INTERNAL - Authorized for Partners

Partner
Ecosystem
Success

Exercise 1 – Build an extension on S/4HANA Cloud

S/4HANA Cloud Developer Extensibility bootcamp

www.sap.com/contactsap

THE BEST RUN

Table of Contents

Introduction

This Hands-On workshop will guide you to build developer extensions in a S/4HANA Cloud ABAP Environment system (S4HC). You will use released BAdIs to implement custom code and enhance an existing business application. You will also create your own transactional UI and use released RAP facades to enhance existing functionality.

We will focus on the development tasks. Deployment of UIs and UI development in general will be only partly touched.

In this tutorial, wherever XXX appears, use the number assigned to you (e.g. 000).

Exercise scope

This hands-on session consists of one big exercise (Part 1) with two optional parts in addition to identify further possibilities of SAP S/4HANA Cloud ABAP Environment.

Part 1 consists of a more detailed version of the official tutorial to create a custom Business Object with a custom Fiori Elements based UI integrated into standard objects.

Part 2 is optional and enhances the use of the ABAP Environment within S/4HANA Cloud. It is basically a starting point to show you what else might be useful leveraging S/4HANA Cloud ABAP Environment.

Prerequisites

Make sure that you have a developer user in the system with necessary authorizations.

Make sure that you have a user with the BR_BPC_EXPERT role assigned to maintain number range intervals.

You have installed SAP ABAP Development Tools (ADT), version 3.16 or later, and have created an ABAP Cloud project for your SAP S/4HANA Cloud System in it. You are familiar with the concept of extensions to the SAP standard and ABAP RESTful Application Programming Model (RAP).

Hints and Tips

Speed up the typing by making use of the Code Completion feature (shortcut: Ctrl+Space) or the prepared Code Snippets mentioned in the next section. You can easily open an object with the shortcut Ctrl+Shift+A.

PART 1: DEVELOP AN ONLINE SHOP APPLICATION USING THE ABAP RESTFUL APPLICATION PROGRAMMING MODEL

Business Scenario

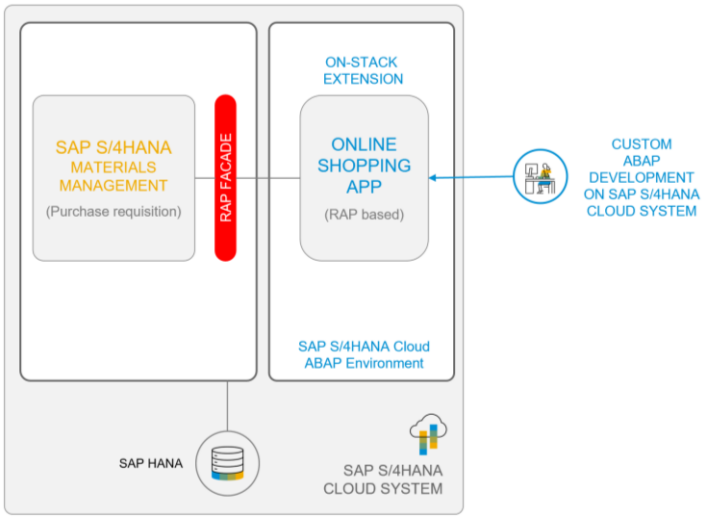
You will develop a custom app (online shop) which will help new employees order their equipment as part of an onboarding process. In the online shop, employees can choose from four starter packages: IT Developer Package, IT Consultant Package, IT Sales Package, and IT General Package. Once a package has been chosen, a new purchase requisition is created via the standard purchase requisitions API. Each starter package contains different items. This will be reflected in the purchase requisition.

The app could be a simple starting point to further support the onboarding process by incorporating more business and custom objects.

Note:

Please note that the business scenario has been developed for training purposes only and might not reflect all the complexity of a real-world scenario. The focus in these hands-on sessions is purely on technical knowledge transfer and therefore optimizations from a business perspective are not part of the training and the business scenario will not be further assessed.

In this scenario, we want to focus only on creating a RAP based Business Object that can invoke the S/4HANA Purchase Requisition API (as RAP Façade) to create a new Purchase Requisition.



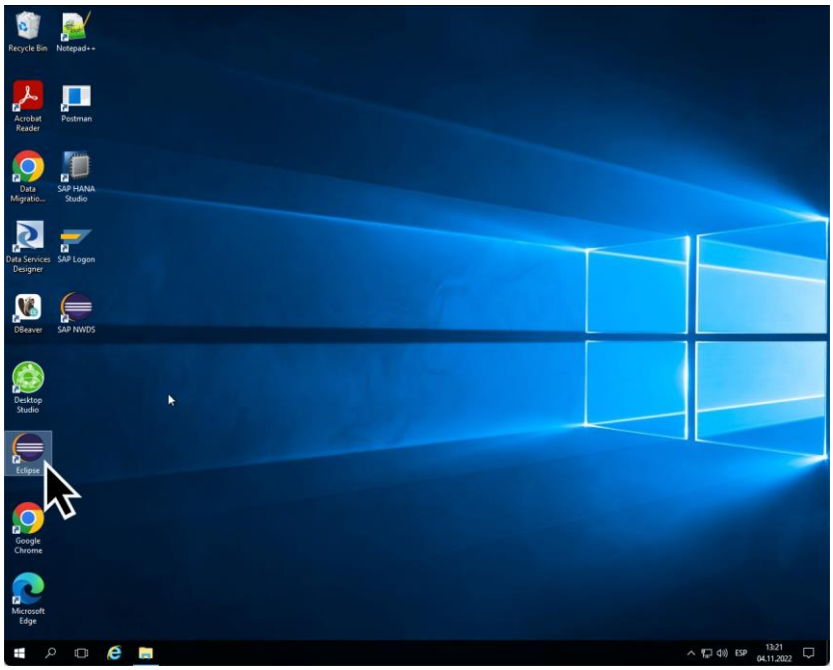
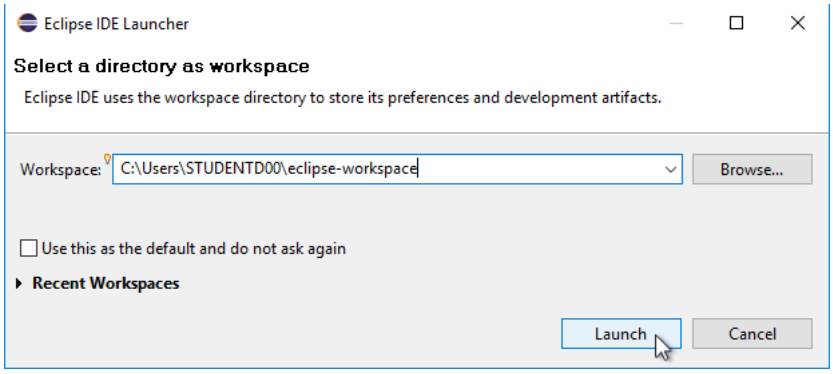
You will learn:

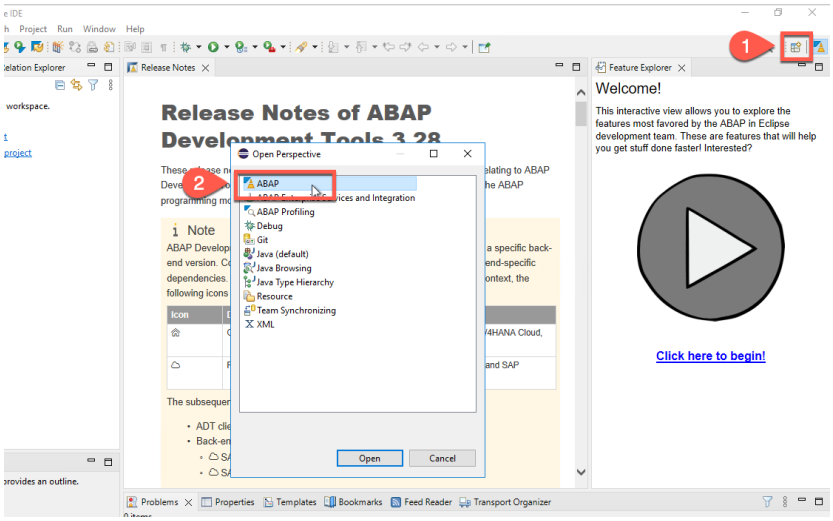
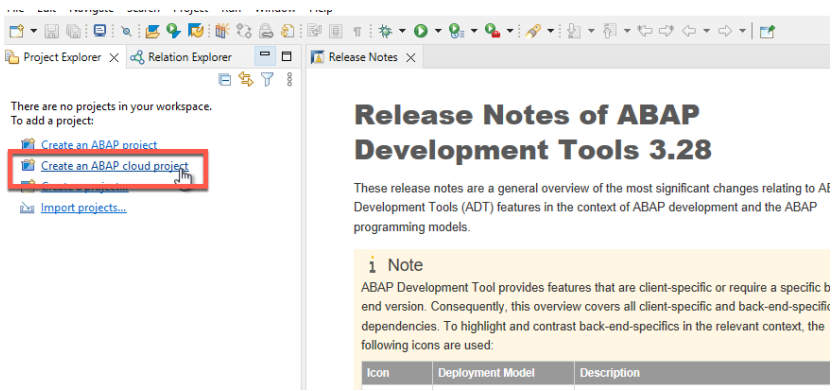
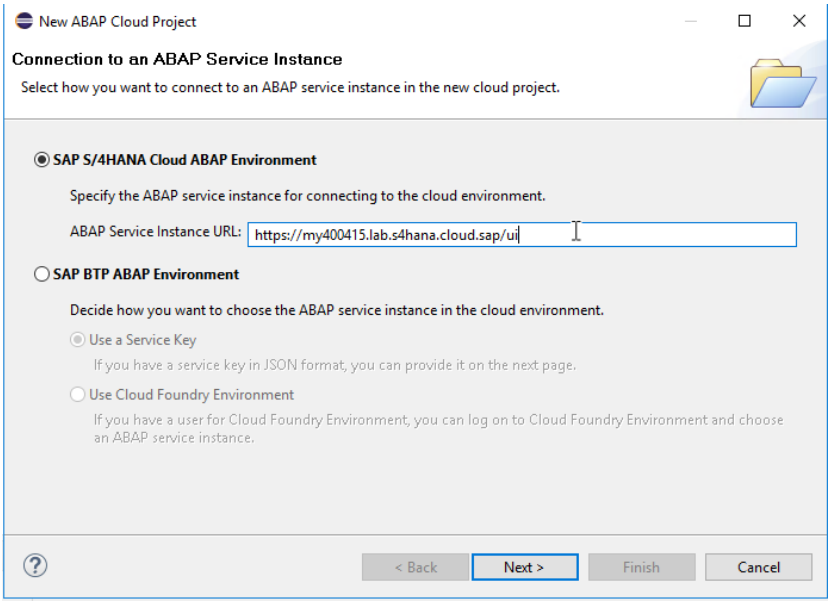
- ✓ How to logon to SAP S/4HANA Cloud ABAP Environment
- ✓ How to create an ABAP package
- ✓ How to create a database table
- ✓ How to create a CDS model and projection view
- ✓ How to create behavior definition & implementation
- ✓ How to integrate with SAP S/4HANA Standard Objects
- ✓ How to create service definition & service binding
- ✓ How to Provide Input Help
- ✓ How to run SAP Fiori Elements Preview

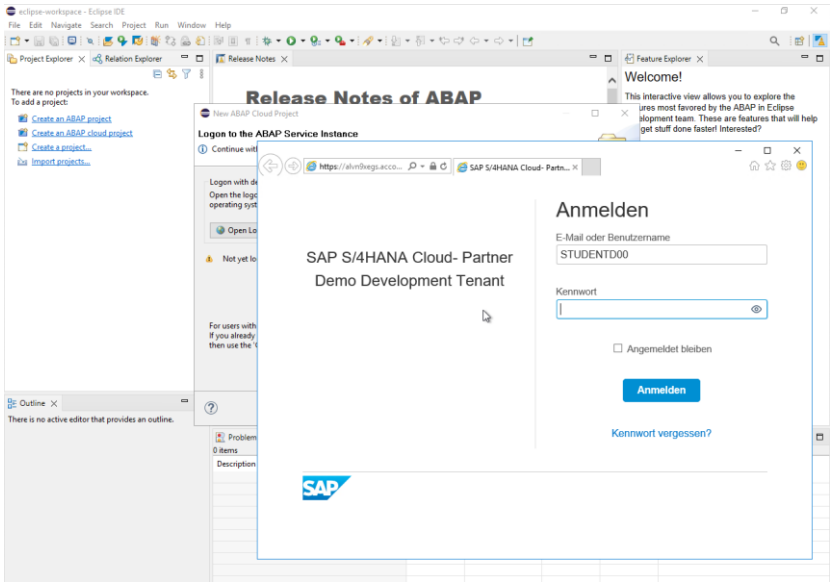
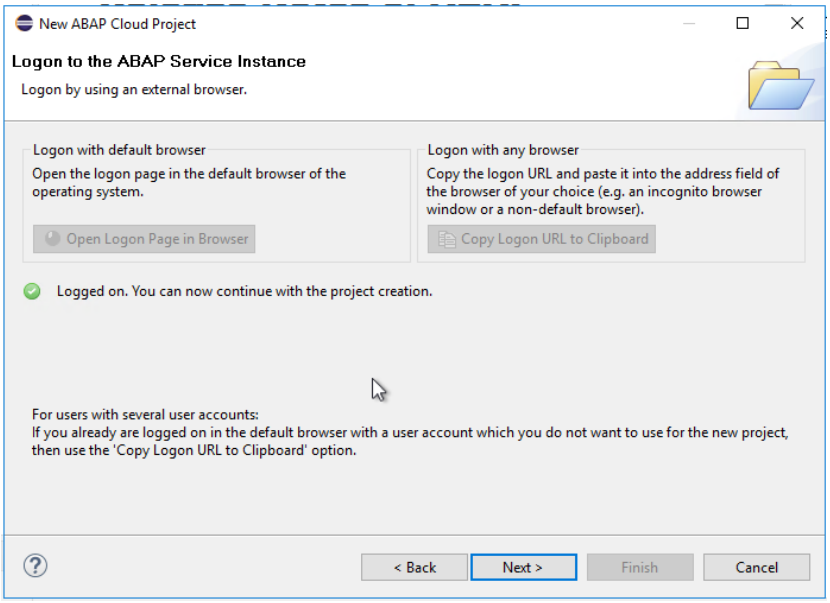
Step 1: Create the RAP Business Object

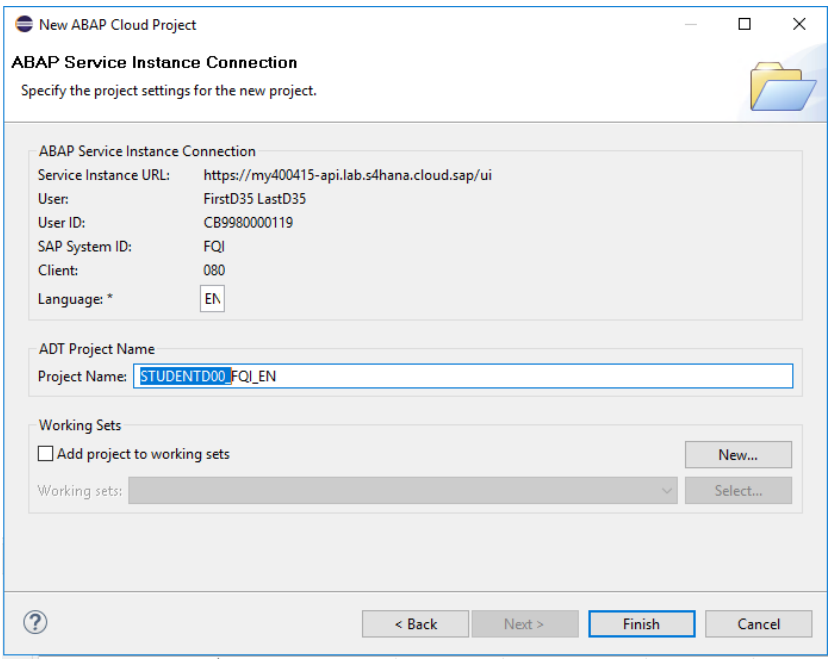
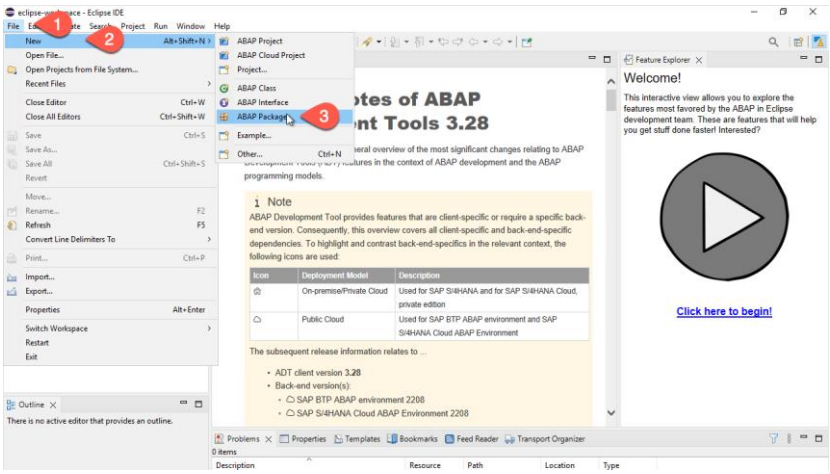
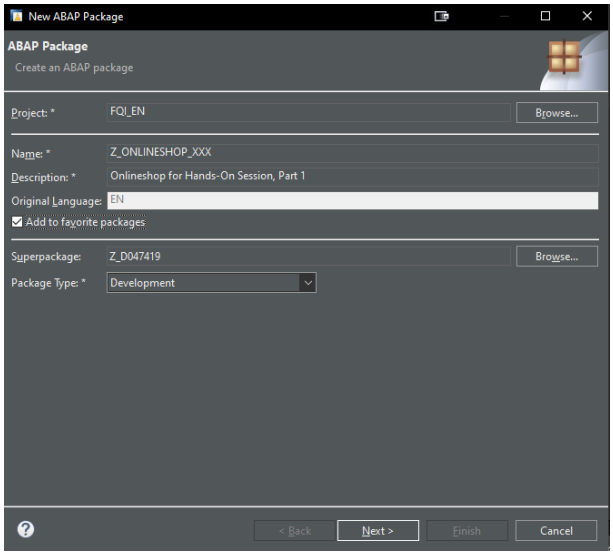
First task is to create a custom object in ABAP (based on RAP) for the online shop data. In this case, things will be simple, and we will have only one table containing the shopping object that will be used to create a purchase requisition.

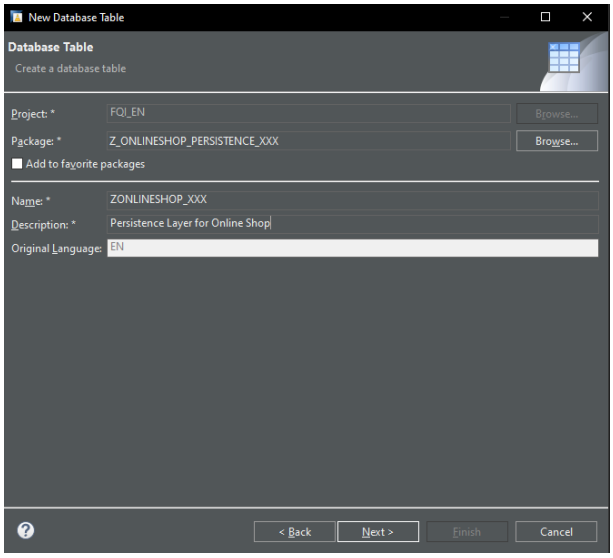
Remember to replace XXX with your individual sequence / number.

Description	Screenshot
Open Eclipse using the icon in the Remote Desktop.	
Create a new workspace and make sure you user you user folder. If the folder already exists, create a new one. Example: C:\Users\STUDENT<XXX>\eclipse-workspace where <XXX> is your ID (e.g. D01).	

Description	Screenshot
Open the ABAP Perspective by clicking the “Open Perspective” icon on the top right and then selecting “ABAP” and finally pressing “Open”.	
On the left side you have the Project Explorer bar. Click on “Create an ABAP Cloud Project”	
Select the first option “SAP S/4HANA Cloud ABAP Environment”. Use the S/4HANA Cloud URL available in the Cheat Sheet. This is connecting you to the Development Tenant. Make sure there are no spaces in between. **Note**: This maybe a different URL than shown in the screenshot depending on when this document is being referred.	

Description	Screenshot
Open the system login page using “Open Logon Page in Browser”. Use your credentials from system “A” in your Cheat Sheet.	
Once the login is done successfully, you can close the browser window and you should see a success message in Eclipse “Logged on….”. Click “Next”.	

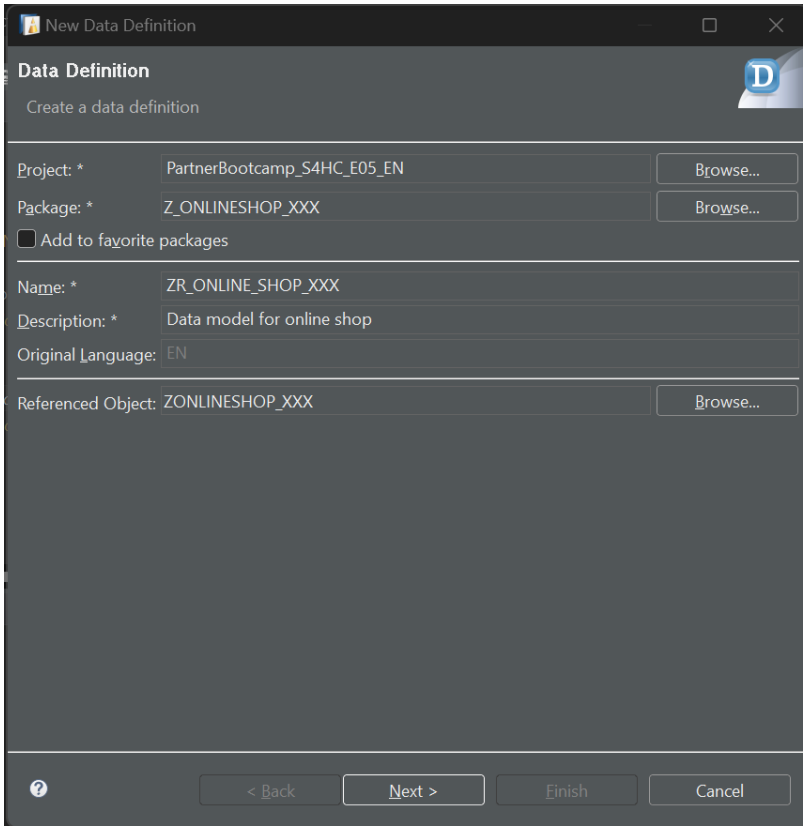
Description	Screenshot
Define the local project name. Use your user ID as a prefix to distinguish your project. Click “Finish”.	
Select File >> New >> ABAP Package	
Define the following parameters: • Name: Z_Onlineshop_XXX • Description: Onlineshop for hands-on session, Part 1 • Check Add to favorite packages Create in the following dialog a new Transport Request and click on Finish.	

Description	Screenshot
Create a database table: 1. Right-click your package Z_Onlineshop_XXX and select New > Other ABAP Repository Object. 2. Search for database table, select it and click Next >. 3. Enter the following database table information: • Name: ZONLINESHOP_XXX • Description: Persistence for Online Shop Select in the following dialog your Transport Request and click on Finish.	

Replace your database definition with the following code:
 	
Save and activate.
	`@EndUserText.label : 'Shop to purchase electronics'`
	`@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE`
	`@AbapCatalog.tableCategory : #TRANSPARENT`
	`@AbapCatalog.deliveryClass : #A`
	`@AbapCatalog.dataMaintenance : #RESTRICTED`
	`define table zonlineshop_xxx {`
	`  key client  : abap.clnt not null;`
	`  key order_uuid : sysuuid_x16 not null;`
	`  order_id  : abap.char(10) not null;`
	`  deliverydate  : abap.dats;`
	`  creationdate  : abap.dats;`
	`  pkgid  : abap.int1;`
	
	`}`

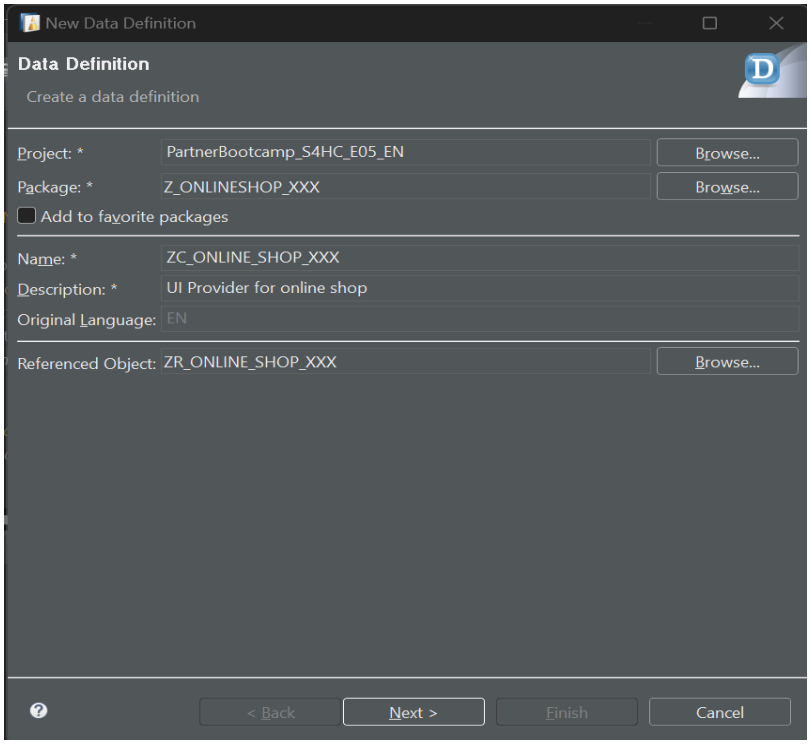
In the same package (Z_Onlineshop_XXX) create a second database table with the following information:
• Name: zshop_as_xxx
• Description: Additional save persistence for Online Shop
 	
Replace your database definition with the following code:
 	
Save and activate.
	`@EndUserText.label : 'Database table for additional save'`
	`@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE`
	`@AbapCatalog.tableCategory : #TRANSPARENT`
	`@AbapCatalog.deliveryClass : #A`
	`@AbapCatalog.dataMaintenance : #RESTRICTED`
	`define table zshop_as_xxx {`
	`  key client  : abap.clnt not null;`
	`  key order_uuid : sysuuid_x16 not null;`
	`  purchasereqn  : abap.string(256);`
	`  purinforecord  : abap.string(256);`
	`  purorder  : abap.string(256);`
	`  costcenter  : kostl;`
	
	`}`

Create a CDS data model	

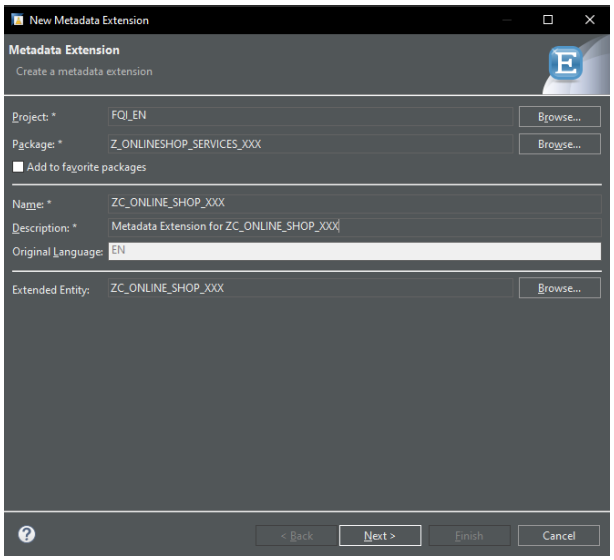
Description	Screenshot
1. Right-click your package Z_Onlineshop_XXX and select New > Other ABAP Repository Object. 2. Search for data definition, select it and click Next >. 3. Enter the following data definition information: • Name: ZR_ONLINE_SHOP_XXX • Description: Data model for online shop Select in the following dialog your Transport Request and click on Finish.	

Replace your data definition with the following code:
 	
Save and activate.
	`@EndUserText.label: 'Data model for online shop'`
	`@AccessControl.authorizationCheck: #NOT_REQUIRED`
	`define root view entity ZR_ONLINE_SHOP_XXX`
	`  as select from zonlineshop_xxx`
	`  association [1..1] to zshop_as_xxx as _Shop on`
	`  $projection.Order_Uuid = _Shop.order_uuid`
	`{`
	
	`  key order_uuid  as Order_Uuid,`
	`  order_id  as Order_Id,`
	`  deliverydate as Deliverydate,`
	`  creationdate as Creationdate,`
	`  pkgid  as PackageId,`
	`  _Shop.costcenter as CostCenter,`
	
	`  /*Associations*/`
	`  _Shop`
	`}`

Create a projection view	

Description	Screenshot
1. Right-click your package Z_Onlineshop_XXX and select New > Other ABAP Repository Object. Search for data definition, select it and click Next >. 2. Enter the following projection view information: • Name: ZC_ONLINE_SHOP_XXX • Description: UI Provider for online shop • Referenced Object: ZR_ONLINE_SHOP_XXX Select in the following dialog your Transport Request and click on Finish.	

Replace your data definition with the following code:
 	
Save and activate.
	`@EndUserText.label: 'shop projection'`
	`@AccessControl.authorizationCheck: #NOT_REQUIRED`
	`@Metadata.allowExtensions: true`
	`define root view entity ZC_ONLINE_SHOP_XXX`
	`  provider contract transactional_query`
	`  as projection on ZR_ONLINE_SHOP_XXX`
	`{`
	`  key Order_Uuid,`
	`  Order_Id,`
	`  Deliverydate,`
	`  Creationdate,`
	`  PackageId,`
	`  CostCenter,`
	`  _Shop.purchasereqn as Purchasereqn`
	`}`

1. Right-click your data definition ZC_ONLINE_SHOP_XXX and select New Metadata Extension. 2. Enter the following metadata extension information: • Name: ZC_ONLINE_SHOP_XXX • Description: UI Annotation definition for online shop Select in the following dialog your Transport Request and click on Finish.	

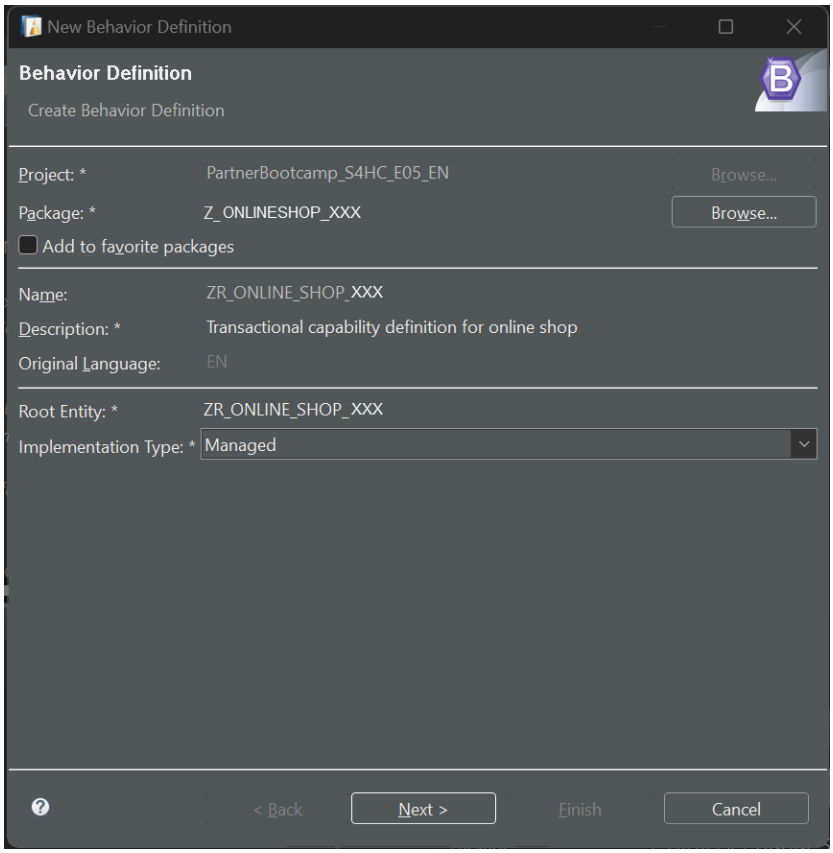
Description	Screenshot
Replace your metadata extension code with the code for the metadata extension shared with you Save and activate.	

```
@Metadata.layer: #CUSTOMER
@Search.searchable: true
@UI: { headerInfo: { typeName: 'Online Shop',
                     typeNamePlural: 'Online Shop',
                     title: { type: #STANDARD, label: 'Online Shop', value:
'order_id' }},
       presentationVariant: [{ sortOrder: [{ by: 'Creationdate',
                                             direction: #DESC }] }] }
annotate view ZC_ONLINE_SHOP_XXX with
{

  @UI.facet: [{ id:       'Orders',
                purpose:  #STANDARD,
                type:     #IDENTIFICATION_REFERENCE,
                label:    'Order',
                position: 10 }]
  Order_Uuid;
  @UI: { lineItem: [{ position: 10,label: 'Order id', importance: #HIGH }],
         identification: [{ position: 10, label: 'Order id' }]
       }
  @Search.defaultSearchElement: true
  Order_Id;

  @UI: { lineItem:[{ position: 20,label: 'Creation date', importance: #HIGH
},
                   { type: #FOR_ACTION, dataAction: 'update_inforecord',
label: 'Update IR' }],
         identification: [{ position: 20, label: 'Creation date' }]
       }
  Creationdate;
  @UI: { identification: [{ position: 30, label: 'Purchase Requisition' }]
}
  Purchasereqn;
  @UI: { lineItem:       [{ position: 40,label: 'Package ID', importance:
#HIGH }],
         identification: [{ position: 40, label: 'Package ID' }]
       }
  @Consumption.valueHelpDefinition: [{ entity: { name: 'ZC_PREPACKAGE-
DITEMS_VH', element: 'PackageId' }}]
  PackageId;
  @UI: { lineItem:       [{ position: 50,label: 'Cost Center', importance:
#HIGH }],
         identification: [{ position: 50, label: 'Cost Center' }]
       }
  @Search.defaultSearchElement: true
  CostCenter;
}
```

Create behavior definitions

1. Right-click your data definition ZR_ONLINE_SHOP_XXX and select New Behavior Definition.
2. Enter the following Behavior Definition information:
   - Description: Transactional capability definition for online shop

Select in the following dialog your Transport Request and click on Finish.



Description	Screenshot
Replace your behavior definition with the following code: Save and activate.	

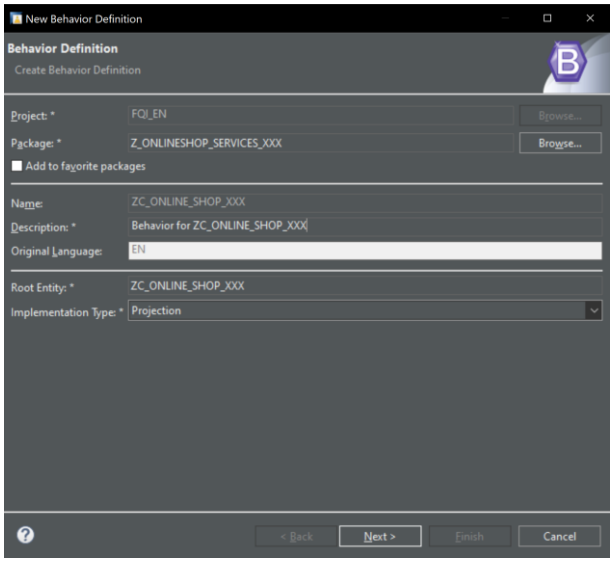
```
managed implementation in class zbp_r_online_shop_xxx unique;

define behavior for ZR_ONLINE_SHOP_XXX alias Online_Shop
persistent table zonlineshop_xxx
with additional save
lock master
authorization master ( instance )
//etag master <field_name>
{
  field ( numbering : managed, readonly ) order_Uuid;
  field ( readonly ) Creationdate, order_id;
  determination calculate_order_id on modify { create; }
  internal action create_pr parameter $self;
 //For demonstration of possible usage only
  internal action set_inforecord;
 //For demonstration of possible usage only
  internal action update_inforecord;
  create;
  update;
  delete;

  mapping for zonlineshop_xxx {
  PackageId = pkgid;
  Order_Id = order_id;
  Creationdate = creationdate;
  Deliverydate = deliverydate;
  Order_Uuid = order_uuid;
  }
}
```

Create a second behavior definition. This time for your projection view, with the following information:

1. Right-click your projection view ZC_ONLINE_SHOP_XXX and select New Behavior Definition.
2. Enter the following Behavior Definition information:
   - Description: Behavior for ZC_ONLINE_SHOP_XXX

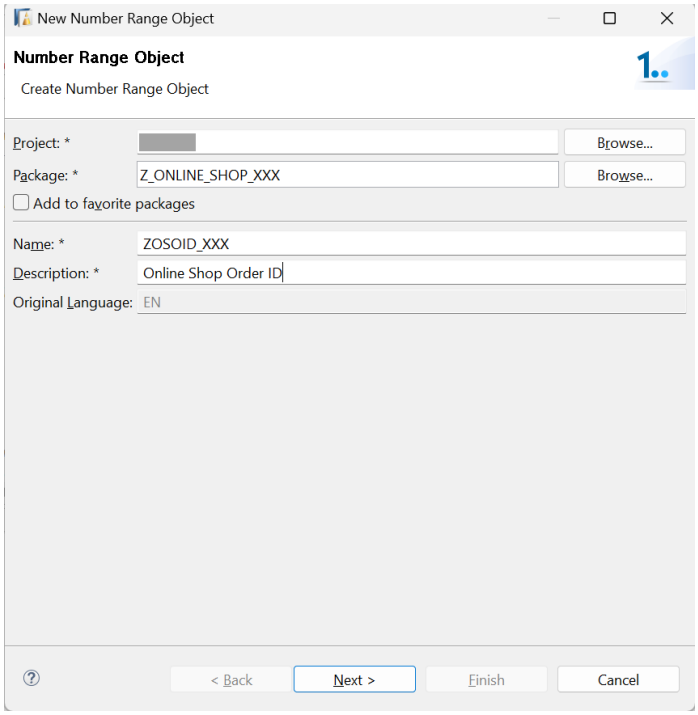
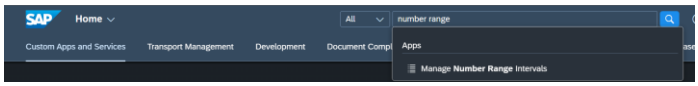
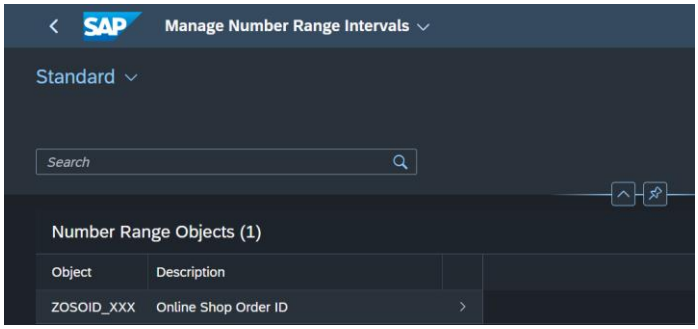
Select in the following dialog your Transport Request and click on Finish.

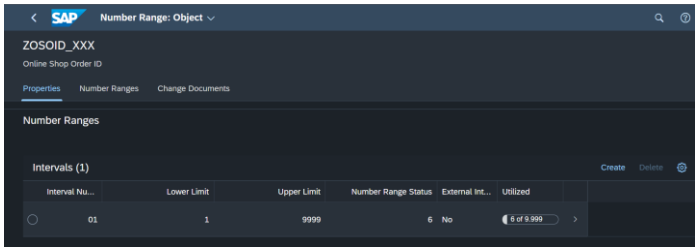
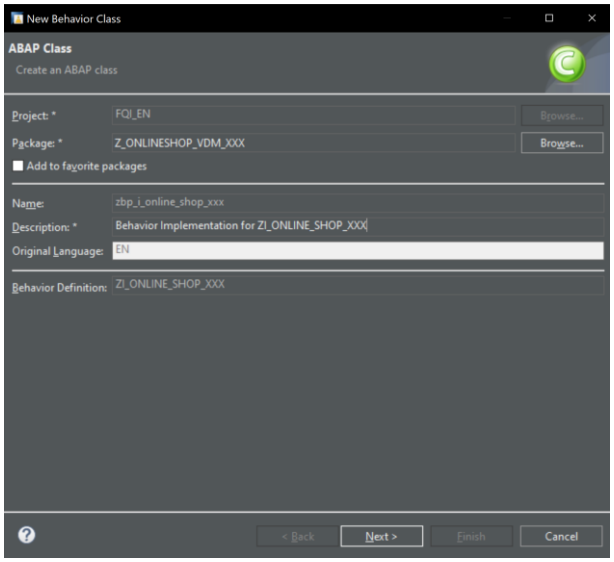
Keep the default source code, save and activate.



Step 2: Create Purchase Requisition within the Business Object

Description	Screenshot
Create behavior implementation	

Description	Screenshot
Create a number range object to handle the OrderID generation. We will use this later in the behavior implementation Name: ZOSOID_XXX Description: Online Shop Order ID Select in the wizard your transport request and click finish	
Provide the Number Length Domain: Z_OLSP_ORD_ID_DOMA Percent Warning: 10 Rolling: true Buffering: No Buffering Buffered Numbers: 0 You could leave the “Sub Type” empty Save and activate	
Open the S/4HANA Cloud system URL from the cheat sheet (system A) Use the search to find the Manage Number Range Intervals app	
Navigate to the Manage Number Range Intervals app Open the Number Range Object created above **NOTE: Be careful NOT to modify objects not owned by you**	

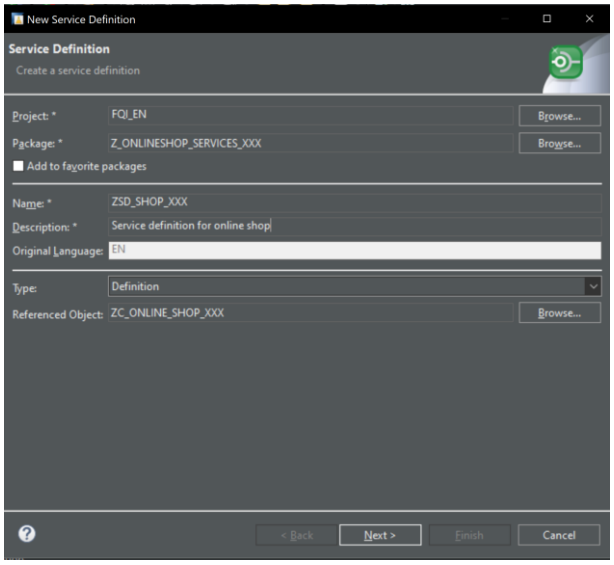
Description	Screenshot
Maintain an interval for the number range object with your group number (replacement of XXX). Open the number range objects and click “Create” in the intervals section Provide information as per the screenshot Interval Number: 01 Lower Limit: 1 Upper Limit: 9999 The External Interval checkbox could be left unchecked. **NOTE: Be careful NOT to modify objects not owned by you**	
Open the behavior definition ZR_ONLINE_SHOP_SOL and use the quick-help suggestion against the statement ‘managed implementation in class zbp_r_online_shop_xxx unique;’ to create new implementation class. Enter the following Behavior Class information: • Description: Behavior Implementation for ZR_ONLINE_SHOP_XXX Select in the following dialog your Transport Request and click on Finish.	

In your Global Class, replace your code with following:
 	
Save and activate.
	`CLASS zbp_r_online_shop_xxx DEFINITION PUBLIC ABSTRACT FINAL FOR BEHAVIOR OF zr_online_shop_xxx.`
	`PUBLIC SECTION.`
	`class-DATA cv_pr_mapped TYPE RESPONSE FOR MAPPED i_purchaserequisitiontp.`
	`ENDCLASS.`
	
	`CLASS zbp_r_online_shop_xxx IMPLEMENTATION.`
	`ENDCLASS.`

Description	Screenshot
In your Local Types, replace your code with the code for the local types of behavior implementation shared with you. **Note**: The code in the screenshot is NOT complete and the complete code has been provided offline for ease of consumption. Save and activate.	

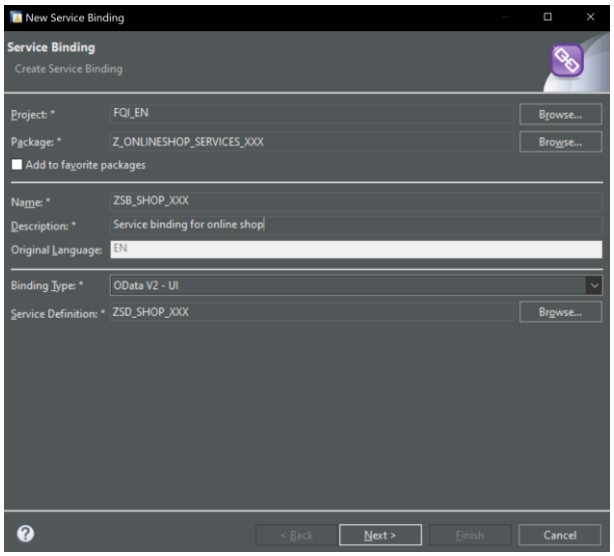
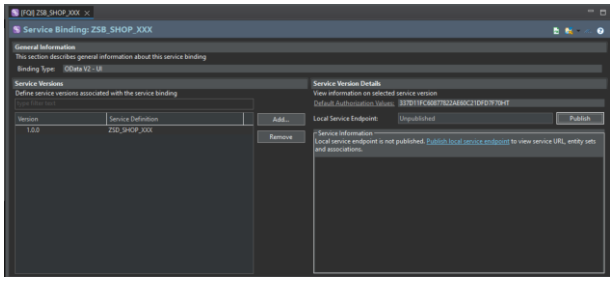
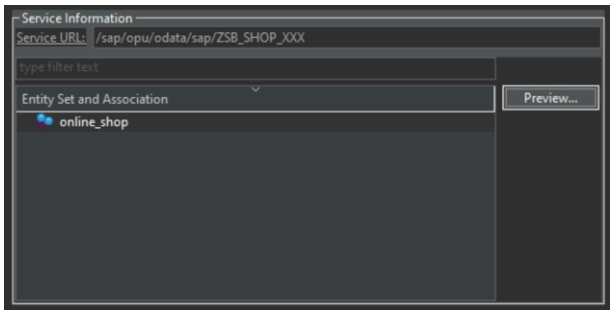
```
CLASS lsc_zbp_r_online_shop_xxx DEFINITION INHERITING FROM cl_abap_behav-
ior_saver.
  PROTECTED SECTION.
    METHODS save_modified REDEFINITION.
ENDCLASS.

CLASS lsc_zbp_r_online_shop_xxx IMPLEMENTATION.

  METHOD save_modified.
    DATA : lt_online_shop_as TYPE STANDARD TABLE OF zshop_as_xxx,
           ls_online_shop_as TYPE                   zshop_as_xxx.
    IF zbp_r_online_shop_xxx=>cv_pr_mapped-purchaserequisition IS NOT INI-
TIAL.
      LOOP AT zbp_r_online_shop_xxx=>cv_pr_mapped-purchaserequisition AS-
SIGNING FIELD-SYMBOL(<fs_pr_mapped>).
        CONVERT KEY OF i_purchaserequisitiontp FROM <fs_pr_mapped>-%pid TO
DATA(ls_pr_key).
        <fs_pr_mapped>-purchaserequisition = ls_pr_key-purchaserequisition.
      ENDLOOP.
    ENDIF.

    IF create-online_shop IS NOT INITIAL.
      " Creates internal table with instance data
      lt_online_shop_as = CORRESPONDING #( create-online_shop ).
      lt_online_shop_as[ 1 ]-purchasereqn =  ls_pr_key-purchaserequisition
.

      INSERT zshop_as_xxx FROM TABLE @lt_online_shop_as.
    ENDIF.
  ENDMETHOD.
ENDCLASS.

CLASS lhc_zbp_r_online_shop_xxx  DEFINITION INHERITING FROM cl_abap_behav-
ior_handler.
  PRIVATE SECTION.

    METHODS get_instance_authorizations FOR INSTANCE AUTHORIZATION
      IMPORTING keys REQUEST requested_authorizations FOR  online_shop RE-
SULT result.

    METHODS create_pr FOR MODIFY
      IMPORTING keys FOR ACTION online_shop~create_pr.

    METHODS update_inforecord FOR MODIFY
      IMPORTING keys FOR ACTION online_shop~update_inforecord.

    METHODS calculate_order_id FOR DETERMINE ON MODIFY
      IMPORTING keys FOR online_shop~calculate_order_id.
    METHODS set_inforecord FOR MODIFY
      IMPORTING keys FOR ACTION online_shop~set_inforecord.


ENDCLASS.

CLASS lhc_zbp_r_online_shop_xxx  IMPLEMENTATION.

  METHOD get_instance_authorizations.
  ENDMETHOD.

  METHOD create_pr.
```

Step 3: Check the Fiori elements preview to test the scenario

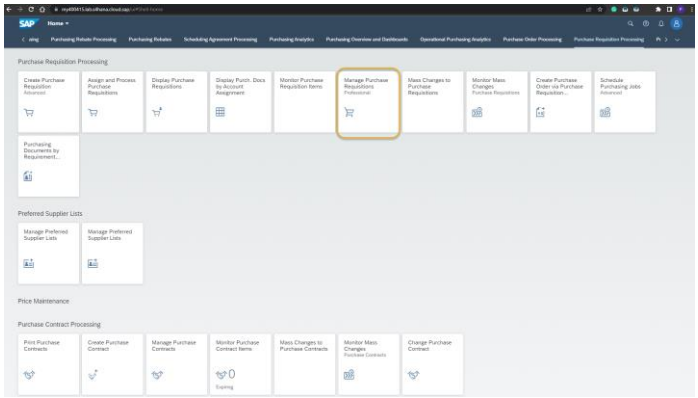
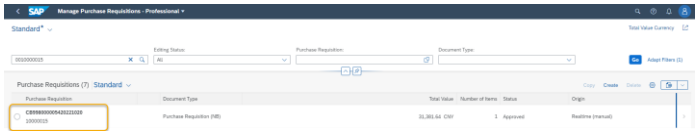
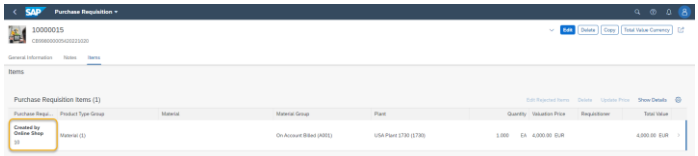
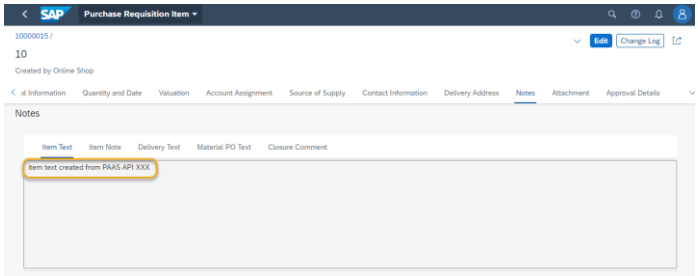
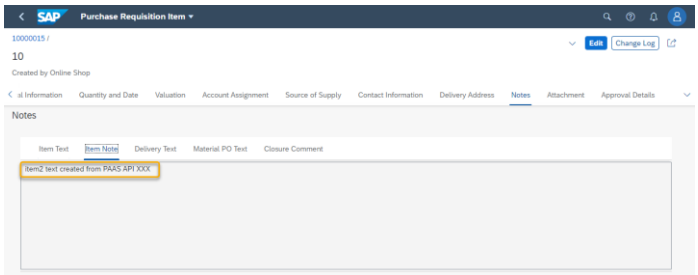
Description	Screenshot
Create service definition	
1. Right-click your projection view ZC_ONLINE_SHOP_XXX and select New Service Definition. 2. Enter the following Service Definition information: • Name: ZSD_SHOP_XXX • Description: Service definition for online shop Select in the following dialog your Transport Request and click on Finish.	

Description	Screenshot

Replace your service definition with the following code:
 	
Save and activate.
	`@EndUserText.label: 'Service definition for online shop'`
	`define service ZSD_SHOP_XXX {`
	`  expose ZC_ONLINE_SHOP_XXX as online_shop;`
	`}`

Create service binding	
1. Right-click your service definition ZSD_SHOP_XXX and select New Service Binding. 2. Enter the following Service Binding information: • Name: ZSB_SHOP_XXX • Description: OData V2 service for online shop • Binding Type: OData V2 – UI • Service Definition: ZSD_SHOP_XXX Select in the following dialog your Transport Request and click on Finish. Save and activate.	
Click Publish to publish your service binding.	
Select online_shop in your service binding and click Preview to open SAP Fiori Elements preview.	
A preview of your new app will open in a browser. Test your app by creating a new entry.	

Step 4: Open the standard Manage Purchase Requisitions (Professional) app to check your purchase requisition

Description	Screenshot
Access Fiori Launchpad	
1. In the Project Explorer, select your system and right click on Properties. 2. Select ABAP Development and copy the system URL without “-api”, paste it in a browser and log in. You should see the URL from the SAP S/4HANA Cloud section in your cheat sheet.	
Access your Purchase Requisiton	
1. Select the Manage Purchase Requisitions tile. 2. In the search field, enter your Purchase Requisition ID (you can find it with the Fiori preview of your business service) and click Go. 3. Select your purchase requisition.	
Check your purchase requisition item and select it. The description “Created by Online Shop” should appear here.	
Select Notes and check your item text. Your ID should appear here instead of XXX.	
Check your item note. Your ID should appear here instead of XXX.	

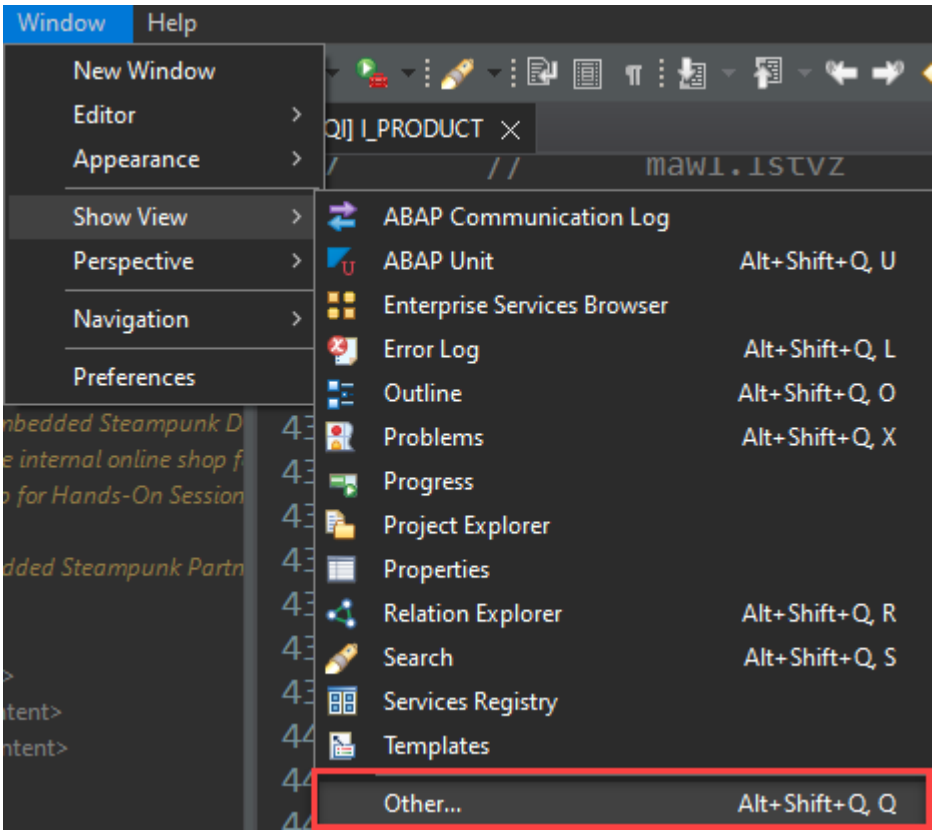
Step 5: Not covered here: Outlook on how to develop and deploy Fiori Elements based App

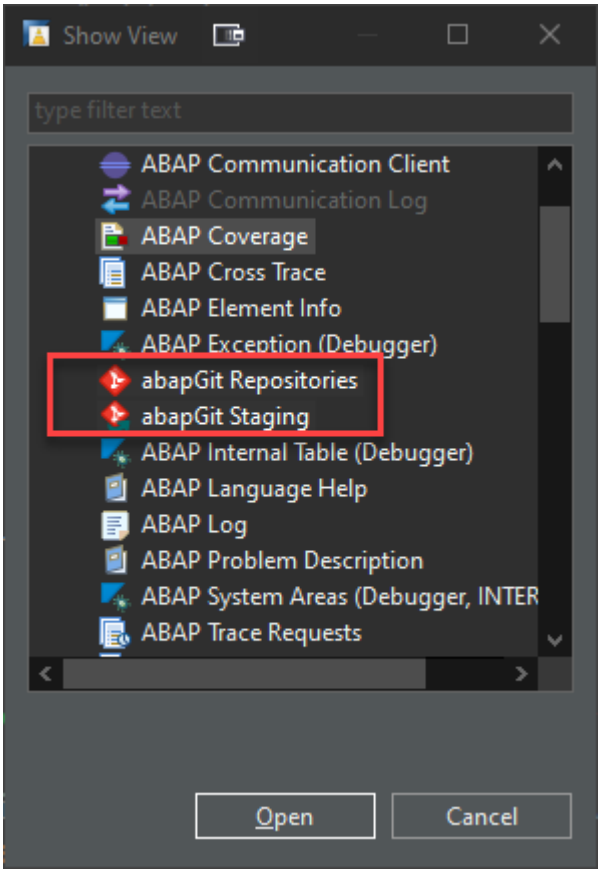
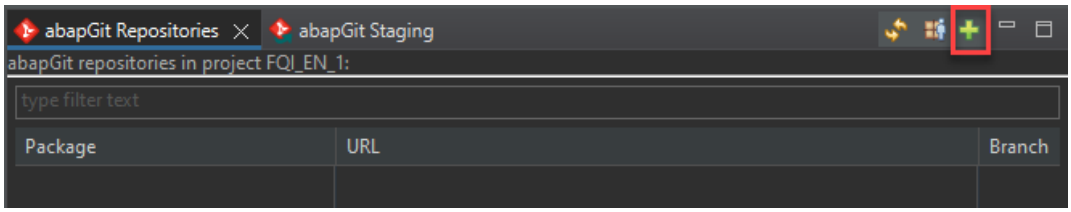
This step will not be described in detail in this tutorial. It is needed in reality to create a SAP Fiori App with Fiori Elements in order to use the scenario in a system.

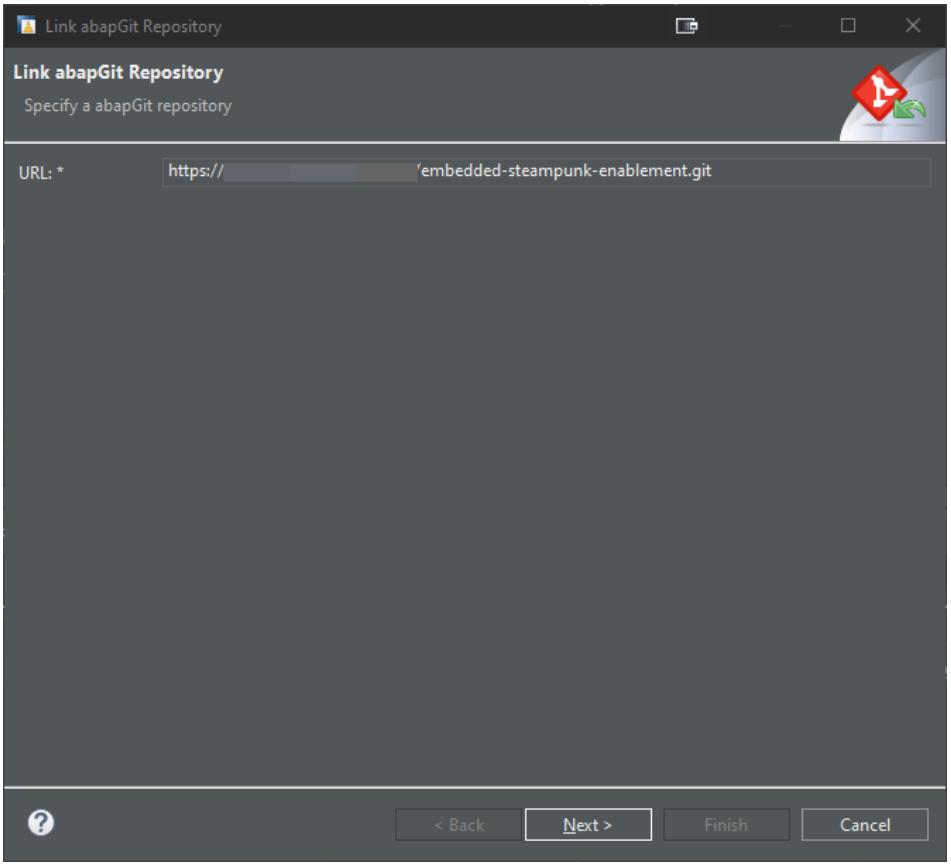
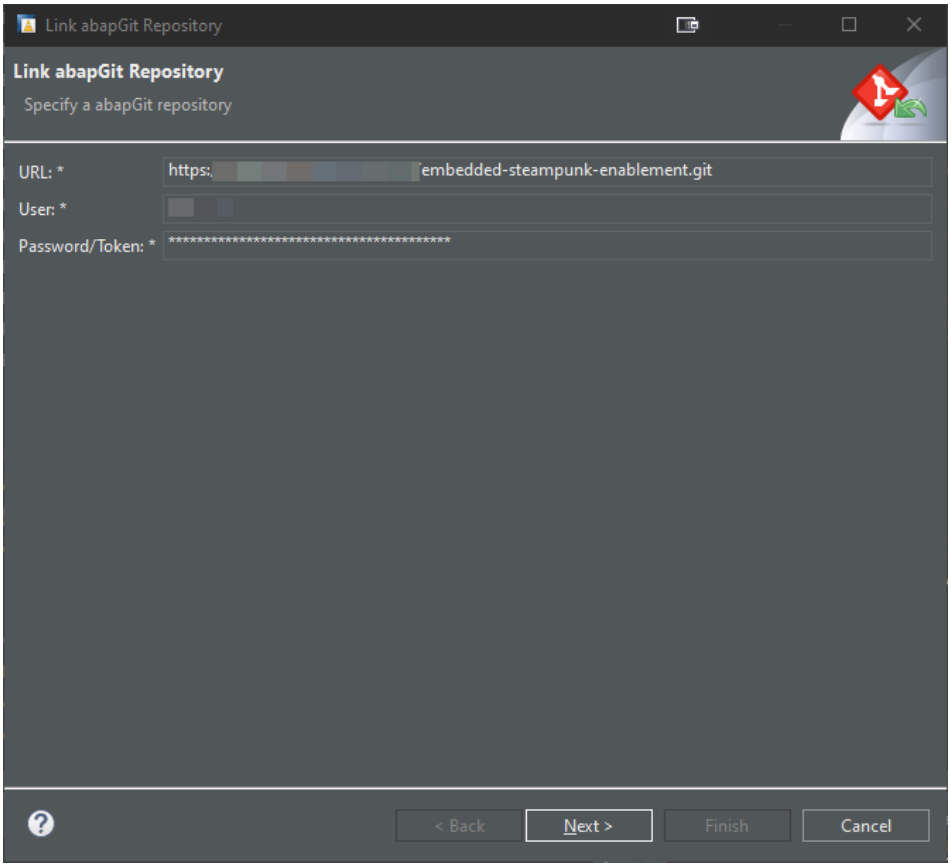
In case you are interested, e.g. to setup this scenario your own development system, all steps needed are described in this tutorial:
Create a SAP Fiori App and Deploy it to SAP S/4HANA Cloud, ABAP Environment

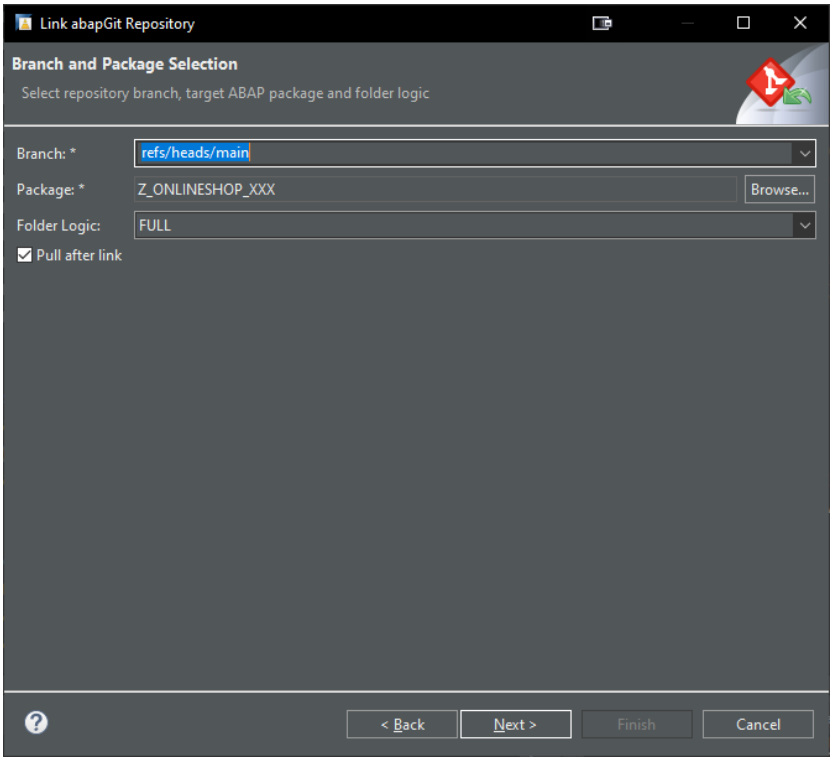
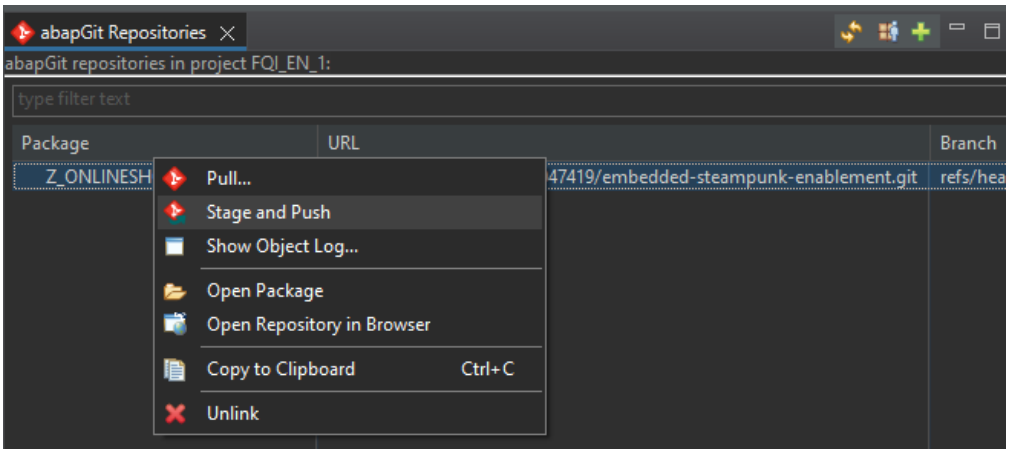
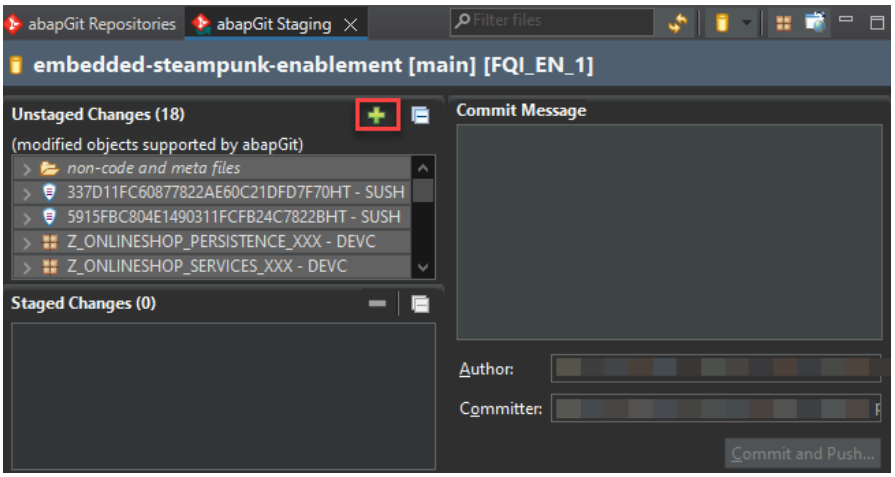
Step 6: Export Coding using abapGit (optional)
You can now export your coding using abapGit. This might be useful to a) get to know abapGit and b) have the example coding stored for later exploration.

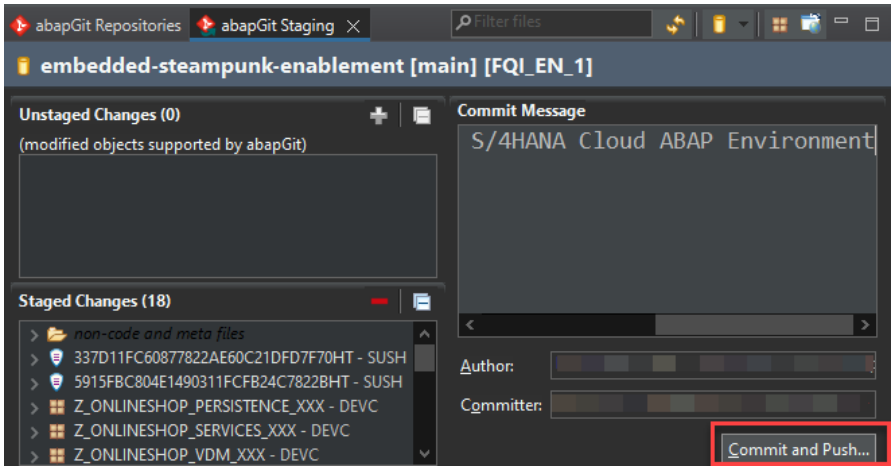
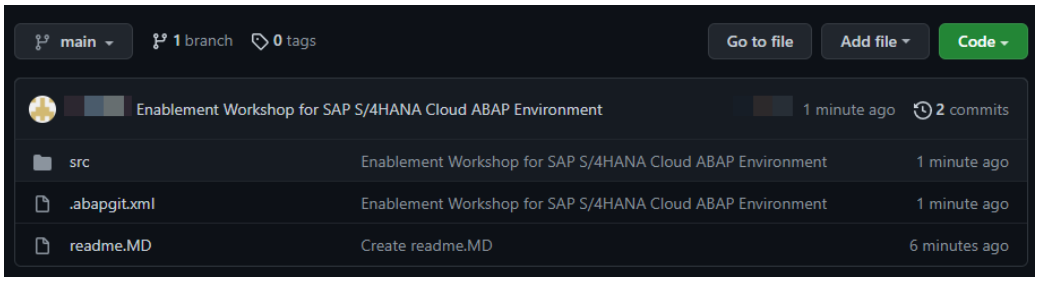
Prerequisites:
- You have a git repository prepared and available
- You have abapGit integrated in your ABAP Development Tools installation

Description	Screenshot
If needed: Add abapGit-View to ADT Choose Window -> Show View -> Other	

Description	Screenshot
Choose abapGit Staging and abapGit Repositories	
Add your Repository in view “abapGit Repositories”	

Description	Screenshot
Enter the link of your repository in the dialog	
Enter valid authentication mechanism, e.g. user & password or user & token	

Description	Screenshot
Choose your package (Z_ONLINESHOP_XXX) Finish the dialog	
Right click your newly mapped package / repository in abapGit Repositories and choose “stage and push”	
Stage your changes by clicking the green +- button while marking the relevant objects.	

Description	Screenshot
Commit your changes with “commit and push” after adding a commit message	
Now you can check your repository (e.g. in browser), it contains the package you just committed.	

PART 2: TEST WITH SIMPLE ABAP CLASS (OPTIONAL)

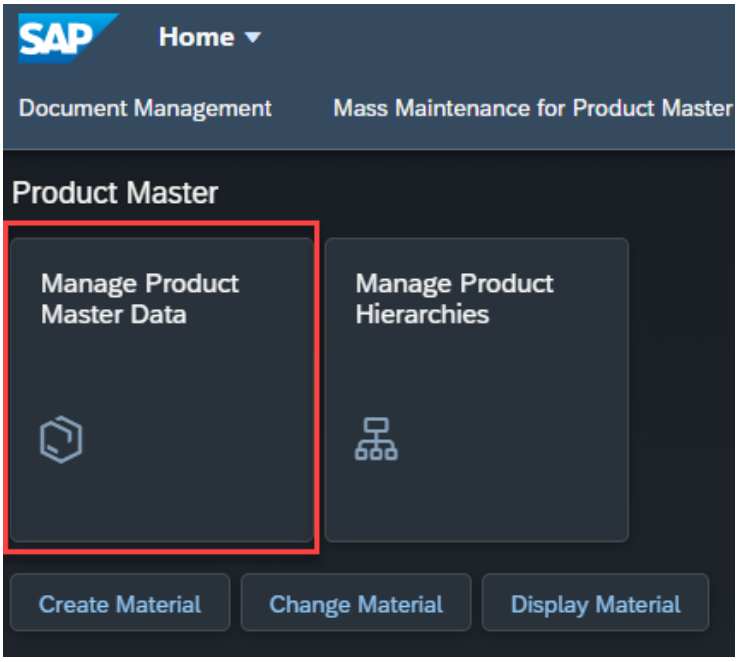
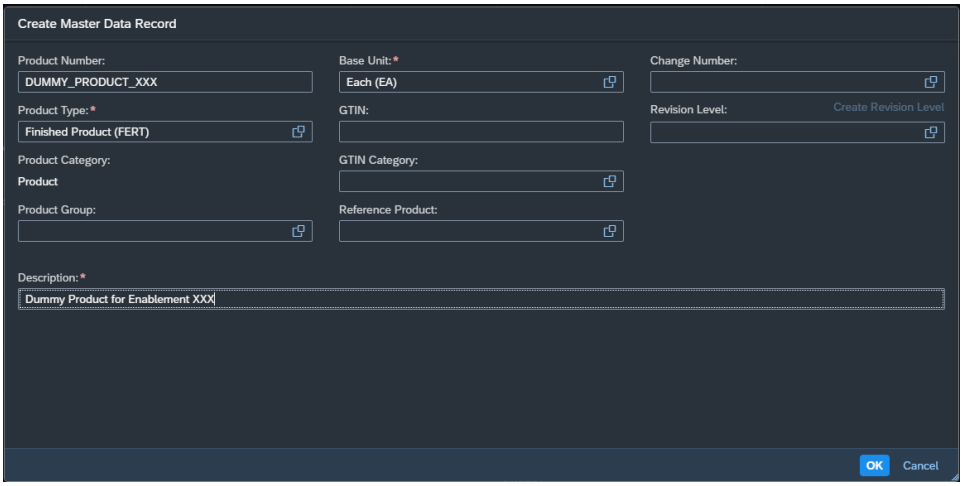
Business Scenario

This optional part is to demonstrate how to develop the basics in the S/4HANA Cloud ABAP Environment. It is the idea on the system as a sandbox to identify the different extension points and access possibilities to SAP Standard objects

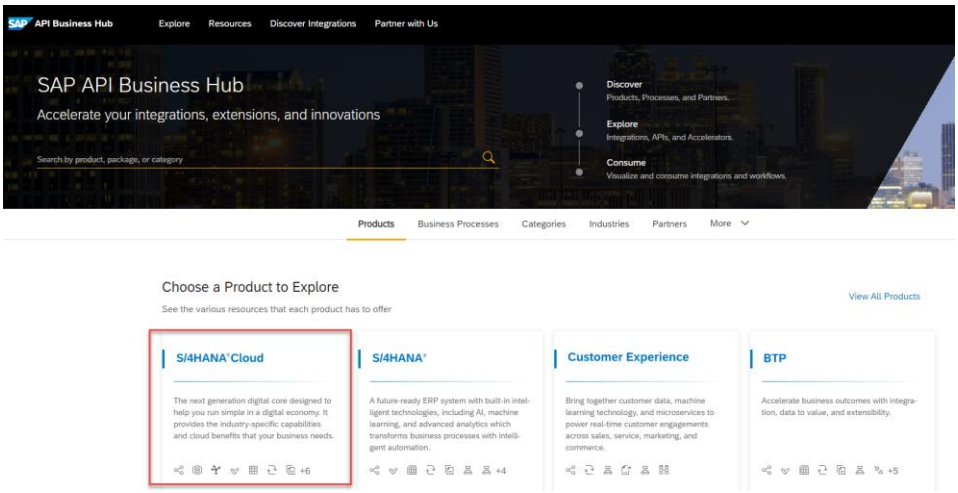
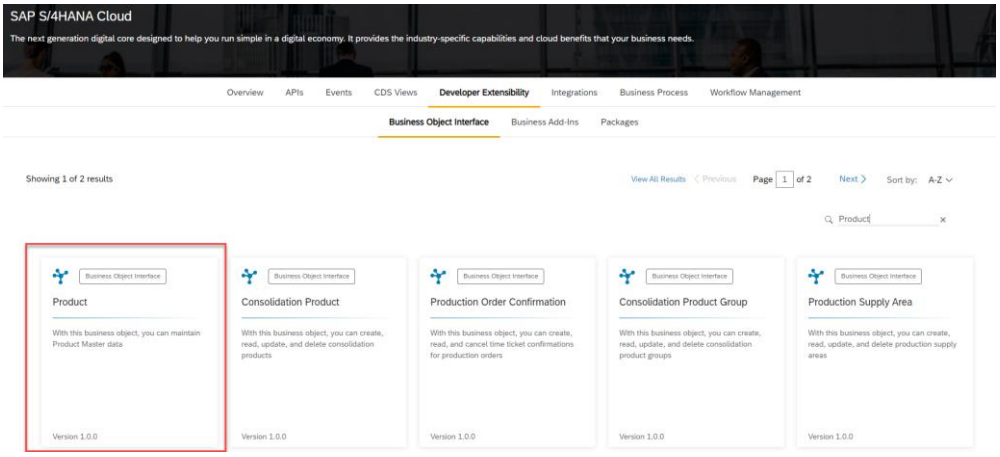
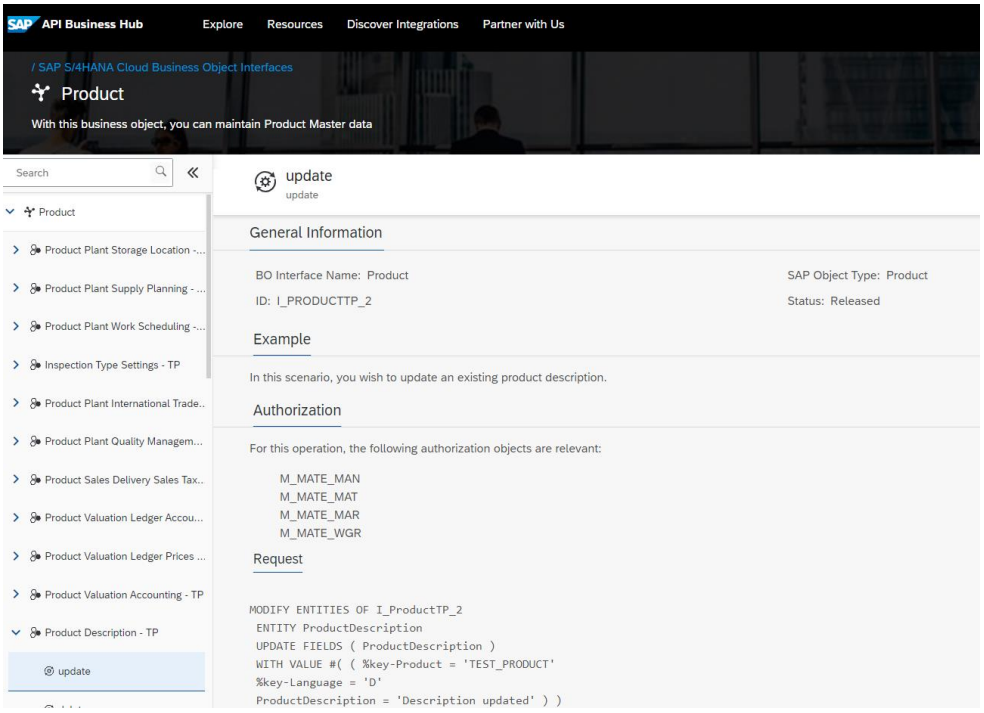
The idea is to simply check a product master record, change it via API and check the success afterwards in the system again.

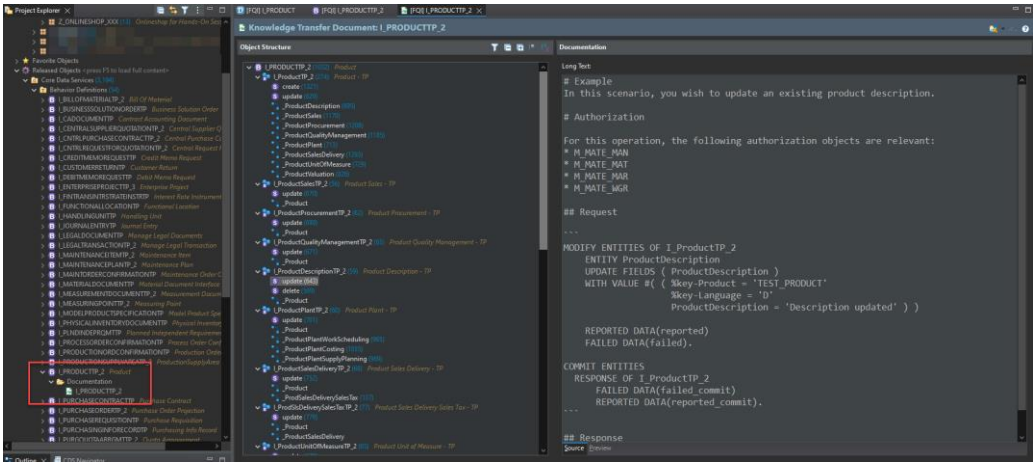
The scenario that is described here is to create a simple product “DUMMY_PRODUCT_XXX” and to change the description.

Step 0: Create/Identify Product Master data in the system

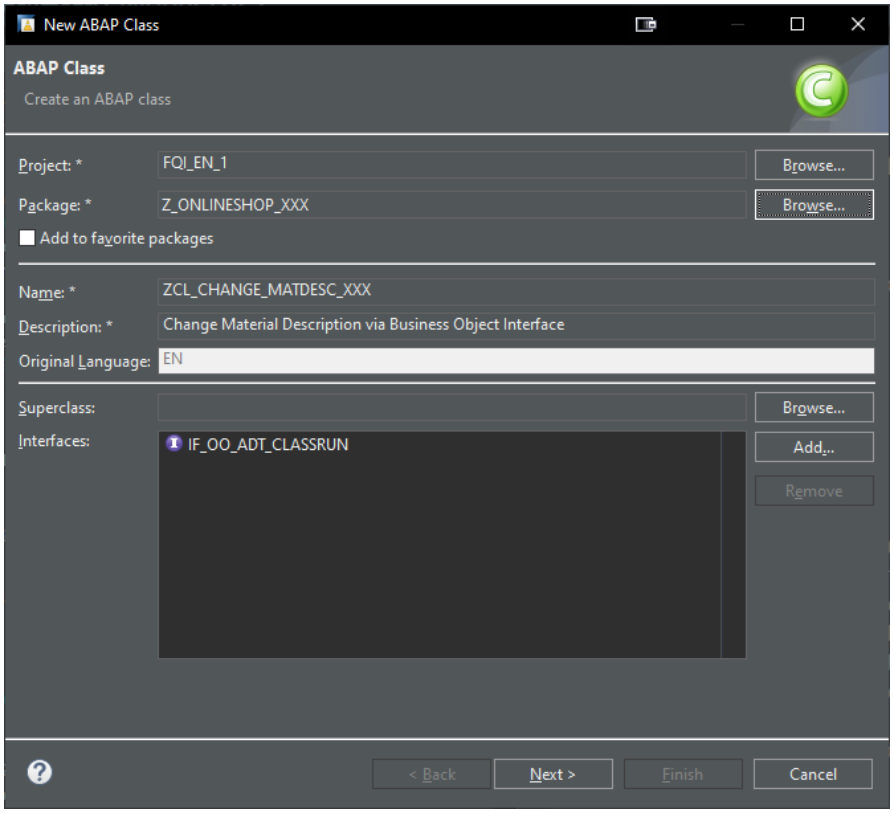
Description	Screenshot
Open the SAP Fiori Launchpad and open the “Manage Product Master Data” Application	
Either choose an existing product or create a new product, e.g. with the data mentioned in the screenshot.	

Step 1: Identify Business Object Interface ("RAP Façade") for Product master in SAP API Business Hub

Description	Screenshot
As we want to manipulate "Product Master Data", we check on SAP API Business Hub for an interface. Navigate in your browser to SAP API Business Hub (https://api.sap.com/) Choose S/4HANA Cloud as Product to explore	
Choose "Developer Extensibility" and "Business Object Interface" and look for the Product API.	
Look into the details to create the Product details and check the exemplary coding.	

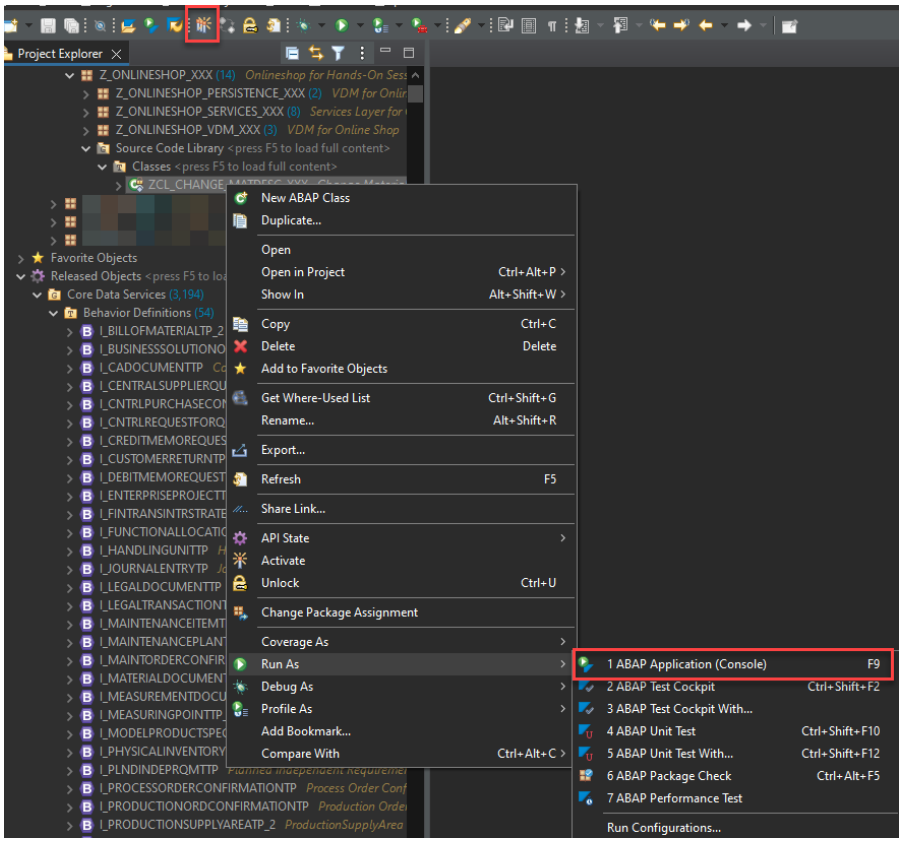
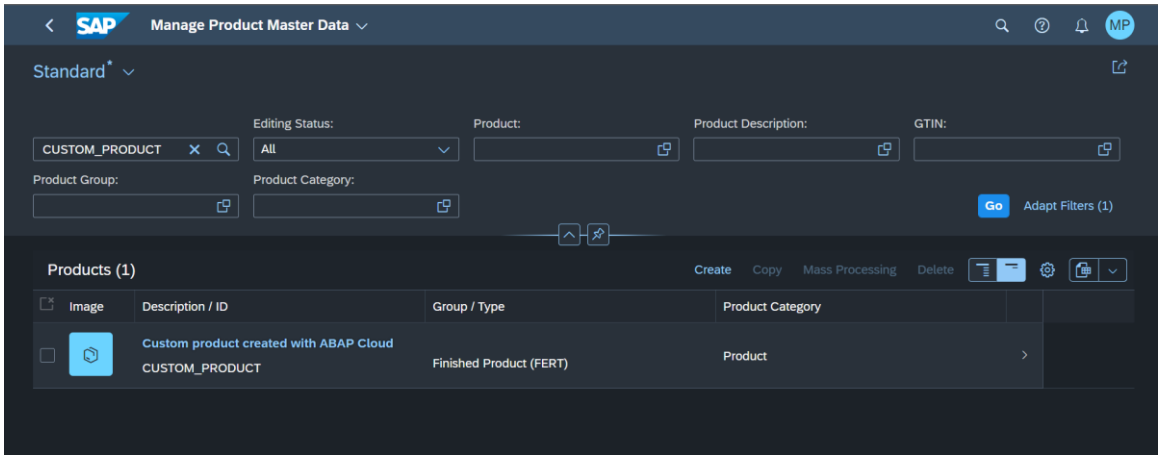
Description	Screenshot
Alternatively, you can also check the interface within ABAP Development Tools, by looking in “Released Objects” for the behavior definition of “I_PRODUCTTP_2” (as described in API Business Hub).	

Step 2: Create runnable ABAP Class

Description	Screenshot
In ABAP Development Tools, right click on your Package and choose “New ABAP Class” and create a class implementing the interface “IF_OO_ADT_CLASSRUN” Create the class and assign it to your transport request / or create a new one.	

Step 3: Utilize the Product Master Interface to create a new product from standard objects

Now implement the logic to create a new product in the “main” method of your newly created class.	CLASS zcl_change_matdesc_xxx DEFINITION PUBLIC FINAL CREATE PUBLIC . PUBLIC SECTION. INTERFACES if_oo_adt_classrun . PROTECTED SECTION. PRIVATE SECTION. ENDCLASS. CLASS zcl_change_matdesc_xxx IMPLEMENTATION. METHOD if_oo_adt_classrun~main. DATA product_detail TYPE TABLE FOR CREATE I_ProductTP_2. product_detail = VALUE #((%cid = 'product1' Product = 'CUSTOM_PRODUCT' %control-Product = if_abap_behv=>mk-on ProductType = 'FERT' %control-ProductType = if_abap_behv=>mk-on BaseUnit = 'EA' %control-BaseUnit = if_abap_behv=>mk-on IndustrySector = 'M' %control-IndustrySector = if_abap_behv=>mk-on)). MODIFY ENTITIES OF I_ProductTP_2 ENTITY Product

	Note: Double-click to enable copy
Activate your class and run the coding.	
Check in Manage Product Master Data app the changes done by the coding.	

Step 4: Discover further APIs to use in "Sandbox"

This task is just used as example, based on this example and the tutorials available, there are some more use cases to be developed in SAP S/4HANA Cloud ABAP Environment. You could extend a process by implementing a Business Add-in (BAdI) or create / manipulate standard objects based on the APIs shown in this workshop.

The list of relevant tutorials can be found here.