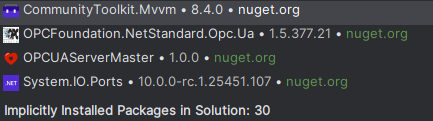
Roteiro de desenvolvimento do código

Extensões usadas no código para comunicação entre a aplicação WPF e servidor do Arduino.



**1. CommunityToolkit.Mvvm (v8.4.0)**

**Função:** É uma biblioteca usada principalmente em aplicações **WPF, MAUI, WinUI ou Xamarin**, para implementar o padrão **MVVM (Model-View-ViewModel)** de forma mais simples.

**Usos principais:**

* Criar *ViewModels* com menos código repetido.
* Implementar **PropertyChanged** automaticamente.
* Comandos assíncronos (como AsyncRelayCommand).
* Simplificar a ligação entre interface (XAML) e lógica (C#).

### ****2. OPCFoundation.NetStandard.Opc.Ua (v1.5.377.21)****

**Função:** Essa é a **biblioteca oficial da OPC Foundation** para comunicação via **OPC UA (Unified Architecture)** — um protocolo industrial usado para troca de dados entre dispositivos, sensores, e sistemas SCADA ou MES.

**Usos principais:**

* Criar **clientes e servidores OPC UA**.
* Conectar seu app a **controladores industriais (PLC)**.
* Ler e escrever variáveis de sensores, máquinas, etc.

### ****3. OPCUAServerMaster (v1.0.0)****

**Função:** Esse parece ser um **pacote personalizado** (não oficial da OPC Foundation).  
Provavelmente ele serve para **gerenciar a criação e configuração de um servidor OPC UA** de forma mais simples — pode ser algo feito para encapsular funções repetitivas.

**Usos prováveis:**

* Inicializar um servidor OPC UA local.
* Definir nós, variáveis e métodos expostos no servidor.
* Facilitar a integração com clientes OPC.

### ****4. System.IO.Ports (v10.0.0-rc.1.25451.107)****

**Função:** Biblioteca usada para **comunicação via porta serial (COM)** — muito comum em projetos com **Arduino, ESP32, sensores, e dispositivos industriais**.

**Usos principais:**

* Enviar e receber dados pela porta serial.
* Ler comandos do Arduino ou enviar instruções.
* Usada junto com o SerialPort.

Código BACK END

### Explicando o código do simulador WPF

Foi criada uma **aplicação WPF** com intuito de receber os dados dos sensores e componente utilizado no Arduino através do servidor os dados foram nomeados como umidade, angulação e velocidade que vão se usados na nomeação e criação dos clientes para cada grandeza. A ideia é que esses valores mudem automaticamente com o tempo, como se fossem dados vindos de sensores e a aplicação comunique recebendo e mandando dados para os sensores e aplicação do Arduino tudo pelo o servidor.

Foram importados alguns **namespaces** com using — cada um tem uma função específica:

* using CommunityToolkit.Mvvm.Input; → serve para usar os comandos [RelayCommand], que facilitam a ligação entre os botões da interface (no XAML) e os métodos do código.
* using System; e using System.Collections.Generic; → trazem classes básicas, como Random, List, EventArgs, etc.
* using System.ComponentModel; e using System.Runtime.CompilerServices; → são usados para implementar o **INotifyPropertyChanged**, que permite atualizar a interface automaticamente quando um valor muda.
* using System.Windows; → traz os elementos principais do WPF, como Window e MessageBox.
* using System.Windows.Threading; → permite usar o DispatcherTimer, um temporizador que dispara eventos em intervalos de tempo definidos.

### Estrutura da classe principal

A classe principal se chama **MainWindow**, e ela herda de Window, o que faz dela a janela principal do programa.  
Ela também implementa a interface **INotifyPropertyChanged**, que é essencial para o WPF entender quando deve atualizar a tela (por exemplo, quando a variável “Distância” muda de valor).

O método OnPropertyChanged() é o coração dessa comunicação:  
sempre que uma propriedade muda, ele “avisa” a interface para exibir o novo valor.

### As propriedades da simulação

As variáveis recebem valores iniciais:

* **Distancia** → começa em 1000.0
* **Umidade** → começa em 55.0
* **Angulação** → começa em 15.0
* **Velocidade** → começa em 80.0
* **Status Simulação** → começa como “Parado”

Cada uma dessas propriedades é configurada para **notificar a interface** quando o valor muda, usando o OnPropertyChanged().  
Isso significa que, se o programa alterar “Distancia”, o valor mostrado na tela muda automaticamente.

### A lista de grandezas

Foi criada uma **lista chamada Grandezas Disponíveis**, que contém os nomes das variáveis simuladas até os recebimentos de dados enviados pelos os sensores através do servidor como:  
Distância, Umidade, Angulação, Velocidade.

Essa lista vai ser usada, por exemplo, para preencher uma **ComboBox** na interface, onde o usuário escolhe qual grandeza quer visualizar para que não seja necessário criar uma interface para cada cliente.

### O motor da simulação

Foi usado nesta parte do código **DispatcherTimer**, chamado \_timer para mostrar os valores mudarem com o tempo .  
Esse temporizador é configurado para **executar o método OnTimerTick() a cada 500 milissegundos (meio segundo)**.

Dentro do OnTimerTick(),foi usada a classe Random para variar os valores das propriedades:

* A **distância** diminui aleatoriamente e volta a 1000 quando chega a zero.
* A **umidade**, **angulação** e **velocidade** mudam suavemente dentro de limites definidos usando Math.Clamp(), para evitar valores absurdos.

Isso cria o efeito de uma simulação contínua, com os valores mudando sozinhos.

### Comandos para iniciar e parar

Com o **CommunityToolkit.Mvvm**, e usado o atributo [RelayCommand] para criar comandos de forma simples.  
Esses comandos podem ser ligados diretamente a botões na interface.

* **IniciarSimulacao()** → inicia o timer e muda o status para “Rodando...”.
* **PararSimulacao()** → para o timer e muda o status para “Parado”.

Desta forma , quando clica no botão de “Iniciar”, a simulação começa a atualizar os valores; e , quando clica no botão de “Parar”, tudo congela.

### Resumindo

Em resumo, esse código cria uma **janela WPF interativa** que simula sensores.  
Ele combina conceitos importantes como:

* **MVVM simplificado** (usando CommunityToolkit);
* **Atualização automática de propriedades** (INotifyPropertyChanged);
* **Simulação com base em tempo** (DispatcherTimer);
* **Integração com interface gráfica** (bindings e comandos).