Enablement-script-backend-session (Build Code)

```
Title:    CAP Development Enablement - Backend Session
Author:   Rodrigo Riveros A.
Date:     Feb 7th, 2024
Comment:  First 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.
This example can be created in BTP Build Code (former BAS) or in Visual
Studio Code, for local development as well.
```

1.- Goals and Objectives

Show CAP functionalities and capabilities that leverage programming options with aided development process offered by SAP Back-end system.
At the end of the second enablement, the expectations are to have a set of tools to create different functionalities according the need (presales, demo, sales or implementation) and options to choose the proper UI architecture according to the customer request.
Also, one of the main concepts that we will show and how to achieve it is the integration to SAP and No-SAP systems on the same back-end to support business process.

2.- Build Code Project

To create the project folder with the basic structure, open your terminal / power shell and go to the folder where you want to create the project. Then execute `cds init <folder-name>` for example **cds init myApplication**

> **Check the folder where your Power Shell open, sometimes by default the windows destination is C:\Windows\system32 and you probably don´t want to create your project in that folder**

2.1.- Create Contracts App Schema

Inside VS code, go to **db** folder and create the file `schema.cds` with the next content:

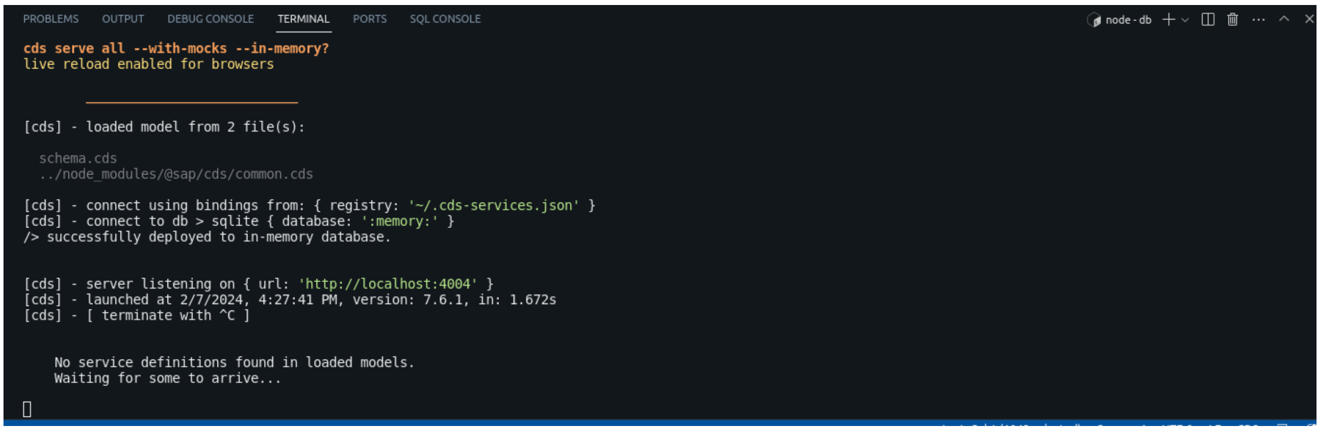
```
namespace p2c.cap.contracts;
using { cuid , managed } from '@sap/cds/common';
```

```
entity Contracts : managed {
        key ID : String;
        description : String;
        beginDate : Date;
        endDate : Date;
        totalMonthValue : Integer;
        totalMonthCurrency : String;
        //businessPartner: Association to BusinessPartners;
        contractItems : Composition of many ContractItems on
contractItems.contract = $self;
        bpName:String;
}

@cds.autoexpose
        entity ContractItems : cuid {
        key contract: Association to one Contracts;
        beginDate: Date;
        endDate: Date;
        price: Integer;
        priceCurrency: String;
        tool: String;
        toolName: String
}
```

2.2.- Check console messages

Execute `cds watch` in the console of the folder and check the messages from the terminal, you can use any terminal or the VSCode embedded one.

Remember always run *cds watch* in the root folder of the project. The command will not work properly if you are inside some sub-folder



```
cds serve all --with-mocks --in-memory?
live reload enabled for browsers

        ___________________________

[cds] - loaded model from 2 file(s):

  schema.cds
  ../node_modules/@sap/cds/common.cds

[cds] - connect using bindings from: { registry: '~/.cds-services.json' }
[cds] - connect to db > sqlite { database: ':memory:' }
/> successfully deployed to in-memory database.


[cds] - server listening on { url: 'http://localhost:4004' }
[cds] - launched at 2/7/2024, 4:27:41 PM, version: 7.6.1, in: 1.672s
[cds] - [ terminate with ^C ]


    No service definitions found in loaded models.
    Waiting for some to arrive...

▯
```

2.3.- Basic services creation

Create basic services for validation of the CAP base capabilities. Both on the `srv` folder in different files.

- Create the initial services for **administration** and **contract management**

admin-service.cds

```
using { p2c.cap.contracts as my } from '../db/schema';

service AdminService @(path: '/admin'){
        entity Contracts as projection on my.Contracts;
        entity ContractItems as projection on my.ContractItems;
}
```

contracts-service.cds

This service is created to show the difference about how CAP let us to show information from the same database with different scope just changing the selection of the fields.

```
using { p2c.cap.contracts as my} from '../db/schema';

service  ContractsService @(path:'/contracts'){

    entity Contracts as select from my.Contracts {
        ID, description
    };

    @readonly entity ContractItems as projection on my.ContractItems;


}
```

2.4.- Data mockup for model

The next step is to add the mock data for the services. For now the in memory database will be used to check the endpoints functionality

- The data files MUST be added into **db/data** folder with the nex specific format
- Mock up data filename rules `namespace-entityName.csv`

⚠ **This first version of the data files will be completed in the next steps, for example with the Business Partner Information that will come from S4HANA integration.**

2.4.1.- Contracts Mock Up Data

The content for **db/data/p2c.cap.contracts-Contracts.csv** file will be:

```
ID,description,beginDate,endDate,totalMonthValue,totalMonthCurrency
CN-100,CCL Mining CO - Perforation 2023,2023-03-01,2023-10-25,0,USD
CN-105,Pacific Transportation INC. South Africa Mining,2022-02-02,2028-04-
02,0,USD
CN-106,Atlantic Transportation INC. - Cooper Mine-Port Movement,2023-02-
02,2026-05-18,0,USD
CN-108,Pure Gold Company - Explosives and Perforation, 2023-02-02,2025-12-
02,0,USD
CN-109,Rusell & CO Diamond Mining - Australian Full Operation
Contract,2023-03-03,2030-10-18,0,USD
```

2.4.2.- Contract Items Mock Up Data

For the file **db/data/p2c.cap.contracts-ContractItems.csv**, we will include the tool reference, but not the toolName field will be added later as a result of direct external query.

Since tools will not be an entity in the schema, we only register the id as tool instead of tool_ID. The last one will be the CAP entity model assignment.

```
ID,contract_ID,beginDate,endDate,price,priceCurrency,tool,toolName

02d29d9d-3538-4140-8381-67ee542928db,CN-100,2023-03-01,2023-03-
30,5700,USD,65d608bafc04cd3d303714a1,EQ-302-DR-01
0b9e270f-0d11-4dd1-bb52-976812bcd32f,CN-100,2023-03-01,2023-03-
10,180000,USD,65d608bafc04cd3d30371375,TR-101-HT-01
63f57654-ee43-4f05-8fa2-7fd3e240517f,CN-105,2022-02-02,2022-02-
02,190,USD,65d608bafc04cd3d303714a3,EQ-302-DR-03
a324e459-a773-4616-8f48-dc60a3ab890f,CN-106,2023-02-02,2023-02-
02,26019,USD,65d608bafc04cd3d30371388,TR-101-HT-20
ac184d7a-b485-493e-be77-9d34c2d76407,CN-108,2023-02-02,2023-02-
02,17000,USD,65d608bafc04cd3d30371377,TR-101-HT-03
cef9a93e-2f97-47ec-8256-7ae3ef7b7704,CN-109,2023-03-03,2023-03-
03,17000,USD,65d608bafc04cd3d30371378,TR-101-HT-04
```

Note the contract value, even in the file is $0 since the value will be calculated in runtime during the contract list request

2.4.1.- Export to local database (Optional)

If you want to work with local database instead of in-memory, you can use the next instructions

~~~
```shell
npm add sqlite3 -D
cds deploy --to sqlite:my.sqlite
```
~~~

The command will generate the database file and this modification in `package.json` file:

~~~
```json5
"db": {
        "kind": "sqlite",
        "credentials": {
          "database": "my.sqlite"
        }
      }
    }
```
~~~

This work model will allow you to understand the join command sintaxis and the name of the joining fields, but it will force to re-create the local database each time yo make some model

2.5 Add business logic to a basic services

This stage is to explain about the cds capabilities to affect a service in the lifecycle management. The basic example will be the "on the fly" calculation of the total contract price and the item calculation according the days and the daily cost for the tool.

> **The "implementation" for a service, means a set of instructions that will affect the service life cycle in different stages, the name convention indicates that the file with the implementation has to be the same name of the service file but with js or ts depending the language used to implement and must be on the same folder. Also the `@impl` annotation can be used to define a different destination for the implementation file if needed..**

The basic examples for implementation will be:

- Calculation of the total contract price **on runtime**
- Calculation of the item price according to days and selected tools
- Item value calculation considering duration of the item and tool daily value

Additionally, the logic generation will be the explanation for the external API consumption considering the use of [functions and actions](Providing Services | CAPire) in the implementation of the service.

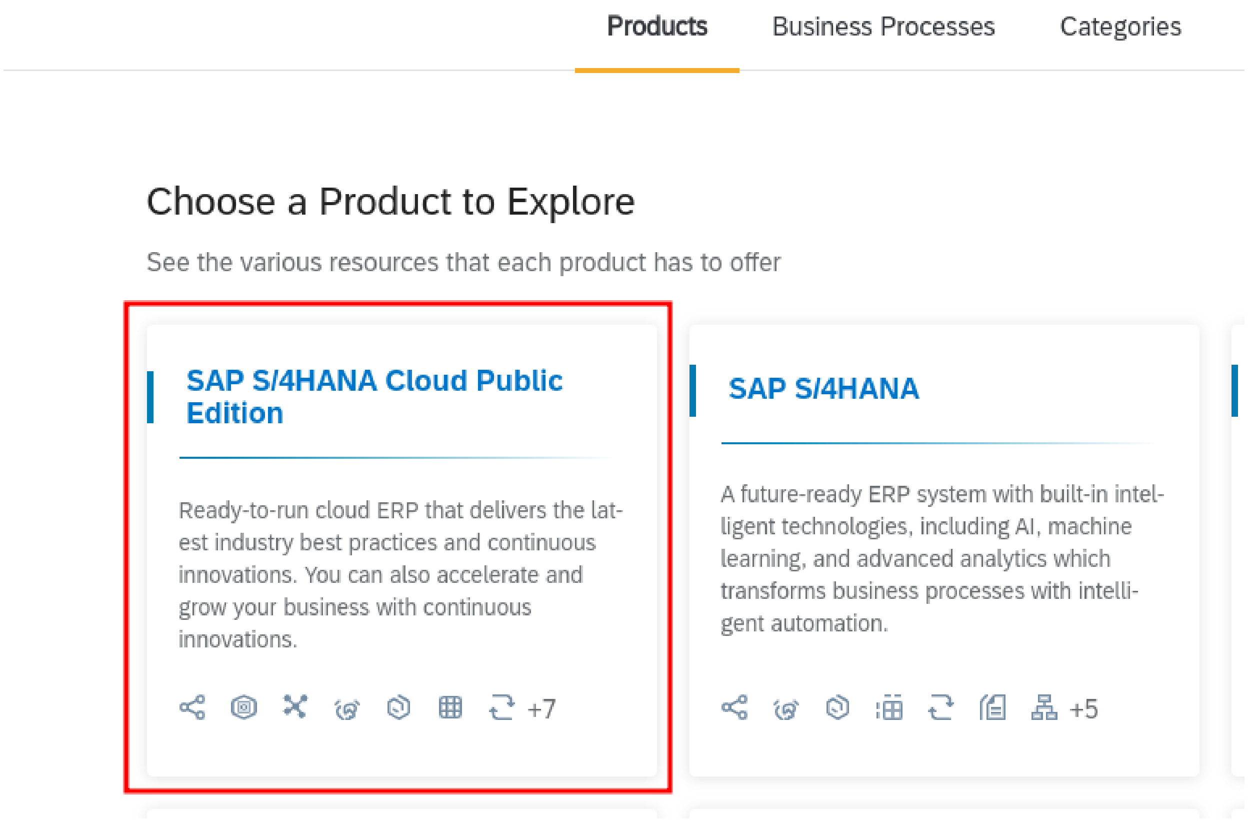
2.5.1.- Create external-test.cds service

```
using { p2c.cap.contracts as my } from '../db/schema';


service DemoService @(path: '/demosrv'){

    //**Business Partner integration from S4HANA */

    @readonly
    // entity BusinessPartners as projection on my.BusinessPartners;
    entity Contracts as projection on my.Contracts;
    entity ContractItems as projection on my.ContractItems;
}
```

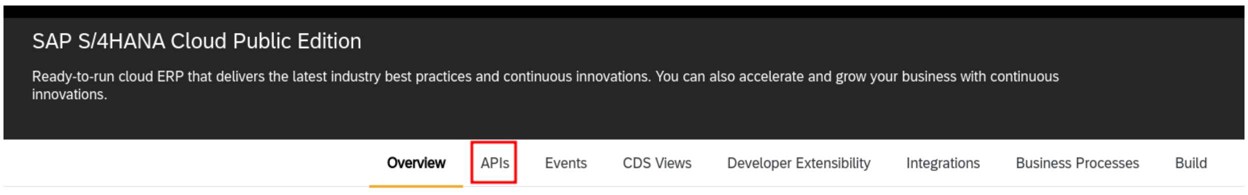
2.5.2.- Add S4HANA external source

First the S4HANA data source will be added to the back end. Go to SAP Business Accelerator HUB and next:

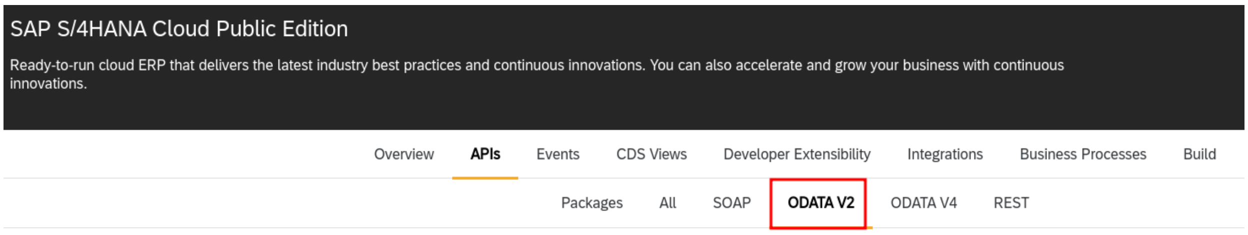
- Get into S4HANA Public


Products
Business Processes
Categories
Choose a Product to Explore
See the various resources that each product has to offer
SAP S/4HANA Cloud Public Edition
Ready-to-run cloud ERP that delivers the latest industry best practices and continuous innovations. You can also accelerate and grow your business with continuous innovations.
+7
SAP S/4HANA
A future-ready ERP system with built-in intelligent technologies, including AI, machine learning, and advanced analytics which transforms business processes with intelligent automation.
+5

- Go to API menu:

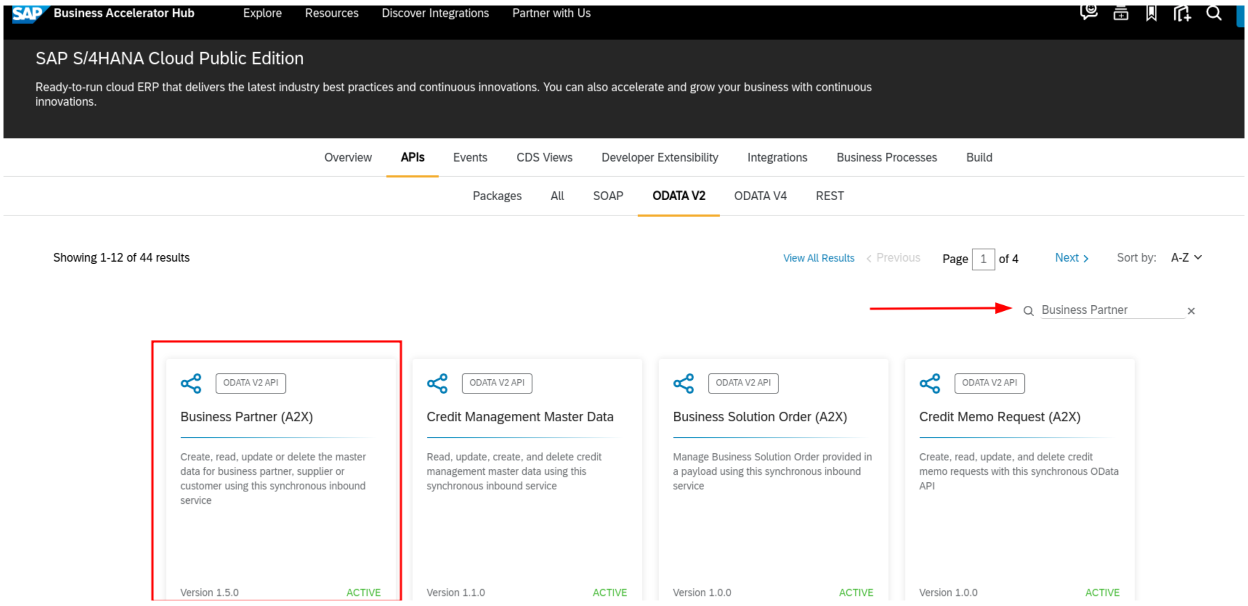

SAP S/4HANA Cloud Public Edition
Ready-to-run cloud ERP that delivers the latest industry best practices and continuous innovations. You can also accelerate and grow your business with continuous innovations.
Overview
APIs
Events
CDS Views
Developer Extensibility
Integrations
Business Processes
Build

- Then get into **OData v2** section:


SAP S/4HANA Cloud Public Edition

Ready-to-run cloud ERP that delivers the latest industry best practices and continuous innovations. You can also accelerate and grow your business with continuous innovations.

- Use the search box with the concept *Business Partner* and then choose the **Business Partner (A2X) API**.**:


SAP S/4HANA Cloud Public Edition

Ready-to-run cloud ERP that delivers the latest industry best practices and continuous innovations. You can also accelerate and grow your business with continuous innovations.

Showing 1-12 of 44 results

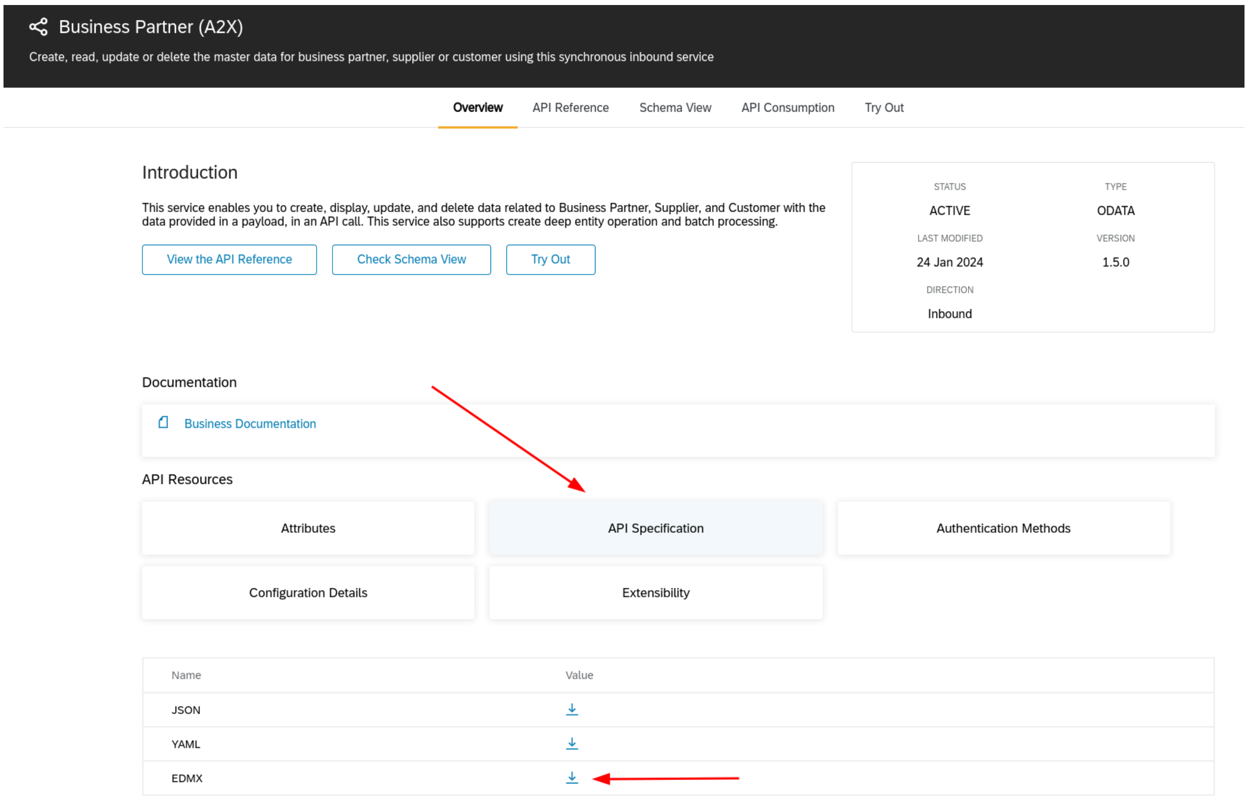
Business Partner (A2X) — Create, read, update or delete the master data for business partner, supplier or customer using this synchronous inbound service — Version 1.5.0 — ACTIVE

Credit Management Master Data — Read, update, create, and delete credit management master data using this synchronous inbound service — Version 1.1.0 — ACTIVE

Business Solution Order (A2X) — Manage Business Solution Order provided in a payload using this synchronous inbound service — Version 1.0.0 — ACTIVE

Credit Memo Request (A2X) — Create, read, update, and delete credit memo requests with this synchronous OData API — Version 1.0.0 — ACTIVE

- Then go to the **API Specification** section and download the EDMX File into the root of the project folder:


Business Partner (A2X)

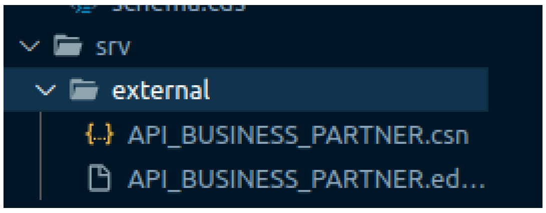
Create, read, update or delete the master data for business partner, supplier or customer using this synchronous inbound service

Introduction

This service enables you to create, display, update, and delete data related to Business Partner, Supplier, and Customer with the data provided in a payload, in an API call. This service also supports create deep entity operation and batch processing.

STATUS	ACTIVE
TYPE	ODATA
LAST MODIFIED	24 Jan 2024
VERSION	1.5.0
DIRECTION	Inbound

Name	Value
JSON	⤓
YAML	⤓
EDMX	⤓

- If everything goes well, automatically a **srv/external** folder will be created in the project tree: 

srv
external
API_BUSINESS_PARTNER.csn
API_BUSINESS_PARTNER.ed...

✓ **in almost all cases, when the `.edmx` file is added to the application tree, the `external` folder will be created into the srv folder. If that not happen, the manual import will be needed:**

```
cds import ~/Downloads/API_BUSINESS_PARTNER.edmx
```

Add the first stage of cds require on package.json, must be automatic, just check if the code is already created.

```
"API_BUSINESS_PARTNER": {
        "kind": "odata-v2",
        "model": "srv/external/API_BUSINESS_PARTNER",
}
```

⚠ **create `data` folder inside `srv/external` and put the file `API_BUSINESS_PARTNER-A_BusinessPartner.csv`, this will make possible to use the Business Partner mockup data when none environment is selected, but need a couple of next steps**

```
BusinessPartner;BusinessPartnerFullName;BusinessPartnerIsBlocked
1000038;Williams Electric Drives;false
1000040;Smith Batteries Ltd;false
1000042;Johnson Automotive Supplies;true
```

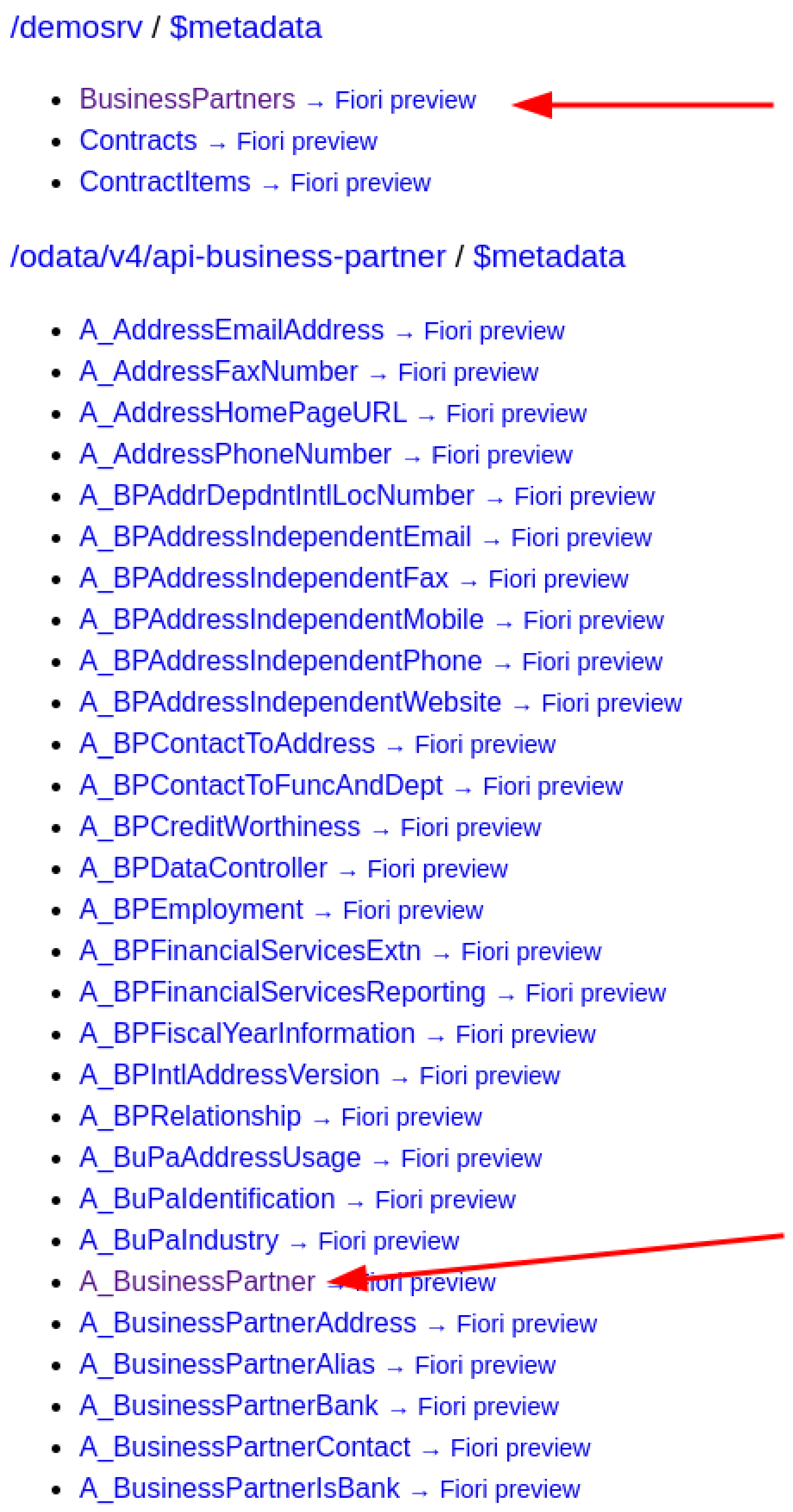
Show that the running app, it will not be using the mock data yet.

2.5.3.- Add the business partner entity to schema.cds

```
using { API_BUSINESS_PARTNER as bpar } from
'../srv/external/API_BUSINESS_PARTNER.csn';

entity BusinessPartners as projection on bpar.A_BusinessPartner {
    key BusinessPartner,
    BusinessPartnerFullName,
    BusinessPartnerIsBlocked
}
```

> ⚠ **UNCOMMENT the line in `external-test.cds` with the business partner entity. Don't forget the @readonly annotation**

Check in `https://localhost:4004` the new content. One change will be the complete api-business-partner definition and the other change is the **/demosrv** service, now has a BusinessPartner entity, only considering the data defined in **schema.cds** file.



/demosrv / $metadata

- BusinessPartners → Fiori preview
- Contracts → Fiori preview
- ContractItems → Fiori preview

/odata/v4/api-business-partner / $metadata

- A_AddressEmailAddress → Fiori preview
- A_AddressFaxNumber → Fiori preview
- A_AddressHomePageURL → Fiori preview
- A_AddressPhoneNumber → Fiori preview
- A_BPAddrDepdntIntlLocNumber → Fiori preview
- A_BPAddressIndependentEmail → Fiori preview
- A_BPAddressIndependentFax → Fiori preview
- A_BPAddressIndependentMobile → Fiori preview
- A_BPAddressIndependentPhone → Fiori preview
- A_BPAddressIndependentWebsite → Fiori preview
- A_BPContactToAddress → Fiori preview
- A_BPContactToFuncAndDept → Fiori preview
- A_BPCreditWorthiness → Fiori preview
- A_BPDataController → Fiori preview
- A_BPEmployment → Fiori preview
- A_BPFinancialServicesExtn → Fiori preview
- A_BPFinancialServicesReporting → Fiori preview
- A_BPFiscalYearInformation → Fiori preview
- A_BPIntlAddressVersion → Fiori preview
- A_BPRelationship → Fiori preview
- A_BuPaAddressUsage → Fiori preview
- A_BuPaIdentification → Fiori preview
- A_BuPaIndustry → Fiori preview
- A_BusinessPartner → Fiori preview
- A_BusinessPartnerAddress → Fiori preview
- A_BusinessPartnerAlias → Fiori preview
- A_BusinessPartnerBank → Fiori preview
- A_BusinessPartnerContact → Fiori preview
- A_BusinessPartnerIsBank → Fiori preview

2.5.4.- Install HTTP query packages - Very IMPORTANT!!

The consumption of external API's need the installation of two specific packages, so add them with the next command:

```
npm install @sap-cloud-sdk/http-client @sap-cloud-sdk/util
```

This packages should work in a local environment so probably you will not need to install with -g option, but if is needed, remember to use sudo command or administrator privileges depending the case

2.5.5.- Add the S4HANA sandbox environment to `package.json`

Next snippet will add the S4HANA Sandbox as a data source for our back-end.

Remember this setup is only for local sandbox development. For the productive use, the access to destination need to be set up directly on BTP. Will be done un further steps.

```
"API_BUSINESS_PARTNER": {
"kind": "odata-v2",
"model": "srv/external/API_BUSINESS_PARTNER",
"[sandbox]": {
        "credentials": {
                "url":
"https://sandbox.api.sap.com/s4hanacloud/sap/opu/odata/sap/API_BUSINESS_PA
RTNER"
}
```

2.5.6.- Create the `external-test.js` file as an implementation for `external-test.cds`,

Add the external entities definition and the default Business Partners call:

It is mandatory to have the bupa connection to show the BP consumption

```
const cds = require('@sap/cds')

module.exports = async (srv) => {

        //* Local Service Entities
        const {Contracts, BusinessPartners, ContractItems } =
srv.entities;

        //* Tools API connection
        // const toolsAPI = await cds.connect.to("toolsManager")

        //*Business Partner Connect
        const bupa = await cds.connect.to('API_BUSINESS_PARTNER')
        const {A_BusinessPartner} = bupa.entities;
        //* Entity BusinessPartners READ -- oData
        srv.on('READ', 'BusinessPartners', async (req) => {
                return await bupa.transaction(req).send({
                        query: req.query,
                        headers: {
                                apikey: "lSnMaBSXTGh7YX1aLfAyN08AdDkVRsfz"
                        }
                })
        })
}
```

⚠ When it is needed to setup a Basic Authentication auth request in the same scenario, will be necessary to setup in a different way. In this example using a communication user, the snippet will be:

```
//* Entity BusinessPartners READ -- oData - PES S4HANA Version - Basic
Auth Credentials Setup

srv.on('READ', 'BusinessPartners', async (req) => {

        const username = 'MIG_BTP_RAP_USER000';
        const password = 'Miguelito@1234567890';

        const credentials =
Buffer.from(`${username}:${password}`).toString('base64');
        const authHeader = `Basic ${credentials}`;

        return await bupa.transaction(req).send({
                query: req.query,
                headers: {
                        'Authorization': authHeader,
                }
        })
})
```

Restart the system with new profile:

```
cds watch --profile sandbox
```

Test the system and check the differences when using `--profile sandbox` about the S4HANA data and API definition

2.5.7- Add Tools External System

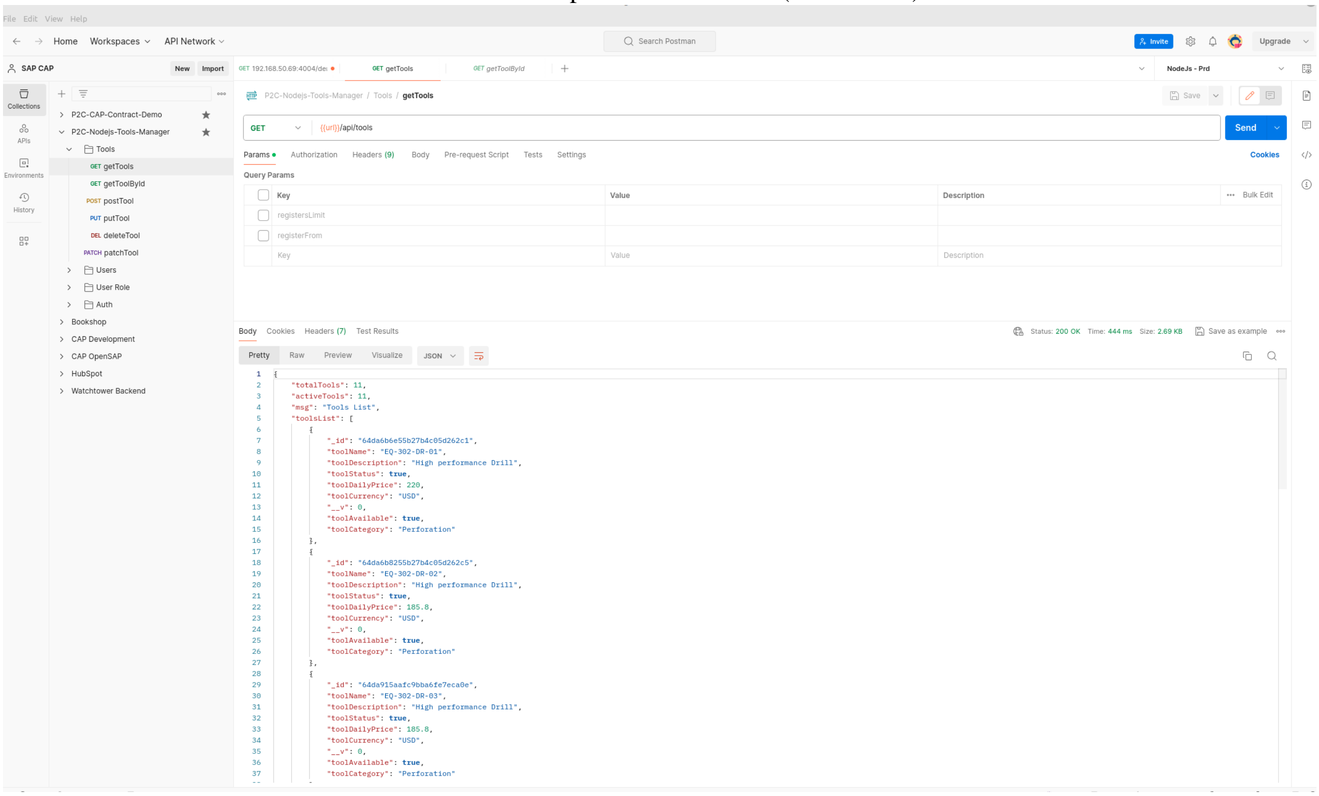
Now it is time to add the connection credentials for Tools Management System.
In this case we will not have the system running on local, so the "localhost:8080" credentials will not apply, but it is a good advise to show that if you are working with some other local installation during your development process, you can integrate that with the same definition.

Add the following code before API_BUSINESS_PARTNER definition in `package.json` inside `cds->requires` section. Explain the production environment definition

```
"toolsManager": {
        "kind": "rest",
        "credentials": {
                "url": "http://localhost:8080",
                "requestTimeout": 3000
        },

"[sandbox]": {
        "credentials": {
                "url": "https://p2c-poc-rest-srv-ts-
production.up.railway.app/"
        }
},

"[production]": {
        "credentials": {
                "destination": "P2C-ToolManager"
        }
  }
},
```

When we create external systems in `package.json` file, there is the possibility to add the login credentials like API, userName and so on directly on the same file, but as a practice we will add that information on every request we made, so if it is needed we can use different credentials for different request.

Now, considering we cannot implement this as entities, we will need to consume external sources as implementation functions. First we will check on REST structure in postman, like:



GET {{url}}/api/tools

Status: 200 OK Time: 444 ms Size: 2.69 KB

```json
{
    "totalTools": 11,
    "activeTools": 11,
    "msg": "Tools List",
    "toolsList": [
        {
            "_id": "64da6b6e55b27b4c05d262c1",
            "toolName": "EQ-302-DR-01",
            "toolDescription": "High performance Drill",
            "toolStatus": true,
            "toolDailyPrice": 220,
            "toolCurrency": "USD",
            "__v": 0,
            "toolAvailable": true,
            "toolCategory": "Perforation"
        },
        {
            "_id": "64da6b8255b27b4c05d262c5",
            "toolName": "EQ-302-DR-02",
            "toolDescription": "High performance Drill",
            "toolStatus": true,
            "toolDailyPrice": 185.8,
            "toolCurrency": "USD",
            "__v": 0,
            "toolAvailable": true,
            "toolCategory": "Perforation"
        },
        {
            "_id": "64da915aafc9bba6fe7eca0e",
            "toolName": "EQ-302-DR-03",
            "toolDescription": "High performance Drill",
            "toolStatus": true,
            "toolDailyPrice": 185.8,
            "toolCurrency": "USD",
            "__v": 0,
            "toolAvailable": true,
            "toolCategory": "Perforation"
```

With the structure of the object, now we can use for cds definition in `external-test.cds`

Add the next functions to the external-test.cds

```
//* Non SAP Source Access Functions

    function getTools() returns {
       totalTools: Integer;
       activeTools: Integer;
       msg: String;

       toolsList: array of {
           _id: String;
           toolName: String;
           toolDescription: String;
           toolStatus: Boolean;
           toolDailyPrice: Integer;
           toolCurrency: String;
           //__v: Integer;
           toolAvailable: Boolean;
           toolCategory: String
       }
   };

   function getToolById(id: String) returns {
       msg: String;
       tool: {
           _id: String;
           toolName: String;
```

```
            toolDescription: String;
            toolStatus: Boolean;
            toolDailyPrice: Integer;
            toolCurrency: String;
            //__v: Integer;
            toolAvailable: Boolean;
            toolCategory: String
        }
    };

    function getToolsByDescription(description: String) returns {
        msg: String;
        availableToolsList: array of {
            _id: String;
            toolName: String;
            toolDescription: String;
            toolStatus: Boolean;
            toolDailyPrice: Integer;
            toolCurrency: String;
            //__v: Integer;
            toolAvailable: Boolean;
            toolCategory: String
        }

    };

   function changeToolStatus(id: String) returns {
        msg: String;
   };
```

Add functions definition to `external-test.js`:

```
  /**
  //* REST API Function Calls for Tools
  */

      //**Function Read Tools -- NO oData

    srv.on('getTools',async (req) => {

        const toolsAPI = await cds.connect.to("toolsManager")
        return await toolsAPI.send({query: 'GET /api/tools', headers:{
            userName: "p2c-comm-user",
            APIKey: "K306GLEZ1TPZZU4CIVGTQFQHXZ2920"
        }})
    })

    //* Function Read Tool By ID -- No oData
    //* URL de Consulta:
http://localhost:4004/demosrv/getToolById(id='64da6b6e55b27b4c05d262c1')

    srv.on('getToolById',async (req) => {

        const { id } = req.data
        const toolsAPI = await cds.connect.to("toolsManager")
        return await toolsAPI.send({query:`GET /api/tools/${id}`, headers:
{
            userName: "p2c-comm-user",
            APIKey: "K306GLEZ1TPZZU4CIVGTQFQHXZ2920"
         }})
    })

    //* Function to get a list of available tools according description
    //* URL de Consulta:
http://localhost:4004/demosrv/getToolsByDescription(description='High
performance Drill')

    srv.on('getToolsByDescription', async (req) => {

        const { description } = req.data
        const toolsAPI = await cds.connect.to("toolsManager")
        return await toolsAPI.send({query:`GET
/api/tools/${description}/1`, headers: {
            userName: "p2c-comm-user",
            APIKey: "K306GLEZ1TPZZU4CIVGTQFQHXZ2920"
        }})
    })


    //* Function to change boolean status of a tool (available), to be
called on
    //* contract item creation

    srv.on('changeToolStatus', async (req) => {

        const { id } = req.data
        const toolsAPI = await cds.connect.to("toolsManager")
        return await toolsAPI.send({query:`PATCH /api/tools/${id}`,
headers: {
            userName: "p2c-comm-user",
            APIKey: "K306GLEZ1TPZZU4CIVGTQFQHXZ2920"
        }})
    })
```

Test the use of functions consuming url like:

```
http://localhost:4004/demosrv/getToolsByDescription(description='High
performance Drill')
```

2.5.8.- Improve Contract Item Creation

The contract item creation with on-the-fly value calculation is the next problem that needs to be solved. To support the task, the life cycle of the application must be intervened, in this case will be before the creation of the item.

Add this code to `external-test.js`

```
//* Contract Item Creation improvement
//* Change Tool Availability in Tool Management System
//* (Calculate item price)
//* (Add tool name)

srv.before('CREATE','ContractItems',async (req) => {

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

//*Change tool availability
await toolsAPI.send({
        query: `PATCH /api/tools/${req.data.tool_ID}`,
                headers: {
                        userName: "p2c-comm-user",
                        APIKey: "K306GLEZ1TPZZU4CIVGTQFQHXZ2920"
                }
        })
        .then(res => {
                const msg = res;
        })

//*Price Calculation and add tool name

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

2.5.9.- Data files adjustment

Before implement the BP name dynamic require, it is mandatory to change the file `p2c.cap.contracts-Contracts.csv` to add the last column for the BP composition. The file now should looks like this:

```
ID,description,beginDate,endDate,totalMonthValue,totalMonthCurrency,busine
ssPartner_BusinessPartner

CN-100,CCL Mining CO - Perforation 2023,2023-03-01,2023-10-25,0,USD,203

CN-105,Pacific Transportation INC. South Africa Mining,2022-02-02,2028-04-
02,0,USD,1018

CN-106,Atlantic Transportation INC. - Cooper Mine-Port Movement,2023-02-
02,2026-05-18,0,USD,1710

CN-108,Pure Gold Company - Explosives and Perforation, 2023-02-02,2025-12-
02,0,USD,1710

CN-109,Rusell & CO Diamond Mining - Australian Full Operation
Contract,2023-03-03,2030-10-18,0,USD,1018
```

If you have an error in the `cds watch --profile sandbox` command after change the Contracts.csv file, check in `schema.cds` and uncomment the businessPartner definition line into the entity definition

The Contracts entity definition in `schema.cds file` should look like:

```
entity Contracts : managed {
        key ID : String;
        description : String;
        beginDate : Date;
        endDate : Date;
        totalMonthValue : Integer;
        totalMonthCurrency : String;
        businessPartner: Association to BusinessPartners;
        contractItems : Composition of many ContractItems on
contractItems.contract = $self;
        bpName:String;
}
```

If you try the Contracts endpoint por the `devsrv` service, the bpName field will be empty or *null*. That is because we need to create the implementation to get that data in runtime

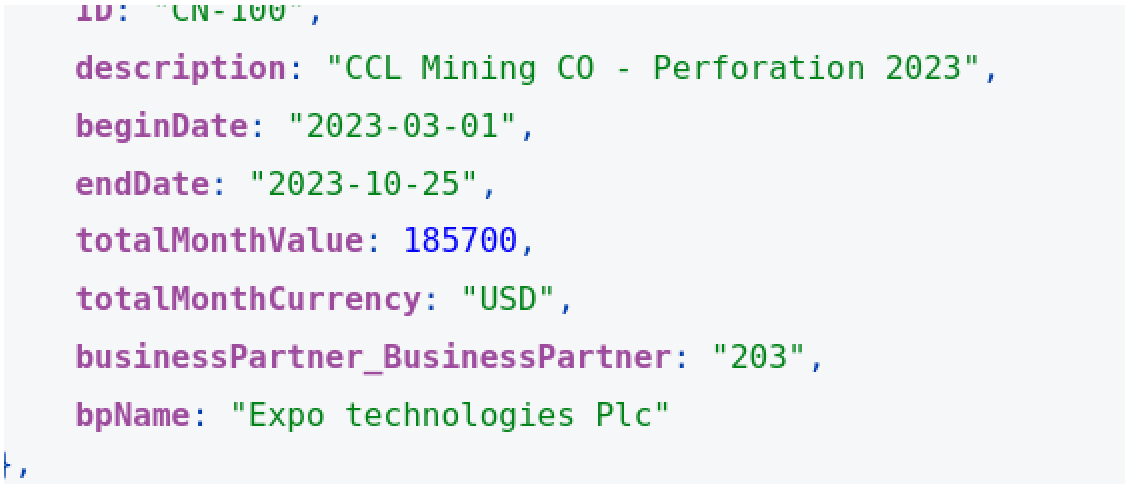
2.5.10.- Runtime Value Calculation

Its important to have the explanation about runtime calculations, that means values that are obtained according to the oData query that is being executed.
Using the application life cycle stages we can request for additional data, that can be a part of master data and operational information as well.
For this case we will implementa the BP Name search and the total contract value calculation.
Add the next snippet to `external-test.js` file:

```
/**
//*Implement Contract Value Calculation
//*Implement Business Partner Call - SAP S4HANA API
*/

//*Total Contract Value Calculation
srv.after('READ','Contracts',async (contracts, req) => {
        const db = srv.transaction(req);
        for(let each of contracts){

                //*Total Contract Value Calculation
                const items = await

db.run(SELECT.from('ContractItems').where({contract_ID:each.ID}))

                for (let i = 0; i < items.length; i++) {
                        const element = items[i];
                        each.totalMonthValue += element.price
                }
        }
})

//*BP Name Completion

srv.after('READ','Contracts',async (contracts, req) => {

        const db = srv.transaction(req);
        for(const cont of contracts){

                //* BP Name Completion
                const query = SELECT(A_BusinessPartner)
                .where({BusinessPartner:
cont.businessPartner_BusinessPartner})
```

```
                  .columns('BusinessPartnerFullName')

                  const bpName = await bupa.transaction().send({
                          query: query,
                          headers: {
                          apikey: "lSnMaBSXTGh7YX1aLfAyN08AdDkVRsfz"
                          }
                  })

                  if(bpName){
                          cont.bpName = bpName[0].BusinessPartnerFullName;
                  } else {
                          cont.bpName = 'BP Name Not Found'
                  }
          }
})
```

✓ **After the modifications, when consuming the Contracts entity, the total value will be the sum of the related items value and the bpName will be integrated from S/4HANA in runtime, the image should be like next.**



```
   ID: "CN-100",
   description: "CCL Mining CO - Perforation 2023",
   beginDate: "2023-03-01",
   endDate: "2023-10-25",
   totalMonthValue: 185700,
   totalMonthCurrency: "USD",
   businessPartner_BusinessPartner: "203",
   bpName: "Expo technologies Plc"
},
```

⚠ **The bpName completion will need also to be restructured because of the different kind of access to the s4hana system:**

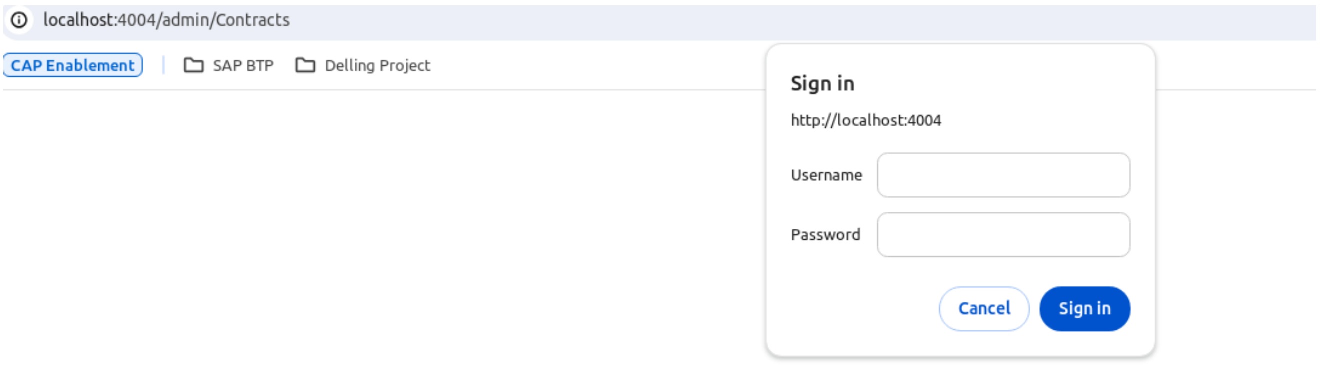
```
//*BP Name Completion - COMM User S4HANA Setup

srv.after('READ','Contracts',async (contracts, req) => {

        const db = srv.transaction(req);
        for(const cont of contracts){

        //* BP Name Completion
        const query = SELECT(A_BusinessPartner)
                .where({BusinessPartner:
cont.businessPartner_BusinessPartner})
                .columns('BusinessPartnerFullName')

        const username = 'MIG_BTP_RAP_USER000';
        const password = 'Miguelito@1234567890';

        const credentials =
Buffer.from(`${username}:${password}`).toString('base64');
        const authHeader = `Basic ${credentials}`;

        const bpName = await bupa.transaction().send({
                query: query,
                headers: {
                        'Authorization': authHeader,
                }
        })

                if(bpName){
                        cont.bpName = bpName[0].BusinessPartnerFullName;
                } else {
                        cont.bpName = 'BP Name Not Found'
                }
        }
})
```

2.6.- Basic authorization model review.

The internal authorization model for applications works based on user roles and actions that can be performed considering the assigned roles to each user. The restriction model will be applied directly to the service on the entity definitions. In this case the `admnin-service.cds` file will be modified to restrict some actions:

```
using { p2c.cap.contracts as my } from '../db/schema';

service AdminService @(path: '/admin'){
        entity Contracts @(restrict : [
                {
                        grant: ['*'],
                        to: ['P2C_ADMIN_ROLE']
                }
        ]) as projection on my.Contracts;

        entity ContractItems as projection on my.ContractItems;
}
```

After the change, when try to consume the **Contracts** entity, a login screen will be displayed, it is necessary to assign the `P2C_ADMIN_ROLE` role to some user.



Now, in the `package.json` define the `[sandbox]` profile security model with basic authentication.
Insert the snippet in the cds --> requires at the begin of the section:

```
"[sandbox]":{
        "//": "---------------- BEGIN Basic local authentication",

        "auth": {
                "kind": "basic",
                "users": {
                        "admin":{
                                "password": "admin",
                                "roles": ["P2C_ADMIN_ROLE",
"P2C_READER_ROLE"]
                        },
                        "reader":{
                                "password": "reader",
                                "roles": ["P2C_READER_ROLE"]
                        }
                }
        },

        "//": "---------------- END OF Basic local authentication"
},
```

✓ **Finally, can test the Contracts entity in the `admin-service.cds` and it should ask for the credentials. If try Contract Items in the same service, will work without problems.**

Just in this case, in `package.json` file, in the auth model for production we will use the option `"restrict_all_services": false`. The snippet should looks like:

```
"requires": {
"[production]": {
        "db": "hana",
        "auth": {
                "kind": "xsuaa",
                "restrict_all_services": false
        }
},
```

3.0.- Deploy to CF from VSCode

The deployment process from VSCode requires some preparation.

- First, need to install CF CLIThe CF CLI commands will be used for all the process.
- In SAP BTP we need to setup a Cloud Foundry Space in a sub account.
- Create a HANA Cloud Instance in the Cloud Foundry Space that will be used as the destination for the deployment

  It is important that the deployment process could need an HDI Container instance already created in HANA. The MTA deploy process will try to create ir, but in case of failures, it could be necessary to create it manually.

- Install **mta build tool**. Using npm execute `npm i -g mbt` this tool will help to create the project descriptor for the application
- **Remember to execute the installation for mta build tools with administrator credentials**
- Install mta plugins:

```
cf add-plugin-repo CF-Community https://plugins.cloudfoundry.org
cf install-plugin multiapps
```

⚠ **Check all the plugins and installations, these tools are mandatory to the deployment process to CloudFoundry environment**

3.1.- The deployment process

3.1.1.- Using SAP HANA Database

While we used SQLite as a low-cost stand-in during development, we're going to use a managed SAP HANA database for production:

```
cds add hana --for production
```

Learn more about using SAP HANA for production.

The command will add a new [production] section like the next example:

```
"[production]": {
        "db": "hana"
}
```

3.1.2.- Using XSUAA-Based Authentication

Configure your app for XSUAA-based authentication. This way, the security model will be consumed on BTP - Cloud Foundry, but the secure configuration about roles & profiles and the related restrictions will be generated according the code annotations

```
cds add xsuaa --for production
```

Now the `[production]` definition in `package.json` will looks like:

```
"[production]": {
        "db": "hana",
        "auth": "xsuaa"
}
```

This will also generate an `xs-security.json` file, with roles/scopes derived from authorization-related annotations in your CDS models. Ensure to rerun `cds compile --to xsuaa`, as documented in the *Authorization* guide whenever there are changes to these annotations.

Learn more about SAP Authorization and Trust Management/XSUAA.

3.1.2.1.- The *basic* authentication model

CAP presents many options for authentication model for development or production process. Even when we force the use of **XSUAA** so we can get advantage of the BTP + CloudFoundry security model and we only need the Roles and Restriction model, when we work in development probably will be better a different option.

For more information about the authentication options, please go to CAP Documentation

The different authentication strategies included in CAP are:

- Dummy authentication
- Mock authentication
- Basic authentication
- JWT authentication
- XSUAA
- IAS-based authentication
- Custom authentication

Each strategy will need a definition in `package.json` file. In this example case we will use **Basic Authentication** for development and **XSUAA** for production.
The setup will require the roles definition in `xs-security.json`, with 2 roles definition **P2C_ADMIN_ROLE** & **P2C_READER_ROLE**:

```
{

        "xsappname": "cap_clean_core_demo",
        "tenant-mode": "dedicated",

        "scopes": [
                {
                "name": "$XSAPPNAME.P2C_ADMIN_ROLE",
                "description": "P2C_ADMIN_ROLE"
                }
        ],
        "attributes": [],
        "role-templates": [
                {
                        "name": "P2C_ADMIN_ROLE",
                        "description": "generated",
                        "scope-references": [
                        "$XSAPPNAME.P2C_ADMIN_ROLE"
                        ],
                        "attribute-references": []
                        },
                {
                        "name": "P2C_READER_ROLE",
                        "description": "generated only for read",
                        "scope-references": [
                        "$XSAPPNAME.P2C_READER_ROLE"
                        ],
                        "attribute-references": []
                }
        ]
}
```

With the roles already defined, the restrictions can be applied to the services. In this cae will be `admin-service.cds` the example for authorizations:

```
using { p2c.cap.contracts as my } from '../db/schema';

service AdminService @(path: '/admin'){
        entity Contracts @(restrict : [
                {
                grant: ['*'],
                to: ['P2C_ADMIN_ROLE']
                }
        ]) as projection on my.Contracts;
        entity ContractItems as projection on my.ContractItems;
}
```

Finally, the definition for the authentication model and the tester users in `package.json`:

Insert before the production authentication definition inside cds require section:

```
"cds": {
        "requires": {
                "//": "---------------- BEGIN Basic local authentication",
                "auth": {
                        "kind": "basic",
                        "users": {
                                "admin":{
                                "password": "hola123",
                                "roles": ["P2C_ADMIN_ROLE",
"P2C_READER_ROLE"]
                                }
                        }
                },

                "//": "---------------- END OF Basic local
authentication",

                "[production]": {
                        "db": "hana",
                        "auth": {
                                "kind": "xsuaa",
                                "restrict_all_services": false
                        }
                },
```

⚠ **Since you could use just the service of the back-end, without the application security model, need to add to the `[production]` section in `package.json`, the instruction: `"restrict_all_services": false` like in the code. With this, the services will be available to be consumed without interface but the security model will not work for all the app.**

✎ **The login/password information are hardcoded just for development process. When the application is deployed to production, the user/password will be managed by the IAS and only the Role assignment will be checked by the platform. Also, the deploy process will take in charge the generation of the role collection and the deploy of the same information.**

3.1.3.- Using MTA-Based Deployment

We'll be using the Cloud MTA Build Tool to execute the deployment. The modules and services are configured in an `mta.yaml` deployment descriptor file, which we generate with:

```
cds add mta
```

Learn more about MTA-based deployment.

✎ **The `cds add mta` command will add the `mta.yaml` application descriptor file that has a summary of the relevant project content for the deployment process. This file will be used as source for the build process**

3.1.4.- Using App Router as Gateway

The *App Router* acts as a single point-of-entry gateway to route requests to. In particular, it ensures user login and authentication in combination with XSUAA.

If the app router is not included, even when only the backend is created, the login and authentication model will not work anyway.

Two deployment options are available:

- **Managed App Router**: for scenarios where *SAP Fiori Launchpad* (FLP) is the entry point to access your applications, the Managed App Router provided by SAP Fiori Launchpad is available. See the end-to-end tutorial for the necessary configuration in `mta.yaml` and on each *SAP Fiori application*.
- **Custom App Router**: for scenarios without SAP Fiori Launchpad, the app router needs to be deployed along with your application. Use the following command to enhance the application configuration:

```
cds add approuter --for production
```

Learn more about the SAP BTP Application Router.

3.1.5.- Optional: Add Multitenancy

To enable multitenancy for production, run the following command:

```
cds add multitenancy --for production
```

If necessary, modifies deployment descriptors such as *mta.yaml* for Cloud Foundry.

Learn more about MTX services.

✓ **you're set!**

The previous steps are required *only once* in a project's lifetime. With that done, we can repeatedly deploy the application.

3.1.6.- Freeze Dependencies

Deployed applications should freeze all their dependencies, including transient ones. Create a *package-lock.json* file for that:

```
npm update --package-lock-only
```

Learn more about dependency management for Node.js

Regularly update your dependencies

You should **regularly update your `package-lock.json`** to consume latest versions and bug fixes. Do so by running this command again, for example each time you deploy a new version of your application.

4.- Build & Assemble

4.1.- Build Deployables with `cds build`

Run `cds build` to generate additional deployment artifacts and prepare everything for production in a local `./gen` folder as a staging area. While `cds build` is included in the next step `mbt build`, you can also run it selectively as a test, and to inspect what is generated:

```
cds build --production
```

Learn more about running and customizing `cds build`.

4.2.- Assemble with `mbt build`

Prepare monorepo setups
The CAP samples repository on GitHub has a more advanced (monorepo) structure, so tell the `mbt` tool to find the `package-lock.json` on top-level:

```
ln -sf ../package-lock.json
```

Now, we use the `mbt` build tool to assemble everything into a single `mta.tar` archive:

```
mbt build -t gen --mtar mta.tar
```

Got errors? See the troubleshooting guide.

Learn how to reduce the MTA archive size during development.

4.3.- Deploy to Cloud

Finally, we can deploy the generated archive to Cloud Foundry:

```
cf deploy gen/mta.tar
```

You need to be logged in to Cloud Foundry.

This process can take some minutes and finally creates a log output like this:

log

```
[…]
Application "<application name>" started and available at
"[org]-[space]-<app name>.landscape-domain.com"
[…]
```

Copy and open this URL in your web browser. It's the URL of your approuter application.

⚠ **The deployment process will end with 2 available links, the app and the service. Remember the authentication model only will run on the app link, but the app link will need to be BTP authenticated, the server will not be blocked, remember the restrict_all_services: false instruction on `package.json` file. Is**

important to show the difference between both links at the evaluation moment in BTP.