Enablement-script-front-end-session (Clean Core)

```
Title:    CAP Development Enablement - Frontend Session
Author:   Rodrigo Riveros A. - rodrigo.riveros@sap.com
Date:     Feb 12 2024
Comment:  Second part of two of technical enablement with CAP on BTP. This
session will cover all the backend creation for a 3 different integrated
systems (SAP + No-SAP inhouse development and this new application that
requires the backend.The second part contains the information about UI
generation according to the available options on SAP BTP.
```

1.- Goals and Objectives

This enablement will show the complete process required to make the backend available for diferente UI consumption. Also the creation of each UI using different options.

2.- Back-end considerations

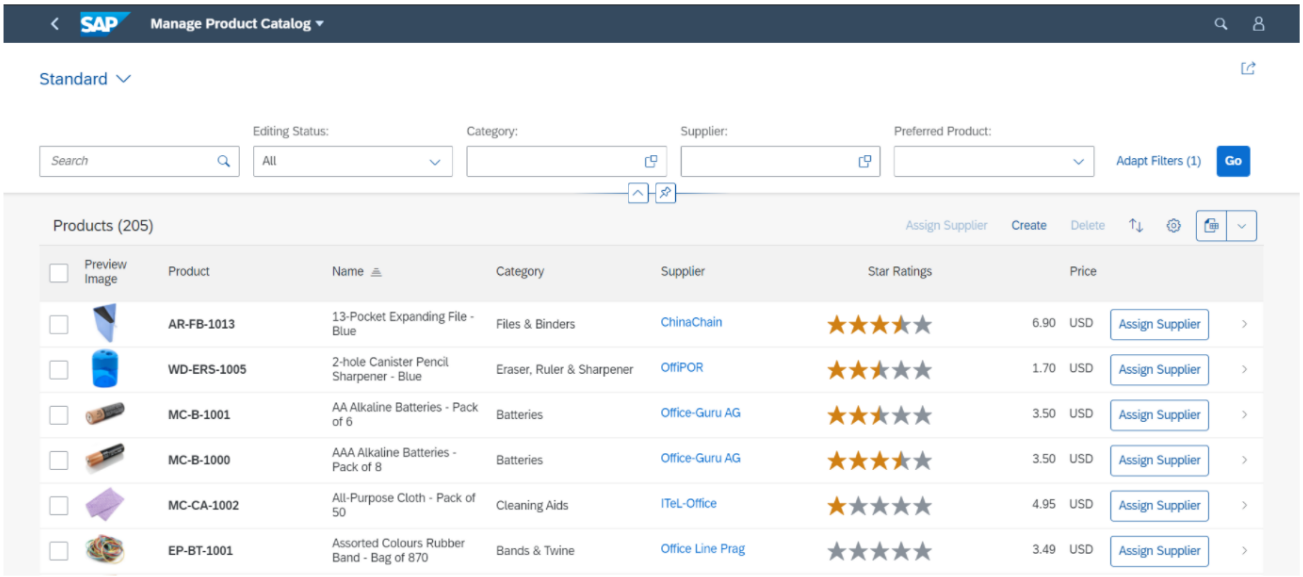
This tutorial will work directly with the CAP Back-end already developed showing the tools that can be used for user interface development.

3.- Fiori Elements intro

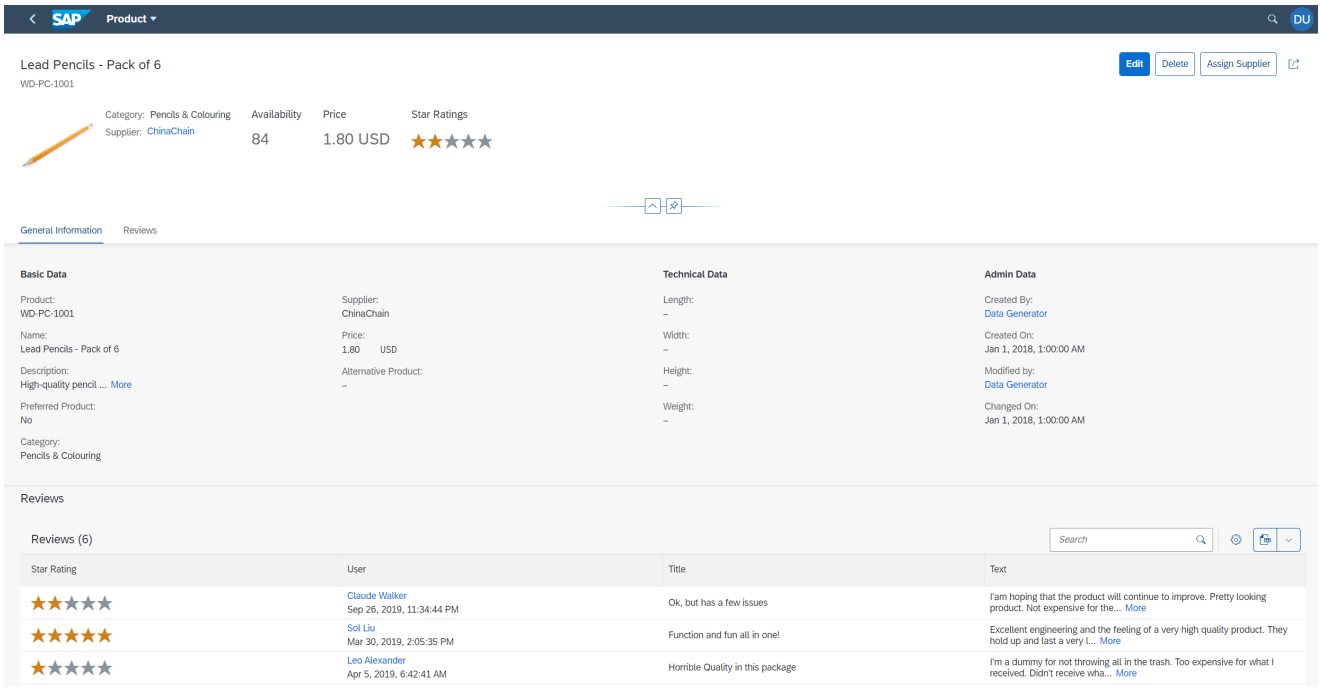
For the initial Fiori elements introduction we will check on UI5 documentation that explain the reason to use Fiori Elements and the **Floorplans** for apps that are available.

Access to UI5 Demo Kit and center the explanation in the List Report and Object Page.

List report:



Object page:



Explain List report characteristics and the setup process (manual and assisted by the App).

3.1.- Create Fiori Elements Floor Plan

In this example we will work with Fiori Extension in VSCode, all the steps can be replicated in Build Code with the same functionality.

> ⚠ **Create a new git branch for Fiori Elements UI before start the Floor Plan Creation Process**

To create a new git branch for fiori-UI, execute in your terminal:

```
git branch fiori-ui
```

To get into the new branch with the control of the project, execute:

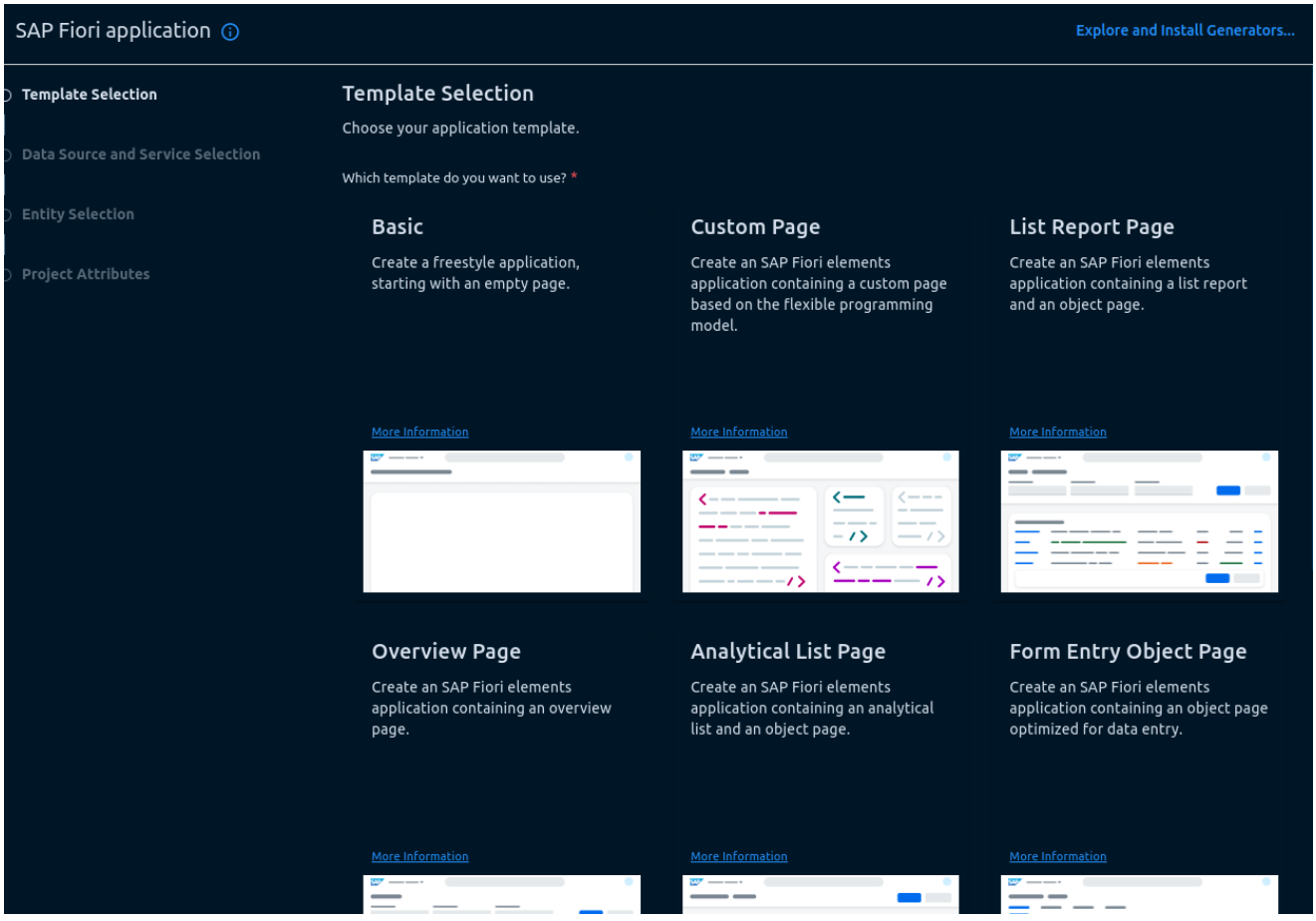
```
git checkout fiori-ui
```

> **With this work model, all the changes will be encapsulated in one branch and if it is required to go back to a previos project instance, the back-end will be safe**

3.1.1.- Use Fiori Generator

To get into the Fiori generator options in VSCode go to **command palette** (`ctrl+shift+p`) and search for *Fiori: Open Application Generator*:

> **In some new installations, there will be a waiting time until the generators are fully installed and ready to use.**



In the selection screen, choose **List Report Page** that will be the Fiori Elements Floor Plan that will be used in the example. Then choose **NEXT**.

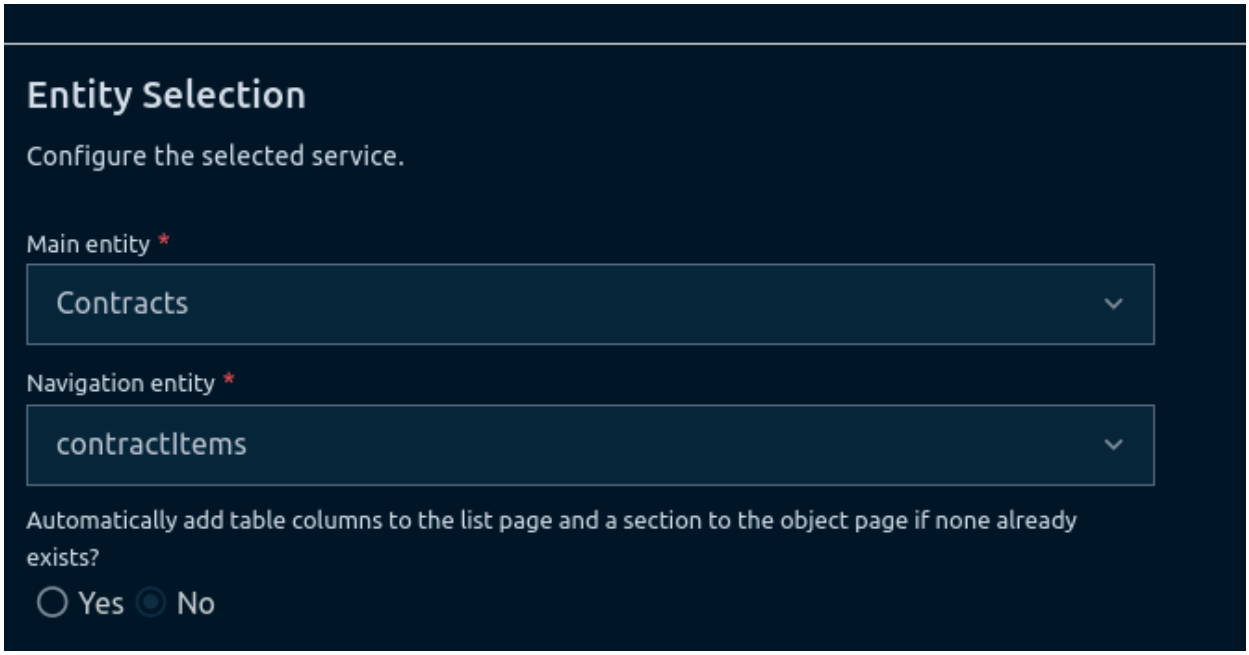
Now need to fill the application characteristics form, selecting Data Source and Services that will be consumed by the UI.

- In Data Source, chose: **Use a Local CAP Project**

- Choose your CAP project
- In the service selection use DemoService(NodeJs) as that service has all the business extensions on the implementation file.
- Click **NEXT**

In Entity Selection Form:

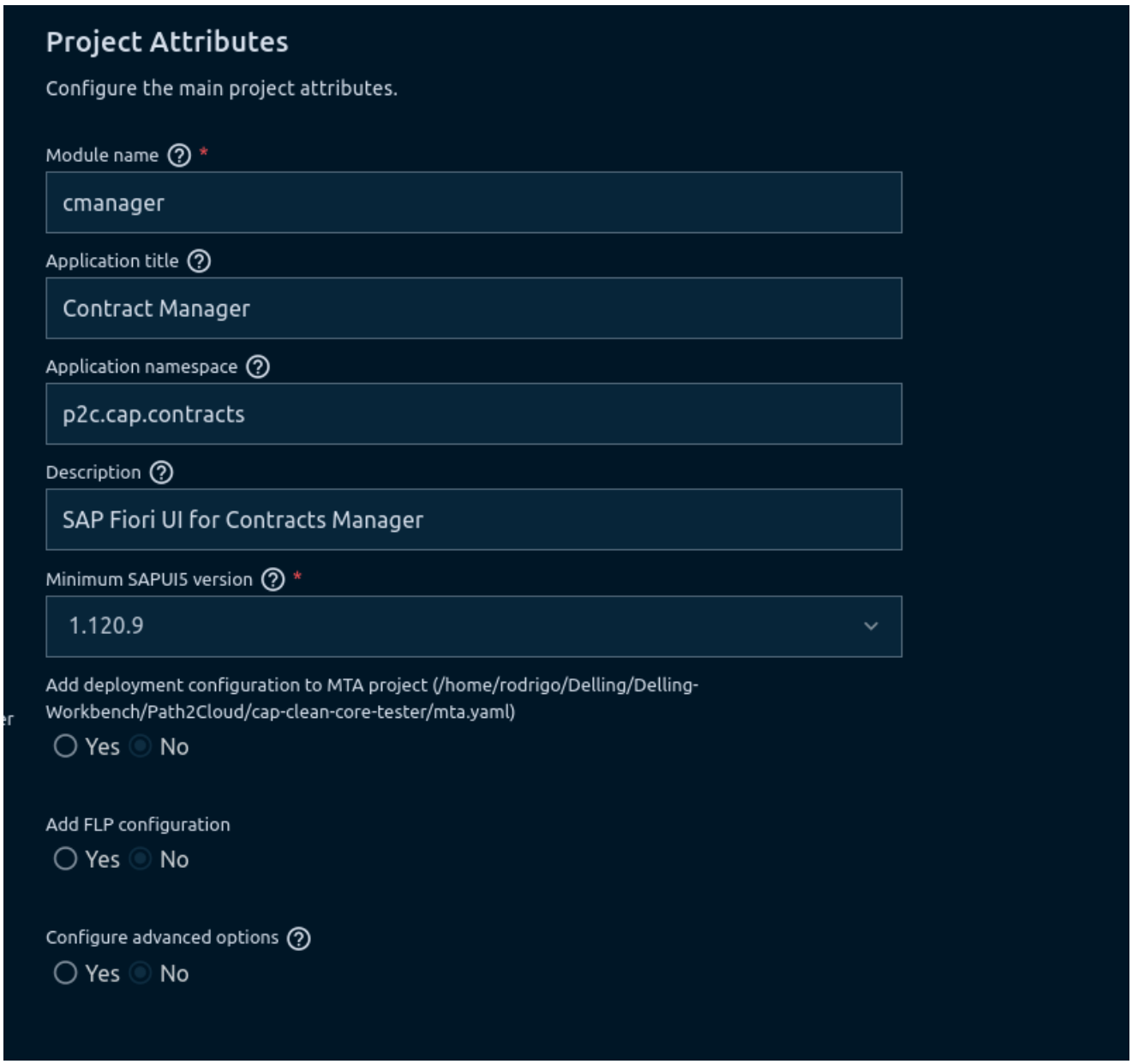
- Choose **Contracts** as the Main entity
- Choose **contractItems** as the Navigation entity
- To the offer of automatic filling columns check **NO**



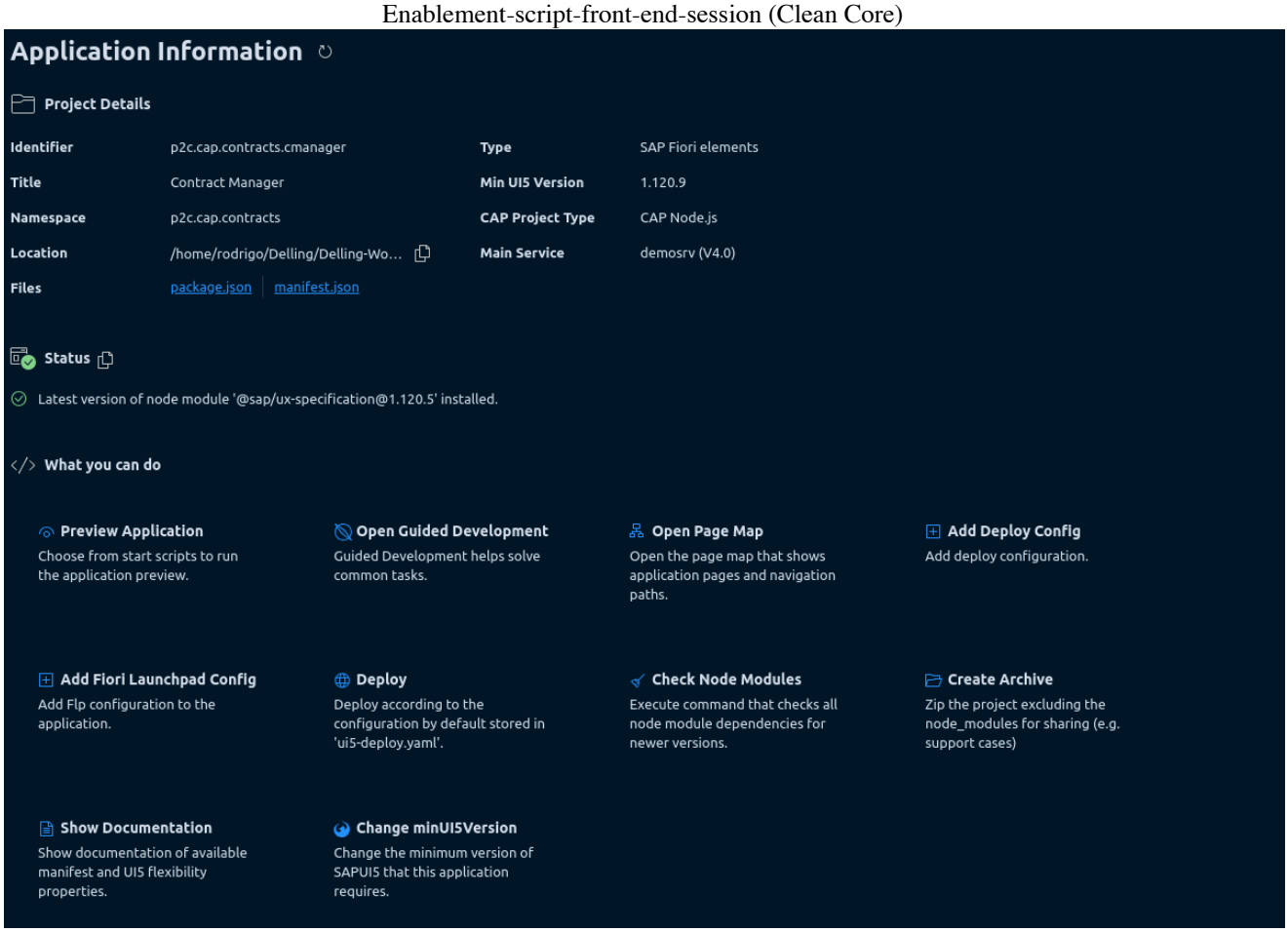
- Finally, fill the project attributes.

Here you can use your own information for project, app title, namespace and description. Try not to change the Minimum SAPUI5 version. For the advanced

configuration answer no to all for this time, next you will now about each functionality and choose what you want to use on discretion.



- Push **FINISH**. The application will start the automatic creation and deployment into the dev system. At the end of the process tyhe screen will show the Application Info page :

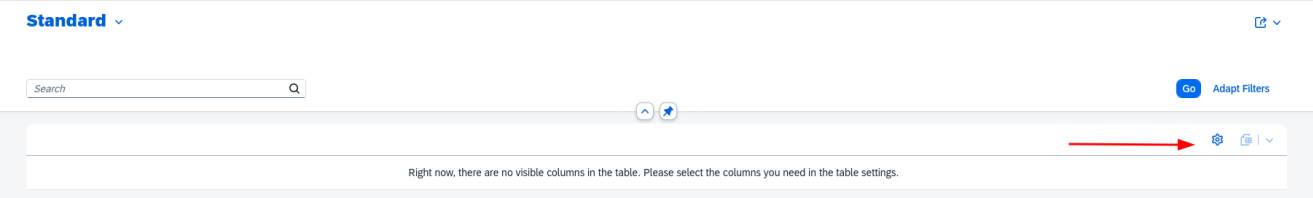


check the information `Latest version of node module '@sap/ux-specification@1.120.5' installed.`, if this package is not installed you will need to before the execution of Fiori Generator.

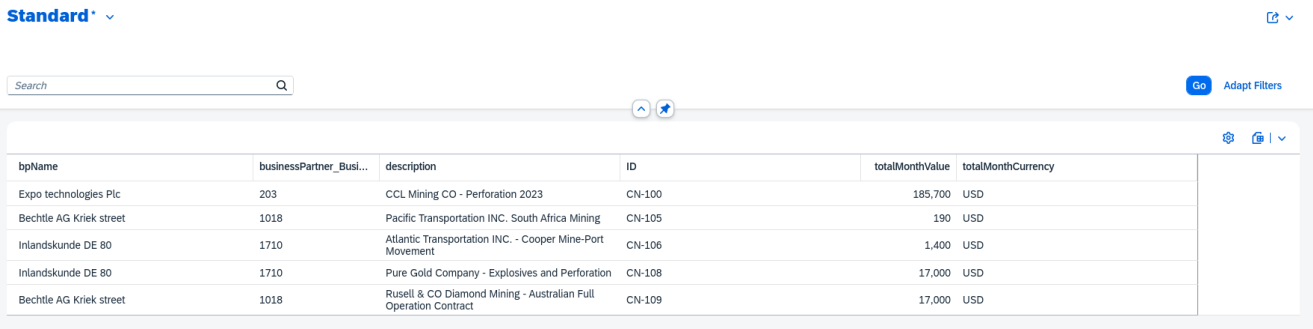
It is very common to close the *Application Info* page, if that happen, go to command palette with `ctrl+shift+p` and in the search box type: `Fiori: Open Application Info`. or with right click on the application name folder and choose *Open Application Info*

- Run `cds watch --profile sandbox` and in the main page the application will be available:

The application will show an empty screen, but the wheel icon will let add table fields now in runtime.

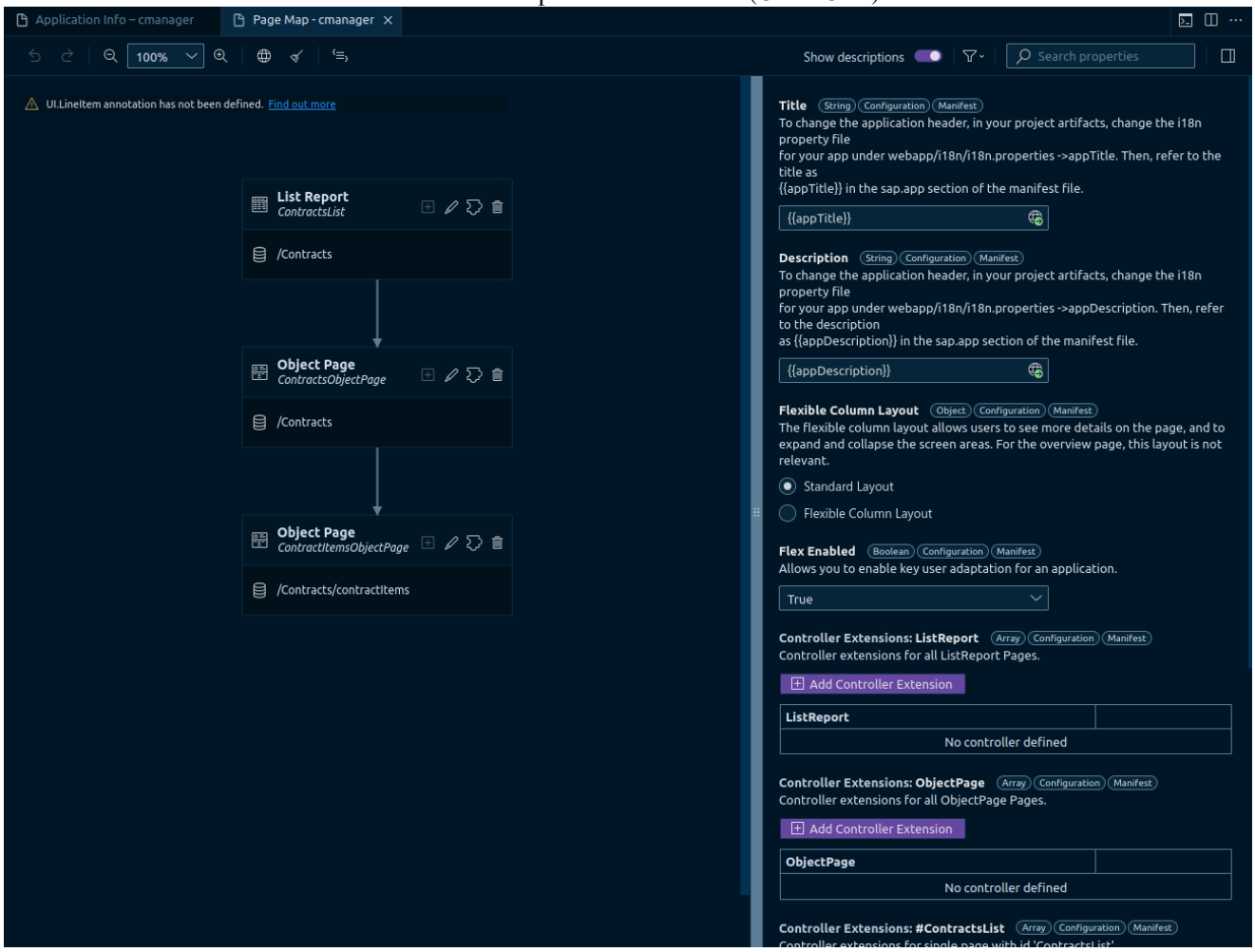


After choose the column fields, press GO button. If the BP information is not added, the UI will show an error, that's because the BP Information is part of the metadata and execute a query to external system. If the BP field is not present, the table can´t be displayed



3.2.- UI Map overview

At this point the application is only the automatic generated code based on the service annotations, to check about the UI structure and how the different annotations are created to change the interface is needed to access the application page map.
Right click on the application folder and choose **show page map**

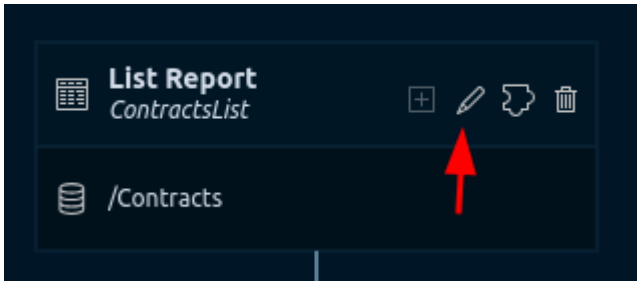


The first sight is the complete application map. It has three pages that will have:

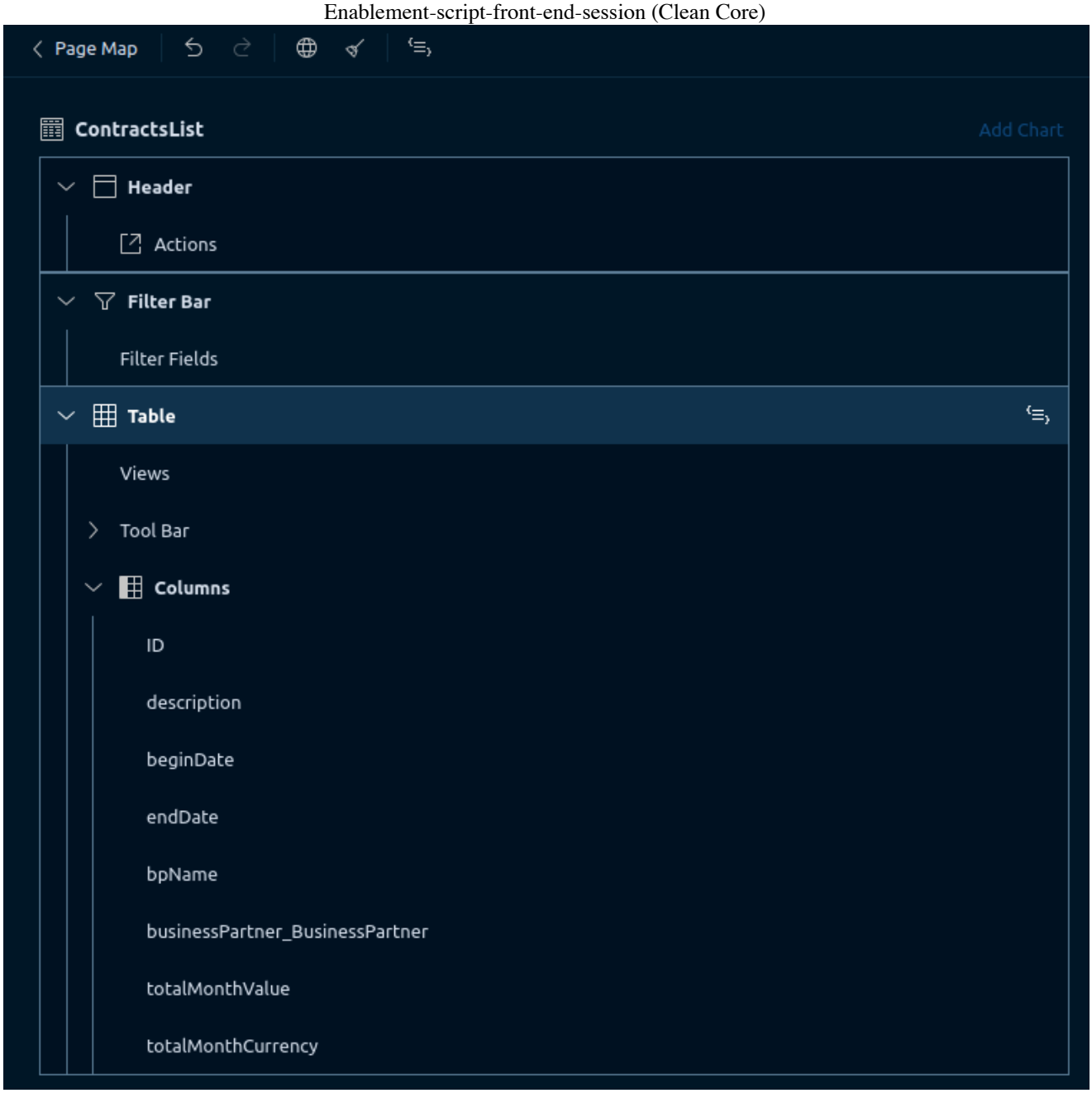
- Contract List
- Contract Object Page (With Contract Items Detail)
- Object Page (Contract Item Detail)
  That are the default creation for the UI based on the creation instructions.

3.3.- Contract entity

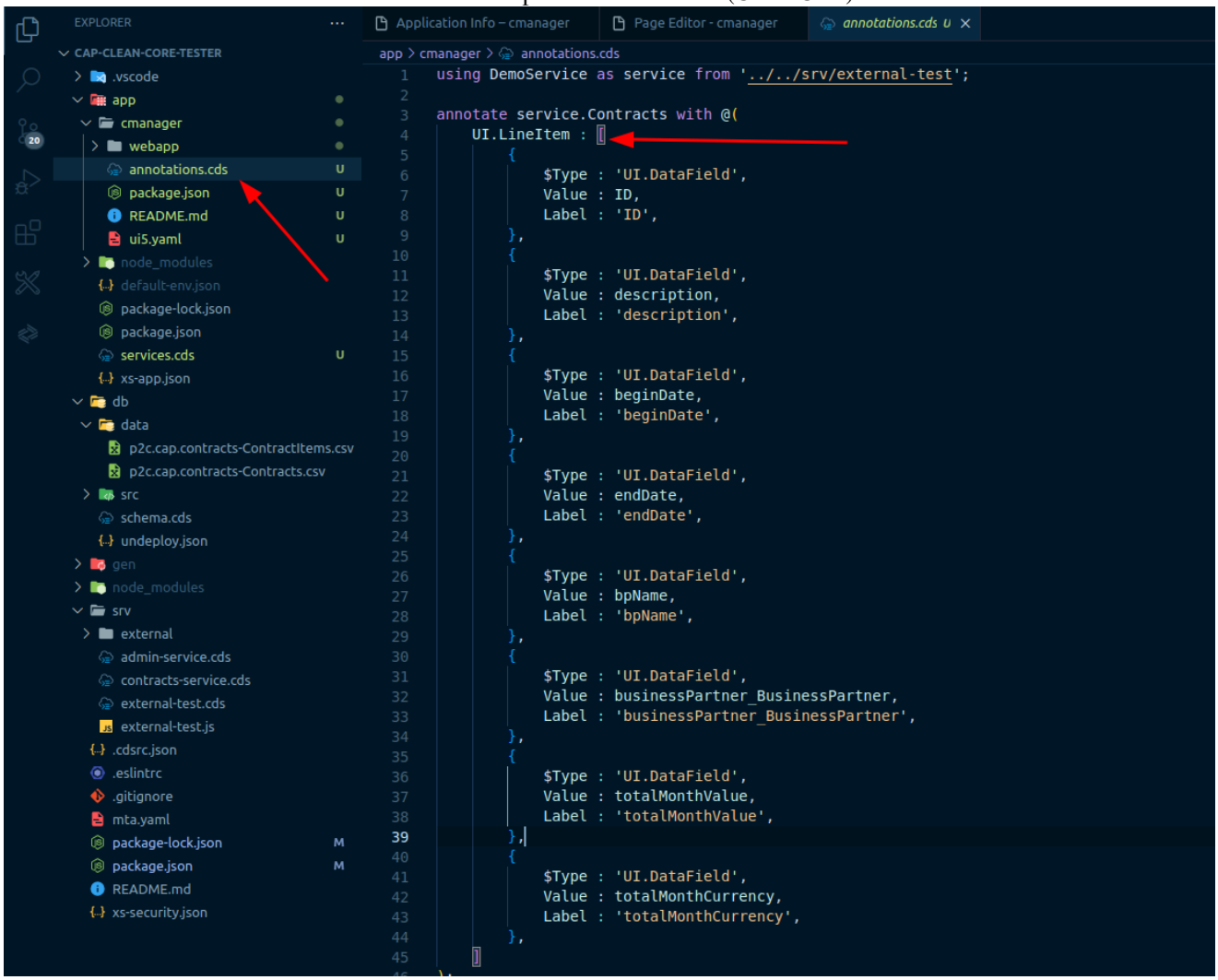
Start working with the Contracts entity, add the columns to the list page and work with filters. To start get into Configure Page item:



The screen will show the schematic vision of the Floor Plan with all the sections. Go to the bottom section **Columns** and in the **+ -> Add Basic Columns** sign start adding the required fields and the column order:
After the column selection, the page map should looks like:



All the configurations created in Fiori Elements are translated to annotations that will be placed in specific scope file. Each app will have the annotations.cds file and the scope will be only for that app. In this case, the column definition will create the LineItem configuration for the Floor Plan (Remember there is only one place where the LineItem can be placed). In this case, after saving, the annotation file will looks like:



> **This is a good way to understand how the annotations are created (plus the documentation), so in next projects will be possible to create the annotations directly, has a pre-defined dictionary for annotations or even review the code crated by some AI assistant like Joule**

3.3.1 Global definitions

There **annotations.cds** file will act only for the app where is created, but inside the **app** folder it is possible to have many applications and sometimes the annotation is needed to apply to all application. In order to do that, create `common.cds` file inside app folder but outside all applications folder.

In this case the annotation will define the Label for each entity component across all applications, the content of the **common.cds** file will be:

```
using { p2c.cap.contracts as srv } from '../srv/external-test.cds';
using from './cmanager/annotations.cds';


annotate srv.Contracts with {
        ID @title : 'Contract';
```

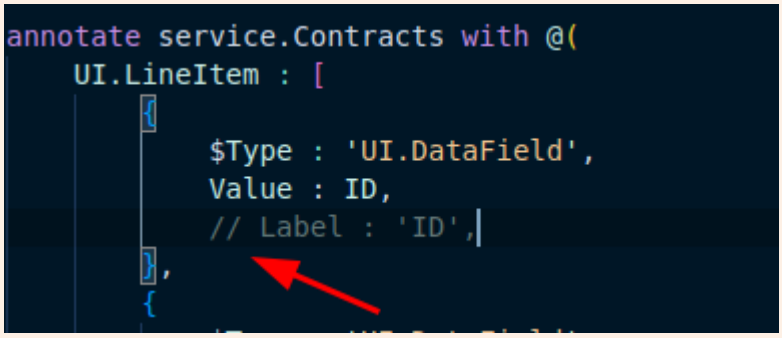
```
        description @title : 'Description';
        beginDate @title : 'Begin Date';
        endDate @title : 'End Date';
        totalMonthValue @title : 'Total Contract Value';
        businessPartner @title : 'BP Number';
        contractItems @title : 'Items';
        bpName @title : 'Business Partner';
};

annotate srv.ContractItems with {
        contract @title : 'Contract ID';
        beginDate @title : 'Begin Date';
        endDate @title : 'End Date';
        price @title : 'Item Price';
        tool @title : 'Tool ID';
        toolName @title : 'Tool Code';
};
```

ⓘ To set up the use of common.cds file is needed to add the use rule to the services.cds file in the applications root folder, the file should looks like:

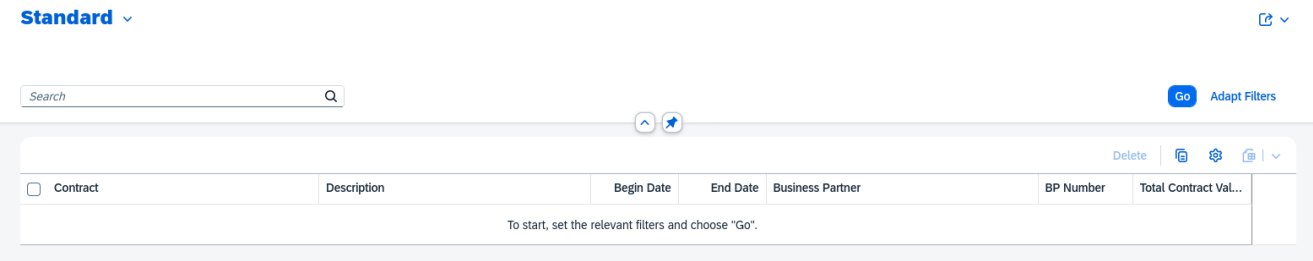
```
using from './cmanager/annotations';
using from './common';
```

In this case, there is only one application so will have only 2 lines, the app one and the common, when the apps number grow, there should be one for each app plus common.

⚠ Since the local annotations are priority against global, to see the effect of the common annotations, there will be needed to comment the label field in the annotations.cds file for each column:

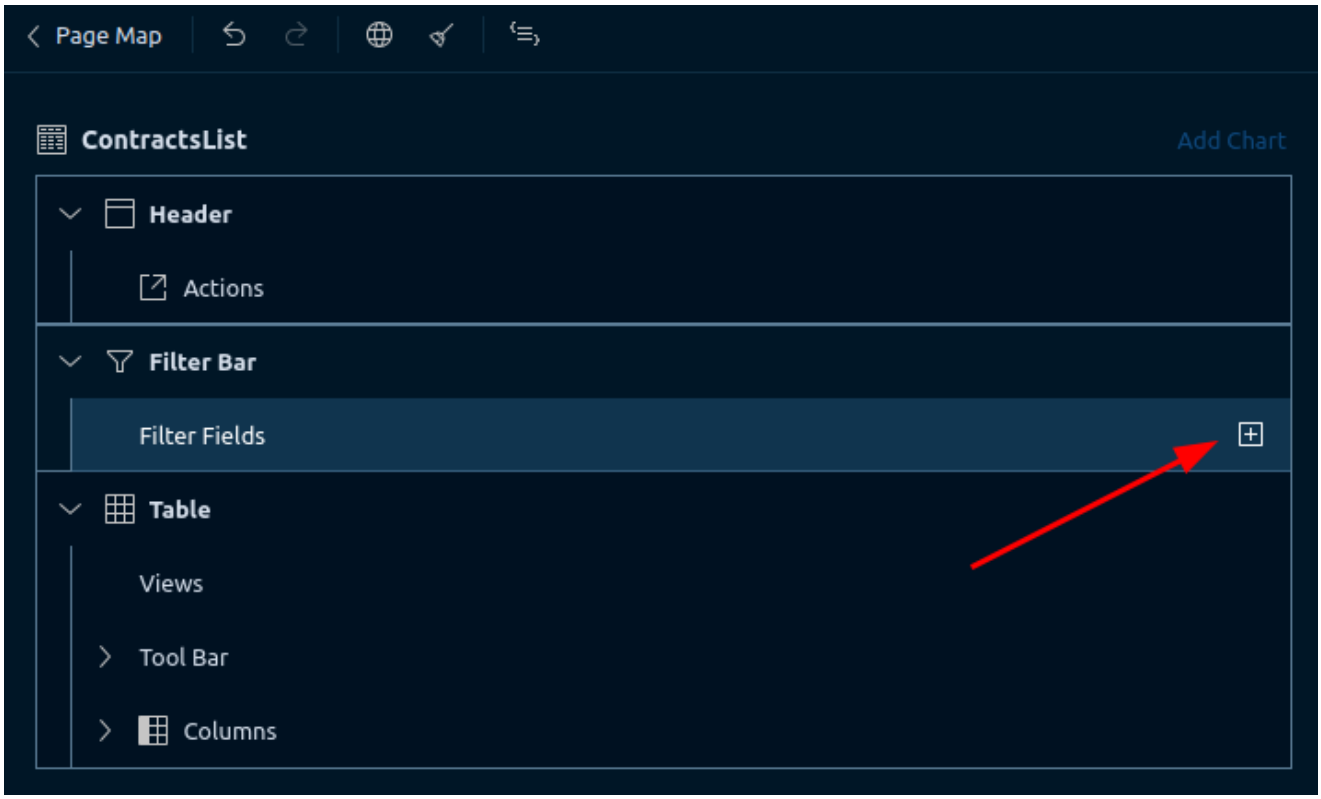


After changes, the initial screen has the defined common columns names:



3.3.2.- Filters basic design

The filter creation follows the same process as the columns in the table.
Go to the page map and in **filter fields** use the **+** and **add filter fields** option to add filters:



If you are not sure about which annotation file contains some definition created in the page map, use the <=> button next to the definition and the editor will open the annotation file.

In this case the filter annotations were positioned in **common.cds** file, since it is only needed for this application, the annotations will be move to the **annotations.cds** file instead.

common.cds U

app > common.cds

```
using { p2c.cap.contracts as srv } from '../srv/external-test.cds';
using from './cmanager/annotations.cds';

annotate srv.Contracts with {
    ID @title : 'Contract';
    description @title : 'Description';
    beginDate @title : 'Begin Date';
    endDate @title : 'End Date';
    totalMonthValue @title : 'Total Contract Value';
    businessPartner @title : 'BP Number';
    contractItems @title : 'Items';
    bpName @title : 'Business Partner';
};


annotate srv.ContractItems with {
    contract @title : 'Contract ID';
    beginDate @title : 'Begin Date';
    endDate @title : 'End Date';
    price @title : 'Item Price';
    tool @title : 'Tool ID';
    toolName @title : 'Tool Code';
};

annotate DemoService.Contracts with @(
    UI.SelectionFields : [
        businessPartner_BusinessPartner,
        bpName,
        endDate,
    ]
);
```

And in the **annotations.cds** file, paste the filters annotations and a comment to keep

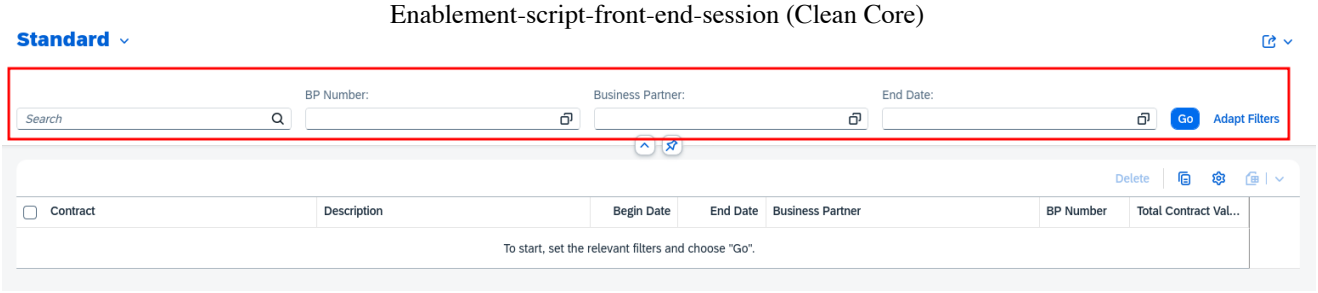
the order into the files:

common.cds U | annotations.cds M

app > cmanager > annotations.cds

```
annotate service.Contracts with @(
    UI.LineItem : [
        {
            $Type : 'UI.DataField',
            Value : beginDate,
            // Label : 'beginDate',
        },
        {
            $Type : 'UI.DataField',
            Value : endDate,
            // Label : 'endDate',
        },
        {
            $Type : 'UI.DataField',
            Value : bpName,
            // Label : 'bpName',
        },
        {
            $Type : 'UI.DataField',
            Value : businessPartner_BusinessPartner,
            // Label : 'businessPartner_BusinessPartner',
        },
        {
            $Type : 'UI.DataField',
            Value : totalMonthValue,
            // Label : 'totalMonthValue',
        },
        {
            $Type : 'UI.DataField',
            Value : totalMonthCurrency,
            // Label : 'totalMonthCurrency',
        },
    ]
);

//* Contracts List Filter Definitions
annotate DemoService.Contracts with @(
    UI.SelectionFields : [
        businessPartner_BusinessPartner,
        bpName,
        endDate,
    ]
);
```

Execute the project to check the filter bar is working with basic functions.



Also it is possible to show the *Single Range* option for filter in the **annotations.cds** file:

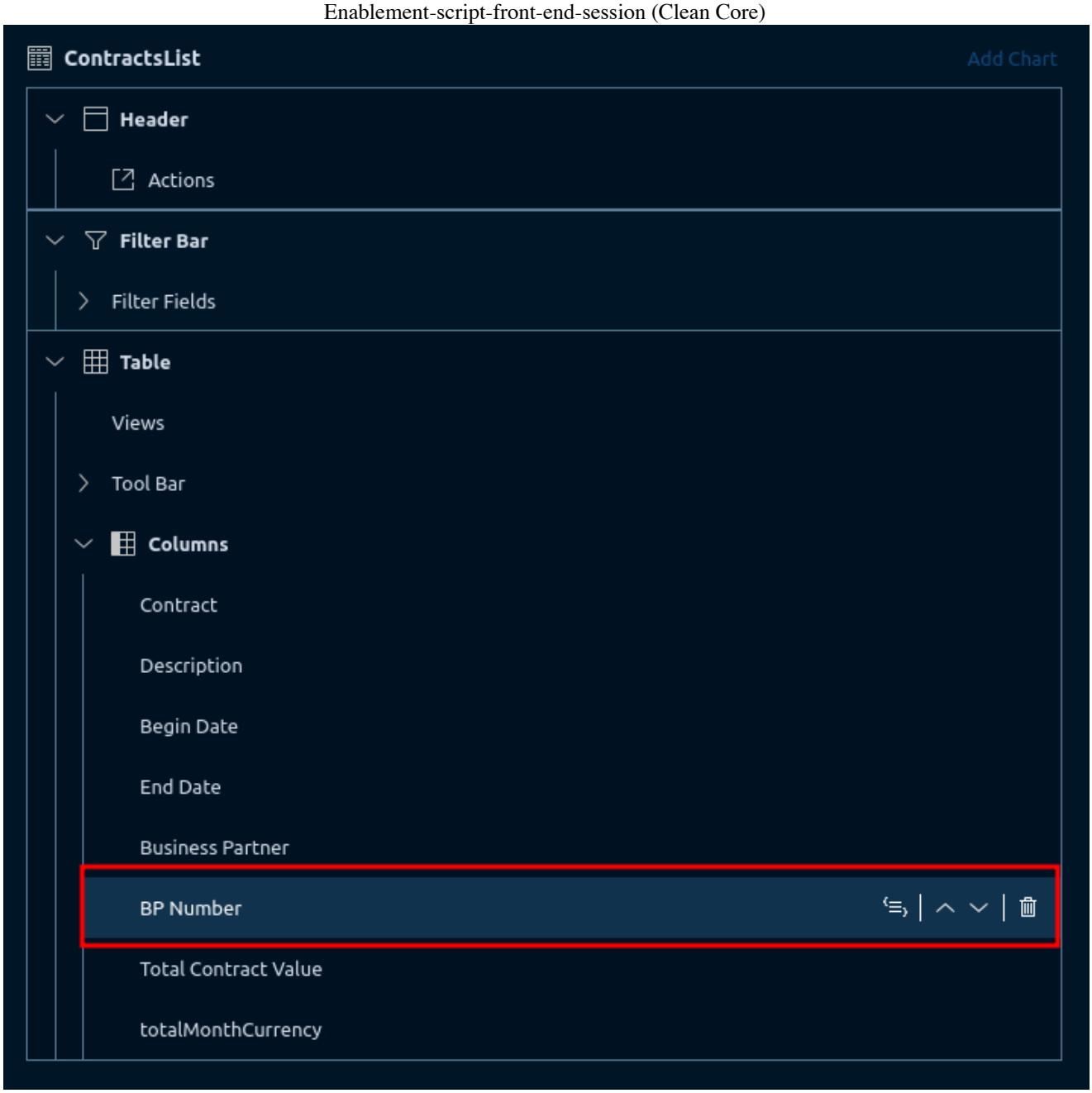
```
//Filter special definitions

annotate service.Contracts with @(
        Capabilities: {
                FilterRestrictions : {
                        FilterExpressionRestrictions: [{
                                Property: 'endDate',
                                AllowedExpressions: 'SingleRange'
                        }]
                },
        }
);
```

3.3.2.- Business Partner Helper

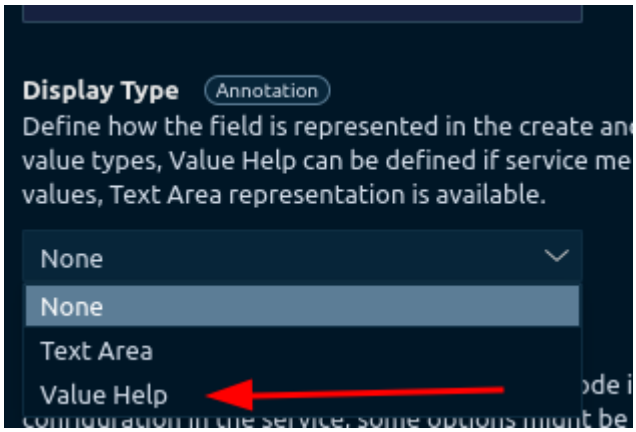
The filters will work in a different way depending of the data that will be used, for dates there are annotations that can help in some way, but for the entities representation is needed to work about the *helper* structure or the way the data will be represented to show and filter information

The business partner filter already created only work with basic capabilities, need some annotations to create a selection list.

Go to page map and find the BP Number column (businessPartner_BusinessPartner).



In the options of the right side go to the **Display Type** *annotation* and choose *value help* in the options box:



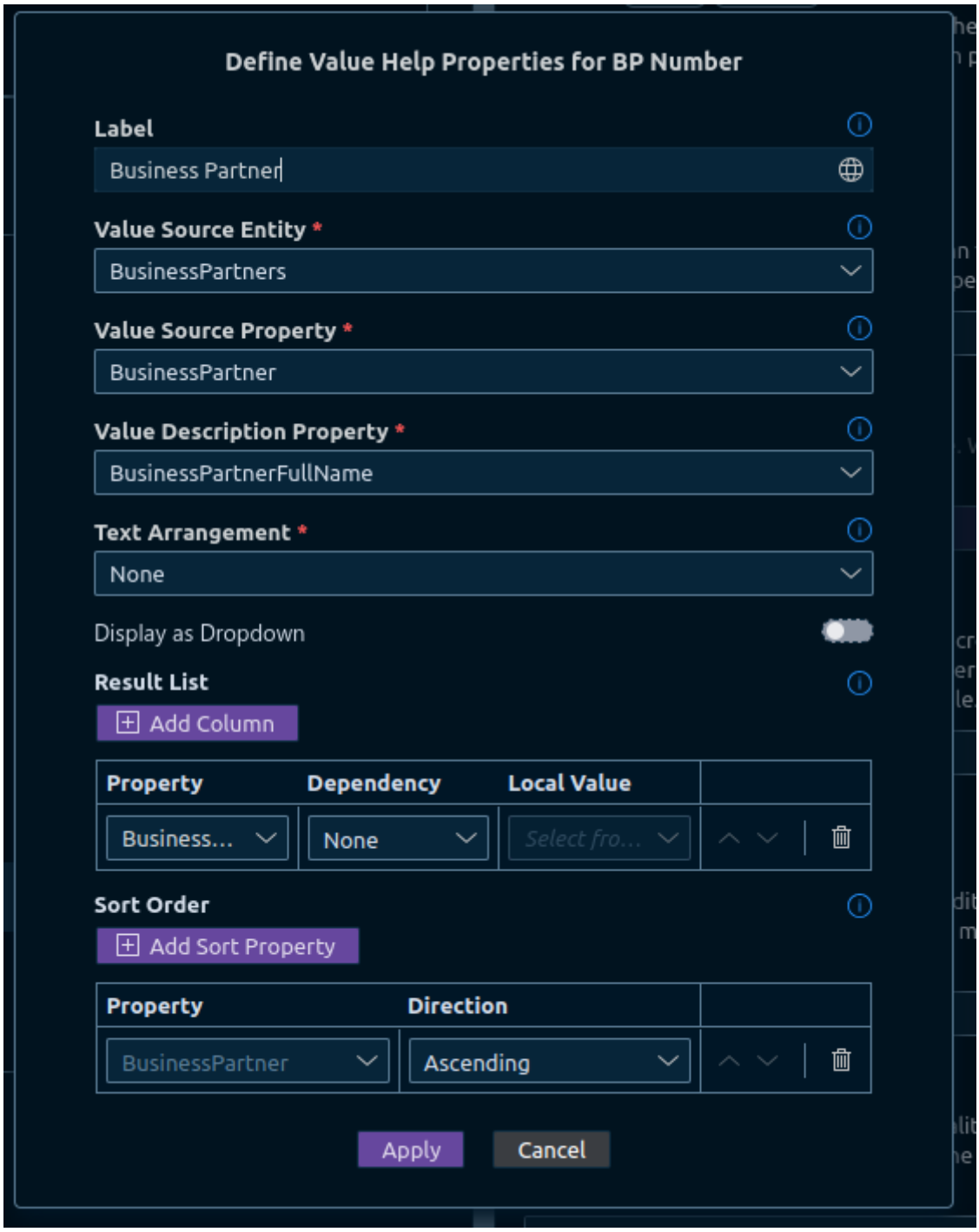
The process will let us to create the value helper for the field, also we can make this global in case we need, putting the annotation inside **common.cds** file.

In this case, the idea is to keep the Business Partner number as the *primary key* parameter and show the text to make the choosing process straightforward. To do this, we will select:

- Value source property: Business Partner
- Value Description property: BusinessPartnerFullName

> **The display as dropdown option will work for few selection options, but in the case of hundreds of business partners it is better to keep it off to have more filter options associated to the field**

The helper configuration will looks like:

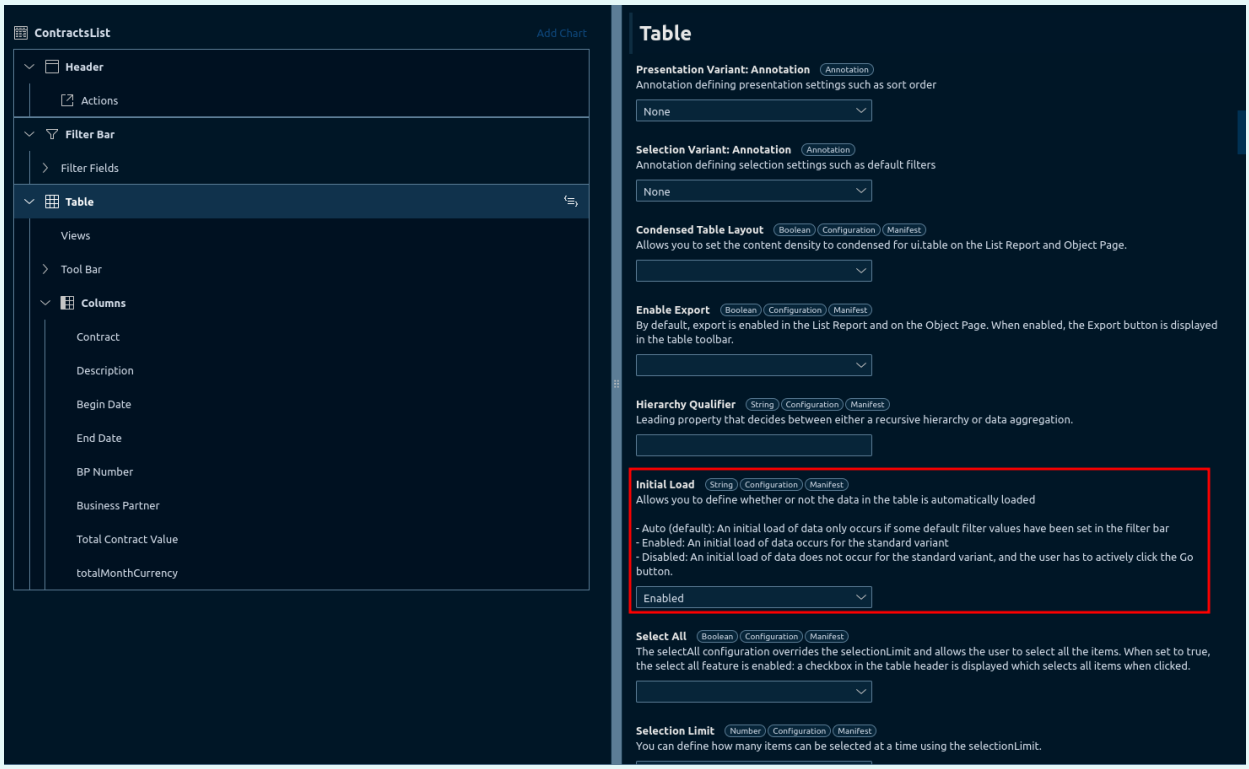


After the definition, the generated code it is into `common.cds` file. The snippet should look like :

```
//*Business Partner Helper Definition

annotate DemoService.Contracts with {
        businessPartner @(Common.ValueList : {
                $Type : 'Common.ValueListType',
                CollectionPath : 'BusinessPartners',
                Parameters : [
                        {
                                $Type : 'Common.ValueListParameterInOut',
                                LocalDataProperty :
businessPartner_BusinessPartner,
                                ValueListProperty : 'BusinessPartner',
                        },
                        {
                                $Type :
'Common.ValueListParameterDisplayOnly',
                                ValueListProperty :
'BusinessPartnerFullName',
                        },
                ],
                Label : 'Business Partner',
                PresentationVariantQualifier :
'vh_Contracts_businessPartner',
        },
        Common.ValueListWithFixedValues : false
)};

annotate DemoService.BusinessPartners with @(
        UI.PresentationVariant #vh_Contracts_businessPartner : {
                $Type : 'UI.PresentationVariantType',
                SortOrder : [
                        {
                                $Type : 'Common.SortOrderType',
                                Property : BusinessPartner,
                                Descending : false,
                        },
                ],
        }
);

annotate DemoService.BusinessPartners with {
        BusinessPartner @Common.Text : BusinessPartnerFullName
};
```

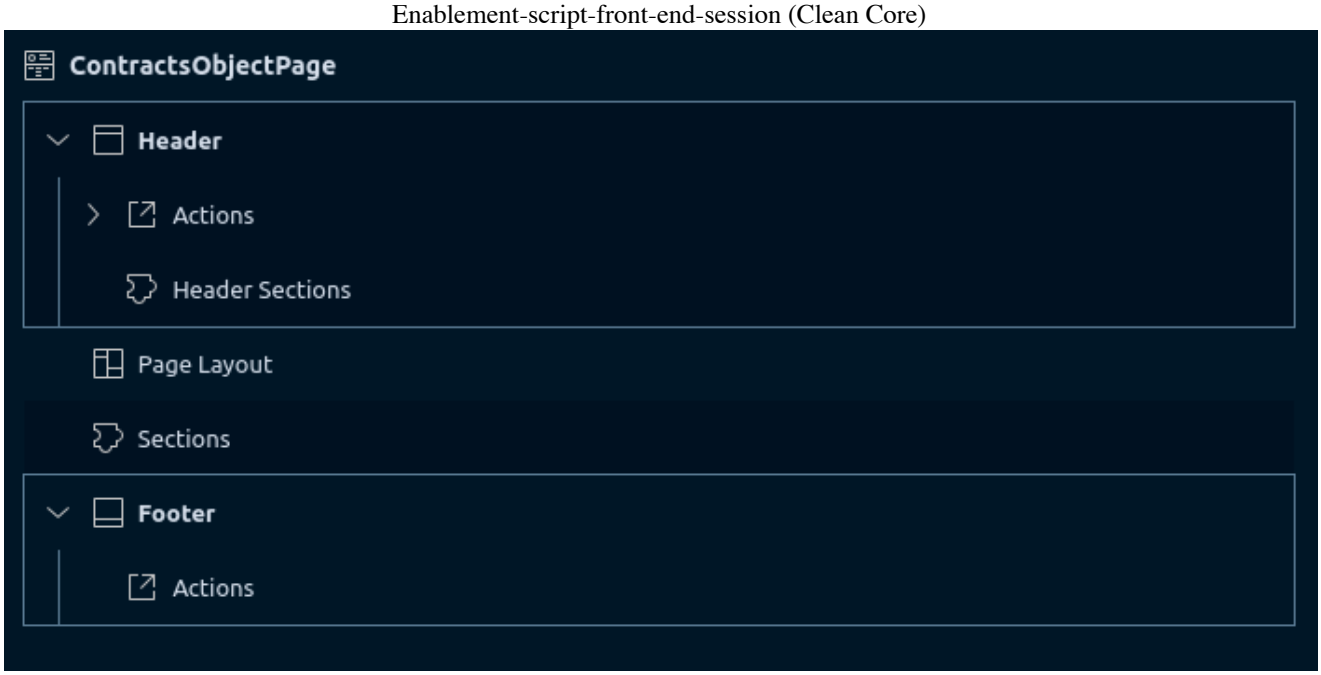
Considering the new representation for entity, the Business Partner name filter is nosense so will be removed from the filter bar by editing the annotations.cds file.

To avoid need to press GO each time to check the values presented by the UI, in the page map, go to Table --> Initial Load option and choose Enable. With this, after each reload the table will be populated with data.



3.3.4.- Field group definition - Contracts detail

To work with contract detail page, need to go to the next floor plan available in the page map **ContractsObjectPage**. According to the floor plan, the available sections will be:

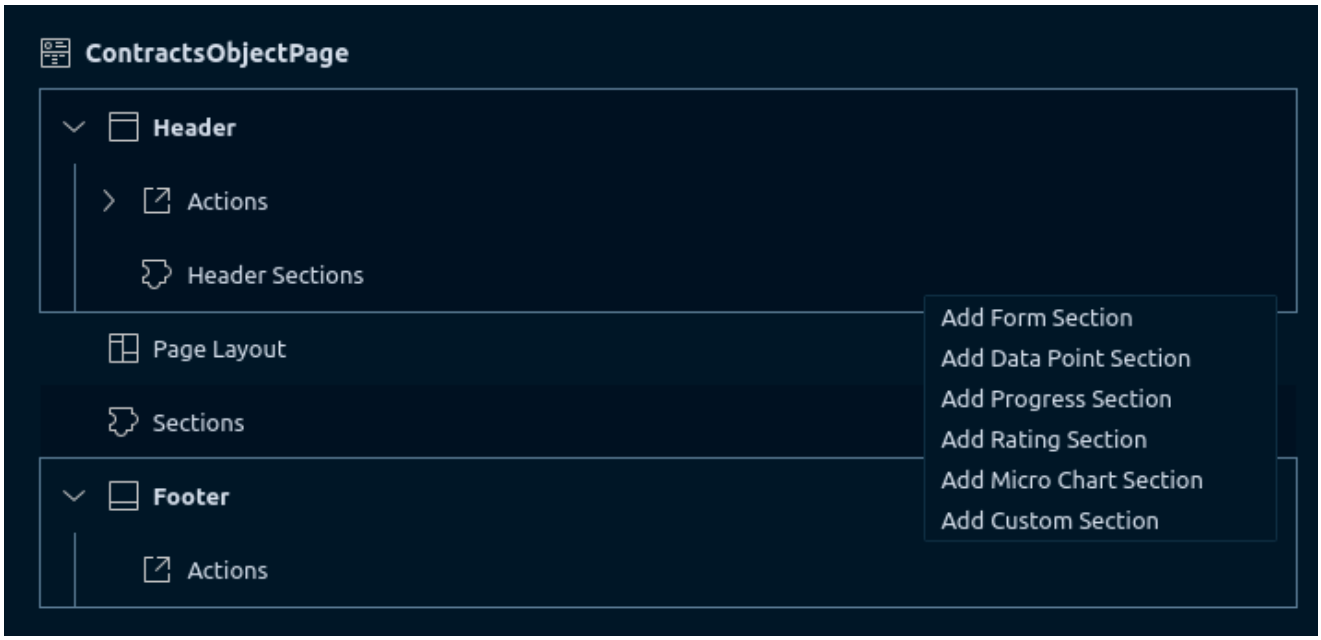


In **Header Sections** the data facets will be created to represent different fields of contract information.
In **Sections** a table will be created with the Contract Items that belongs to the contract.

3.3.4.1 Header sections

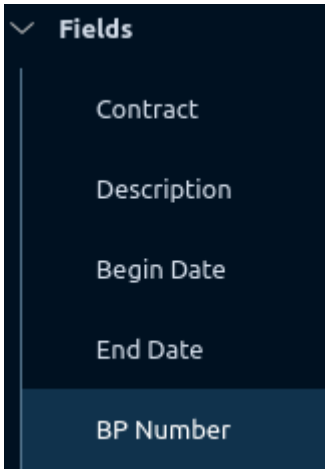
According to the floor plan, the header section can have different facets, in this case we will use Form section and Data Point section. For more information about the rest of the options can go to: [Fiori elements documentation]

Choose **Header Sections -> Add form section** :

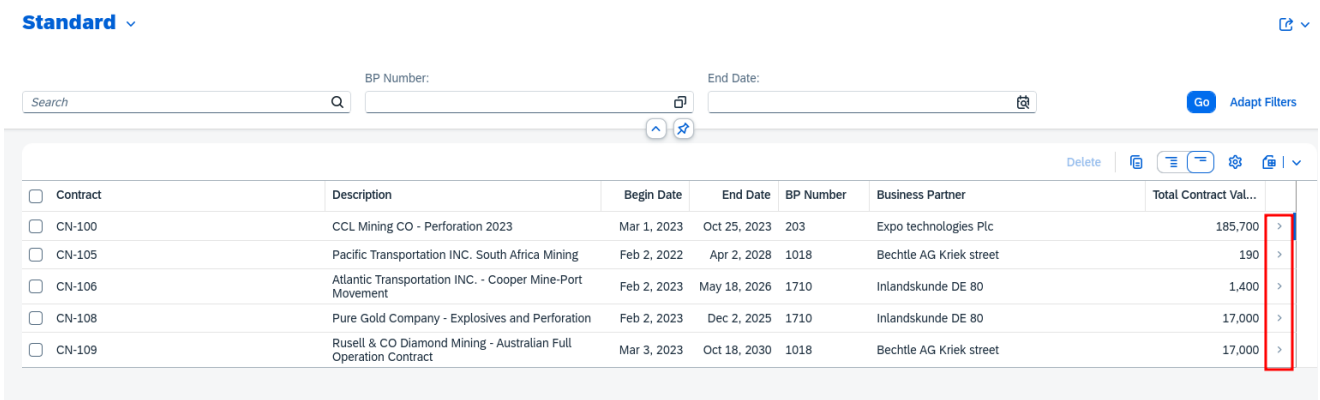


In label use **Contract Header**. After that there will be a **Form -> Fields** section.

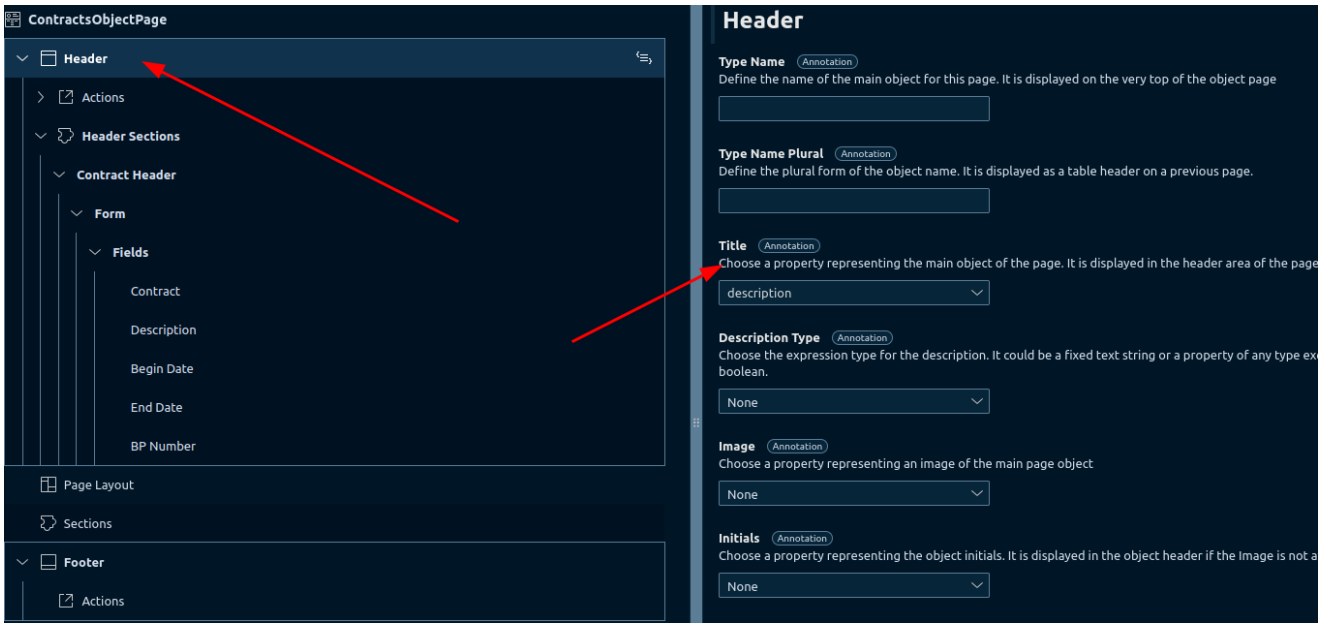
Add the fields required, in this example:



Just with this, if open the application and choose the right arrow on the contract line, will go to the contract detail page with the selected fields:



The header by default take the ID field as the main title. To change this parameters and add logos, images and other header details go to **Header --> Title** and choose a field to be used as a main title, in this example the main title will be the *description* field.



Now the creation process for a data point will be the same process. in the Header Sections add a Data Point Section and choose the *TotalMonthValue* field as the number to show. After

that, the header will looks like:

At this point, all the annotations that support this design could be created on **common.cds** file or in **annotations.cds** file, the editor will use any of both, but since this configurations are only for this application, the next code should be on **annotations.cds**:

> **Next annotations create the facets and the field groups that will be included on each facet. Also create the data point definition included into the header.**

```
//*Contract Details Header Facets

annotate DemoService.Contracts with @(
        UI.HeaderFacets : [
                {
                        $Type : 'UI.ReferenceFacet',
                        Label : 'Contract Header',
                        ID : 'ContractHeader',
                        Target : '@UI.FieldGroup#ContractHeader',
                },
                {
                        $Type : 'UI.ReferenceFacet',
                        ID : 'totalMonthValue',
                        Target : '@UI.DataPoint#totalMonthValue',
                },
        ],

        UI.FieldGroup #ContractHeader : {
                $Type : 'UI.FieldGroupType',
                Data : [
                        {
                                $Type : 'UI.DataField',
                                Value : ID,
                        },{
                                $Type : 'UI.DataField',
```

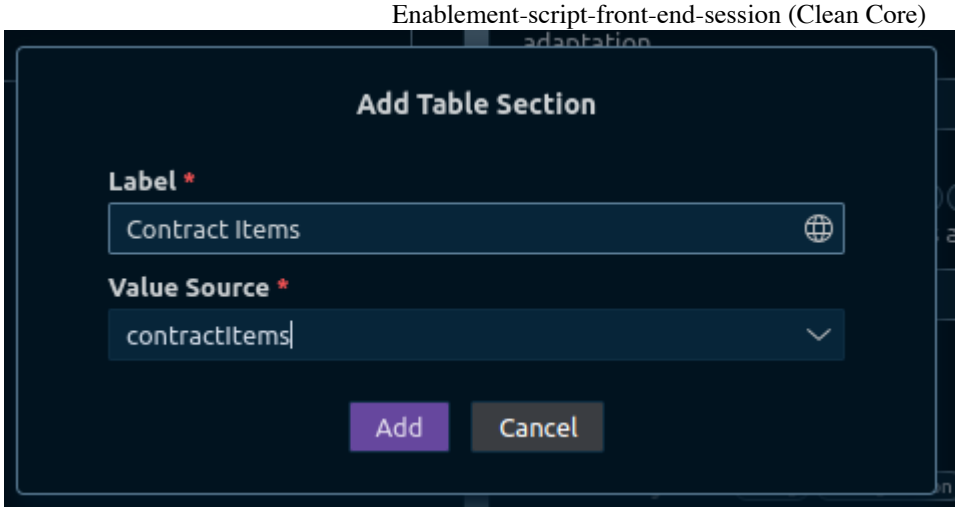
```
                                Value : description,
                        },{
                                $Type : 'UI.DataField',
                                Value : beginDate,
                        },{
                                $Type : 'UI.DataField',
                                Value : endDate,
                        },{
                                $Type : 'UI.DataField',
                                Value : businessPartner_BusinessPartner,
                        },],
        }
);

annotate DemoService.Contracts with @(
        UI.HeaderInfo : {
                Title : {
                        $Type : 'UI.DataField',
                        Value : description,
                },
                TypeName : '',
                TypeNamePlural : '',
        }
);

annotate DemoService.Contracts with @(
        UI.DataPoint #totalMonthValue : {
                $Type : 'UI.DataPointType',
                Value : totalMonthValue,
                Title : 'Total Contract Value',
        }
);
```

As the data point will use the contract value and it is important to show the data with currency, this annotation added to the common.cds file will add both fields in a cross application scenario.

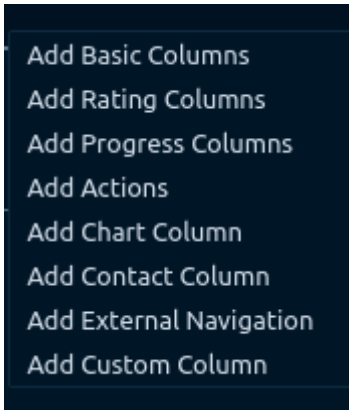
```
annotate DemoService.Contracts with {
        totalMonthValue @Measures.ISOCurrency : totalMonthCurrency
};
```

3.3.4.2 Contract Item Sections

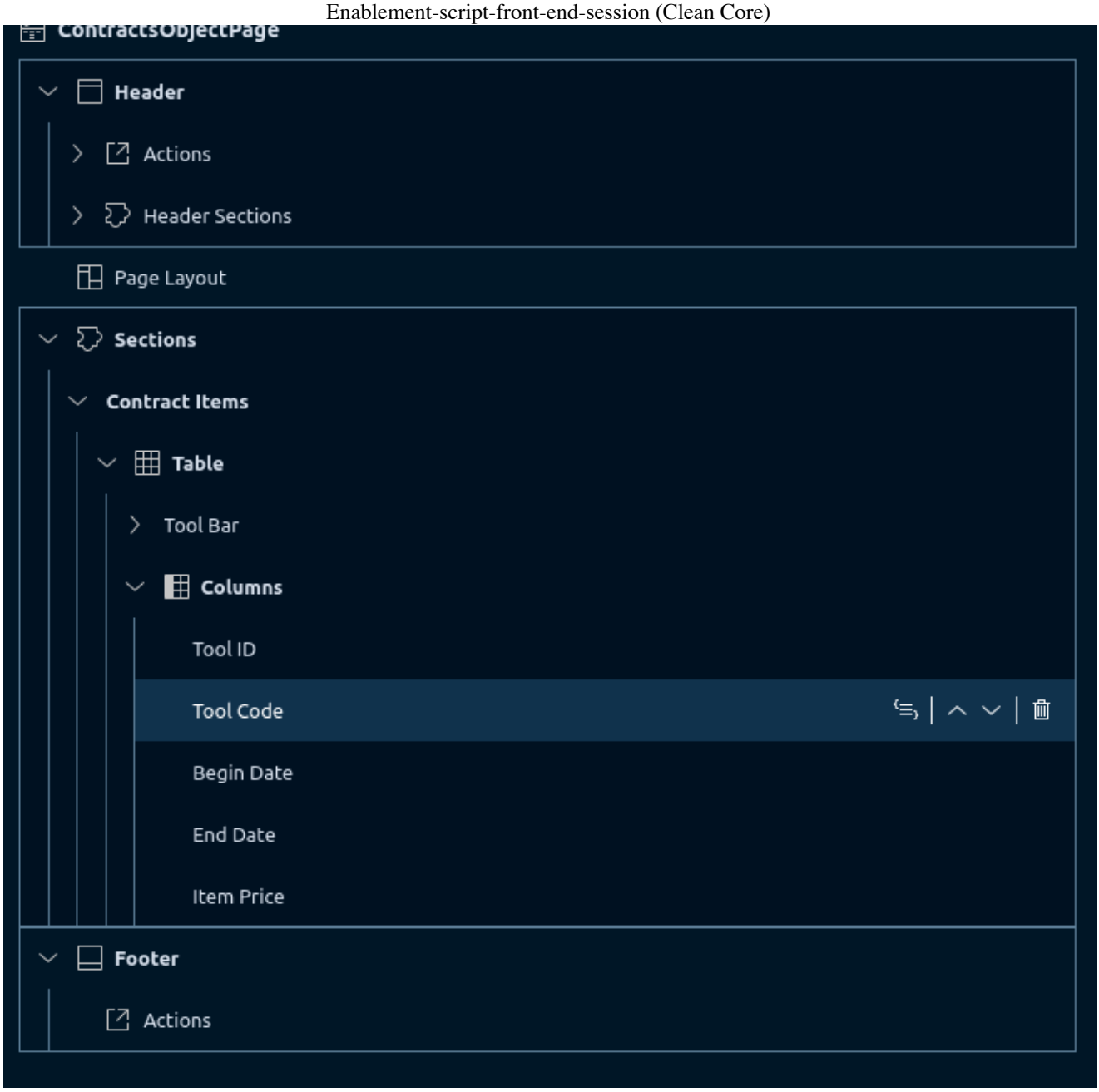
For the contract items list related to the contracts, go to the **Sections -> Add Table Section** with the values for the example:



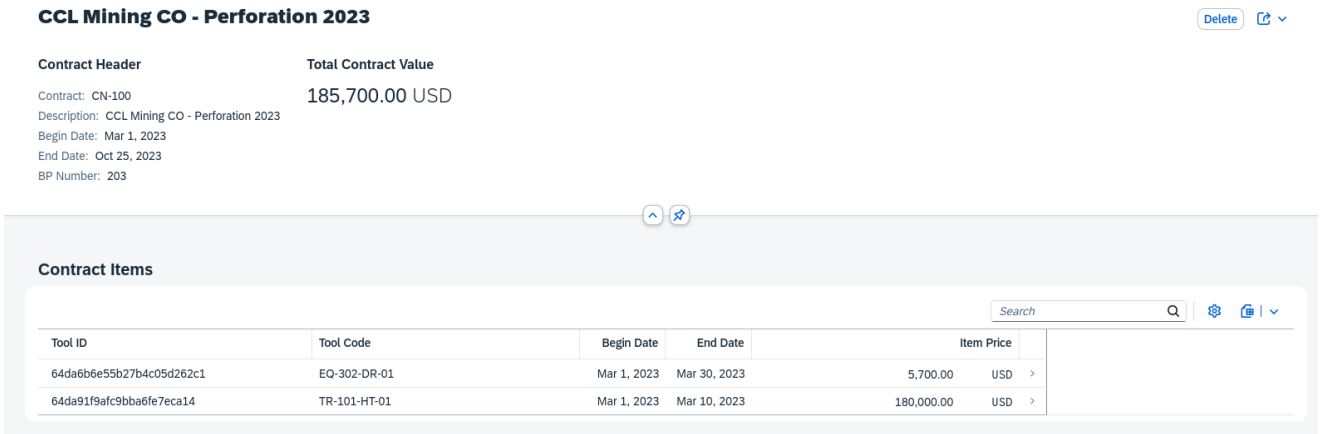
In the new Contract Items sections go to **Table -> Columns** and choose the table fields. As in the previous example, there will be the chance to use different column types according to needs, for this example only basic will be used.



With the next configuration:



The application now shows the item list table as part of the detail page:



The annotations that support the item list creation added to the **annotations.cds** file, are:

```
annotate DemoService.Contracts with @(
```

```
UI.Facets : [
        {
                $Type : 'UI.ReferenceFacet',
                Label : 'Contract Items',
                ID : 'ContractItems',
                Target : 'contractItems/@UI.LineItem#ContractItems',
        },
]);

annotate DemoService.ContractItems with @(

UI.LineItem #ContractItems : [
        {
                $Type : 'UI.DataField',
                Value : tool,
        },
        {
                $Type : 'UI.DataField',
                Value : toolName,
        },
        {
                $Type : 'UI.DataField',
                Value : beginDate,
        },
        {
                $Type : 'UI.DataField',
                Value : endDate,
        },
        {
                $Type : 'UI.DataField',
                Value : price,
        },
]);
```

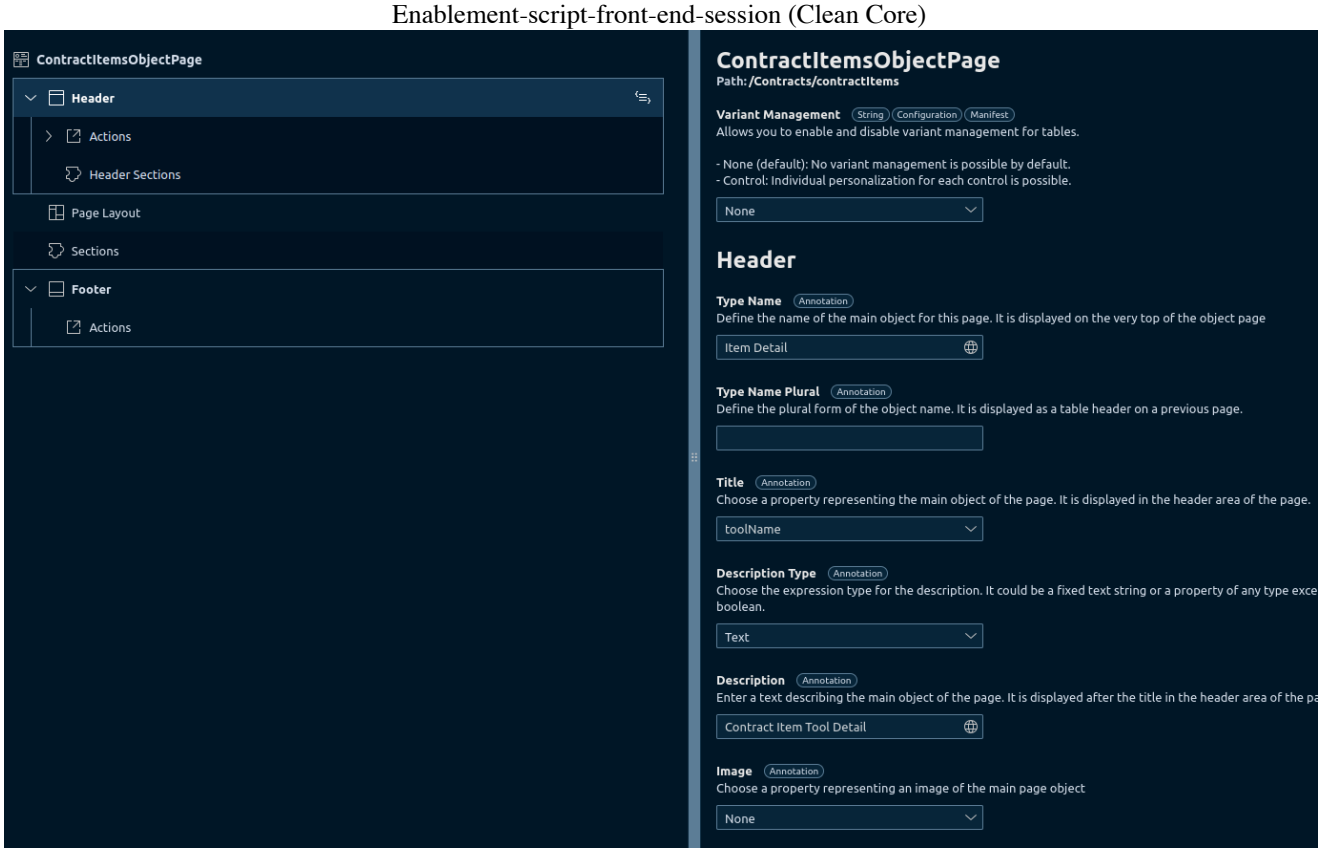
3.4.- Contract Item Detail Page

Now in the page map, define the content for the item detail page, the third and last. This will show the item information in case the individual treatment is needed.

The definition process is the same as previous views, considering this point can access to all previous entitites involved with the detail, that means we can consume Contracts and contractItems information.

> ⚠ **As tools are not involved as entity, we cannot browse the tool information in the same way as the rest of the entities. The consumption of a REST API in a elements UI will need a different deep development process.**

In header section we can choose Title, Description and other description fields:



As same the previous page we can choose the facets for the header section and the content for the central page Sections.

> **Note that this floor plan has different objects available than the previous one. In this case for the central section the table list is not available. For mor information visit [Fiori Elements Object Page Documentation]**

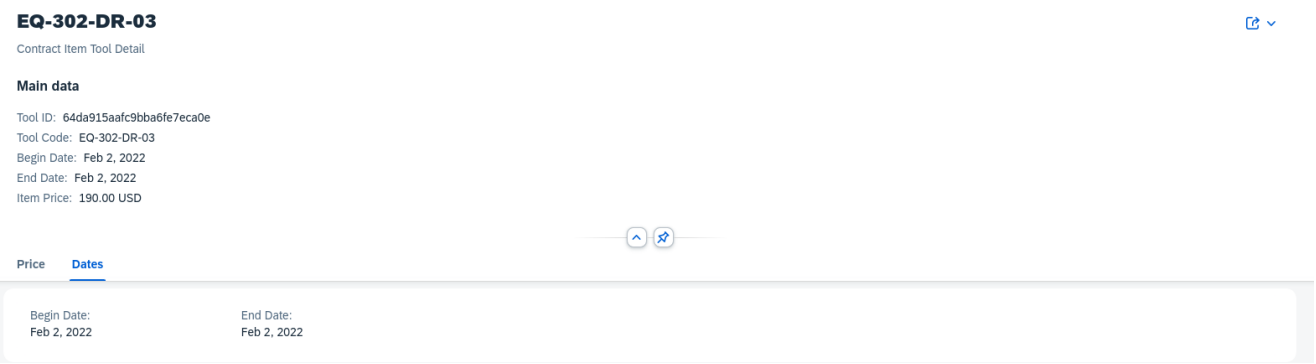
After a configuration with the editor, the generated annotations in **annotations.cds** file will be:

```
//* Contract Items Definition

annotate DemoService.ContractItems with @(

UI.HeaderFacets : [
        {
                $Type : 'UI.ReferenceFacet',
                Label : 'Main data',
                ID : 'Maindata',
                Target : '@UI.FieldGroup#Maindata',
        },
]);

annotate DemoService.ContractItems with @(
        UI.Facets : [
                {
                        $Type : 'UI.CollectionFacet',
```

```
                        Label : 'Price',
                        ID : 'Data',
                        Facets : [
                                {
                                        $Type : 'UI.ReferenceFacet',
                                        Label : 'Price',
                                        ID : 'Price',
                                        Target : '@UI.FieldGroup#Price',
                                },],
                        },
                        {
                                $Type : 'UI.ReferenceFacet',
                                Label : 'Dates',
                                ID : 'BusinessPartner1',
                                Target :
'@UI.FieldGroup#BusinessPartner1',
                        },]
);

annotate DemoService.ContractItems with @(
        UI.HeaderInfo : {
                TypeName : 'Item Detail',
                TypeNamePlural : '',
                Description : {
                        $Type : 'UI.DataField',
                        Value : 'Contract Item Tool Detail',
                },
                Title : {
                                $Type : 'UI.DataField',
                                Value : toolName,
                },
        }
);

annotate DemoService.ContractItems with @(
        UI.FieldGroup #Maindata : {
                $Type : 'UI.FieldGroupType',
                Data : [
                        {
                                $Type : 'UI.DataField',
                                Value : tool,
                        },{
                                $Type : 'UI.DataField',
                                Value : toolName,
                        },{
                                $Type : 'UI.DataField',
                                Value : beginDate,
                        },{
                                $Type : 'UI.DataField',
```

```
                        Value : endDate,
                    },{
                        $Type : 'UI.DataField',
                        Value : price,
                },],
        }
);

annotate DemoService.ContractItems with @(
        UI.Identification : [
        {
                $Type : 'UI.DataField',
                Value : tool,
        },{
                $Type : 'UI.DataField',
                Value : toolName,
        },]
);

annotate DemoService.ContractItems with @(
        UI.FieldGroup #Price : {
                $Type : 'UI.FieldGroupType',
                Data : [
                        {
                                $Type : 'UI.DataField',
                                Value : price,
                        },{
                                $Type : 'UI.DataField',
                                Value : priceCurrency,
                                Label : 'priceCurrency',
                        },
                ],
        }
);

annotate DemoService.ContractItems with @(
        UI.FieldGroup #BusinessPartner1 : {
                $Type : 'UI.FieldGroupType',
                Data : [
                        {
                                $Type : 'UI.DataField',
                                Value : beginDate,
                        },{
                                $Type : 'UI.DataField',
                                Value : endDate,
                        },
                ],
        }
);
```

The screen will looks like:



3.5.- Creation and draft

To add the save and edit capabilities is needed to activate the **@draft** annotation for the service, define the **read only** fields and the restrictions that can apply for the elements.

First, create the `capabilities.cds` file inside **cmanager** folder and add the semantic keys and draft permission for the service:

```
using DemoService as service from '../../srv/external-test';

annotate service.Contracts with @odata.draft.enabled;
annotate service.Contracts with @Common.SemanticKey: [ID];
annotate service.ContractItems with @Common.SemanticKey: [contract_ID];
```

Next, need to add the capabilities references to the `services.cds` file:

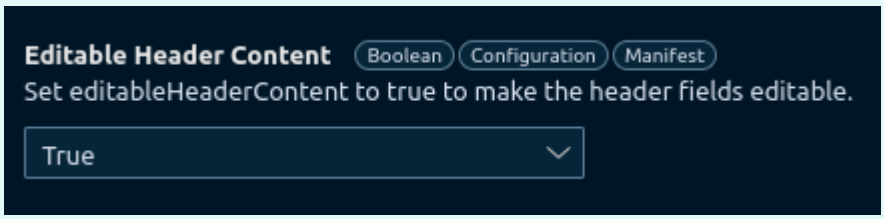
```
using from './cap_clean_core_demo.cmgr/annotations';
using from './cap_clean_core_demo.cmgr/capabilities';
using from './common';
```

Finally, to avoid creation errors, set *Read Only* fields in `annotations.cds` file :

```
// Read Only Fields for Creation

annotate service.Contracts with {
        totalMonthValue @readonly;
        bpName @readonly
}

annotate service.ContractItems with {
        price @readonly;
        toolName @readonly;
}
```

With this last change, all the draft and C.R.U.D capabilities will be enabled cross the interface.

By default all the header information edition is disabled in all pages, to change that, llok for the Editable Header Content and change to true.



In Fiori Elements, the contract item creation needs to be implemented in a different way, because of *draft* management structure, so in the external-test.cds file will be necessary to change value calculation process from *on* life cycle point to *after* like the snippet:

```
//* This method work with fiori elements!!!

srv.after('READ','ContractItems',async (ContractItems, req) => {
        for(let item of ContractItems){
                if((item.beginDate) && (item.endDate)){

                //* Days duration calculation
                let beginDate = new Date(`${ item.beginDate }`).getTime();
                let endDate = new Date(`${ item.endDate }`).getTime();
                let daysDiff = endDate - beginDate;let difference =
daysDiff/(1000*60*60*24);

                //*One day Contract Item
                if(difference == 0){difference = 1}


                //* Get tool information

                await toolsAPI.send({
                        query: `GET /api/tools/${item.tool}`,
                        headers: {
                                userName: "p2c-comm-user",
                                APIKey: "K306GLEZ1TPZZU4CIVGTQFQHXZ2920"
                        }
                })
                .then (res => {
                        item.toolName = res.tool.toolName;
                        item.price = res.tool.toolDailyPrice * difference;
                })
```

```
        }
}
})
```

4.- VueJs Intro

Explain VueJs capabilities, the different development models (Options and Composition API). Explain the *two ways* to develop a frontend using VueJs and how the first one can be easily used for a demo, pre sales process, showcase and to have their own application book with easy to rise a model.

4.1.- VueJs - The *fast* way

One of the easiest way to understand how vuejs works and interact with the CAP Backend is treating it like an html + js file importing all the functional libraries. This focus could work in a perfect way but, it is absolutely lack of professional maintenance and **it should be used only for demo and learning purposes.**

4.1.1.- Create `index.html`

First, create an empty `index.html` file on app/vuejs folder . *Don´t forget to create the folder first*
Show the basic html content with a *hello*:

```
<!DOCTYPE html>

<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Document</title>
</head>
<body>
        Hello World
</body>

</html>
```

4.1.2.- Add Bootstrap - VueJs - Axios as script call

Explain why each of the chosen technologies and what will be the use for each one:

Add in head:

```
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.
css" rel="stylesheet" integrity="sha384-
4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9"
crossorigin="anonymous">
        <title>Contract Management System</title>
</head>
```

Add at the end of the body section **inside body**:

```
<!-- Bootstrap JS -->

<!-- <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle
.min.js" integrity="sha384-
HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
crossorigin="anonymous"></script> -->

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.mi
n.js" integrity="sha384-
I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r"
crossorigin="anonymous"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.min.js
" integrity="sha384-
Rx+T1VzGupg4BHQYs2gCW9It+akI2MM/mndMCy36UVfodzcJcF0GGLxZIzObiEfa"
crossorigin="anonymous"></script>

<!-- VueJS -->

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>



<!-- Axios -->

<script
src="https://cdnjs.cloudflare.com/ajax/libs/axios/1.5.0/axios.min.js"
integrity="sha512-
aoTNnqZcT8B4AmeCFmiSnDlc4Nj/KPaZyB5G7JnOnUEkdNpCZs1LCankiYi01sLTyWy+m2P+W4
XM+BuQ3Q4/Dg==" crossorigin="anonymous" referrerpolicy="no-referrer">
</script>
```

Finally add after body section:

```
<script src="app.js"></script>
```

> **In this stage you can show the display changes in hello world because of the bootstrap inclusion.**

4.1.3.- Create `app.js` file.

In the same folder, next to the index.html, add the `app.js` file and define the main global variables.

```
//* Global variables //

const GET = (url) => axios.get('/demosrv'+url)
const POST = (cmd, data) => axios.post('/demosrv'+cmd,data)
```

> **The global variables are created to simplify the axios endpoint consumption making easier the endpoint setup change in case of need**

Next, add the empty create vuejs app definition. Explain the `mount(#app)` command at the end.

```
const contractsApp = Vue.createApp({

}).mount('#app')
```

4.1.3.- HTML Contract List

To start we need to replicate the contracts list and we will create a table with basic options to show data. Also we will need to define the `contracList[]` array to show how the answer of the API will be added to the tools and deployed with a v-for.

But first define `<body>` options:

```
<body class="container" style="margin-top: 70px;">
        <div id="app">
        </div>

Cerrar body despues de los scripts

</body>
```

Next, add the contracts table definition, for headers:

```
 <div class="container-lg mx-auto d-grid gap-3">
            <h1>Contract Management System</h1>
            <!-- <input type="text" placeholder="Search..."
@input="search"> -->

            <table id="contracts-table" class="table table-hover table-
bordered caption-top">
                <!-- <caption>Contracts List</caption> -->
                <thead class="text-center table-primary">
                    <th>Contract ID</th>
                    <th>Description</th>
                    <th>Begin Date</th>
                    <th>End Date</th>
                    <th>Total Value</th>
                    <th>Currency</th>
                    <th>BP Number</th>
                    <th>Business Partner</th>
                </thead>
                <tbody class="table-group-divider">
                    <tr class="text-justify" v-for="contract in
contractsList" v-bind:id="contract.ID" v-on:click="details">
                        <td>{{ contract.ID }}</td>
                        <td>{{ contract.description}}</td>
                        <td>{{ contract.beginDate}}</td>
                        <td>{{ contract.endDate}}</td>
                        <td class="text-end">{{ contract.totalMonthValue
}}</td>
                        <td class="text-center">{{
contract.totalMonthCurrency }}</td>
                        <td class="text-center">{{
contract.businessPartner_BusinessPartner}}</td>
                        <td>{{ contract.bpName}}</td>
                    </tr>
                </tbody>
            </table>
            <div class="col-2">
                <button class="btn btn-primary" @click="createContract">
Create Contract </button>
         </div>
 </div>
```

The vue options were removed and need to be added when show the data binding withe the array and the events use:

```
v-for="contract in contractsList" v-bind:id="contract.ID" v-
on:click="details"
```

4.1.4.- Create data binding in `app.js`

In the app.js file, inside the application definition, create the data segment with the arrays and the contracts get function:

```
data(){
        return {
                contractsList: [],
                contractItems: [],
                toolsList:[],
        }
},
methods: {

        //*Get Contracts List
        async getContracts () {
                const contracts = await GET('/Contracts')
                this.contractsList = contracts.data.value
        },
}
```

Explain how the data management work and remember to insert the code with v-for in the table row definition:

Show the first result.

4.1.5.- Create contract items component in `index.html`

Add the index component snippet in the index.html:

```
 <!--Contract Items Component -->

        <div class="container-lg mx-auto d-grid gap-3" v-if="itemsDisplay"
style="margin-top: 40px;">
            <table id="contracts-table" class="table table-hover table-
bordered caption-top">
                <caption>Contract Items for contract: <b>{{
contract.description }}</b></caption>
                <thead class="text-center table-info">
                    <th>Contract ID</th>
                    <th>Tool ID</th>
                    <th>Tool Description</th>
                    <th>Begin Date</th>
                    <th>End Date</th>
                    <th>Item Price</th>
                    <th>Currency</th>
```

```
                </thead>
                <tbody class="table-group-divider">
                    <tr class="text-justify" v-for="item in
contractItems">
                        <td>{{ item.contract_ID }}</td>
                        <td>{{ item.tool_ID }}</td>
                        <td>{{ item.toolName }}</td>
                        <td>{{ item.beginDate }}</td>
                        <td>{{ item.endDate }}</td>
                        <td class="text-end">{{ item.price }}</td>
                        <td class="text-center">{{ item.priceCurrency }}
</td>
                    </tr>
                </tbody>
            </table>
            <div class="col-2">
                <button class="btn btn-info" @click="createItem"> Create
Item </button>
            </div>
        </div>
```

4.1.6.- Create contracts component in `index.html`

```
 <!-- Create Contract Component -->
        <div class="col-4" style="margin-top: 40px;" v-
if="contractCreate">
            <form>
                <div class="mb-3">
                  <label for="contract_ID" class="form-label">Contract
ID</label>
                  <input type="type" class="form-control" v-
model="contract.ID" >
                </div>
                <div class="mb-3">
                  <label for="description" class="form-
label">Description</label>
                  <input type="type" class="form-control" v-
model="contract.description" >
                </div>
                <div class="mb-3">
                  <label for="beginDate" class="form-label">Begin
Date</label>
                  <input type="type" class="form-control" v-
model="contract.beginDate" >
                </div>
                <div class="mb-3">
                  <label for="Contract_ID" class="form-label">End
Date</label>
```

```
                  <input type="type" class="form-control" v-
model="contract.endDate" >
                </div>
                <div class="mb-3">
                  <label for="Contract_ID" class="form-
label">Currency</label>
                  <input type="type" class="form-control" v-
model="contract.totalMonthValue_currency_code" >
                </div>
                <div class="mb-3">
                  <label for="businessPartner" class="form-
label">BusinessPartner</label>
                  <input type="type" class="form-control" v-
model="contract.businessPartner_BusinessPartner" >
                </div>
                <button type="button" class="btn btn-primary"
@click="submitNewContract">Submit</button>
                <button type="button" class="btn btn-secondary"
@click="cancelContractCreation" style="margin-left: 20px;">Cancel</button>
              </form>
        </div>
```

4.1.7.- Create item component in `index.html`

```
 <!-- Create Item Component -->
        <div class="col-4" style="margin-top: 40px;" v-if="itemCreate">
            <form>
                <div class="mb-3">
                  <label for="contract_ID" class="form-label">Contract
ID</label>
                  <input type="type" class="form-control" v-
model="contractItem.contract_ID" >
                </div>
                <div class="mb-3">
                  <label for="beginDate" class="form-label">Begin
Date</label>
                  <input type="type" class="form-control" v-
model="contractItem.beginDate" >
                </div>
                <div class="mb-3">
                  <label for="endDate" class="form-label">End Date</label>
                  <input type="type" class="form-control" v-
model="contractItem.endDate" >
                </div>
                <div class="mb-3">
                  <label for="price_value" class="form-
label">Price</label>
                  <input type="type" class="form-control" v-
```

```
model="contractItem.price_value" disabled>
                </div>
                <div class="mb-3">
                  <label for="price_currency_code" class="form-
label">Currency</label>
                  <input type="type" class="form-control" v-
model="contractItem.price_currency_code" >
                </div>
                <div class="mb-3">
                  <label for="tool_ID" class="form-label">Tool ID</label>
                  <input type="type" class="form-control" v-
model="contractItem.tool_ID" >
                </div>
                <div class="mb-3">
                  <label for="toolName" class="form-label">Tool</label>
                  <input type="type" class="form-control" v-
model="contractItem.toolName" disabled>
                </div>
                <div class="mb-3">
                  <label for="toolSelect" class="form-label">Tool</label>
                  <select class="form-select" v-
model="contractItem.tool_ID">
                    <option disabled value="">Select tool</option>
                    <option v-for="tool in toolsList" :value="tool._id"  >
{{ tool.toolName }}</option>
                  </select>
                </div>
                <button type="button" class="btn btn-primary"
@click="submitNewItem">Submit</button>
                <button type="button" class="btn btn-secondary"
@click="cancelItemCreation" style="margin-left: 20px;">Cancel</button>
              </form>
        </div>

    </div>
```

4.1.8.- Replace data return in `app.js`

```
return {
        contract: {
                ID: '',
                description: '',
                beginDate: '',
                endDate: '',
                totalMonthValue: 0,
                totalMonthCurrency: '',
                businessPartner_BusinessPartner: '',
```

```
                bpName: ''
        },

        contractItem: {
                contract_ID: '',
                beginDate: '',
                endDate: '',
                price: 0,
                priceCurrency: '',
                tool_ID: '',
                toolName: ''
        },

contractsList: [],
contractItems: [],
toolsList:[],

itemsDisplay: false,
contractCreate: false,
itemCreate: false

}
```

4.1.9.- Add contracts methods in `app.js`

```
async details(event){
            this.itemsDisplay = true
            const { id } = event.currentTarget
            const contractObject = await GET(`/Contracts/${ id }`)
            this.contract = contractObject.data
            const items = await GET('/ContractItems'+`?$filter=
(contract_ID eq '${ id }')`)
            this.contractItems = items.data.value
        },
        createContract(){
            this.contractCreate = !this.contractCreate
            this.clearContractObject()
            this.itemsDisplay = false
        },
        cancelContractCreation(){
            this.contract = ''
            this.contractCreate = false
        },
        clearContractObject(){
            this.contract = {
                ID: '',
                description: '',
                beginDate: '',
```

```
                endDate: '',
                totalMonthValue_value: 0,
                totalMonthValue_currency_code: '',
                businessPartner_BusinessPartner: '',
                bpName: ''
            }
        },
        async submitNewContract(){

            try {
                await POST('/Contracts', this.contract)
                .then(resp => {
                    console.log(resp.data);
                    alert('Contract created sucessfully. ID: ' +
resp.data.ID)
                    this.getContracts()
                    this.clearContractObject()
                    this.contractCreate = false
                })

            } catch (error) {
                console.log(error);
            }
        },
```

4.1.10.- Add contracts items methods in `app.js`

```
 //* Contract Items

        createItem(){
            this.itemCreate = !this.contractCreate
            // this.clearContractObject()
            // this.itemsDisplay = false
        },
        cancelItemCreation(){
            this.clearItemtObject()
            this.itemCreate = false
        },
        clearItemtObject(){
            this.contractItem = {
                contractItem: {
                    contract_ID: '',
                    beginDate: '',
                    endDate: '',
                    price_value: 0,
                    price_currency_code: '',
                    tool_ID: '',
```

```
                    toolName: ''
                },
            }
        },
        async submitNewItem(){

            try {
                await POST('/ContractItems', this.contractItem)
                .then(resp => {
                    console.log(resp.data);
                    alert('Item added sucessfully. ID: ' +
resp.data.contract_ID)
                    this.getContracts()
                    this.clearContractObject()
                    this.clearItemtObject()
                    this.contractCreate = false
                    this.itemCreate = false
                })

            } catch (error) {
                console.log(error);
            }
        },

        //*Tools
        async getTools(){
            try {
               const tools =  await GET('/getTools()')
               this.toolsList = tools.data.toolsList

            } catch (error) {
                console.log(error);
            }
        }
    }
```

Finally add the initial call function for tools:

```
contractsApp.getTools();
```