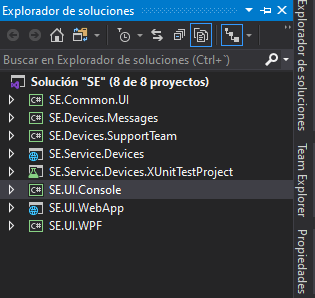
**SYSTEM DESCRIPTION**

1. Solution Description



* 1. SE.Common.UI:

Common classes like DTOs, Enums and ViewModels for UI projects. Also some functionality as DeviceService that holds an HttpClient to connect Service Device WebAPI and retrieve data from there. Data creation is done through NServiceBus commands.

* 1. SE.Devices.Messages:

NServiceBus Command and event classes. These are the messages conveyed through the Bus that helps the communication between services and Apps.

* 1. SE.Devices.SupportTeam:

Console App that subcribes to Events and show a message on the console. This Project mimics a Team that receives a notification and acts accordingly.

* 1. SE.Services.Devices:

Service in charge of providing Devices information and also to create devices. It exposes a method GetDevicesAsync as “api/Device” in its Get verb.

Also its deviceProvider class exposes a method accountable of creating devices. This method it´s not exposed in the WebApi as it is used by the Command Handler when a message arrives from UIs.

.Net Core 3.1 WebAPI Project. It uses DI and EF in-Memory

* 1. SE.Services.Devices.XUnitTestProject:

Test Unit Project that tests Device Service Provider internal class that is in charge of retrieving and storing Devices.

It has dependencies on SE.Common.UI and SE.Devices.Messages.

.Net Core 3.1 Web App Project. It uses DI, HttpClientFactory

* 1. SE.UI.Console:

Console App that is only able to send a “CreateDevice” Command. When Console App starts, it asks for json file path. If Device Service is able to create the Device, it will send an event. Console App also is subscribed to “Device Created” and will receive the message. This process also applies when another UI creates a device, console App will also receive the message event, as long as it is up and running. Console App lifecycle lives just for 1 attempt of creation no matter it success or fails.

It has dependencies on SE.Common.UI and SE.Devices.Messages.

.Net Core 3.1 Console App Project.

* 1. SE.UI.WebApp

Web Application to manage devices. Connects via get rest call to SE.Services.Devices to retrieve devices. When it´s time for the creation of a device, it sends a command to NServiceBus in order to the device service to pick it up and process it.

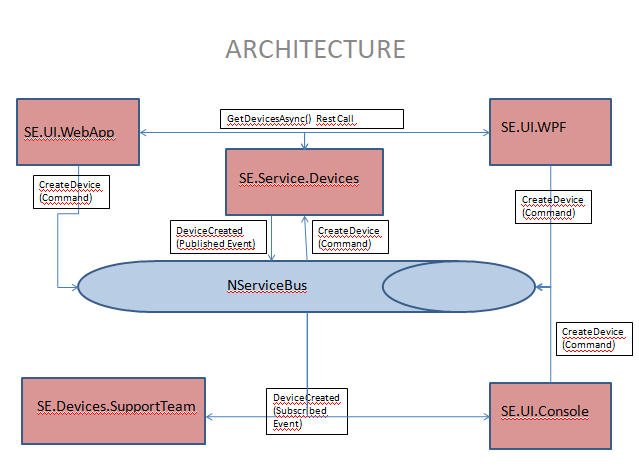
It has dependencies on SE.Common.UI and SE.Devices.Messages.

.Net Core 3.1 Web App Project. It uses DI, HttpClientFactory

* 1. SE.UI.WPF

Same idea than SE.UI.WebApp but for desktop. Use of MVVM with DataBinding. Trying to avoid code-behind, eventhough some times is inevitable

1. System Architecture



1. Considerations
   1. ID field is not present in the UIs, as I assume that is the primary key of the Database. Because of that we don´t use as an input in the UI. Because we don´t update devices on the UI, I decided not to send Id when calling the DeviceService web API.
   2. I take Uniqueness as a definition for a device where SerialNumber, Model and Brand will note ver repeat in any other device of the same Type. We could have a wáter counter with (SN,M,B)=(1,M1,B1) but also an electricity counter with (SN,M,B)=(1,M1,B1) and also a gateway with (SN,M,B)=(1,M1,B1) because the type is already a discriminator between them. As I said above ID will be primary key for devices, so will not count in device UNIQUENESS.  
      As In-Memory is not relational database I will manually check for this UNIQUENESS
   3. I understand that even if I can do some client-side validations, I will only do validation in the back-end.
   4. NserviceBus for WPF is configured as send-only, so WPF will only send commands to DeviceService. Console App will send commands but also will be subscribed to DeviceService events, which means that will be know when a device is created not only by itself but by other UIs. Finally WebApp has configured a req-res callback with DeviceService, so it will always know what happen with its CreateDevice Command, as it will be replied.