LOGALI

Teoría

ABAP Cloud Developer Extensibility – Acciones

SAP S/4HANA Cloud – Modelo de extensibilidad Clean Core

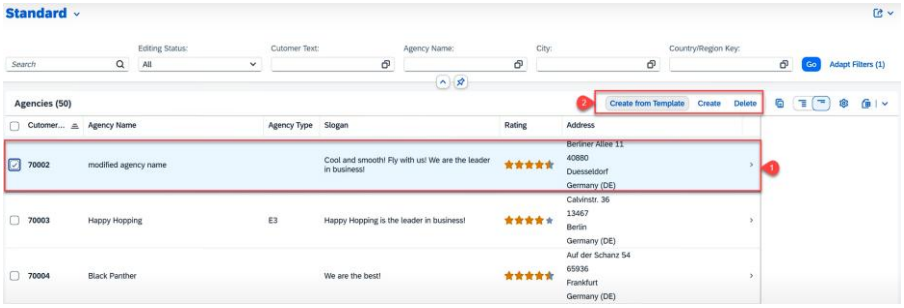
Contenido

1. ABAP Cloud Developer Extensibility – Acciones

1.1. Requerimiento – Extensión RAP Nueva Acción

En esta lección se aborda la extensión de una aplicación estándar mediante la adición de un nuevo botón con una nueva acción y funcionalidad. El objetivo es ampliar las capacidades estándar de la aplicación para satisfacer un requerimiento empresarial específico.

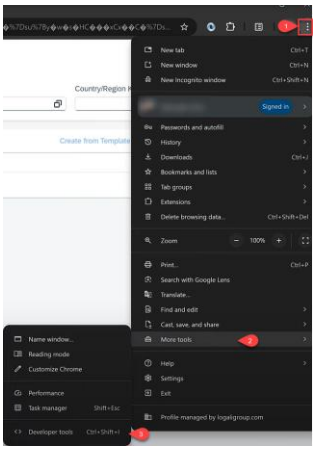
Al seleccionar cualquier registro existente, las acciones o botones se habilitan. Se debe agregar visualmente el botón y realizar todas las extensiones necesarias de los objetos para disponer de la funcionalidad requerida.



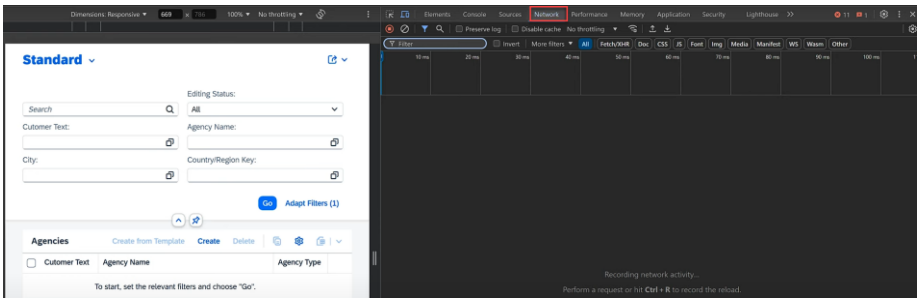
Identificación de Peticiones:

En la pestaña Network en las herramientas de desarrollador. Las cuales se pueden habilitar en el navegador al estar en la aplicación requerida a modificar de la siguiente manera:

- Al presionar la tecla **F12.**
- Al presionar el conjunto de teclas **Ctrl + Shift + I.**
- Al seguir la siguiente ruta: Presionar el botón  al lado derecho del icono de la cuenta de Google si se encuentra en el navegador de Google Chrome. Luego seleccionar “More Tools” y por último “Developer Tools”.

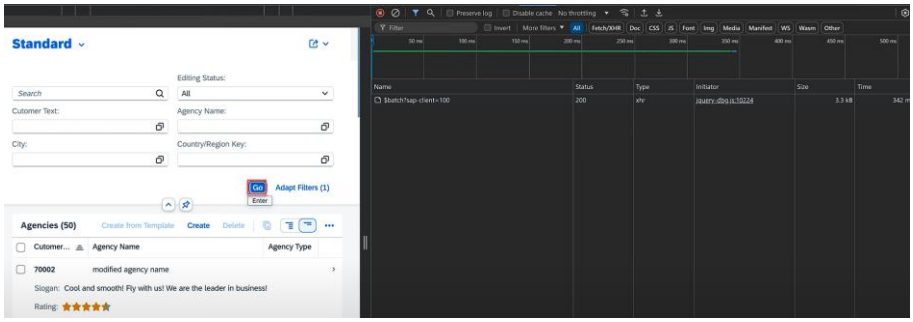


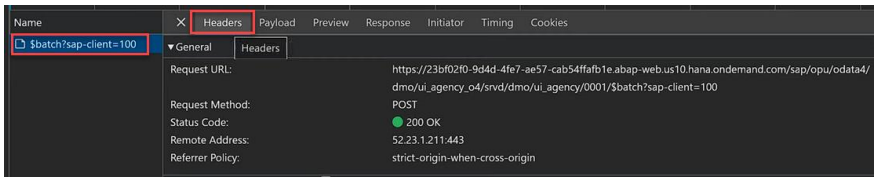
Al presionar el botón “Go” en la aplicación se mostrarán las peticiones de tipo batch.



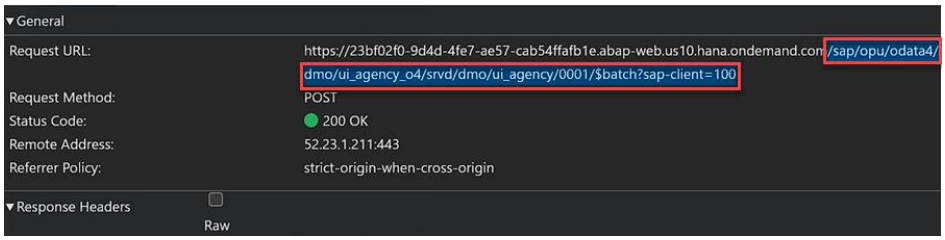
Identificación del Servicio y Service Binding:

Se identifican las peticiones tipo batch a un servicio OData. Los detalles de estas peticiones se encuentran en la sección headers. El sección “Request URL” tenemos información relevante como:

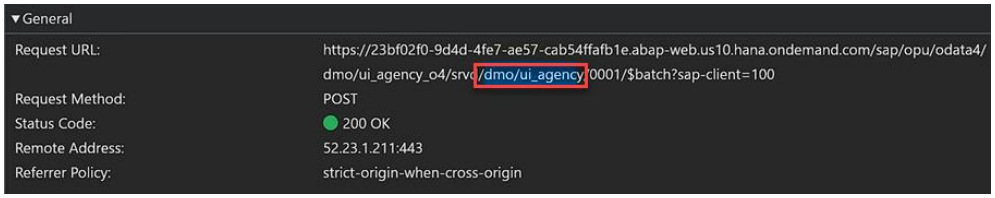




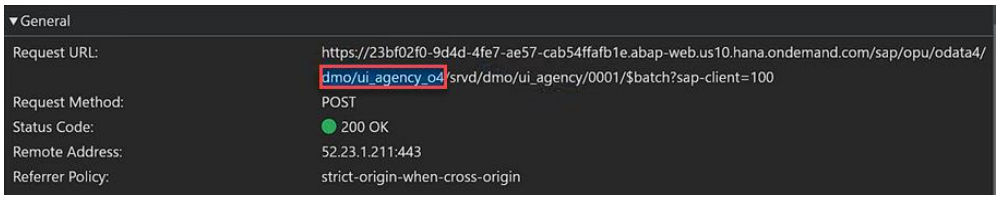
- **URL Relativa:** Identificar la URL relativa que apunta al servicio invocado.



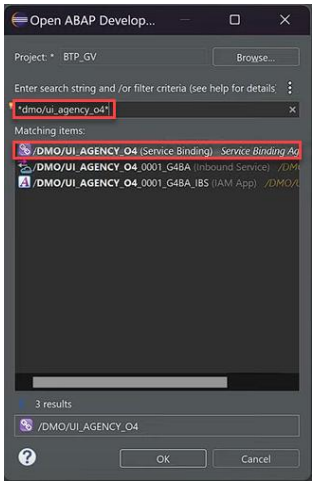
- **Servicio Odata:** El nombre del servicio Odata a la que se le hizo la llamada.



- **Service Binding:** El nombre del service binding.



Con el nombre del service binding se puede acceder a este recurso del backend a través de Eclipse. Utilizando el ABAP Development Object, en donde se recomienda coloca el nombre del servicio entre "*" de tal manera que permita buscar en un rango más amplio el archivo.

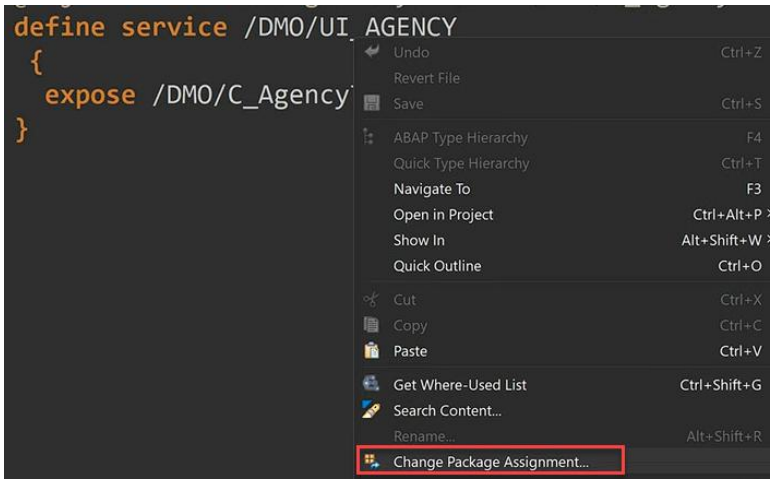


Identificación del paquete de desarrollo de la aplicación:

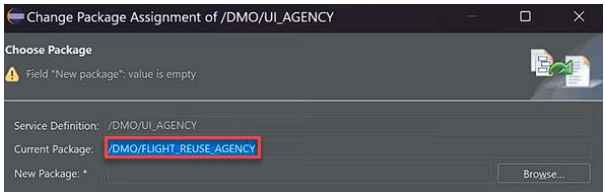
Para identificar el paquete de desarrollo, se puede navegar desde el Service Binding hasta el Service Definition. Una vez allí, posicionar el cursor en cualquier línea del código y hacer clic derecho para abrir el menú de opciones.

```
@EndUserText.label: 'Agency'
@AbapCatalog.extensibility.extensible: true
@ObjectModel.leadingEntity.name: '/DMO/C_AgencyTP'
define service /DMO/UI_AGENCY
 {
  expose /DMO/C_AgencyTP as Agency;
}
```

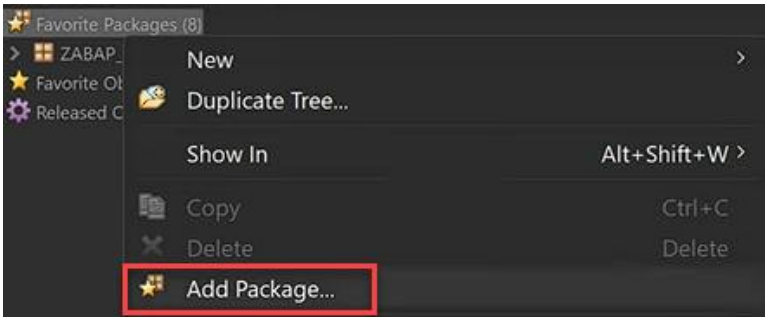
Al seleccionar la opción “Change Package Assignment…”.



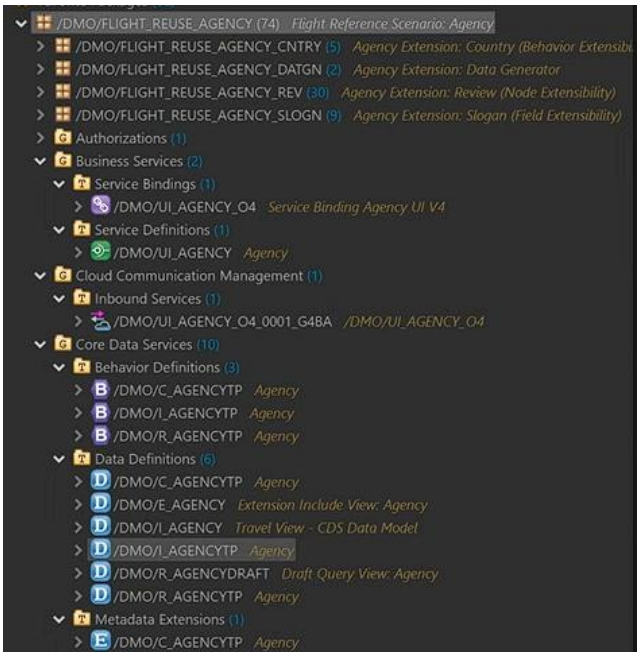
Se puede identificar el paquete actual del proyecto de la aplicación.



Para posteriormente añadir el paquete en favoritos.



Donde se puede analizar todos los objetos del proyecto para añadir la extensión de la funcionalidad.

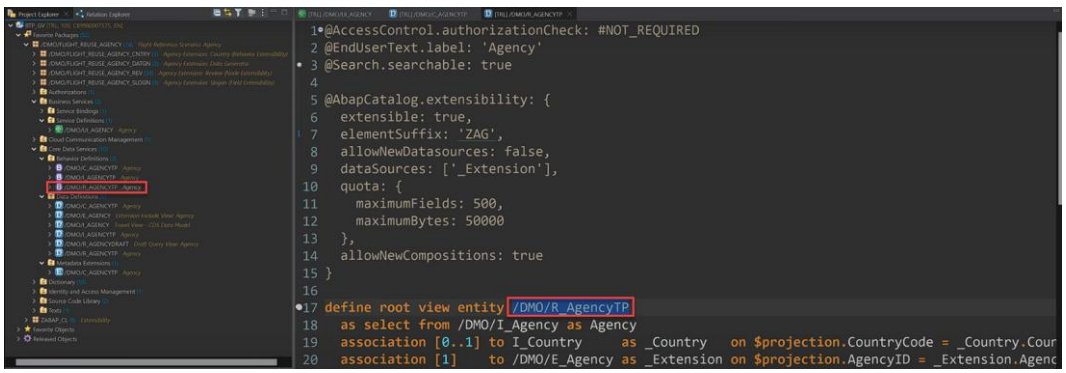


1.2. Behavior Definition Root – Creación Extensión

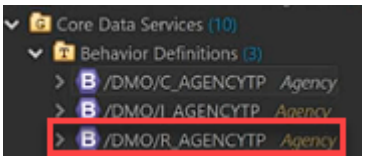
Para la creación de una nueva acción o botón en una aplicación estándar de tipo RAP, es necesario crear una extensión de comportamiento basado en el Behavior Definition de la entidad raíz permitiendo agregar nuevas funcionalidades a una aplicación estándar.

Identificación del Desarrollo y Artefactos a Extender:

Se debe navegar desde el Service Binding, pasando por el Service Definition hasta la entidad de consumo o de proyección y luego a la entidad de tipo raíz. A través de esto, se identifica la definición del comportamiento correspondiente.



El cual posee el mismo nombre que la entidad raíz.



Las acciones se definen en el Behavior Definition de la entidad raíz de la aplicación. Estas acciones son las funcionalidades básicas y esenciales que la aplicación puede ejecutar. Dicha entidad debe poseer características de extensión implementadas de la siguiente manera:

```
managed implementation in class /dmo/bp_r_agencytp unique;
strict ( 2 );
with draft;
extensible
{
  with determinations on modify;
  with determinations on save;
  with validations on save;
  with additional save;
}

define behavior for /DMO/R_AgencyTP alias /DMO/Agency
persistent table /dmo/agency
draft table /dmo/agency_d query /DMO/R_AgencyDraft
lock master
total etag LastChangedAt
authorization master ( global )
etag master LocalLastChangedAt
late numbering
extensible
```

Además es necesario que la aplicación tenga implementada un Behavior Definition de una entidad de interfaz. Esta entidad permite que los desarrolladores amplíen y mejoren los objetos de negocio sin alterar la lógica subyacente. Basada en la entidad raíz (Root Entity), la entidad de interfaz integra anotaciones y proyecciones necesarias para gestionar datos de manera eficiente y coherente. La cual actúa como una puerta que permite implementar nuevas funcionalidades en la aplicación. Si la interfaz posee características de extensión, se puede utilizar dicha interfaz como "puerta" para introducir nuevas acciones en el nodo principal de la aplicación.

```
interface;
use draft;
extensible;

define behavior for /DMO/I_AgencyTP alias /DMO/Agency
{
  use create;
  use update;
  use delete;

  use action Resume;
  use action Edit;
  use action Activate;
  use action Discard;
  use action Prepare;
}
```

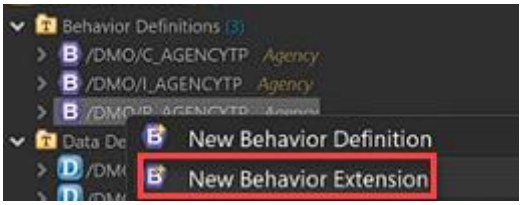
Los elementos necesarios para las extensiones deben estar presentes de igual forma en el behavior definition de consumo. Esto garantiza que la aplicación pueda manejar las nuevas funcionalidades que queremos agregar.

```
projection;
strict(2);
use draft;
extensible;

define behavior for /DMO/C_AgencyTP alias /DMO/Agency
extensible
{
  use create;
  use update;
  use delete;

  use action Resume;
  use action Edit;
  use action Activate;
  use action Discard;
  use action Prepare;
}
```

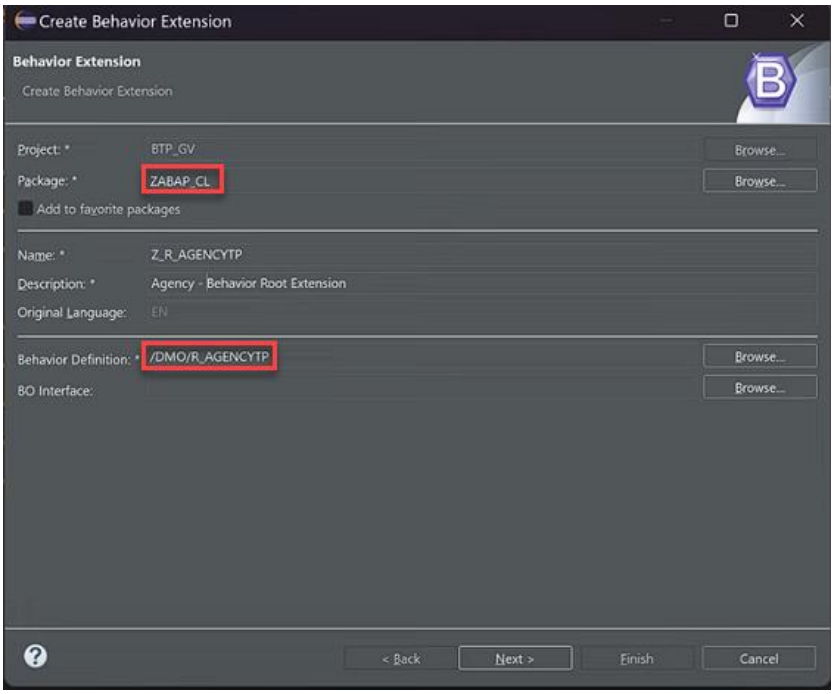
En una configuración habitual, los elementos necesarios para las extensiones están presentes tanto en el nodo raíz (Root) como en la parte de consumo de datos. Esto garantiza que la aplicación pueda manejar las nuevas funcionalidades que requieran agregar.

Proceso de Creación de un Behavior Extension:

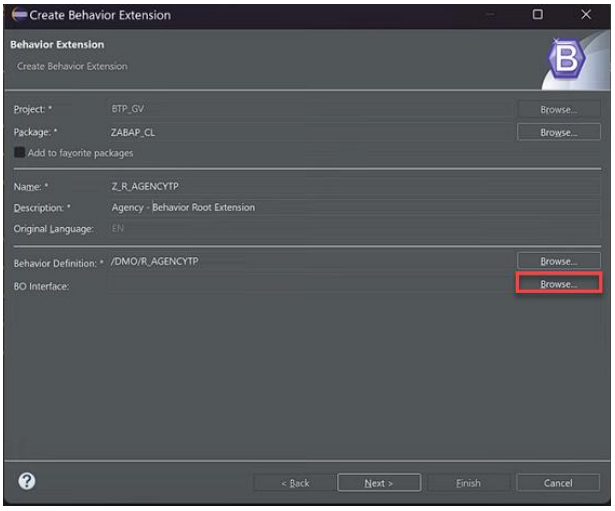
Para crear una extensión de comportamiento, es necesario localizar la definición de comportamiento (Behavior Definition) de la entidad raíz dentro del paquete de la aplicación. Luego, haz clic derecho en el nombre del objeto para desplegar el menú de opciones y seleccionar "New Behavior Extension".



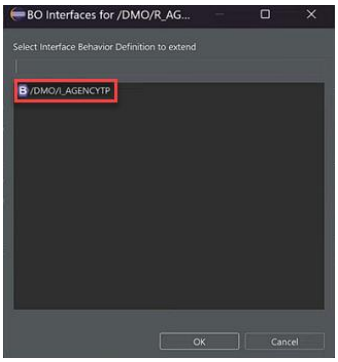
En la siguiente ventana es fundamental colocar el paquete Z donde se estarán realizando los desarrollos de las extensiones. También se puede validar que el nombre del Behavior Definition de la entidad raíz se encuentra ya asignado.



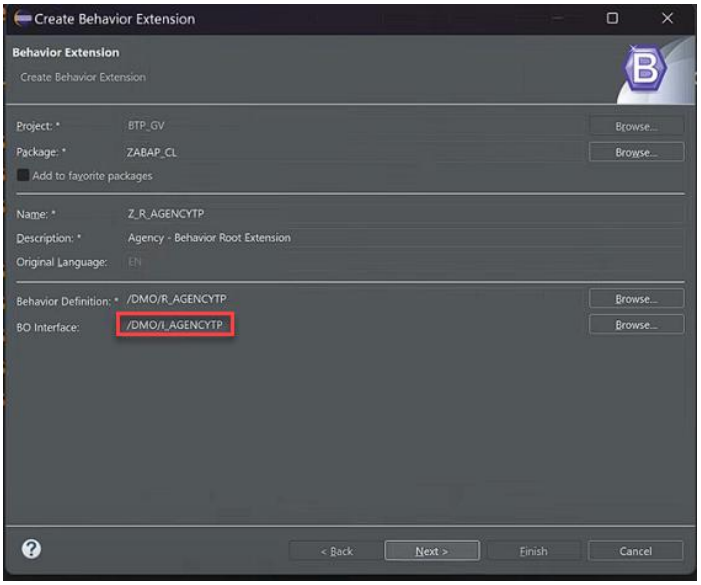
Luego es necesario especificar el Behavior Definition de la entidad interfaz. Utilizando el botón “Browse” en el campo BO: Interface.



Se puede seleccionar rápidamente el Behavior Definition de la entidad interfaz más rápidamente.



Para finalizar con el proceso presionando el botón “Next” para agregar los cambios realizados en el paquete en una orden de transporte.



El Behavior extensión se mostrará de la siguiente manera:

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;
```

1.3. Behavior Definition Root – Extensión Nueva Acción

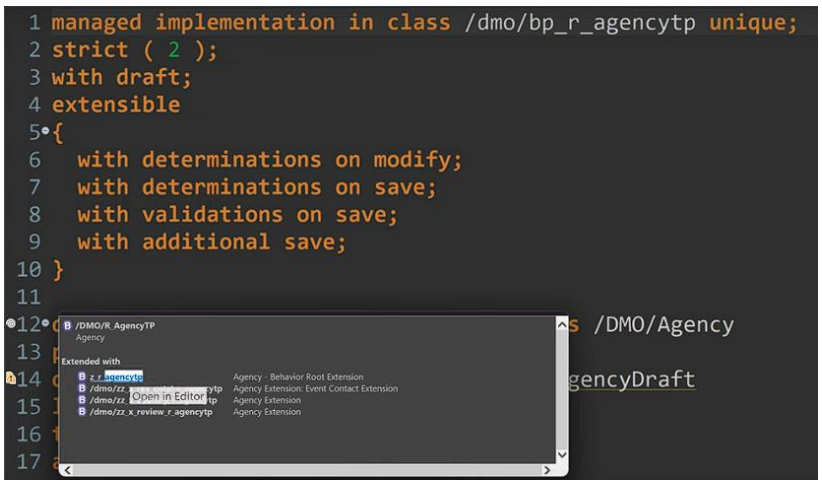
La extensión de una aplicación estándar de tipo RAP mediante la creación de una nueva acción o botón en la interfaz de usuario implica la definición y personalización del comportamiento de la aplicación. Este proceso incluye la identificación de los artefactos necesarios dentro del paquete de desarrollo, así como la definición de nuevas acciones utilizando nodos y alias existentes. Al utilizar extensiones en la definición de comportamiento (Behavior Definition), se pueden agregar acciones adicionales que modifican o amplían las funcionalidades estándar de la aplicación. La interfaz y las extensiones de metadatos

(metadata extensions) juegan un papel crucial en habilitar y visualizar estas nuevas funcionalidades, asegurando que las acciones añadidas se integren de manera coherente en la aplicación estándar.

Continuando con el proceso, después de crear la extensión de comportamiento (Behavior Extension), se puede verificar que la relación está disponible dentro de la definición de comportamiento (Behavior Definition) de la entidad raíz. Para hacerlo, selecciona el símbolo , ubicado en la línea donde se encuentra la sentencia **define behavior for**. al desplegar el menú de relaciones.

```
1  managed implementation in class /dmo/bp_r_agencytp unique;
2  strict ( 2 );
3  with draft;
4  extensible
5  {
6    with determinations on modify;
7    with determinations on save;
8    with validations on save;
9    with additional save;
10 }
11
12 define behavior for /DMO/R_AgencyTP alias /DMO/Agency
13 persistent table /dmo/agency
14 draft table /dmo/agency_d query /DMO/R_AgencyDraft
15 lock master
16 total etag LastChangedAt
17 authorization master ( global )
18 etag master LocalLastChangedAt
19 late numbering
20 extensible
```

Luego es necesario seleccionar , ubicado en la línea donde se encuentra la sentencia **define root view entity**, para desplegar el menú de relaciones de la entidad de consumo.



En donde es importante saber, que se debe utilizar el alias del nodo que se requiere extender dentro de la definición del comportamiento en de

la entidad raíz. Y se procede a la definición de la nueva acción en la extensión de comportamiento.

```
managed implementation in class /dmo/bp_r_agencytp unique;
strict ( 2 );
with draft;
extensible
{
  with determinations on modify;
  with determinations on save;
  with validations on save;
  with additional save;
}

define behavior for /DMO/R_AgencyTP alias /DMO/Agency
persistent table /dmo/agency
draft table /dmo/agency_d query /DMO/R_AgencyDraft
lock master
total etag LastChangedAt
authorization master ( global )
etag master LocalLastChangedAt
late numbering
extensible
```

Recordando que se utilizará el alias del nodo y no el nombre de la fuentes de datos a pesar de que en este caso sean iguales.

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{

}
```

Luego se procede a definir la acción para luego seleccionar una autorización para la acción. Las posibles autorizaciones EML que se establecen dentro del Behavior Definition se explican a profundidad en las documentaciones de RAP, disponibles en su curso o máster correspondiente en la plataforma de Logali.

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  action (au)
}
```

En este caso no se establece ninguna autorización con la sentencia **authorization : none.** Para luego definir el nombre de la acción, donde en obligatorio colocar el prefijo "zz" al nombre debido a que la definición es un elemento personalizado. Seguido de la sentencia **result [1] $self;** para finalizar con la instrucción.

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  action (authorization : none) zzAssignType result [1] $self;
}
```

1.4. Extensión CDS Interfaz – Elementos Requeridos

Este proceso se centra en la extensión de una aplicación estándar de tipo RAP mediante la creación de una nueva acción o botón en la interfaz de usuario. La extensión implica la definición de lógica y la implementación de funcionalidades adicionales que se ejecutarán cuando el usuario interactúe con la nueva acción.

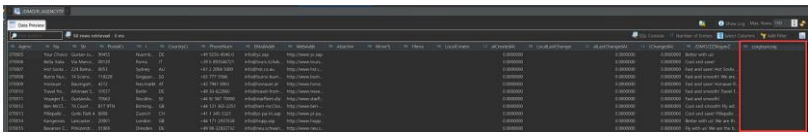
Dado que la entidad de tipo interfaz permite implementar nuevas funcionalidades sin alterar la lógica subyacente. Y como dicha entidad selecciona los datos de la entidad raíz que ya tiene el campo personalizado agregado, se debe proyectar dicho campo en la entidad de interfaz a través de una extensión. Esto permitirá utilizarlo en la lógica del código fuente en el behavior pool, explicación que se realizará más adelante en el desarrollo del documento, dando vida a este método.

```
define root view entity /DMO/I_AgencyTP
  provider contract transactional_interface
  as projection on /DMO/R_AgencyTP as Agency
{
      @ObjectModel.text.element: ['Name']
  key AgencyID,

      @Search.defaultSearchElement: true
      @Search.fuzzinessThreshold: 0.8
      @Semantics.text: true
      @Semantics.organization.name: true
      Name,
```

Esto se puede validar navegando desde la entidad interfaz a la entidad raíz.

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Agency'
@Search.searchable: true

@AbapCatalog.extensibility: {
  extensible: true,
  elementSuffix: 'ZAG',
  allowNewDatasources: false,
  dataSources: ['_Extension'],
  quota: {
    maximumFields: 500,
    maximumBytes: 50000
  },
  allowNewCompositions: true
}

define root view entity /DMO/R_AgencyTP
  as select from /DMO/I_Agency as Agency
  association [0..1] to I_Country     as _Country   on $projection.CountryCode = _Cou
  association [1]    to /DMO/E_Agency as _Extension on $projection.AgencyID = _Exten
```

Y ejecutándola al presionar la tecla **F8** para ver su contenido.



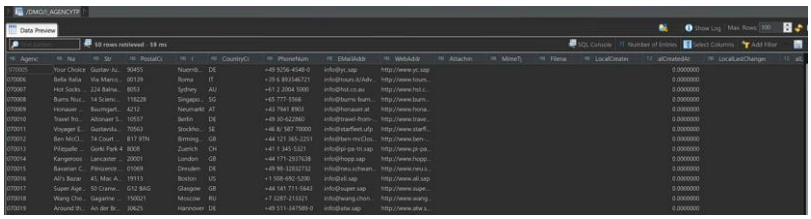
Dado a que no es posible utilizar la lectura del nodo raíz directamente en la interfaz de usuario, se debe recurrir a la entidad interfaz habilitada en la definición de comportamiento asociada.

```
interface;
use draft;
extensible;

define behavior for /DMO/I_AgencyTP alias /DMO/Agency
{
  use create;
  use update;
  use delete;

  use action Resume;
  use action Edit;
  use action Activate;
  use action Discard;
  use action Prepare;
}
```

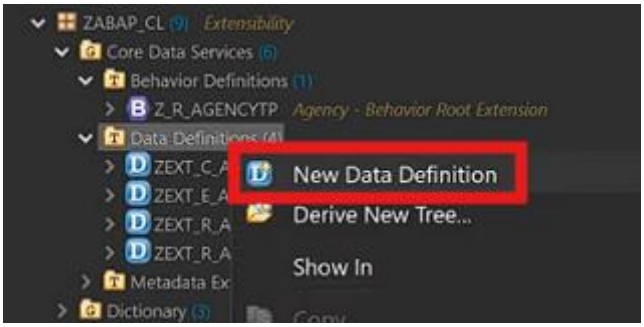
Donde al navegar a dicha entidad de interfaz y pulsar F8, se puede comprobar que el campo no está disponible.

```
@Metadata.ignorePropagatedAnnotations: true
@EndUserText.label: 'Agency'
@Search.searchable: true
@AccessControl.authorizationCheck: #NOT_REQUIRED

@AbapCatalog.extensibility: {
  extensible: true,
  dataSources: ['AGENCY'],
  elementSuffix: 'ZAG',
  quota: {
    maximumFields: 500,
    maximumBytes: 50000
  }, allowNewCompositions: true
}

define root view entity /DMO/I_AgencyTP
  provider contract transactional_interface
  as projection on /DMO/R_AgencyTP as Agency
```

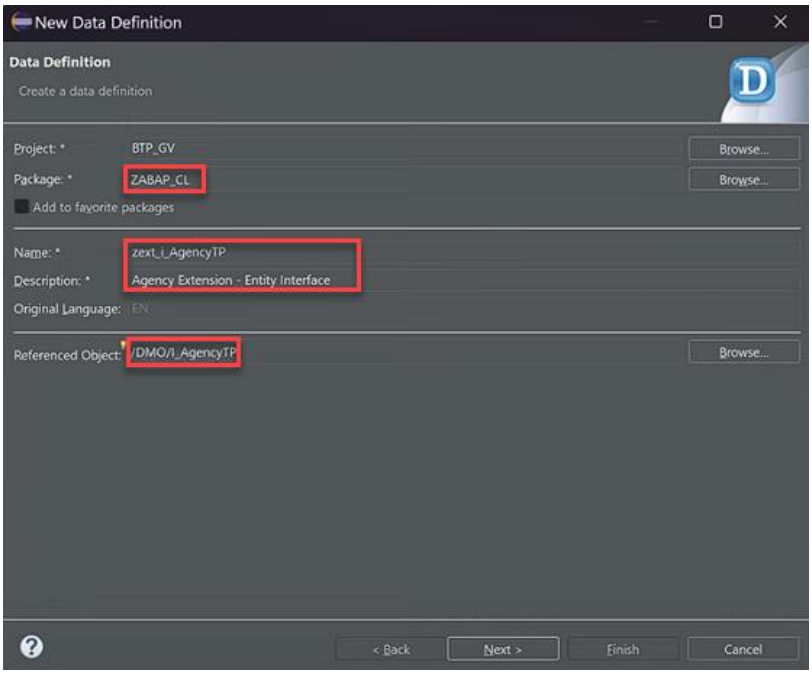
Ya que por el momento no ha sido agregado.



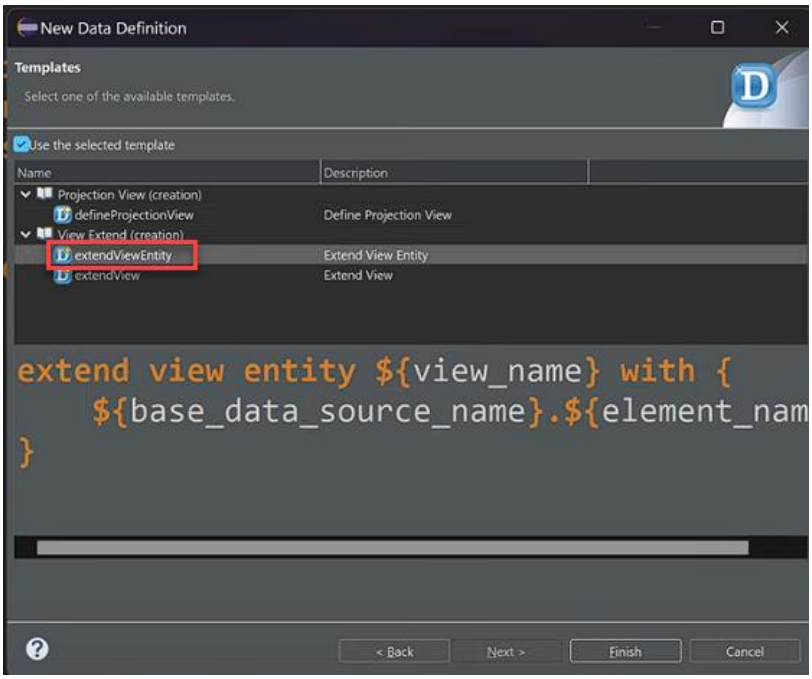
Por lo que se necesita crear una vista de extensión para proyectar el campo. Como se hizo anteriormente con la entidad raíz.



Se crea el nuevo Data Definition colocando el paquete Z donde se estarán agregando las ampliaciones al programa estándar por medio de extensiones, un nombre para el objeto donde por lo general se utiliza el sufijo “zz”. No obstante, en este apartado se debe emplear la nomenclatura específica utilizada por el cliente en su organización. Además, se debe colocar el nombre de la entidad raíz en el campo "Referenced Object".



Recordando seleccionar la plantilla **extendViewEntity** que se encuentra en la carpeta **View Extend (creation).**



Donde para proyectar el campo desde la entidad raíz a la entidad interfaz, es necesario utilizar el nombre del alias de la entidad raíz.

```
  }, allowNewCompositions: true
}

define root view entity /DMO/I_AgencyTP
  provider contract transactional_interface
  as projection on /DMO/R_AgencyTP as Agency
{
      @ObjectModel.text.element: ['Name']
  key AgencyID,

      @Search.defaultSearchElement: true
      @Search.fuzzinessThreshold: 0.8
      @Semantics.text: true
      @Semantics.organization.name: true
      Name,

      @Semantics.address.street: true
      Street,

      @Semantics.address.zipCode: true
      PostalCode,
```

Y especificar el campo proyectado.

```
extend view entity /DMO/I_AgencyTP with {
    Agency.zzagtypezag
}
```

1.5. Acción – Implementación Behavior Pool

Las funcionalidades adicionales que se ejecutarán en la interfaz de usuario cuando el usuario interactúe con un nuevo botón. Conllevan en su definición la implementación de la lógica que se ejecutará en la nueva acción a través de una clase de tipo Behavior Pool, que extiende la definición de comportamiento raíz de una aplicación estándar de tipo RAP.

La lógica de las acciones de una aplicación RAP se desarrollan en la clase Behavior Pool implementada en el Behavior Definition de la entidad raíz. Pero debido a que la lógica de la nueva acción personalizada no se puede agregar directamente en la clase, se hace uso de la extensión de comportamiento.

```
managed implementation in class /dmo/bp_r_agencytp unique;
strict ( 2 );
with draft;
extensible
{
  with determinations on modify;
  with determinations on save;
  with validations on save;
  with additional save;
}

define behavior for /DMO/R_AgencyTP alias /DMO/Agency
persistent table /dmo/agency
draft table /dmo/agency_d query /DMO/R_AgencyDraft
lock master
total etag LastChangedAt
authorization master ( global )
etag master LocalLastChangedAt
late numbering
extensible
```

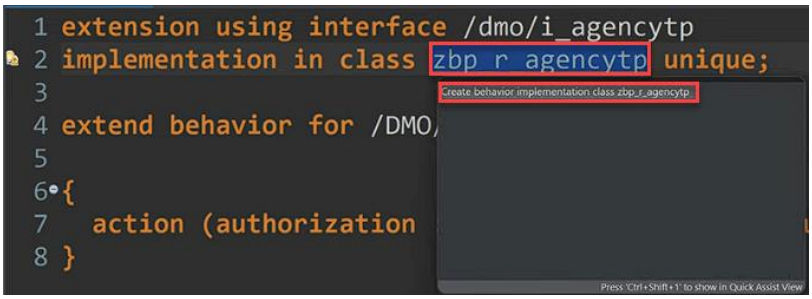
Creación de la Clase Behavior Pool de extensión :

En la carpeta **Core Data Services** del paquete de desarrollo "Z" donde se han estado agregando las ampliaciones al programa estándar a través de extensiones. En esta sección, se selecciona la extensión de comportamiento correspondiente.

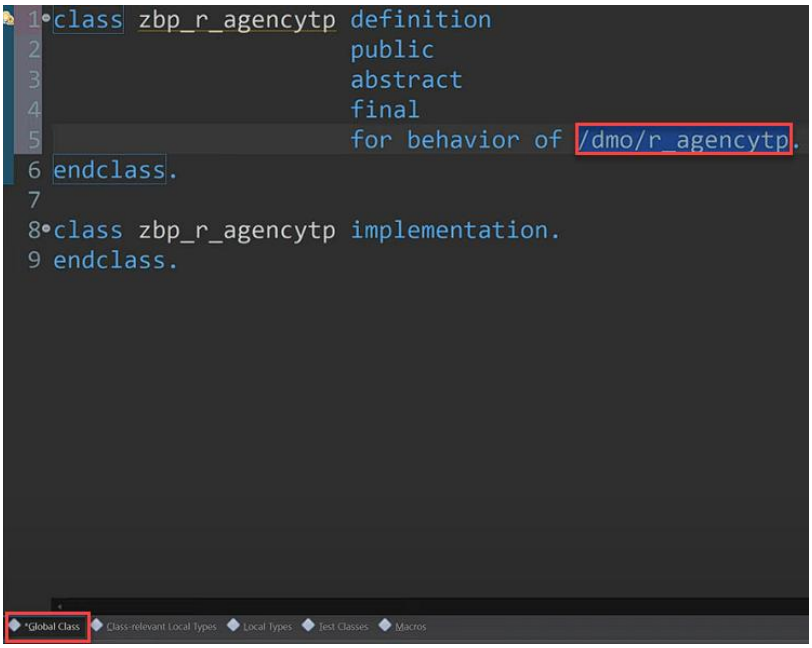
Luego se selecciona el nombre propuesto para la nueva clase Behavior Pool, que se encuentra al finalizar la sentencia **implementation in class.**

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  action (authorization : none) zzAssignType result [1] $self;
}
```

Se utiliza la opción de propuesta sobre el mensaje de advertencia para crear la clase. Al hacer doble clic en la opción “Create behavior implementation class”, se creará dicha clase finalizando el proceso al incorporar el artefacto en el paquete de desarrollo "Z" en una orden de transporte.



En la pestaña “Global Class” se hace referencia a la entidad raíz, verificando que la lógica añadida en esta clase de extensión "behavior pool" afectará dicha entidad.



Al posicionar el cursor encima del nombre de la acción se comprueba que la lógica de esta por medio de un método en dicha clase no ha sido implementada.

Por lo que al hacer clic en el nombre de la acción y utilizar la ayuda del editor y seleccionar la opción “Add Method for action in new local…”, permitirá la implementación del método asociado a la acción.

```
1 extension using interface /dmo/i_agencytp
2 implementation in class zbp_r_agencytp unique;
3
4 extend behavior for /DMO/Agency
5
6{
7   action (authorization : none) zzAssignType result [1] $self;
8 }
```

En la pestaña Local Types se puede comprobar que el método ha sido implementado.

```
CLASS lhc_/dmo/agency DEFINITION INHERITING FROM cl_abap_behavior_handler.

  PRIVATE SECTION.

    METHODS zzAssignType FOR MODIFY
      IMPORTING keys FOR ACTION /DMO/Agency~zzAssignType RESULT result.

ENDCLASS.

CLASS lhc_/dmo/agency IMPLEMENTATION.

  METHOD zzAssignType.
  ENDMETHOD.

ENDCLASS.

*"* use this source file for the definition and implementation of
*"* local helper classes, interface definitions and type
*"* declarations
```

Y es dentro de ese método que se agregará la lógica asociada a la acción. Sin embargo en primera instancia no podemos utilizar la sentencia **in local mode**, porque realmente esta clase no se encuentra en el nodo de la entidad raíz.

```
method zzAssignType.

   read entities of /DMO/R_AgencyTP in local mode
        entity /DMO/Agency
        fields ( CountryCode zzagtypezag )
        with corresponding #( keys )
        result data(agencies).


endmethod.
```

Por lo tanto para eliminar la advertencia se debe eliminar la sentencia **in local mode** si se va a consultar directamente desde la entidad raíz.

Por otro lado, se puede utilizar la entidad de interfaz para realizar consultas, ya que la clase hace referencia al nodo correspondiente de dicha entidad. Esto permite el uso de la sentencia **in local mode**, mejorando significativamente el rendimiento de las consultas.

```
method zzAssignType.

  read entities of /DMO/I_AgencyTP in local mode
       entity /DMO/Agency
       fields ( CountryCode zzagtypezag )
       with corresponding #( keys )
       result data(agencies).

  loop at agencies assigning field-symbol(<agency>).

    case <agency>-CountryCode.
      when 'US'.
        <agency>-zzagtypezag = 'AM'.
      when others.
        <agency>-zzagtypezag = 'EU'.
    endcase.

  endloop.


endmethod.
```

Posteriormente agregar la lógica correspondiente que en este caso el objetivo será de que al seleccionar un registro permitirá cambiar el estado del campo Agency Type dependiendo del código del país.

```
method zzAssignType.

  read entities of /DMO/I_AgencyTP in local mode
       entity /DMO/Agency
       fields ( CountryCode zzagtypezag )
       with corresponding #( keys )
       result data(agencies).

  loop at agencies assigning field-symbol(<agency>).

    case <agency>-CountryCode.
      when 'US'.
        <agency>-zzagtypezag = 'AM'.
      when others.
        <agency>-zzagtypezag = 'EU'.
    endcase.

  endloop.

  modify entities of /DMO/I_AgencyTP in local mode
         entity /DMO/Agency
         update fields ( zzagtypezag )
         with corresponding #( agencies ).


endmethod.
```

Para luego utilizar el parámetro result para actualizar la interfaz de usuario.

```
  modify entities of /DMO/I_AgencyTP in local mode
         entity /DMO/Agency
         update fields ( zzagtypezag )
         with corresponding #( agencies ).

  result = value #( for agency in agencies ( %tky   = agency-%tky
                                             %param = agency ) ).


endmethod.
```

1.6. Behavior Definition Projection – Habilitar Extensión

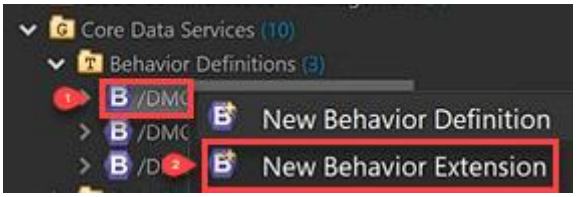
El proceso para habilitar la acción en la interfaz de usuario comienza con la extensión de la definición de comportamiento de tipo projection. Esta extensión permite habilitar la nueva acción creada sobre la definición del comportamiento en la entidad de consumo.

La habilitación de las acciones se logra mediante la inclusión de la sentencia **"use action"** en la definición del comportamiento de la entidad de consumo o proyección. Esta acción puede validarse dentro de la propia definición del comportamiento. Además, se puede verificar que dicha definición permite realizar extensiones para habilitar nuevas acciones, mediante el uso de la instrucción extensible.

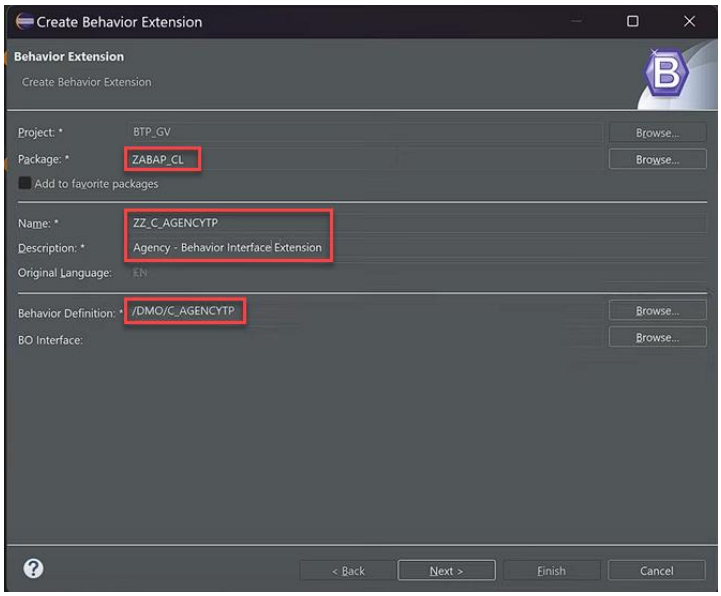
```
projection;
strict(2);
use draft;
extensible;

define behavior for /DMO/C_AgencyTP alias /DMO/Agency
extensible
{
  use create;
  use update;
  use delete;

  use action Resume;
  use action Edit;
  use action Activate;
  use action Discard;
  use action Prepare;
}
```

Creación de la Extensión de comportamiento para la entidad de consumo o proyección:

Para la creación de la extensión de comportamiento para la entidad de consumo se puede hacer clic derecho en el nombre de la definición en la carpeta “Core Data Services” del paquete de desarrollo Z donde se estarán realizando los desarrollos de las extensiones y seleccionar la opción “New Behavior Extension”.



De manera que en la siguiente ventana se especifica el paquete Z, un nombre, descripción y por la acción realizada anteriormente ya se encuentra especificada en la definición de comportamiento de la entidad de consumo a la que hace referencia.



Generando el siguiente elemento de extensión.

```
extension for projection;

extend behavior for /DMO/Agency
{
}
```

La habilitación de la acción se logra mediante la agregación de la sentencia "use action" en la definición de la extensión. Se verifican y activan los cambios, asegurando que la implementación sea correcta y libre de advertencias. Es importante mencionar que el nombre de la acción se debe colocar manualmente debido a que el editor de texto no reconoce el nombre de la acción a extender para su habilitación. El nombre debe ser copiado desde la extensión de comportamiento basado en el Behavior Definition de la entidad raíz.

```
extension for projection;

extend behavior for /DMO/Agency
{
  use action zzAssignType;
}
```

1.7. Metadata Extensions – Exponer Acción en UI

Los Metadata Extensions son esenciales para personalizar y adaptar la interfaz de usuario a las necesidades específicas del requerimiento, mejorando así la funcionalidad y usabilidad de la aplicación. Permiten realizar cambios significativos sin comprometer la estabilidad del sistema, facilitando la actualización y mantenimiento de la UI. Por lo que se utilizará para habilitar y configurar un nuevo botón en la interfaz de una tabla, que permitirá modificar el tipo de agencia de acuerdo con la lógica implementada en el behavior pool.

Es por medio del Metadata extension extendida de la entidad de consumo creado con anterioridad que se puede implementar un nuevo botón en la interfaz de usuario y vincularlo a la acción que ejecuta la lógica agregada en la clase Behavior Pool, que extiende la definición de comportamiento de la entidad raíz.

```
@Metadata.layer: #CUSTOMER
annotate entity /DMO/C_AgencyTP with
{
  @UI: { lineItem:       [ { position: 21, importance: #HIGH } ],
         fieldGroup:     [ { position: 21, qualifier: 'General_FG' } ] }
  zzagtypezag;

}
```

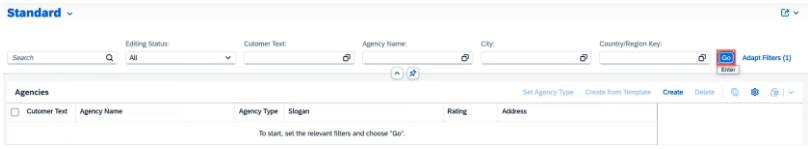
Vinculación del Botón en el Ítem de Línea:

Para agregar el botón en la interfaz, es necesario especificar por medio de la anotación **@UI.lineItem**, la posición utilizando la propiedad **position**. Generalmente, se establece en la posición 10 para que aparezca en primer lugar entre los botones del encabezado de la tabla en la interfaz de usuario. Además, se debe definir el tipo de acción que realizará el botón mediante la propiedad type, en este caso, **#FOR_ACTION**. Posteriormente, se vincula la propiedad **dataAction** con el nombre de la acción implementada, que en este ejemplo es

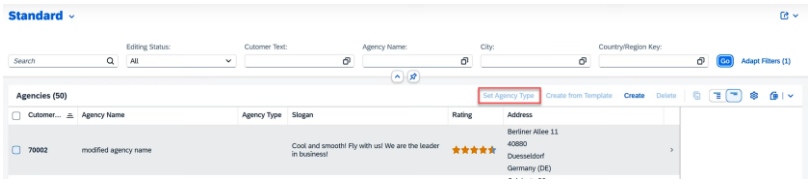
"zzAssignType". Finalmente, se asigna una etiqueta al botón utilizando la propiedad **Label**, de modo que este sea el texto visible en el botón dentro de la interfaz de usuario.

```
@Metadata.layer: #CUSTOMER
annotate entity /DMO/C_AgencyTP with
{
  @UI: { lineItem:       [ { position: 21, importance: #HIGH },
                           { position: 10,
                             type: #FOR_ACTION,
                             dataAction: 'zzAssignType',
                             label: 'Set Agency Type' } ],
         fieldGroup:     [ { position: 21, qualifier: 'General_FG' } ] }
  zzagtypezag;

}
```

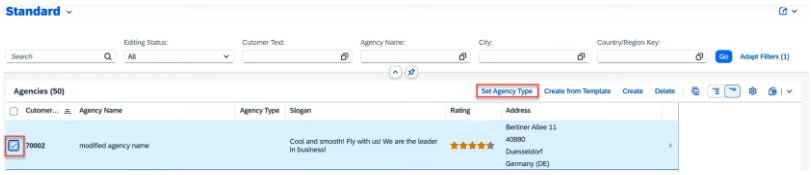
Al ir a la aplicación en la interfaz de usuario y presionar el botón "Go" para mostrar los registros consultados de la capa de persistencia.



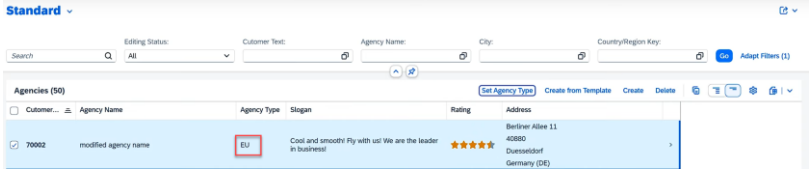
Se puede comprobar que el botón se encuentra habilitado en la interfaz de usuario pero deshabilitado.



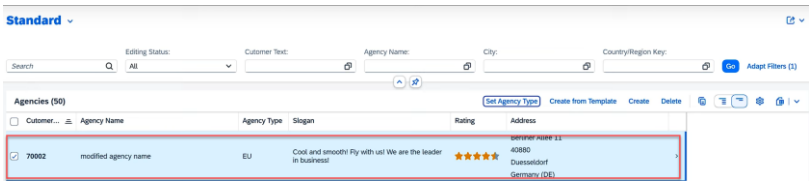
Por otro lado al seleccionar un registro se habilitará y al presionar dicho botón ejecutará la lógica personalizada vinculada a la acción.



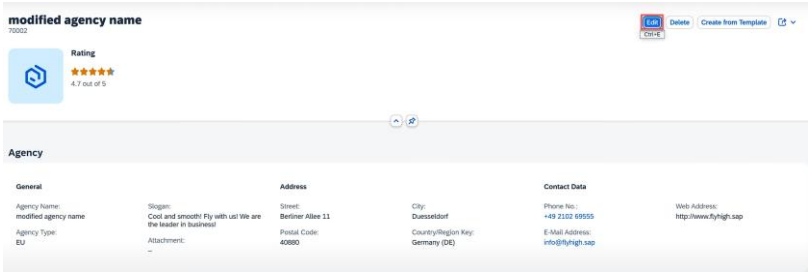
En este este caso cambiar el campo "Agency Type", dependiendo del código de país.



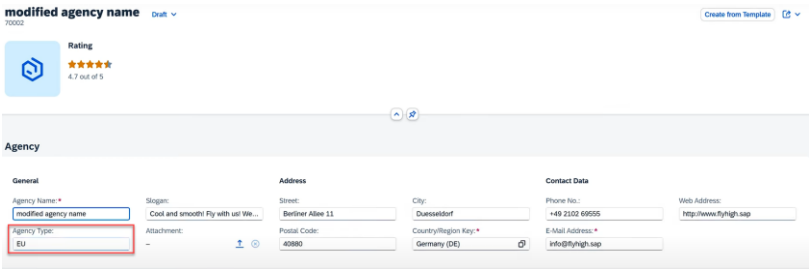
Sin embargo al seleccionar un registro de la tabla.



Y presionar el botón “Edit”.



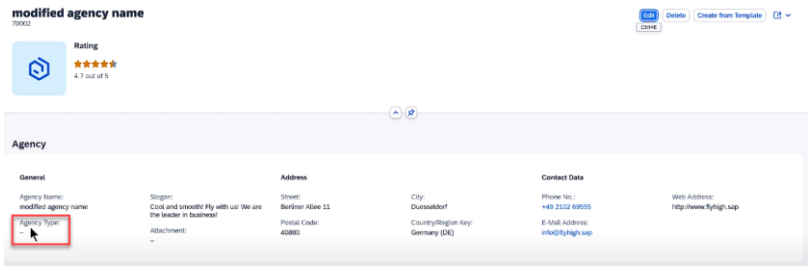
Por el momento permitirá la modificación manualmente del campo personalizado agregado por medio de extensiones.



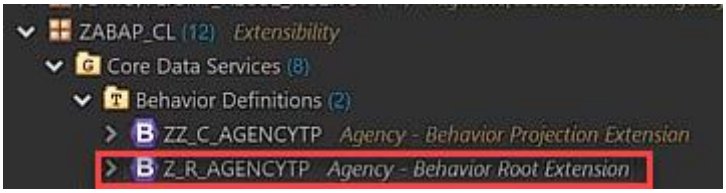
1.8. Campo Solo Lectura

La restricción de solo lectura (read-only) en campos específicos de la interfaz de usuario se utiliza para garantizar la integridad de los datos y limitar el acceso de edición a ciertos campos críticos. Esta funcionalidad se implementa mediante extensiones de metadata (metadata extensions), permitiendo que los campos no sean modificables por los usuarios, mientras se configuran acciones específicas a través de botones en la interfaz.

Para establecer el campo personalizado como no editable o campo de sólo lectura al momento de editar el registro.



Es necesario agregar el nuevo comportamiento del campo en la extensión de comportamiento de la entidad raíz. Recordando que para acceder a dicho elemento se debe acceder a la carpeta “Behavior Definitions” en el paquete Z donde se estarán realizando los desarrollos de las extensiones.



De manera que es necesario ampliar la definición desarrollada.

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  action ( authorization : none ) zzAssignType result [1] $self;
}
```

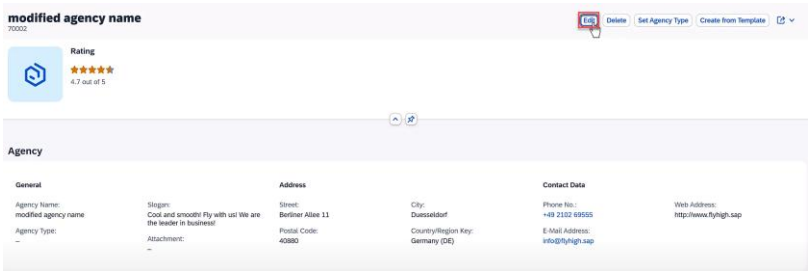
Agregando la instrucción **field,** con la propiedad **readonly,** seguido del nombre del campo que se establecerá como campo de solo lectura.

```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  field ( readonly ) zzagtypezag;
  action ( authorization : none ) zzAssignType result [1] $self;
}
```

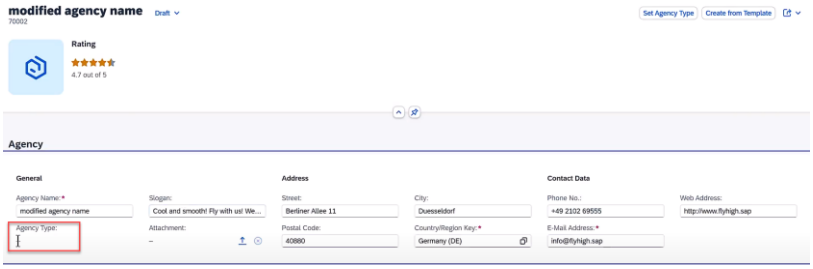
Además de que es posible colocar el botón asociado a la acción en el menú de edición. Por lo que para implementar dicha acción se debe agregar la anotación **@UI.identification.** Con las mismas propiedades establecidas en la anotación **@UI.lineItem.**

```
@Metadata.layer: #CUSTOMER
annotate entity /DMO/C_AgencyTP with
{
  @UI: { lineItem:       [ { position: 21, importance: #HIGH },
                           { position: 10,
                             type: #FOR_ACTION,
                             dataAction: 'zzAssignType',
                             label: 'Set Agency Type' } ],
         fieldGroup:     [ { position: 21, qualifier: 'General_FG' } ],
         identification: [{ position: 10,
                            type: #FOR_ACTION,
                            dataAction: 'zzAssignType',
                            label: 'Set Agency Type'  }] }
  zzagtypezag;

}
```

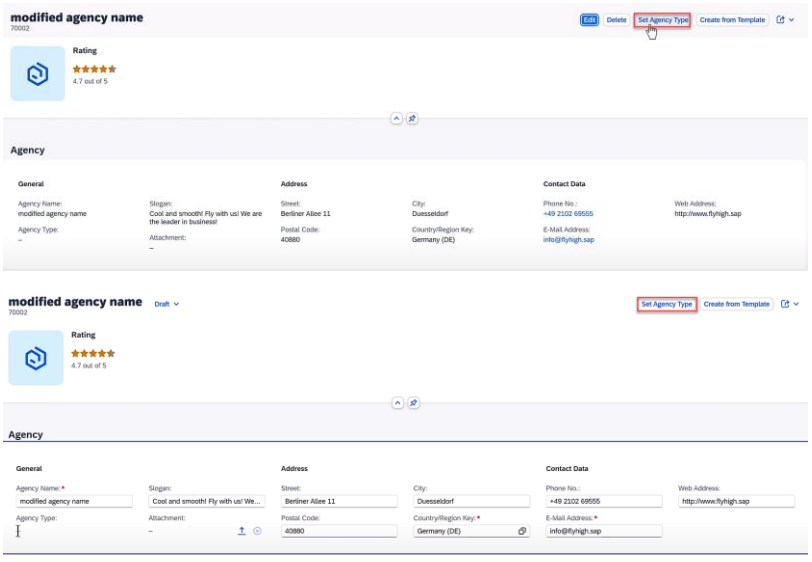
Para comprobar los cambios realizados al seleccionar un registro en la tabla para su modificación al presionar el botón “Edit”.



Se puede evidenciar que el campo no se pueda editar.



Además de que en la visualización y edición del registro seleccionado aparecerá el botón con la acción personalizada.



Permitiendo de esta manera cambiar el estado del campo Agency Type dependiendo del código de país, desde el menú de edición.