)

<Border.Shadow>  
 <Shadow Brush="Black" Offset="10,10" Radius="20" Opacity="0.5"/>  
</Border.Shadow>

### ✋ GestureRecognizers (toques e gestos)

<Image>  
 <Image.GestureRecognizers>  
 <TapGestureRecognizer Tapped="OnImagemTocada" />  
 </Image.GestureRecognizers>  
</Image>

### 🔄 RefreshView (puxar para atualizar)

<RefreshView IsRefreshing="{Binding AAtualizar}" Command="{Binding AtualizarDadosCommand}">  
 <CollectionView ItemsSource="{Binding Dados}" />  
</RefreshView>

### 👈 SwipeView (deslizar para ações)

<SwipeView>  
 <SwipeView.RightItems>  
 <SwipeItems Mode="Execute">  
 <SwipeItem Text="Apagar" BackgroundColor="Tomato" Invoked="ApagarTarefa" />  
 </SwipeItems>  
 </SwipeView.RightItems>  
 <Label Text="Tarefa 1" />  
</SwipeView>

## 🧠 Psicologia aplicada ao design deste tutorial:

* **Segmentação**: Cada tema dividido por bloco para facilitar o foco.
* **Exemplos mínimos**: Mostramos apenas o necessário para ver o efeito.
* **Aprendizagem ativa**: Estás sempre a experimentar com código real.
* **Andaimagem cognitiva**: começamos simples e vamos adicionando camadas.

## ✅ O que aprendeste

* Diferença entre Page, Layout e Control
* Usar os controlos visuais mais comuns
* Trabalhar com listas dinâmicas (CollectionView, CarouselView)
* Modificar a interface com Clip, Border, Shadow
* Adicionar gestos, "swipes", e atualizações com RefreshView e SwipeView

# 🧱 Parte 5 – Layouts em .NET MAUI: organiza o teu mundo visual

🎯 Objetivo: aprender a **organizar elementos numa app** com os diferentes tipos de layout, criando UIs bonitas, funcionais e adaptáveis a qualquer ecrã.

## 🧠 Antes de mais… o que é um “Layout”?

Imagina que estás a montar uma casa:

* **Controls** são os móveis (botões, caixas de texto…)
* **Layouts** são o chão onde colocas os móveis — organizam o espaço
* Sem um bom layout, tudo fica desarrumado (mesmo que os móveis sejam bonitos)

## 📐 Tipos principais de layout em .NET MAUI

| Layout | Para quê? |
| --- | --- |
| Grid | Tabelas invisíveis com linhas e colunas |
| VerticalStackLayout | Empilhar elementos de cima para baixo |
| HorizontalStackLayout | Colocar elementos lado a lado |
| ScrollView | Permitir scroll de conteúdo |

Vamos explorar **cada um** com exemplos simples, sempre a pensar em reduzir a complexidade ✨

## 🧩 1. Grid – o mais poderoso

Ideal para UIs organizadas por linhas e colunas (ex: calculadoras, ecrãs de login)

### 🛠️ Exemplo prático: Mini-Calculadora

<Grid ColumnDefinitions="\*,\*,\*,\*"  
 RowDefinitions="4\*,\*,\*,\*,\*"  
 ColumnSpacing="2"  
 RowSpacing="2">  
   
 <!-- Ecrã -->  
 <Label Grid.Row="0" Grid.ColumnSpan="4"  
 FontSize="40" Text="123"   
 BackgroundColor="Beige"  
 HorizontalTextAlignment="End"  
 Padding="10"/>  
  
 <!-- Botões -->  
 <Button Grid.Row="1" Grid.Column="0" Text="7" />  
 <Button Grid.Row="1" Grid.Column="1" Text="8" />  
 <Button Grid.Row="1" Grid.Column="2" Text="9" />  
 <Button Grid.Row="1" Grid.Column="3" Text="+" />  
</Grid>

🧠 **Dica cognitiva**: usa sempre rascunhos antes de codificar em Grid. Visualiza o “tabuleiro” com linhas e colunas numeradas. Simplifica muito!

## 📏 2. StackLayouts – empilha com simplicidade

### a) VerticalStackLayout

Empilha de cima para baixo

<VerticalStackLayout Spacing="10">  
 <Label Text="Nome:" />  
 <Entry Placeholder="Escreve aqui..." />  
 <Button Text="Submeter" />  
</VerticalStackLayout>

### b) HorizontalStackLayout

Coloca lado a lado

<HorizontalStackLayout Spacing="10">  
 <Label Text="Dia:" />  
 <DatePicker />  
</HorizontalStackLayout>

🎯 **Quando usar StackLayouts?**

* Prototipagem rápida
* Formularios simples
* Agrupar pequenos conjuntos de elementos

## 📜 3. ScrollView – quando há conteúdo a mais

Permite que o utilizador faça scroll (ex: textos longos, listas de cartões)

<ScrollView>  
 <VerticalStackLayout Spacing="15">  
 <Label Text="Capítulo 1: A História do Universo" />  
 <Label Text="(texto muito longo aqui...)" />  
 <Label Text="Capítulo 2: Buracos Negros" />  
 <!-- E por aí fora -->  
 </VerticalStackLayout>  
</ScrollView>

🎯 Ótimo para:

* Biografias
* Termos e condições
* Apps que têm de se adaptar a **muitos tamanhos de ecrã**

## 🎯 Dica visual: Combina layouts!

Podes colocar StackLayouts dentro de Grids, ou ScrollViews a envolver tudo.  
**Exemplo típico**:

<ScrollView>  
 <VerticalStackLayout Spacing="20">  
 <Image Source="perfil.png" />  
   
 <Grid ColumnDefinitions="\*,\*">  
 <Label Text="Nome:" />  
 <Entry />  
 </Grid>  
  
 <Button Text="Guardar" />  
 </VerticalStackLayout>  
</ScrollView>

## 🧠 Psicologia cognitiva aplicada neste tutorial:

| Estratégia | Como foi usada |
| --- | --- |
| **Segmentação** | Divisão clara entre tipos de layout |
| **Andaimagem** | Começámos com o mais simples e adicionámos complexidade aos poucos |
| **Repetição com variação** | Vários exemplos de StackLayouts com funções diferentes |
| **Exemplos visuais** | Todos os blocos de código com função visível clara |

## ✅ O que aprendeste nesta parte

* O que são layouts e para que servem
* Como usar Grid, VerticalStackLayout, HorizontalStackLayout e ScrollView
* Como posicionar elementos com Row, Column, ColumnSpan
* Combinar layouts para interfaces mais ricas
* Tornar a UI adaptável a diferentes tamanhos de ecrã

## 🚀 Desafio prático para ti\*

Cria uma app chamada **MiniPerfil** com:

* Foto no topo
* Campos de nome, e-mail e data de nascimento
* Botão “Guardar”
* Tudo dentro de um ScrollView com VerticalStackLayout

\* Solução no GitHub (link no final do documento) 😉

# 🎯 Parte 6 – Layouts Avançados em .NET MAUI: flexibilidade máxima na tua app

## 🧠 Objetivo

Neste tutorial, vais aprender a:

* Usar **Grid** para layouts complexos
* Transformar qualquer layout numa lista dinâmica com **BindableLayout**
* Posicionar elementos com precisão com **AbsoluteLayout**
* Combinar tudo isso para criar **UIs reais**, como a do Outlook

## 🧩 1. Pensar como um Grid

🎨 Dica visual: pensa na tua app como uma folha quadriculada.

### Exemplo prático: ecrã de perfil

Tens 3 zonas principais:

* Cabeçalho (nome, botões)
* Detalhes (foto, resumo de skills)
* Biografia

🔹 Criamos 3 **linhas** com alturas relativas: 2\*, 9\*, 3\*  
🔹 A linha de topo usa 3 **colunas**: \*, 5\*, \*

<Grid RowDefinitions="2\*,9\*,3\*"  
 ColumnDefinitions="\*,5\*,\*">  
 <!-- Cabeçalho, corpo e bio -->  
</Grid>

🧠 **Dica cognitiva:** Reutiliza o Grid até em UIs que não parecem uma grelha — vais ganhar **precisão** sem perder **simplicidade**.

## 🔄 2. BindableLayout – listas sem CollectionView

✨ Permite transformar qualquer layout (ex: Stack, Flex) num **layout dinâmico** com dados.

### Exemplo com FlexLayout – app de filmes

<FlexLayout BindableLayout.ItemsSource="{Binding Filmes}" Wrap="Wrap">  
 <BindableLayout.ItemTemplate>  
 <DataTemplate>  
 <VerticalStackLayout>  
 <Image Source="{Binding PosterUrl}" />  
 <Label Text="{Binding Titulo}" />  
 <Label Text="{Binding Rating, StringFormat='{0} ⭐'}" />  
 </VerticalStackLayout>  
 </DataTemplate>  
 </BindableLayout.ItemTemplate>  
</FlexLayout>

📌 Ideal para mostrar "chips", "tags", ou pequenos cartões!

❗ Usa CollectionView se precisares de seleção, scroll ou interações complexas.

## 📍 3. AbsoluteLayout – quando queres um botão “flutuante” (FAB)

🧠 Pensa como se estivesses a colocar autocolantes num ecrã com coordenadas XY.

### Exemplo – botão no canto inferior direito:

<AbsoluteLayout>  
 <ScrollView>  
 <!-- Conteúdo principal -->  
 </ScrollView>  
  
 <Button Text="+"   
 WidthRequest="60" HeightRequest="60" CornerRadius="30"  
 BackgroundColor="DodgerBlue"  
 AbsoluteLayout.LayoutBounds="0.9,0.9,60,60"  
 AbsoluteLayout.LayoutFlags="PositionProportional" />  
</AbsoluteLayout>

🎯 O botão fica sempre a 90% da largura e altura do ecrã.

❗ Usa com moderação — se exagerares, podes perder responsividade.

## 🧑‍🎨 4. Exemplo completo: App de filmes interativa (MauiMovies)

### Funcionalidades:

* Mostra filmes da API [themoviedb.org](https://themoviedb.org)
* Filtro por género
* Detalhes em popups

### Ferramentas usadas:

* BindableLayout + FlexLayout para mostrar géneros e filmes
* CollectionView para popups
* CommunityToolkit.Maui para popups personalizados

### 🧠 Psicologia aplicada:

* Repetição com variação (usamos Flex em 2 contextos)
* Minimização de distrações (cada componente está isolado)
* Feedback imediato (popups, selecção visual)

## ✅ O que levas deste tutorial

* Grid não é só para grelhas quadradas – é versátil!
* BindableLayout permite listas leves sem CollectionView
* AbsoluteLayout dá-te controlo absoluto de posição
* Combinar layouts é a chave para criar UIs reais

# 🧭 Parte 7 – Páginas e Navegação em .NET MAUI (com VS Code)

🎯 Objetivo: Aprender a criar várias páginas numa app .NET MAUI e navegar entre elas de forma simples e moderna com **Shell**, **tabs**, **flyout menus** e **passagem de dados**.

## 🧠 1. Por que precisamos de várias páginas?

Imagina uma app com:

* Um **ecrã de login**
* Uma **lista de produtos**
* Um **relatório de inventário**

Seria caótico tentar meter tudo numa só página! Usamos **páginas separadas** para manter a clareza e tornar a navegação natural para o utilizador.

## 📄 2. ContentPage – a página base

Todas as páginas derivam de ContentPage. Ela tem uma **propriedade Content**, onde colocamos o layout principal da página.

### Exemplo simples:

<ContentPage ... Title="Página de Exemplo">  
 <VerticalStackLayout Padding="30">  
 <Label Text="Bem-vindo à app!" />  
 </VerticalStackLayout>  
</ContentPage>

🧠 Ponto importante: sempre que possível, usa um Layout (como StackLayout, Grid, etc.) como conteúdo da página — não controles isolados como só um botão.

## 🧭 3. Tipos de navegação em .NET MAUI

| Tipo de navegação | Descrição |
| --- | --- |
| **Hierárquica** | Pilha de páginas. Ex: botão “voltar”. |
| **Flyout** | Menu lateral (hambúrguer) com links para páginas. |
| **Tabbed** | Abas com ícones/texto, como nas apps de redes sociais. |
| **Shell (💎)** | Abstração que simplifica todos os modelos anteriores. |

Vamos focar-nos no **Shell**, que é mais simples, poderoso e moderno para apps multi-página.

## 🛠️ 4. Começar com Shell

### a) Cria um novo projeto com Shell (no terminal):

dotnet new maui -n MauiStockTake  
cd MauiStockTake

Este projeto já vem com:

* AppShell.xaml → estrutura de navegação
* MainPage.xaml → a página de arranque

## 📑 5. Criar várias páginas

Cria uma pasta chamada Pages com estas páginas:

* LoginPage.xaml
* InputPage.xaml
* ReportPage.xaml

Cada uma com um VerticalStackLayout e um Label com o nome da página.

<!-- Exemplo: InputPage.xaml -->  
<ContentPage Title="Input">  
 <VerticalStackLayout>  
 <Label Text="Página de Inventário" />  
 </VerticalStackLayout>  
</ContentPage>

## 🧭 6. Navegação com Shell: TabBar + Flyout

### a) AppShell.xaml (estrutura da app)

<Shell  
 x:Class="MauiStockTake.AppShell"  
 xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
 xmlns:pages="clr-namespace:MauiStockTake.Pages"  
 FlyoutBehavior="Flyout">  
  
 <!-- Menu lateral com botão de logout -->  
 <Shell.FlyoutHeader>  
 <Label Text="Bem-vindo à MauiStockTake!" FontSize="24" Padding="20"/>  
 </Shell.FlyoutHeader>  
  
 <MenuItem Text="Logout" IconImageSource="logout.png" />  
  
 <!-- Barra de abas -->  
 <TabBar>  
 <Tab Title="Inventário" Icon="icon\_input.svg" Route="input">  
 <ShellContent ContentTemplate="{DataTemplate pages:InputPage}" />  
 </Tab>  
 <Tab Title="Relatório" Icon="icon\_report.svg" Route="report">  
 <ShellContent ContentTemplate="{DataTemplate pages:ReportPage}" />  
 </Tab>  
 </TabBar>  
</Shell>

🧠 Psicologia: a estrutura está dividida em partes lógicas e repetidas, facilitando a memorização e identificação de padrões.

## 🔄 7. Navegação programática (GoToAsync)

### a) Para ir da InputPage para uma nova página (ex: detalhes):

await Shell.Current.GoToAsync("productdetails", new Dictionary<string, object>  
{  
 { "Product", myProduct }  
});

### b) Registar a página de destino (ex: ProductPage)

Routing.RegisterRoute("productdetails", typeof(ProductPage));

## 📦 8. Passar dados entre páginas com Shell

### a) Decorar a página destino com [QueryProperty]

[QueryProperty(nameof(Product), "Product")]  
public partial class ProductPage : ContentPage  
{  
 public Product Product { get; set; }  
}

🧠 Excelente exemplo de andaimagem: primeiro aprendes a navegar sem dados, depois com dados simples, e agora com objetos complexos!

## 🔁 9. Ciclo de vida das páginas

| Método | Quando acontece |
| --- | --- |
| OnAppearing() | Quando a página **entra em cena** |
| OnDisappearing() | Quando **sai de cena** |
| OnNavigatedTo() | Antes de aparecer |
| OnBackButtonPressed() | Ao carregar no botão “voltar” |

### Exemplo:

protected override async void OnAppearing()  
{  
 base.OnAppearing();  
 var items = await db.GetInventoryItems();  
 InventoryList.ItemsSource = items;  
}

## 📌 10. Resumo (Resumo = retenção 💪)

* .NET MAUI suporta **páginas múltiplas** com ContentPage
* **Shell** simplifica navegação com **Tabs**, **Flyouts** e **rotas**
* Podes navegar com GoToAsync() e passar **dados**
* Usa [QueryProperty] para receber parâmetros
* Aproveita os **métodos de ciclo de vida** para carregar dados ou limpar recursos

## 🎯 Desafio para fixar\*

Cria uma app chamada MiniGestor com:

* Página de Login
* Página de Lista de Tarefas
* Página de Estatísticas
* Usa Shell com Tabs e um botão de logout no menu lateral
* Ao clicar numa tarefa, abre uma página de detalhes com os dados da tarefa

\* Solução no GitHub (link no final do documento) 😉

# 🏢 Parte 8 – Desenvolver Apps Empresariais com .NET MAUI

## 🎯 Objetivo

Aprender a construir **apps robustas, escaláveis e profissionais**, usando práticas como:

* Separar lógica da interface
* Usar serviços (Services) para funcionalidades
* Fazer login com autenticação OAuth2 (ex: IdentityServer)
* Usar **injeção de dependência** e o **host builder**
* Ligar a APIs seguras com tokens (JWT)
* Estruturar apps em soluções empresariais reutilizáveis

## 🧠 1. Por que separar a lógica da interface?

Analogia: UI = volante do carro | Serviço = motor

Colocar toda a lógica no *code-behind* das páginas torna a app difícil de manter. Em vez disso:

* Criamos **interfaces** que definem o que queremos fazer
* Implementamos isso em **serviços reutilizáveis**
* A UI apenas “pede” ao serviço que faça o trabalho

## 🛠️ 2. Exemplo: Página de Login com serviço de autenticação

### a) Criar interface IAuthService

public interface IAuthService  
{  
 Task<bool> LoginAsync();  
}

🧠 Aqui definimos o que queremos (login), sem dizer como.

### b) Criar implementação mock: MockAuthService

public class MockAuthService : IAuthService  
{  
 public Task<bool> LoginAsync() => Task.FromResult(true);  
}

Útil para testar sem ainda termos a API ou autenticação real pronta.

### c) Ligar à UI (LoginPage.xaml.cs)

private readonly IAuthService \_authService = new MockAuthService();  
  
private async void LoginButton\_Clicked(object sender, EventArgs e)  
{  
 LoginButton.IsEnabled = false;  
 LoggingIn.IsVisible = true;  
  
 var loggedIn = await \_authService.LoginAsync();  
  
 LoggingIn.IsVisible = false;  
  
 if (loggedIn)  
 await Navigation.PopModalAsync();  
 else  
 await DisplayAlert("Erro", "Não foi possível iniciar sessão.", "OK");  
}

## 🔐 3. Fazer login real com IdentityServer

### a) Fluxo OAuth2 simplificado:

1. Abrimos o navegador com WebAuthenticator
2. O utilizador faz login
3. O IDP devolve um **token JWT**
4. O token é usado para chamadas à API

### b) Criar AuthService com OidcClient

Instalar o pacote:

dotnet add package IdentityModel.OidcClient

### Implementar:

public class AuthService : IAuthService  
{  
 private readonly OidcClient \_client;  
  
 public AuthService(IBrowser browser)  
 {  
 \_client = new OidcClient(new OidcClientOptions  
 {  
 Authority = Constants.AuthorityUri,  
 ClientId = Constants.ClientId,  
 Scope = Constants.Scope,  
 RedirectUri = Constants.RedirectUri,  
 Browser = browser  
 });  
 }  
  
 public async Task<bool> LoginAsync()  
 {  
 var result = await \_client.LoginAsync(new LoginRequest());  
 if (result.IsError) return false;  
  
 AuthHandler.AuthToken = result.AccessToken;  
 return true;  
 }  
}

## 🌐 4. Ligar a APIs seguras com tokens

Usamos um **DelegatingHandler** que coloca o token em todas as chamadas automaticamente.

### a) AuthHandler.cs

public class AuthHandler : DelegatingHandler  
{  
 public static string AuthToken { get; set; }  
  
 protected override async Task<HttpResponseMessage> SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)  
 {  
 request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", AuthToken);  
 return await base.SendAsync(request, cancellationToken);  
 }  
}

## 💉 5. Usar Injeção de Dependência (DI) com o Generic Host Builder

### a) No MauiProgram.cs

using IBrowser = IdentityModel.OidcClient.Browser.IBrowser;  
  
builder.Services.AddSingleton<IBrowser, AuthBrowser>();  
builder.Services.AddSingleton<IAuthService, AuthService>();  
builder.Services.AddTransient<LoginPage>();  
builder.Services.AddApiClientServices(new ApiClientOptions  
{  
 BaseUrl = "https://[o\_teu\_ngrok].ngrok.io"  
});

🧠 Princípio cognitivo: **organização visual** ajuda a memorizar padrões como AddSingleton, AddTransient, etc.

## 🧠 6. Padrões de Arquitetura Profissionais

### Clean Architecture (CA) – Como o projeto está estruturado:

| Camada | O que contém |
| --- | --- |
| **Domain** | Entidades e regras de negócio (sem deps) |
| **Application** | Casos de uso (usa o Domain) |
| **Infrastructure** | Acesso a dados, serviços externos |
| **Presentation** | WebAPI, .NET MAUI, Blazor... |

🎯 O projeto Shared guarda os DTOs que são **partilhados entre frontend e backend**, evitando duplicação de código.

## 🌍 7. Visão “Enterprise” — reutilização máxima

Se quiseres adicionar um site (Blazor) ou app para clientes no futuro, podes **reutilizar tudo**:

* Modelos
* Serviços de acesso à API
* Lógica de autenticação

💡 A app MAUI passa a ser só mais uma “cara” do mesmo cérebro!

## ✅ Resumo Cognitivo

* Separa lógica de negócio da UI (menos frágil, mais reutilizável)
* Usa IAuthService e depois AuthService com OAuth2
* WebAuthenticator + OidcClient = login seguro
* DelegatingHandler adiciona o token automaticamente
* Usa o Generic Host Builder para registar tudo
* Segue princípios de **injeção de dependência**
* Estrutura profissional com **Clean Architecture**
* Reutiliza código entre frontend e backend

## 🚀 Próximo desafio

Criamos juntos uma app “Catálogo de Produtos” com autenticação, listagem protegida por token e serviços reutilizáveis?  
  
\* Solução no GitHub (link no final do documento) 😉

# 🎯 Parte 9 – MVVM em .NET MAUI: apps organizadas, robustas e fáceis de manter

## 🧠 O que é MVVM?

O MVVM é como uma orquestra bem organizada:

* 🎻 **Model** → os instrumentos (regras, dados, lógica do mundo real)
* 👨‍🎤 **View** → o palco (interface visual da app)
* 🧠 **ViewModel** → o maestro (controla o que é mostrado e como)

### Porquê usar MVVM?

* Código mais limpo 🧼
* Mais fácil de manter 🔧
* Mais fácil de testar 🧪
* Separação de responsabilidades (UI ≠ lógica de negócio)

## 🔧 Refatorar a app MauiTodo com MVVM

### 1. Criar a ViewModel

Cria uma nova pasta ViewModels no projeto e um ficheiro MainViewModel.cs.

public class MainViewModel : INotifyPropertyChanged  
{  
 public event PropertyChangedEventHandler PropertyChanged;  
  
 protected void OnPropertyChanged([CallerMemberName] string propertyName = "") =>  
 PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  
}

### 2. Propriedades e comandos

public ObservableCollection<TodoItem> Todos { get; set; } = new();  
public string NewTodoTitle { get; set; }  
public DateTime NewTodoDue { get; set; } = DateTime.Now;  
public ICommand AddTodoCommand { get; set; }  
public ICommand CompleteTodoCommand { get; set; }  
  
private readonly Database \_database;

### 3. Métodos principais

private async Task Initialise()  
{  
 var todos = await \_database.GetTodos();  
 foreach (var todo in todos) Todos.Add(todo);  
}  
  
public async Task AddNewTodo()  
{  
 var todo = new TodoItem { Title = NewTodoTitle, Due = NewTodoDue };  
 var inserted = await \_database.AddTodo(todo);  
  
 if (inserted != 0)  
 {  
 Todos.Add(todo);  
 NewTodoTitle = string.Empty;  
 NewTodoDue = DateTime.Now;  
 OnPropertyChanged(nameof(NewTodoTitle));  
 OnPropertyChanged(nameof(NewTodoDue));  
 }  
}  
  
public async Task CompleteTodo(TodoItem item)  
{  
 await \_database.UpdateTodo(item);  
 OnPropertyChanged(nameof(Todos));  
}

### 4. Construtor da ViewModel

public MainViewModel()  
{  
 \_database = new Database();  
 AddTodoCommand = new Command(async () => await AddNewTodo());  
 CompleteTodoCommand = new Command<TodoItem>(async (item) => await CompleteTodo(item));  
 \_ = Initialise();  
}

## 🖼️ 5. Atualizar a UI (MainPage.xaml)

Usa bindings em vez de lógica no code-behind:

<Entry Text="{Binding NewTodoTitle}" />  
<DatePicker Date="{Binding NewTodoDue}" />  
<Button Text="Adicionar" Command="{Binding AddTodoCommand}" />  
<CollectionView ItemsSource="{Binding Todos}">  
 <!-- Template dos itens -->  
</CollectionView>

## 💡 Resolver a limitação do CheckBox

Problema: o CheckBox **não tem propriedade Command**!

### ✅ Solução: usar **EventToCommandBehavior** da CommunityToolkit

1. Instala o pacote:

dotnet add package CommunityToolkit.Maui

1. No MainPage.xaml, adiciona:

xmlns:behaviors="clr-namespace:CommunityToolkit.Maui.Behaviors;assembly=CommunityToolkit.Maui"

1. Usa o comportamento:

<CheckBox IsChecked="{Binding Done, Mode=TwoWay}">  
 <CheckBox.Behaviors>  
 <behaviors:EventToCommandBehavior  
 EventName="CheckedChanged"  
 Command="{Binding Source={x:Reference TodoPage}, Path=BindingContext.CompleteTodoCommand}"  
 CommandParameter="{Binding .}" />  
 </CheckBox.Behaviors>  
</CheckBox>

🧠 Isto permite ligar o CheckBox a um comando e passar o item como parâmetro. 💥

## 🧼 Organizar melhor: BaseViewModel

Para não repetires código como INotifyPropertyChanged, cria uma BaseViewModel:

public class BaseViewModel : INotifyPropertyChanged  
{  
 public event PropertyChangedEventHandler PropertyChanged;  
 protected void OnPropertyChanged([CallerMemberName] string name = "") =>  
 PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));  
  
 public string Title { get; set; }  
 public bool IsLoading { get; set; }  
 public INavigation Navigation { get; set; }  
}

## 🧪 Exemplo avançado: MauiStockTake com MVVM

Cada página (InputPage, LoginPage, ReportPage) tem a sua ViewModel, que:

* Exponibiliza propriedades e comandos
* Liga-se a serviços (ProductService, InventoryService, etc.)
* Trata da lógica sem tocar na UI

A UI só tem:

* Layout
* Ligações (Binding) à ViewModel

Exemplo: InputPage.xaml

<SearchBar Text="{Binding SearchTerm}" SearchCommand="{Binding SearchProductsCommand}" />  
<CollectionView ItemsSource="{Binding SearchResults}" SelectedItem="{Binding SelectedProduct}" />  
<Stepper Value="{Binding Count}" />  
<Button Command="{Binding AddCountCommand}" />

## 🎓 Vantagens do MVVM

| Vantagem | Como se aplica aqui |
| --- | --- |
| 🔁 Reutilização | ViewModels podem ser testados ou partilhados |
| 🧼 Separação de lógica | Cada camada tem uma única responsabilidade |
| 🔎 Testabilidade | Testar lógica sem precisar de UI |
| 🔗 Extensibilidade | Adicionar funcionalidades sem quebrar a UI |

## 🚀 Desafio para ti (Solução no final do documento 😉)

Refatora uma app simples (como a "FindMe") para usar o padrão MVVM:

* Cria uma FindMeViewModel
* Usa Command para partilhar a localização
* Passa a usar INotifyPropertyChanged para mostrar o nome e estado

# 🎨 Parte 10 – Estilos, Temas e Layouts Multi-Plataforma em .NET MAUI

## 🎯 Objetivo

Aprender a:

* Estilizar os elementos visuais da app
* Suportar **modo claro e escuro**
* Criar **temas personalizados** (como “tema praia” 🌴)
* Adaptar a interface a **telefones, tablets e computadores**
* Reagir a mudanças de estado com **triggers** e **visual states**
* Criar janelas e menus para apps de desktop

## 🧠 Princípio-chave: **Consistência visual = Experiência de Utilizador intuitiva**

Evita confundir o utilizador com botões e cores inconsistentes. Através de **estilos e temas**, podes:

* Usar as mesmas cores e formas em toda a app
* Definir o aspeto dos botões, páginas, e layouts de forma centralizada
* Reagir automaticamente ao modo claro/escuro do sistema operativo

## 🛠️ 1. Criar estilos (e evitar repetir código)

### a) Onde ficam os estilos?

No ficheiro Styles.xaml, que por sua vez usa as cores definidas em Colors.xaml.

### b) Exemplo de estilo para botões:

<Style TargetType="Button">  
 <Setter Property="BackgroundColor"   
 Value="{AppThemeBinding Light={StaticResource Primary}, Dark={StaticResource White}}" />  
 <Setter Property="HeightRequest" Value="50" />  
 <Setter Property="CornerRadius" Value="25" />  
 <Setter Property="MinimumWidthRequest" Value="100" />  
</Style>

🧠 A tua app agora pode usar **botões arredondados e coerentes**, sem repetir código!

## 🖼️ 2. Estilizar páginas com gradientes

Queremos um fundo em tons de praia 🌊 com um **degradê diagonal**.

### a) Define no Colors.xaml:

<Color x:Key="PrimaryBackground">#8dacb9</Color>  
<Color x:Key="SecondaryBackground">#b4bcc7</Color>

### b) Usa um Brush no Styles.xaml:

<Setter Property="Background">  
 <Setter.Value>  
 <LinearGradientBrush StartPoint="0,1" EndPoint="1,0">  
 <GradientStop Color="{StaticResource PrimaryBackground}" Offset="0" />  
 <GradientStop Color="{StaticResource SecondaryBackground}" Offset="1" />  
 </LinearGradientBrush>  
 </Setter.Value>  
</Setter>

🧠 Ao aplicar este estilo à Page, todas as páginas da app terão um fundo moderno e consistente!

## 🌙 3. Suporte a modo claro/escuro (automático)

### a) Define as cores para o modo escuro:

<Color x:Key="PrimaryDark">#7b98aa</Color>  
<Color x:Key="PrimaryDarkBackground">#141d31</Color>  
<Color x:Key="SecondaryDarkBackground">#212f51</Color>

### b) Usa AppThemeBinding:

<Setter Property="TextColor"  
 Value="{AppThemeBinding Light={StaticResource Primary}, Dark={StaticResource PrimaryDark}}" />

🎯 A app adapta-se automaticamente ao tema do sistema operativo!

## 🧢 4. Criar temas personalizados (ex: Tema "Praia")

Cria dois ficheiros:

* DefaultTheme.xaml
* SandyTheme.xaml

Cada tema inclui:

* Paleta de cores
* Estilos

### a) Alternar o tema em runtime:

App.Current.Resources.MergedDictionaries.Clear();  
App.Current.Resources.MergedDictionaries.Add(new SandyTheme());

🧠 Com AppThemeBinding, os dois temas reagem a claro/escuro automaticamente. 🎯

## 🚦 5. Triggers: reagir a estados

Exemplo: só permitir clicar no botão “Submeter” quando a contagem for > 0:

<Button Text="Submeter" Command="{Binding AddCommand}">  
 <Button.Triggers>  
 <DataTrigger TargetType="Button" Binding="{Binding Count}" Value="0">  
 <Setter Property="IsEnabled" Value="False" />  
 </DataTrigger>  
 </Button.Triggers>  
</Button>

## 🎭 6. Visual State Manager: diferentes visuais por estado

Exemplo: quando um item for selecionado numa CollectionView, mudar o fundo:

<Style TargetType="VerticalStackLayout" x:Key="ProductSelector">  
 <Setter Property="VisualStateManager.VisualStateGroups">  
 <VisualStateGroupList>  
 <VisualStateGroup x:Name="CommonStates">  
 <VisualState x:Name="Normal" />  
 <VisualState x:Name="Selected">  
 <VisualState.Setters>  
 <Setter Property="BackgroundColor" Value="{StaticResource Secondary}" />  
 </VisualState.Setters>  
 </VisualState>  
 </VisualStateGroup>  
 </VisualStateGroupList>  
 </Setter>  
</Style>

## 🖥️ 7. Layouts multi-plataforma

Queremos que a app tenha:

* Layout compacto no telemóvel 📱
* Layout expandido no desktop 🖥️

### Exemplo com OnIdiom:

<Label Text="{OnIdiom Phone='Produto', Desktop='Nome do Produto'}" />  
<CollectionView ItemSpacing="{OnIdiom Phone=30, Desktop=10}" />

🧠 Muito útil para mostrar mais colunas no desktop e simplificar no telemóvel.

## 🪟 8. Funcionalidades de desktop: menus e janelas

### a) Menu na barra superior (apenas visível no desktop):

<ContentPage.MenuBarItems>  
 <MenuBarItem Text="Ajuda">  
 <MenuFlyoutItem Text="Sobre" Command="{Binding ShowAboutCommand}" />  
 </MenuBarItem>  
</ContentPage.MenuBarItems>

### b) Abrir uma nova janela com o conteúdo da página “Sobre”:

public void ShowAboutPage()  
{  
 var window = new Window(new AboutPage())  
 {  
 Title = "Sobre",  
 Width = 300,  
 Height = 300  
 };  
 Application.Current.OpenWindow(window);  
}

## ✅ O que levas deste tutorial

| Conceito | Para quê? |
| --- | --- |
| Styles.xaml | Aplicar visual coerente a toda a app |
| AppThemeBinding | Suportar modo claro/escuro |
| Triggers | Reagir dinamicamente a estados |
| VisualStateManager | Aparência personalizada por estado |
| OnIdiom | Layouts que se adaptam a telemóvel/desktop |
| MenuBarItems + Window | UX de desktop com janelas e menus |

# 

# 🎛️ Parte 11 – Controlo Total: Criação de Controles Personalizados em .NET MAUI

## 🎯 Objetivo

Aprender a:

* Criar controlos reutilizáveis com **ContentView**
* Usar **Bindable Properties** para integrar com MVVM
* Personalizar controlos nativos em **cada plataforma**
* Criar **handlers** para adaptar o visual com precisão
* Preparar os teus controlos para serem **partilhados entre apps**

## 🧠 Porquê criar controlos personalizados?

Quando crias interfaces repetidas (ex: Stepper + Label), é melhor transformar isso num **componente reutilizável**.

Exemplo típico: Um Stepper para contar produtos — mas com problemas de UX se tiveres de contar 546 itens...

🎯 Solução? Criar o teu próprio Stepper com:

* Dois botões (+ e -)
* Um campo de entrada (Entry) para editar diretamente o valor

## 🧱 1. Criar um controlo personalizado com ContentView

### a) Estrutura do layout (MildredStepper.xaml)

<ContentView ...>  
 <Grid ColumnDefinitions="50,120,50">  
 <Button Text="-" Clicked="MinusButton\_Clicked" MinimumWidthRequest="50" />  
 <Entry x:Name="ValueEntry" FontSize="42" TextChanged="ValueEntry\_TextChanged" />  
 <Button Text="+" Clicked="PlusButton\_Clicked" MinimumWidthRequest="50" />  
 </Grid>  
</ContentView>

### b) Código por trás (MildredStepper.xaml.cs)

public partial class MildredStepper : ContentView  
{  
 public int Value { get; set; }  
  
 public MildredStepper()  
 {  
 InitializeComponent();  
 ValueEntry.Text = "0";  
 }  
  
 private void PlusButton\_Clicked(object sender, EventArgs e)  
 {  
 Value++;  
 ValueEntry.Text = Value.ToString();  
 }  
  
 private void MinusButton\_Clicked(object sender, EventArgs e)  
 {  
 if (Value > 0)  
 {  
 Value--;  
 ValueEntry.Text = Value.ToString();  
 }  
 }  
  
 private void ValueEntry\_TextChanged(object sender, TextChangedEventArgs e)  
 {  
 if (e.NewTextValue.StartsWith("-"))  
 {  
 ValueEntry.Text = Value.ToString();  
 return;  
 }  
  
 if (int.TryParse(e.NewTextValue, out var value))  
 {  
 Value = value;  
 }  
 }  
}

🧠 Aprendizagem ativa: já estás a criar um controlo completo, mas sem preocupações com data binding — isso vem a seguir.

## 🔗 2. Ligar o controlo à ViewModel com Bindable Properties

### a) Criar propriedade vinculável (bindable)

public static readonly BindableProperty ValueProperty = BindableProperty.Create(  
 nameof(Value),  
 typeof(int),  
 typeof(MildredStepper));  
  
public int Value  
{  
 get => (int)GetValue(ValueProperty);  
 set => SetValue(ValueProperty, value);  
}

Agora podes usar:

<controls:MildredStepper Value="{Binding Count}" />

🧠 Este é o coração da integração MVVM. Ao fazer isto, o teu controlo fala com o ViewModel!

## 🧩 3. Adicionar propriedades extra (ex: IsEnabled com DataTrigger)

public static new readonly BindableProperty IsEnabledProperty = BindableProperty.Create(  
 nameof(IsEnabled),  
 typeof(bool),  
 typeof(MildredStepper),  
 true,  
 propertyChanged: OnIsEnabledChanged);  
  
private static void OnIsEnabledChanged(BindableObject bindable, object oldValue, object newValue)  
{  
 if (bindable is MildredStepper stepper)  
 {  
 bool isEnabled = (bool)newValue;  
 stepper.ValueEntry.IsEnabled = isEnabled;  
 stepper.PlusButton.IsEnabled = isEnabled;  
 stepper.MinusButton.IsEnabled = isEnabled;  
 }  
}

🧠 Combina isto com um DataTrigger na página para ativar/desativar o stepper consoante o produto estiver selecionado.

## 🧪 4. Melhorar a experiência do utilizador

* Usar teclado **numérico** no Entry (Keyboard="Numeric")
* Impedir números negativos
* Mostrar feedback visual (opcional: Label com mensagens)

## 🧙 5. Personalizar o controlo por plataforma com **Handlers**

### a) Criar uma subclasse BorderlessEntry:

public class BorderlessEntry : Entry  
{  
 public BorderlessEntry()  
 {  
 Microsoft.Maui.Handlers.EntryHandler.Mapper.AppendToMapping("RemoveBorder", (handler, view) =>  
 {  
 if (view is BorderlessEntry)  
 {  
#if ANDROID  
 handler.PlatformView.Background = null;  
 handler.PlatformView.SetBackgroundColor(Android.Graphics.Color.Transparent);  
#elif IOS || MACCATALYST  
 handler.PlatformView.BackgroundColor = UIKit.UIColor.Clear;  
 handler.PlatformView.Layer.BorderWidth = 0;  
 handler.PlatformView.BorderStyle = UIKit.UITextBorderStyle.None;  
#elif WINDOWS  
 handler.PlatformView.BorderThickness = new Microsoft.UI.Xaml.Thickness(0);  
 handler.PlatformView.Background = null;  
 handler.PlatformView.FocusVisualMargin = new Microsoft.UI.Xaml.Thickness(0);  
#endif  
 }  
 });  
 }  
}

🧠 Isto permite alterar só este controlo, sem afetar os outros Entry.

### b) Usar BorderlessEntry no MildredStepper:

<controls:BorderlessEntry x:Name="ValueEntry" Keyboard="Numeric" ... />

## 🪄 6. Criar bibliotecas de controlos

Imagina que queres usar o MildredStepper noutra app da empresa.

### a) Cria um projeto tipo:

dotnet new maui-lib -n MildredControls

### b) Move o controlo para essa biblioteca

* Atualiza namespaces
* Adiciona a biblioteca ao projeto principal

🎯 Podes até criar pacotes NuGet para partilhar com outros devs!

## ✅ Resumo Cognitivo

| Conceito | Utilidade |
| --- | --- |
| ContentView | Cria controlos reutilizáveis |
| BindableProperty | Permite data binding com ViewModels |
| Handlers | Personalizam controlos em cada plataforma |
| BorderlessEntry | Subclasse que remove bordas da Entry |
| Control Libraries | Partilha de UI entre apps |

## 🚀 Desafio prático\*

Cria um controlo chamado RatingSelector com 5 estrelas (ícones), que permite selecionar de 1 a 5 e envia esse valor como uma bindable property.

\* Solução no GitHub (link no final do documento) 😉

# 🚀 Parte 12 – Publicar Apps com GitHub Actions: do VS Code à App Store

## 🎯 Objetivo

* Personalizar o aspeto da tua app (ícone, splash screen, nome)
* Preparar versões de produção
* Automatizar o processo com **CI/CD** usando **GitHub Actions**
* Publicar em **Google Play** e **App Store**

## 📱 1. Dar personalidade à tua app

### 1.1 Substituir o ícone da app

1. Substitui os ficheiros:
   * icon-bg.svg → renomeia para appicon.svg
   * icon-fg.svg → renomeia para appiconfg.svg
   * Coloca-os em Resources/AppIcon
2. O .NET MAUI junta automaticamente o fundo e o topo do ícone.

### 1.2 Personalizar o splash screen

1. Substitui o splash.svg em Resources/Splash
2. No .csproj, define:

<MauiSplashScreen Include="Resources/Splash/splash.svg" Color="#74A0B7" BaseSize="256,256" />

🧠 Isto faz a tua app parecer “acabada”, mesmo antes de abrir!

## 🏷️ 2. Nome, identificador e versão

No teu .csproj:

<!-- Nome da app -->  
<ApplicationTitle>MauiStockTake</ApplicationTitle>  
  
<!-- ID único (reverso do domínio) -->  
<ApplicationId>com.mildredssurfshack.mauistocktake</ApplicationId>  
  
<!-- Versão -->  
<ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>  
<ApplicationVersion>1</ApplicationVersion>

🎯 O ApplicationId é o **bilhete de identidade** da tua app para as lojas. O ApplicationVersion serve para os updates automáticos.

## 🤖 3. Introdução ao GitHub Actions (CI/CD)

### O que é?

Um **robot na cloud** que:

* Compila o teu código
* Cria ficheiros .aab (Android) e .ipa (iOS)
* Assina com certificados
* Publica automaticamente nas lojas

## 📁 4. Criar o ficheiro do workflow

1. Cria as pastas no repositório:

.github/workflows

1. Dentro de workflows, cria:

build-and-deploy.yaml

1. Estrutura base:

name: Build and Deploy MauiStockTake  
  
on:  
 push:  
 branches:  
 - main  
 workflow\_dispatch:

## ⚙️ 5. Primeiro job: Build Android

jobs:  
 build-android:  
 name: Build Android  
 runs-on: windows-latest  
  
 steps:  
 - name: Checkout  
 uses: actions/checkout@v2  
  
 - name: Setup .NET  
 uses: actions/setup-dotnet@v1  
 with:  
 dotnet-version: 7.0.x  
  
 - name: Install MAUI  
 run: dotnet workload install maui  
  
 - name: Build Android  
 run: dotnet publish MauiStockTake.UI.csproj -f:net7.0-android -c:Release

🎯 Aqui podes acrescentar parâmetros para **assinar o APK/AAB** e depois usar upload-artifact.

## 🍏 6. Segundo job: Build iOS

Este precisa de:

* Um certificado .p12
* Um perfil de provisionamento .mobileprovision
* Correr num agente **macOS** (idealmente self-hosted)

Passos:

* Fazer checkout do repositório
* Decodificar os segredos
* Instalar o .NET + MAUI
* Usar dotnet publish com os parâmetros de assinatura

🧠 Usa comandos base64 para codificar os ficheiros .p12 e .mobileprovision e guarda-os como **GitHub Secrets**.

## 📦 7. Último job: Deploy para as lojas

Este job:

* Espera pelos builds (needs)
* Faz download dos ficheiros assinados
* Usa actions comunitárias:
  + [r0adkll/upload-google-play](https://github.com/r0adkll/upload-google-play)
  + [apple-actions/upload-testflight-build](https://github.com/apple-actions/upload-testflight-build)

### Exemplo Android:

- name: Upload AAB  
 uses: r0adkll/upload-google-play@v1  
 with:  
 serviceAccountJsonPlainText: ${{ secrets.GCP\_SERVICE\_ACCOUNT }}  
 packageName: com.mildredssurfshack.mauistocktake  
 releaseFiles: ./\*Signed.aab  
 track: internal  
 status: completed

## 🔐 8. Como guardar os segredos?

No GitHub:

* Vai a **Settings > Secrets and Variables > Actions**
* Adiciona:
  + ANDROID\_KEYSTORE
  + APPLE\_CERT
  + GCP\_SERVICE\_ACCOUNT
  + APPSTORE\_API\_PRIVATE\_KEY
  + etc.

Usa nomes simples e consistentes. Evita line breaks nos valores.

## ✅ Resumo final

| Passo | Para quê? |
| --- | --- |
| Ícone + splash | Tornar a app profissional |
| .csproj com metadata | Nome, versão e ID da app |
| GitHub Actions (CI/CD) | Automatizar compilação, assinatura e publicação |
| Secrets | Proteger chaves e ficheiros sensíveis |
| Jobs por plataforma | Android, iOS e deploy separados |

## 🚀 Próximo passo sugerido\*

Montar este workflow completo com ficheiro yaml, keystore, .p12, e uploads para teste  
  
  
\* Solução no GitHub (link no final do documento) 😉

# 📂 Consulta as soluções no GitHub

As soluções práticas dos exercícios estão disponíveis aqui:

🔗 [github.com/luiscunhacsc/GitHub-MAUI](https://github.com/luiscunhacsc/GitHub-MAUI)