Ejercicio A.6: Hacer Extensible nuestra extensión

Para un código limpio, mantenible y escalable, los desarrollos tienen que cumplir con **el principio abierto/cerrado**. Así los desarrollos tienen que:

- Estar abierto a extensiones
- Estar cerrado a modificaciones

Cuando tenemos un módulo cerrado a modificaciones, vamos a poder:

- Instalarlo en múltiples implantaciones
- Re-usarlo en otros módulos
- Mantener una única versión
- Actualizar todas las implantaciones cuando haya modificaciones

Cuando tenemos un módulo abierto a extensiones, vamos a poder:

- Alterar el funcionamiento del módulo original sin tocar el código base.
- Adaptarnos a las necesidades de cada implantación.

Datos de conexión

- Entorno de desarrollo:
  - Servidor: localhost\NAVDEMO
  - BBDD: BCCU0_AppDevelopment
  - Autenticación: Windows

Indicaciones para el ejercicio

En vscode, vamos a crear la extensión HolaMundo. Vamos a modificar la extensión de formar que sea extendible.

Indicaciones paso a paso

Sigue los siguientes pasos en el entorno de desarrollo C/SIDE:

1. En vscode, crea un nuevo proyecto y descarga los símbolos.
   1.1. Pulsa F1 para abrir la paleta de comandos.
   1.2. Ejecuta el comando AL: Go!
   Nombre del proyecto: HolaMundo Base
   1.3. Modifica el fichero launch.json

```
"serverInstance": " BCCU0_AppDevelopment ",
"authentication": "Windows",
```

   1.4. Ejecuta el comando AL: Download symbols.

2. Abre el fichero HelloWorl.al.
   2.1. Crea el procedimiento MyAction en una nueva codeunit.
   2.2. Mueve la instrucción Message al procedimiento.
   2.3. En el trigger OnOpenPage, llama al procedimiento MyAction.

```
pageextension 50100 CustomerListExt extends "Customer List"
{
    trigger OnOpenPage();
    var
        MyActionMgt: Codeunit MyActionMgt;
    begin
        MyActionMgt.MyAction;  //>> Llamar al procedimiento
    end;
}

codeunit 50100 MyActionMgt
{
    //>> Crear este procedimiento
    procedure MyAction()
    begin
        Message('App published: Hello world');
    end;
}
```

3. Antes y después de la acción principal, lanza eventos para que otras extensiones se puedan suscribir y extender la funcionalidad:

   Nota: Aunque los procedimientos todavía no existen, por ahora escribe la llamada. En siguientes pasos vamos a crear los procedimientos y a determinar qué parámetros deben recibir.

```
codeunit 50100 MyActionMgt
{
    procedure MyAction()
    begin
        OnBeforeMyAction(); //>> Lanzar evento
        Message('App published: Hello world');
        OnAfterMyAction(); //>> Lanzar evento
    end;
}
```

4. Crea los procedimientos OnBeforeMyAction y OnAfterMyAction. Declaralos como eventos de integración.

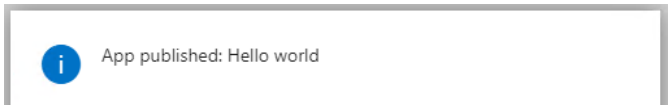
```
//>> Crear eventos de integración
[IntegrationEvent(false, false)]
local procedure OnBeforeMyAction()
begin
end;

[IntegrationEvent(false, false)]
local procedure OnAfterMyAction()
begin
end;
```

Al declarar eventos, hay dos parámetros que debemos informar:

https://docs.microsoft.com/en-us/dynamics-nav/includesender-property

- IncludeSender. Especifica si se exponen las funciones globales del objeto que contiene el evento publicador.
- GlobalVarAccess, Especifica si se exponent las variables globales del objeto que contiene el evento publicador.

5. Publica la App con el comando AL: publish without debugging
6. En el cliente web, abre la lista de clientes. El siguiente mensaje se va a mostrar:

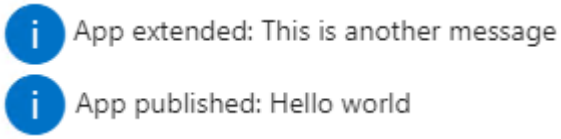


Ahora vamos a ampliar la funcionalidad, sin tocar el código base, tal y como lo haríamos desde una nueva extensión.

7. Crea una nueva codeunit
8. Crea un procedimiento y suscríbelo al evento OnBeforeMyAction

```
codeunit 50101 ExtendMyAction
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::MyActionMgt, 'OnBeforeMyAction', '', false , false)]
    local procedure OnBeforeMyAction_ShowAnotherMessage()
    begin
        Message('App extended: This is another message');
    end;
}
```

9. Publica la App con el comando AL: publish without debugging
10. En el cliente web, abre la lista de clientes. Ahora deberías recibir los dos mensajes, en este orden:



Con la arquitectura actual podemos ampliar la funcionalidad ejecutando código OnBefore y OnAfter, pero **no podemos sustituir la función principal**.

Vamos a ajustar la arquitectura para permitir que la función principal sea sustituida:

11. Edita el procedimiento MyAction:
    - Crea la variable local Handled
    - Añade la variable como parámetro de la función OnBeforeMyAction

```
codeunit 50100 MyActionMgt
{
    procedure MyAction()
    var
        Handled : Boolean; //>> Crear variable
    begin
        OnBeforeMyAction(Handled); //>> Pasarle la variable a la función
        Message('App published: Hello world');
        OnAfterMyAction();
    end;
```

12. Edita el procedimiento OnBeforeMyAction. Añade el parámetro Handled, que se recibe por referencia.

```
  //>> Añadir parámetro, recibido por referencia
  [IntegrationEvent(false, false)]
  local procedure OnBeforeMyAction(var Handled : Boolean)
  begin
  end;
```

13. Edita el procedimiento MyAction:
    13.1. Crea el procedimiento MainMyAction
    13.2. Mueve la instrucción Message al procedimiento.

```
codeunit 50100 MyActionMgt
{
    procedure MyAction()
    var
        Handled : Boolean;
    begin
        OnBeforeMyAction(Handled);
        MainMyAction(Handled);  //>> Llamar a procedimiento
        OnAfterMyAction();
    end;

    //>> Crear este procedimiento
    local procedure MainMyAction(var Handled: Boolean)
    begin
        Message('App published: Hello world'); //>> Mover el message aquí
    end;
```

14. Edita el procedimiento MainMyAction.
    Al inicio del procedimiento comprueba el valor de la variable Handled.
    Si es true, pon un exit para que no se ejecute el código de la función.

```
local procedure MainMyAction(var Handled: Boolean)
begin
    if Handled then //>> Añadir
        exit;       //>> Añadir

    Message('App published: Hello world');
end;
```

15. Edita el procedimiento OnBeforeMyAction_ShowAnotherMessage
    15.1. Añade el parámetro Handled
    15.2. Asigna el valor true a la variable.

```
codeunit 50101 ExtendMyAction
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::MyActionMgt, 'OnBeforeMyAction', '', false, false)]
    local procedure OnBeforeMyAction_ShowAnotherMessage(var Handled: Boolean) //>> Añadir parámetro Handled
    begin
        Message('App extended: This is another message.');
        Handled := true; //>> Indica que el código ya está gestionado
                         //>> está "handled", por lo que no se debe
                         //>> ejecutar el main
    end;
}
```

16. Publica la App con el comando AL: publish without debugging
17. En el cliente web, abre la lista de clientes. Ahora deberías recibir únicamente un mensaje, el del OnBefore:

Referencias:

https://community.dynamics.com/nav/b/navmarkbrummel/archive/2015/11/25/the-handled-pattern