Conectar **WPF (Frontend)** con **Node.js (Backend)** es una arquitectura muy común: WPF se encarga de la interfaz bonita en Windows y Node.js se encarga de los datos y la lógica en el servidor.

Aquí tienes los pasos para crear esa API y cómo recibir los datos en C#.

Parte 1: El Servidor (Node.js + Express)

Primero, crearemos una API que devuelva una lista de "Tareas" o "Pendientes".

1. **Crea una carpeta** llamada `mi-api-node` y abre una terminal ahí.
2. **Inicializa el proyecto:** `npm init -y`
3. **Instala Express:** `npm install express`
4. **Crea el archivo** **`server.js`** y pega este código:

JavaScript
```
const express = require('express');
const app = express();
const PORT = 3000;

// Middleware para entender JSON
app.use(express.json());

// Datos de ejemplo (Simulando una base de datos)
const tareas = [
    { id: 1, titulo: "Aprender XAML", completada: true },
    { id: 2, titulo: "Configurar Node.js", completada: false },
    { id: 3, titulo: "Conectar WPF con API", completada: false }
];

// RUTA GET: Devuelve las tareas
app.get('/api/tareas', (req, res) => {
    res.json(tareas);
});

app.listen(PORT, () => {
    console.log(`Servidor API corriendo en http://localhost:${PORT}`);
});
```

5. **Corre el servidor:** `node server.js`.

Parte 2: El Cliente (WPF + C#)

Ahora vamos a hacer que tu aplicación WPF "llame" a esa URL para obtener los datos.

1. El XAML (**`MainWindow.xaml`**)

Solo necesitamos un botón y una lista para mostrar los datos.

XML
```
<StackPanel Margin="20">
    <Button Content="Obtener Tareas desde Node.js" Click="BtnCargar_Click"
Margin="0,0,0,10"/>
    <ListBox x:Name="lstTareas" Height="200" DisplayMemberPath="titulo"/>
</StackPanel>
```

2. El Código C# (`MainWindow.xaml.cs`)

Usaremos la clase `HttpClient`. Para que esto funcione, asegúrate de tener instalado el paquete NuGet `System.Text.Json` (viene por defecto en .NET moderno).

C#

```
using System.Net.Http;
using System.Text.Json;

// ... dentro de tu clase MainWindow ...

private async void BtnCargar_Click(object sender, RoutedEventArgs e)
{
    using (HttpClient client = new HttpClient())
    {
        try
        {
            // 1. Llamar a la API de Node
            string response = await
client.GetStringAsync("http://localhost:3000/api/tareas");

            // 2. Deserializar el JSON a una lista de objetos
            var opciones = new JsonSerializerOptions {
PropertyNameCaseInsensitive = true };
            var lista = JsonSerializer.Deserialize<List<Tarea>>(response,
opciones);

            // 3. Mostrar en el ListBox
            lstTareas.ItemsSource = lista;
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al conectar con el servidor: " +
ex.Message);
        }
    }
}

// Clase para representar el objeto que viene de Node
public class Tarea {
    public int Id { get; set; }
    public string Titulo { get; set; }
    public bool Completada { get; set; }
}
```

¿Qué acabamos de hacer?

1. **Node.js** expone una "puerta" (endpoint) en `http://localhost:3000/api/tareas`.
2. **Express** envía los datos en formato **JSON** (el lenguaje universal de las APIs).
3. **WPF** actúa como un cliente: pide el texto, lo convierte en objetos de C# y los dibuja en la pantalla.

Por qué esto es poderoso:

- Podrías cambiar el servidor de Node por uno en Python o Go, y **WPF ni se enteraría**.
- Podrías conectar una aplicación Android a la misma API de Node.