

Develop SAP Business One extensions on the SAP Cloud Platform



TABLE OF CONTENTS

PREREQUISITES	4
i. Download and Install Development Tools	4
ii. Create a SAP Cloud platform Neo trial account	5
iii. Activate Web IDE Full Stack service.....	6
iv. Activate a Cloud Foundry trial account.....	9
STEP 1: CREATE A BUILD HIGH FIDELITY PROTOTYPE	10
STEP 2: IMPORT YOUR BUILD PROTOTYPE INTO WEBIDE	10
i. Create the Project.....	10
ii. Test the project with mock data.....	13
iii. Create a destination pointing to your backend server	14
iv. Connect to your real B1 backend server.....	14
v. Extra SAP Business One backend configuration steps	16
STEP 3: CLONE A NODEJS APP	21
STEP 4: DEPLOY THE NODEJS APP INTO SAP CLOUD FOUNDRY	22
i. SAP Cloud Platform Cloud Foundry Environment.....	22
ii. Create the backing services	23
iii. Deploy the smbmkmt app	24
iv. Configure the SMB Mktplace backend	25
v. Initialize the SMB Mktplace backend	26
vi. Test the SMB Mktplace backend /SimilarItems API	26
STEP 5: CONSUME THE NODEJS APP FROM THE SAP FIORI APP	27
i. Create a destination pointing to your smbmkmt backend	27
ii. Create a new JSON model	28
iii. Change the Image control in the Page1.view.xml file.....	28
iv. Create a FileUploader control.....	29
v. Bind the Matching Items Table to our backend properties	29
vi. Implement the Page1.controller.js	32
STEP 6: ADD A NEW SERVICE TO THE NODEJS APPLICATION	35
STEP 7: CALL THE NEW NODEJS SERVICE FROM YOUR SAP FIORI APP	38



The objective of this hands on is to put in practice how to develop SAP Business One extensions on SAP Cloud Platform.

The exercise will be composed by

- Step 1: Create a Build prototype connecting to B1
- Step 2: Import the Build prototype into a SCP WebIDE Fiori application and connect to your real B1 backend
- Step 3: Clone an existing NodeJS application
- Step 4: Deploy the server side NodeJS application to SAP Cloud Foundry
- Step 5: Modify the SAP Fiori app to consume the server side NodeJS application
- Step 6: Add a new service to the NodeJS application and consume it from SAP Fiori
- Step 7: Call the new NodeJS service from the SAP Fiori app

This hands-on exercise will require several steps, please follow them in the proposed order as each step is counting on the precedent steps.

PREREQUISITES

i. Download and Install Development Tools

Download and install git version control on your system from the following link

<https://git-scm.com/downloads>



We will also make use of SAP Cloud Platform Cloud Foundry Environment.

To do so, we need the Cloud Foundry command line interface (CLI)

You can download it and install if the CF CLI for your operating system on.

<https://github.com/cloudfoundry/cli#downloads>

Downloads

Installing using a package manager

Mac OS X and Linux using [Homebrew](#) via the [cloudfoundry tap](#):

```
brew install cloudfoundry/tap/cf-cli
```

Debian and Ubuntu based Linux distributions:

```
# ...first add the Cloud Foundry Foundation public key and package repository to your system
wget -q -O - https://packages.cloudfoundry.org/debian/cli.cloudfoundry.org.key | sudo apt-key add -
echo "deb https://packages.cloudfoundry.org/debian stable main" | sudo tee /etc/apt/sources.list.d/cloudfo
# ...then, update your local package index, then finally install the cf CLI
sudo apt-get update
sudo apt-get install cf-cli
```

Enterprise Linux and Fedora systems (RHEL6/CentOS6 and up):

```
# ...first configure the Cloud Foundry Foundation package repository
sudo wget -O /etc/yum.repos.d/cloudfoundry-cli.repo https://packages.cloudfoundry.org/fedora/cloudfoundry-c
# ...then, install the cf CLI (which will also download and add the public key to your system)
sudo yum install cf-cli
```

Installers and compressed binaries

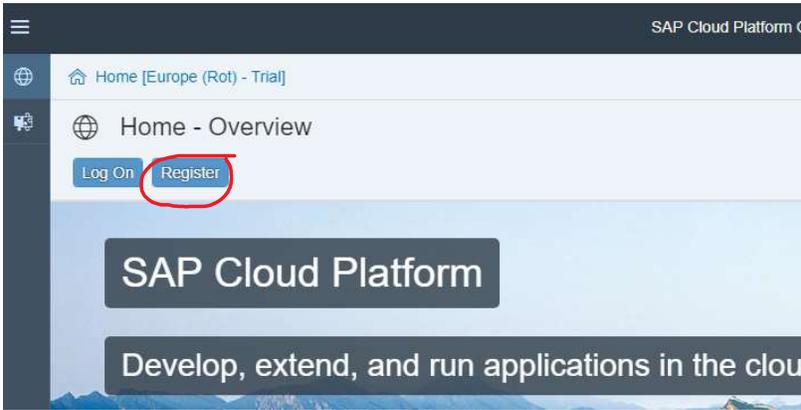
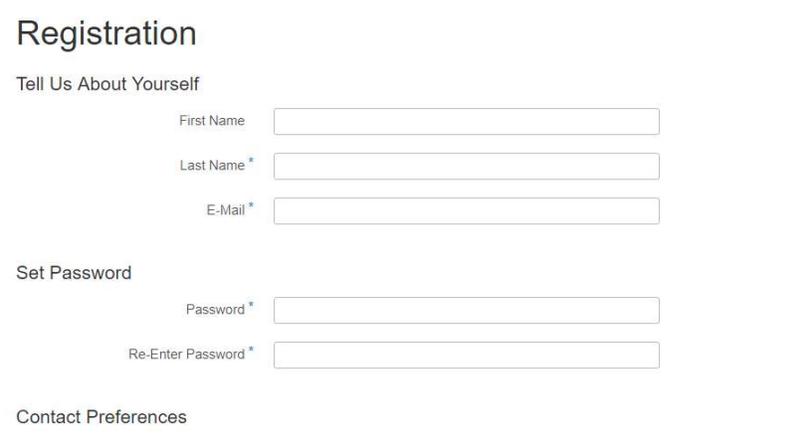
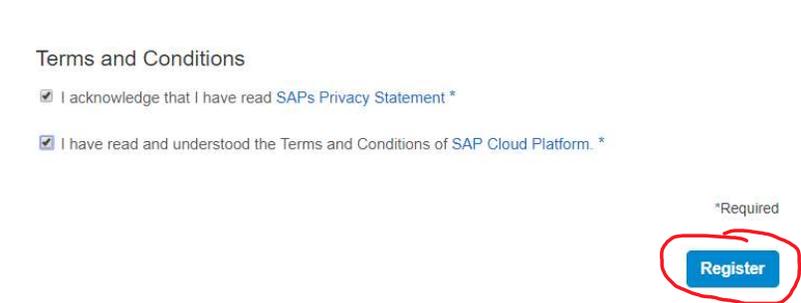
	Mac OS X 64 bit	Windows 64 bit	Linux 64 bit
Installers	pkg	zip	rpm / deb
Binaries	tgz	zip	tgz

ii. Create a SAP Cloud platform Neo trial account

The exercises proposed in this hands on are implemented on top of the SAP Cloud Platform.

If you have already a trial SAP Cloud Platform account, you can skip this step.

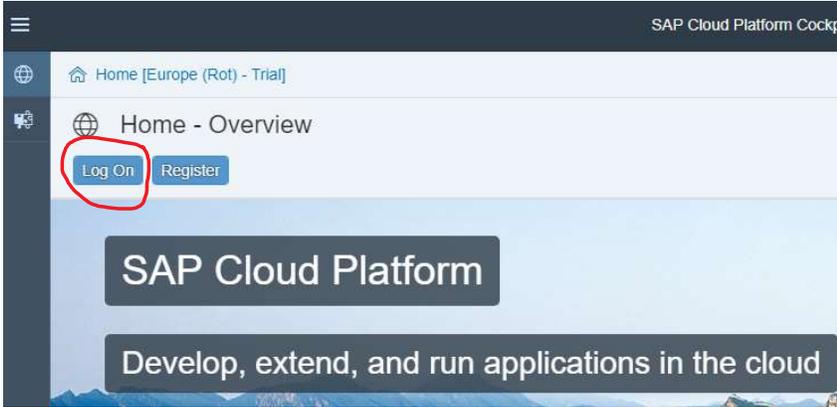
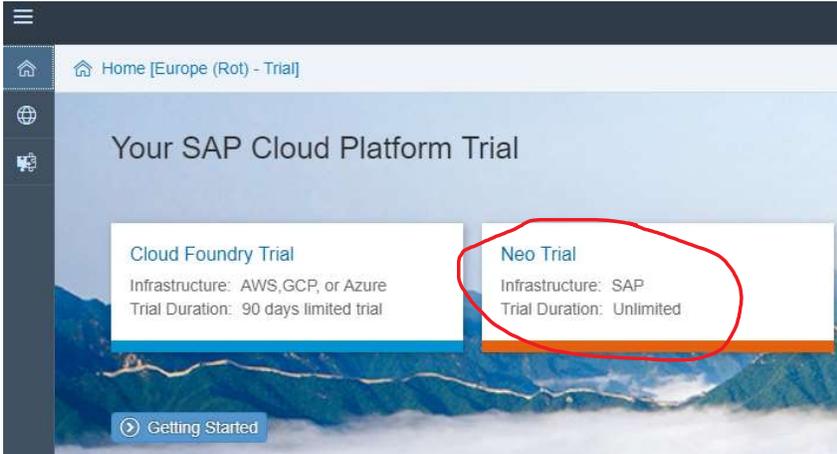
To create a trial SAP Cloud Platform account, go to the following link:

Explanation	Screenshot
<p>To create a trial SAP Cloud Platform account, go to the following link: https://account.hanatrial.ondemand.com</p> <p>Press the Register button</p>	 <p>The screenshot shows the SAP Cloud Platform Home - Overview page. At the top, there is a navigation bar with a home icon and the text 'Home [Europe (Rot) - Trial]'. Below this, there is a section titled 'Home - Overview' with two buttons: 'Log On' and 'Register'. The 'Register' button is circled in red. Below the buttons, there is a large banner with the text 'SAP Cloud Platform' and 'Develop, extend, and run applications in the cloud'.</p>
<p>Enter all your details</p>	 <p>The screenshot shows the SAP Cloud Platform Registration form. The form is titled 'Registration' and has three sections: 'Tell Us About Yourself', 'Set Password', and 'Contact Preferences'. The 'Tell Us About Yourself' section has three input fields: 'First Name', 'Last Name *', and 'E-Mail *'. The 'Set Password' section has two input fields: 'Password *' and 'Re-Enter Password *'. The 'Contact Preferences' section is empty. At the bottom right, there is a blue 'Register' button circled in red.</p>
<p>Accept the terms and conditions by checking both check boxes and press "Register".</p>	 <p>The screenshot shows the SAP Cloud Platform Terms and Conditions section. It has two checkboxes: 'I acknowledge that I have read SAPs Privacy Statement *' and 'I have read and understood the Terms and Conditions of SAP Cloud Platform. *'. Both checkboxes are checked. At the bottom right, there is a blue 'Register' button circled in red.</p>

iii. Activate Web IDE Full Stack service

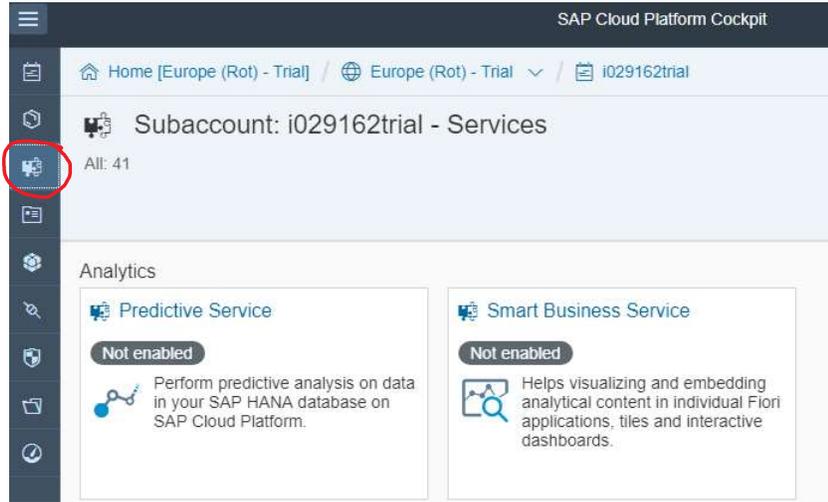
We will use Web IDE Full Stack for the creation and implementation of our application. Web IDE is offered as a service on the SAP Cloud Platform.

To activate Web IDE Full Stack service please follow the steps here below, if you already have Web IDE Full Stack service active in your account please skip this step.

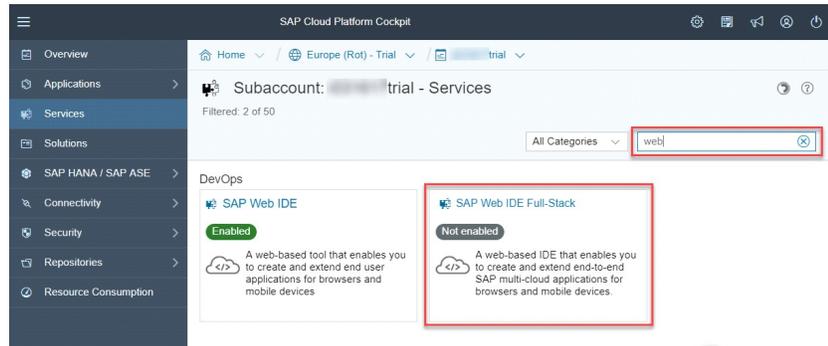
Explanation	Screenshot
<p>Open your trial SAP Cloud Platform account from the following link: https://account.hanatrial.ondemand.com</p>	
<p>Press the Log On button if you are not automatically logged in</p>	 A screenshot of the SAP Cloud Platform Home - Overview page. The page has a dark blue header with a hamburger menu icon on the left and 'SAP Cloud Platform Cockp' on the right. Below the header, there's a navigation bar with a home icon and 'Home [Europe (Rot) - Trial]'. The main content area has a light blue background with a globe icon and 'Home - Overview'. Below this, there are two buttons: 'Log On' and 'Register'. The 'Log On' button is circled in red. Further down, there's a large dark blue box with 'SAP Cloud Platform' in white, and another dark blue box with 'Develop, extend, and run applications in the cloud' in white.
<p>After login if you are proposed between Cloud Foundry Trial and Neo Trial please choose Neo Trial.</p>	 A screenshot of the 'Your SAP Cloud Platform Trial' page. The page has a dark blue header with a hamburger menu icon on the left and 'Home [Europe (Rot) - Trial]' on the right. Below the header, there's a navigation bar with a home icon and 'Home [Europe (Rot) - Trial]'. The main content area has a light blue background with a globe icon and 'Your SAP Cloud Platform Trial'. Below this, there are two trial options: 'Cloud Foundry Trial' and 'Neo Trial'. The 'Neo Trial' option is circled in red. The 'Cloud Foundry Trial' option shows 'Infrastructure: AWS,GCP, or Azure' and 'Trial Duration: 90 days limited trial'. The 'Neo Trial' option shows 'Infrastructure: SAP' and 'Trial Duration: Unlimited'. At the bottom, there's a blue button with a right arrow and 'Getting Started'.

Explanation	Screenshot
-------------	------------

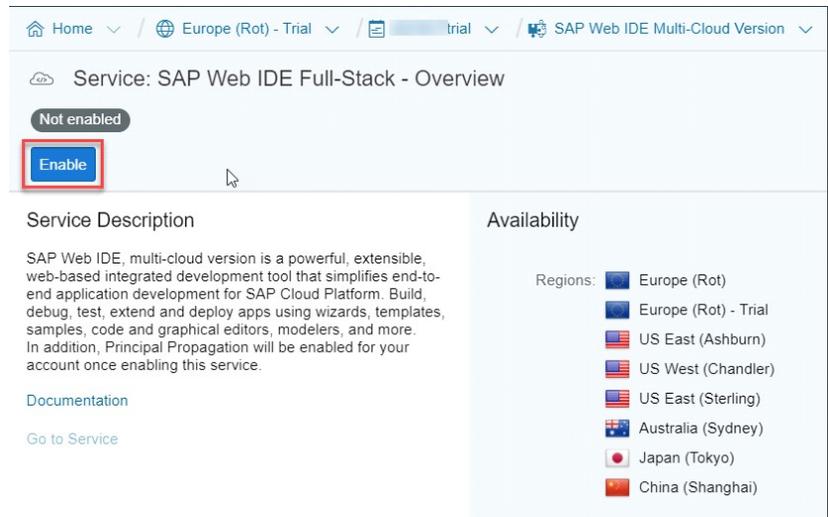
Select the Services icon on the left side bar.

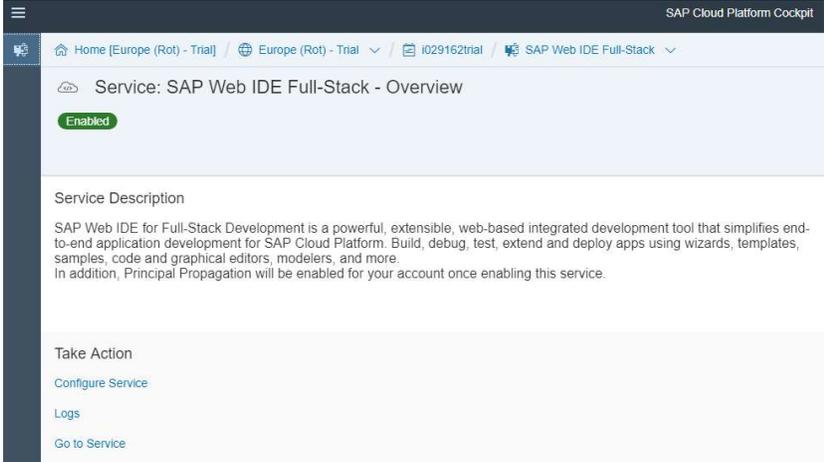
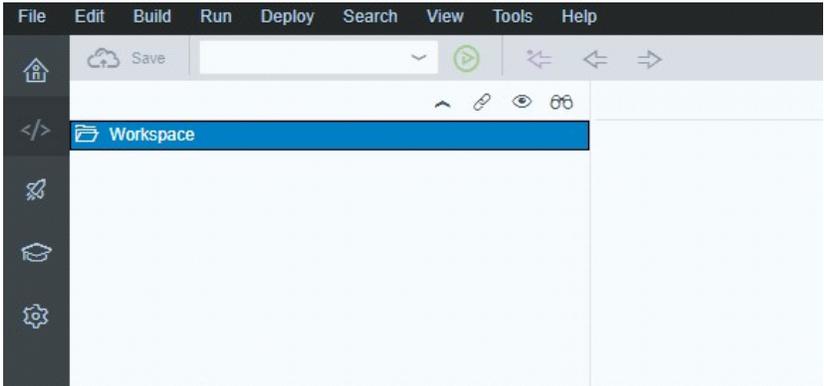


Enter **Web** in the search edit text.
Click on the SAP Web IDE Full Stack box.

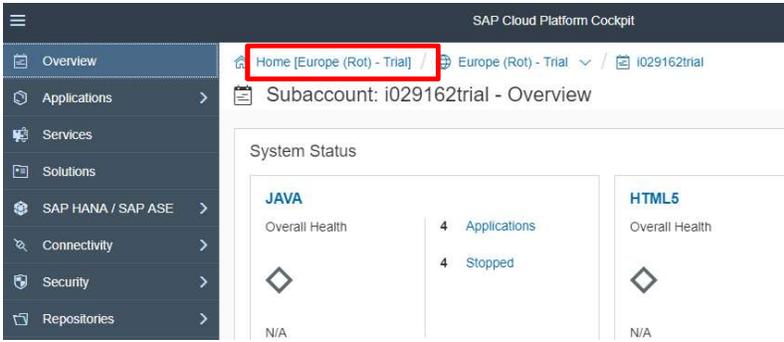
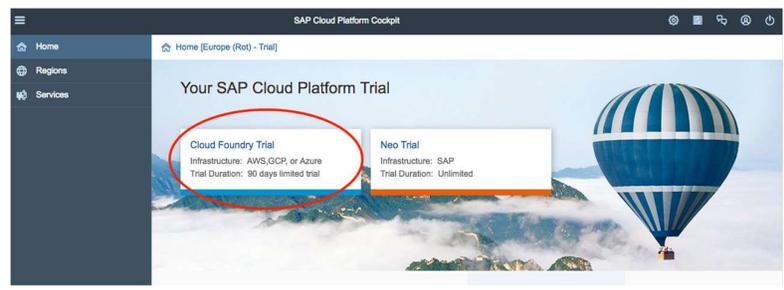
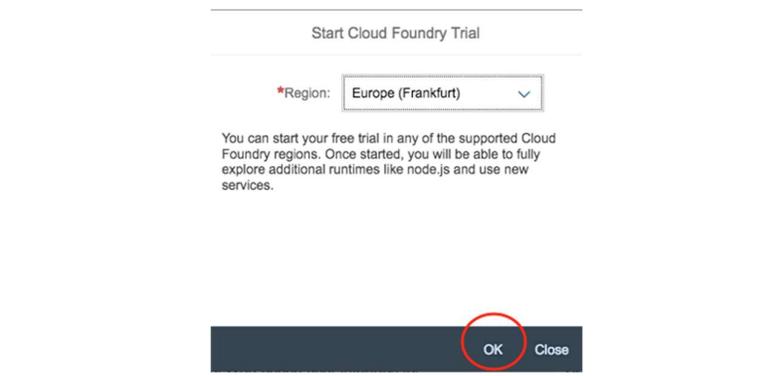
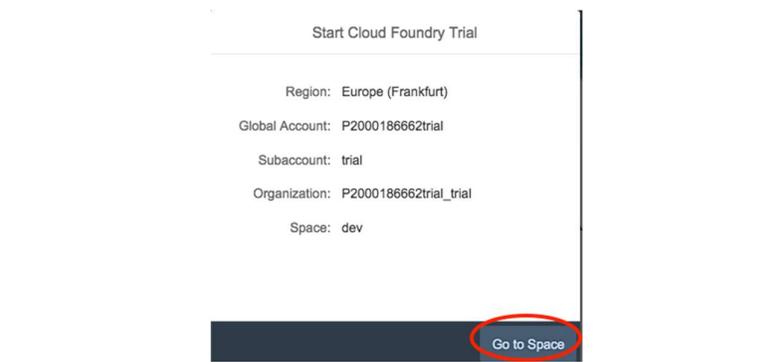


Click **Enable**.
This may take a few minutes.



Explanation	Screenshot
<p>Once Enabled select the link Go to Service to open Web IDE Full Stack.</p>	 <p>The screenshot shows the SAP Cloud Platform Cockpit interface. At the top, it says 'SAP Cloud Platform Cockpit'. Below that, there's a breadcrumb trail: 'Home [Europe (Rot) - Trial] / Europe (Rot) - Trial / i029162trial / SAP Web IDE Full-Stack'. The main heading is 'Service: SAP Web IDE Full-Stack - Overview' with a green 'Enabled' button. Under 'Service Description', it states: 'SAP Web IDE for Full-Stack Development is a powerful, extensible, web-based integrated development tool that simplifies end-to-end application development for SAP Cloud Platform. Build, debug, test, extend and deploy apps using wizards, templates, samples, code and graphical editors, modelers, and more. In addition, Principal Propagation will be enabled for your account once enabling this service.' Under 'Take Action', there are links for 'Configure Service', 'Logs', and 'Go to Service'.</p>
<p>Web IDE opens with an empty Workspace unless you already developed applications with Web IDE in the past.</p>	 <p>The screenshot shows the SAP Web IDE interface. It has a menu bar with 'File', 'Edit', 'Build', 'Run', 'Deploy', 'Search', 'View', 'Tools', and 'Help'. Below the menu is a toolbar with a 'Save' button, a search icon, and navigation arrows. The main workspace is empty. On the left, there's a sidebar with icons for home, code editor, and settings. A 'Workspace' folder is highlighted in the sidebar.</p>

iv. **Activate a Cloud Foundry trial account**

Explanation	Screenshot
<p>We need to activate the SAP Cloud Platform Cloud Foundry Environment.</p> <p>On the SAP Cloud Platform Dashboard click on the HOME option.</p>	
<p>First, we need to activate the SAP Cloud Platform Cloud Foundry Environment.</p> <p>From the SAP CP Home Screen, click on Cloud Foundry Trial.</p>	
<p>Select the Trial Region that most suits you. And Click on OK</p>	
<p>This will initialize your Cloud Foundry Trial and create a DEV space (where the solutions will be deployed).</p>	

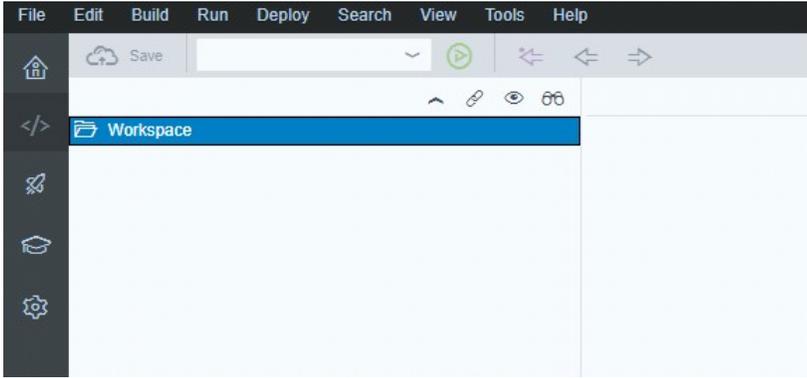
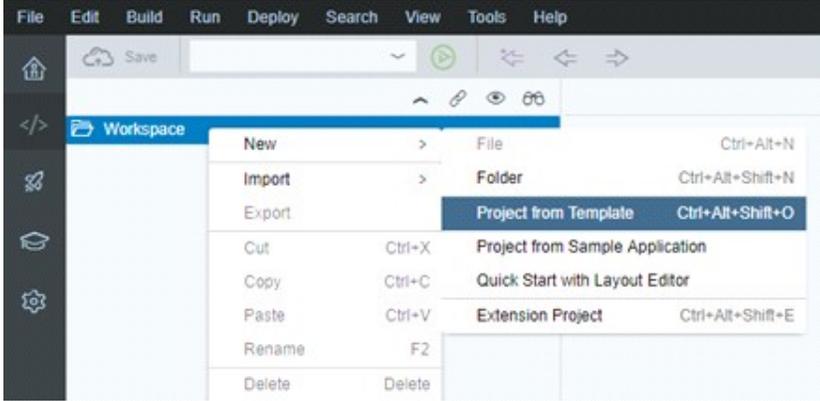
STEP 1: CREATE A BUILD HIGH FIDELITY PROTOTYPE

Follow the steps of the HandsOn_Build_B1_Instructions.pdf document.

STEP 2: IMPORT YOUR BUILD PROTOTYPE INTO WEBIDE

The objective of this first exercise is to create a SAP Fiori app from your Build prototype.

i. Create the Project

Explanation	Screenshot
<p>Open SAP Web IDE Full Stack.</p> <p>Check the prerequisites sections “Create a SAP Cloud platform trial account” and “Activate Web IDE Full Stack service” if you don’t know how to open WebIDE Full Stack.</p>	 <p>The screenshot shows the SAP Web IDE Full Stack interface. The top menu bar includes File, Edit, Build, Run, Deploy, Search, View, Tools, and Help. Below the menu bar is a toolbar with icons for Save, Run, and navigation. The left sidebar contains icons for Home, Code, Run, and Settings. The main workspace area shows a folder named 'Workspace' selected in the left sidebar.</p>
<p>Right click on your Workspace and select New -> Project from Template.</p>	 <p>The screenshot shows the SAP Web IDE Full Stack interface with the context menu open for the 'Workspace' folder. The menu items are: New (Ctrl+Alt+N), Import (Ctrl+Alt+Shift+N), Export, Cut (Ctrl+X), Copy (Ctrl+C), Paste (Ctrl+V), Rename (F2), and Delete (Delete). The 'Project from Template' option is highlighted, with its keyboard shortcut Ctrl+Alt+Shift+O. Other options include 'Project from Sample Application', 'Quick Start with Layout Editor', and 'Extension Project' (Ctrl+Alt+Shift+E).</p>

Explanation	Screenshot
-------------	------------

Select the **BUILD Project** template.

Press **Next**.

If you don't see this template, change the Category to **BUILD Project** or **All categories**.

Enter a **Project Name**.

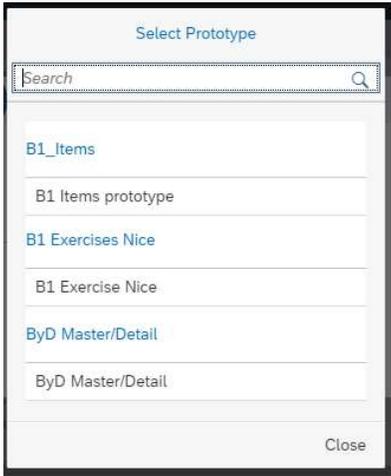
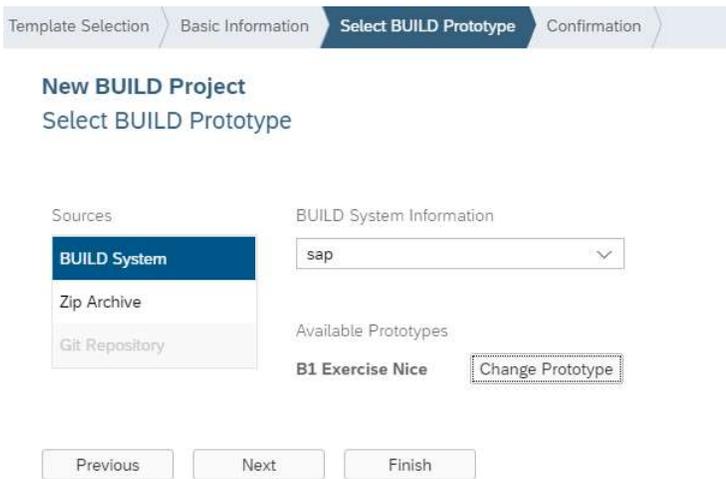
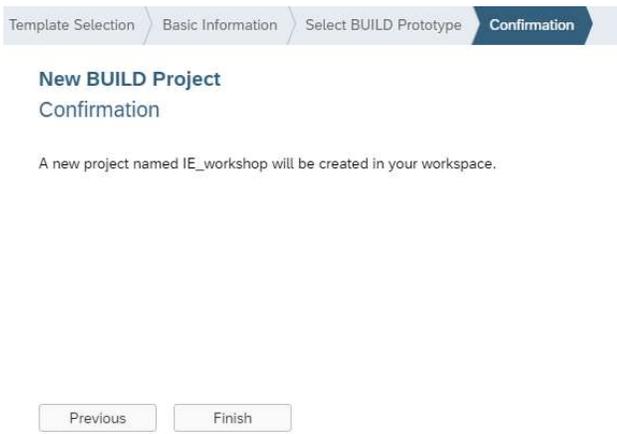
Press **Next**.

Select **BUILD System** source.

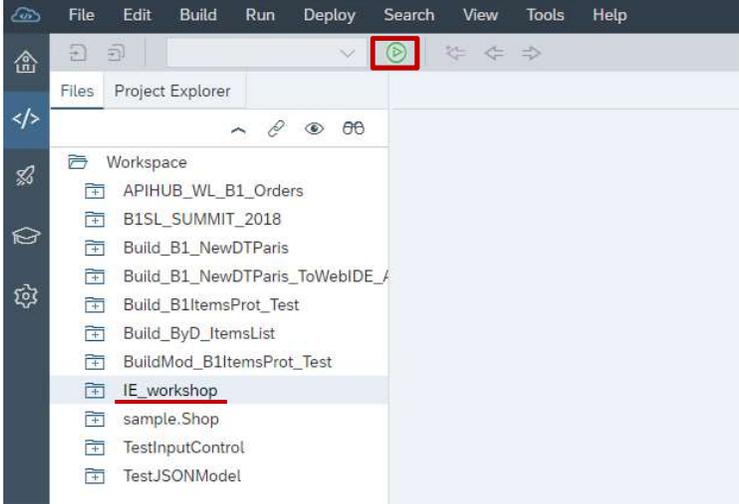
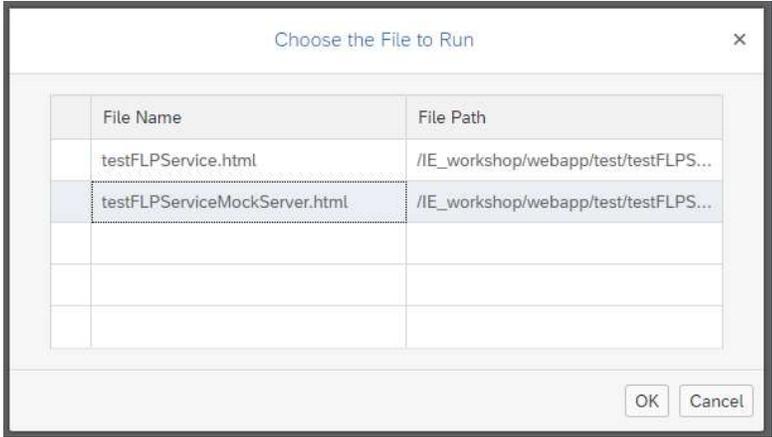
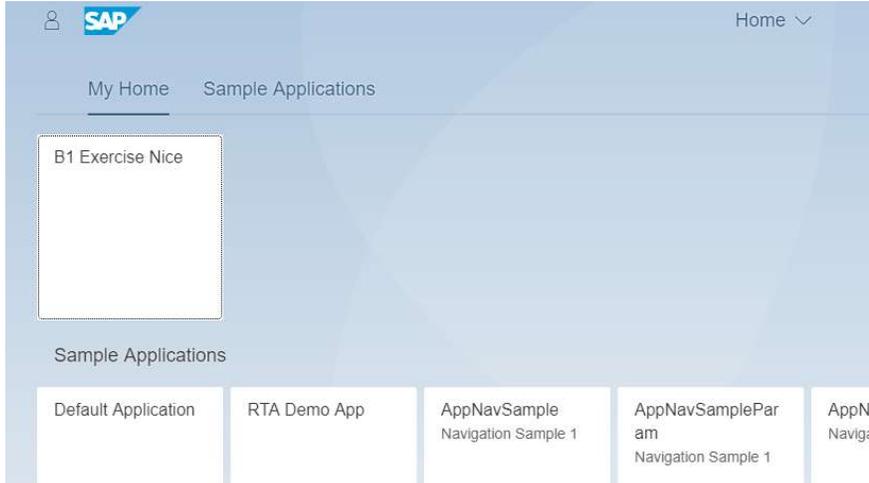
Select **standard** BUILD System Information.

Press **Select Prototype** to get the list of available BUILD prototypes.

You might be prompted to enter your SAP Community User Name and Password. Enter your credentials and press Log In.

Explanation	Screenshot
<p>Select the prototype you shared previously in BUILD.</p> <p>Press Close.</p>	
<p>Press Next.</p>	
<p>Press Finish.</p>	

ii. Test the project with mock data

Explanation	Screenshot												
<p>Go to your workspace, the new project should be listed.</p> <p>Select your project and press the Run button.</p> 													
<p>Select the testFLPServiceMockServer.html to run the application with the mock data we prepared in BUILD.</p> <p>Press OK.</p>	 <table border="1" data-bbox="597 982 1271 1241"> <thead> <tr> <th>File Name</th> <th>File Path</th> </tr> </thead> <tbody> <tr> <td>testFLPService.html</td> <td>//IE_workshop/webapp/test/testFLPS...</td> </tr> <tr> <td>testFLPServiceMockServer.html</td> <td>//IE_workshop/webapp/test/testFLPS...</td> </tr> <tr> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	File Name	File Path	testFLPService.html	//IE_workshop/webapp/test/testFLPS...	testFLPServiceMockServer.html	//IE_workshop/webapp/test/testFLPS...						
File Name	File Path												
testFLPService.html	//IE_workshop/webapp/test/testFLPS...												
testFLPServiceMockServer.html	//IE_workshop/webapp/test/testFLPS...												
<p>A new tab will be open and show an SAP Fiori launchpad.</p> <p>Select the tile on the launchpad that corresponds to the name your BUILD prototype application.</p>													

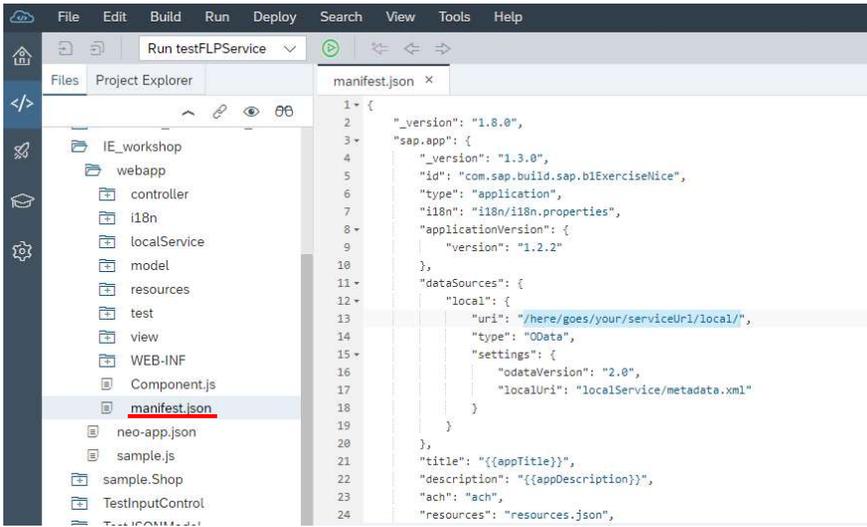
Explanation	Screenshot																				
<p>Your application will now open and show the views you designed with BUILD.</p>	 <p>The screenshot shows the SAP Fiori 'Shoe Store' application. At the top, there's a navigation bar with 'SAP' and a dropdown menu 'B1 Exercise Nice'. Below that, the page title is 'Shoe Store'. There are two tabs: '23 All Items' and '50 Matching Items'. A table titled 'Items (3)' is displayed with columns: Item Code, Description, Quantity, Price, and Currency. The table contains three rows of data:</p> <table border="1"> <thead> <tr> <th>Item Code</th> <th>Description</th> <th>Quantity</th> <th>Price</th> <th>Currency</th> </tr> </thead> <tbody> <tr> <td>A00001</td> <td>J.B. Officeprint 1420</td> <td>197</td> <td>200</td> <td>GBP</td> </tr> <tr> <td>A00002</td> <td>J.B. Officeprint 1111</td> <td>391</td> <td>100</td> <td>GBP</td> </tr> <tr> <td>B0001</td> <td>B0001</td> <td>0</td> <td>0</td> <td></td> </tr> </tbody> </table>	Item Code	Description	Quantity	Price	Currency	A00001	J.B. Officeprint 1420	197	200	GBP	A00002	J.B. Officeprint 1111	391	100	GBP	B0001	B0001	0	0	
Item Code	Description	Quantity	Price	Currency																	
A00001	J.B. Officeprint 1420	197	200	GBP																	
A00002	J.B. Officeprint 1111	391	100	GBP																	
B0001	B0001	0	0																		

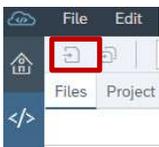
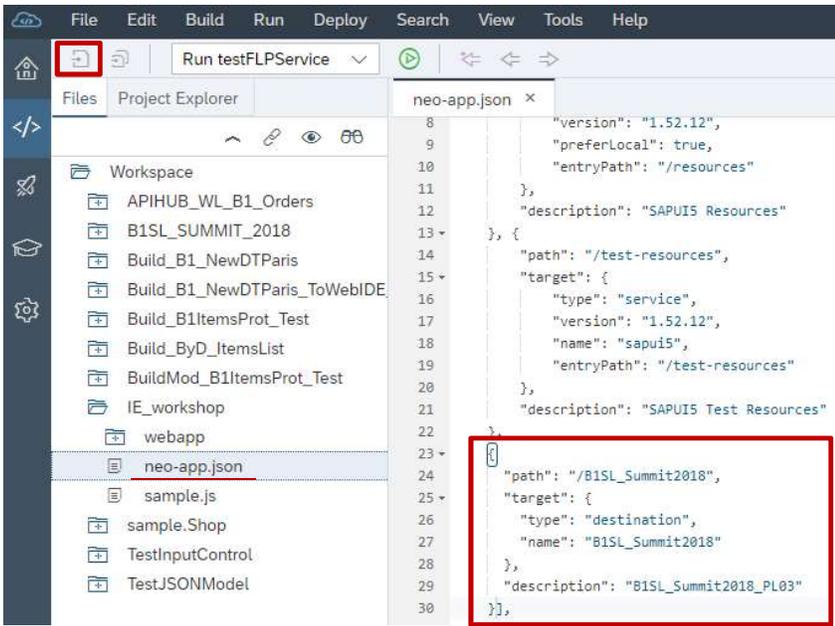
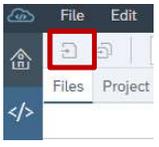
iii. Create a destination pointing to your backend server

On the SAP Community [From SAP API Business Hub to your SAP Business One system](#) blog dedicated to SAP API Business Hub it is explained how to create a destination in SAP Cloud Platform pointing to your SAP Business One backed server. Please check step number 1 of this blog to learn how to create a destination.

iv. Connect to your real B1 backend server

We have imported the BUILD prototype into a WebIDE SAP Fiori project but we are still not connected to a real backend server. This section will show you how to modify the SAP Fiori project to connect to your real B1 backend server.

Explanation	Screenshot
<p>In SAP Web IDE workspace, expand your project.</p> <p>In the webapp folder, open the manifest.json file with the code editor.</p>	 <p>The screenshot shows the SAP Web IDE workspace. The Project Explorer on the left shows the 'webapp' folder expanded, with 'manifest.json' selected. The code editor on the right shows the content of 'manifest.json' with line numbers 1 through 24. The JSON content is as follows:</p> <pre> 1 { 2 "_version": "1.8.0", 3 "sap.app": { 4 "_version": "1.3.0", 5 "id": "com.sap.build.sap.b1ExerciseNice", 6 "type": "application", 7 "i18n": "i18n/i18n.properties", 8 "applicationVersion": { 9 "version": "1.2.2" 10 }, 11 "dataSources": { 12 "local": { 13 "uri": "/here/goes/your/serviceUri/local/", 14 "type": "OData", 15 "settings": { 16 "odataVersion": "2.0", 17 "localUri": "localService/metadata.xml" 18 } 19 } 20 }, 21 "title": "{{appTitle}}", 22 "description": "{{appDescription}}", 23 "ach": "ach", 24 "resources": "resources.json", </pre>

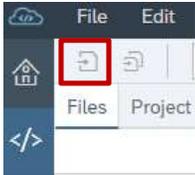
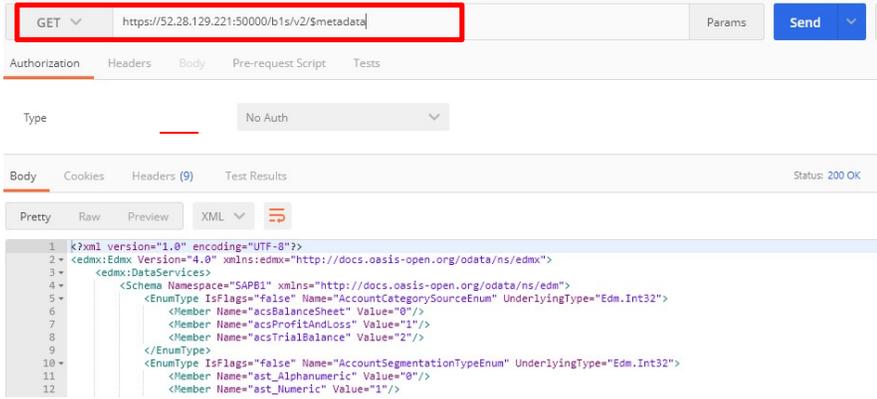
Explanation	Screenshot
<p>In the manifest.json file.</p> <p>Replace the uri property value under dataSources section with your backend OData service path.</p> <p>The uri is built from your destination name (in my case /B1SL_Summit2018) plus the root Service Layer path for OData v4 (/b1s/v2/ available from SAP Business One 9.3 PL04).</p>	 <pre> 1 { 2 "_version": "1.8.0", 3 "sap.app": { 4 "_version": "1.3.0", 5 "id": "com.sap.build.sap.b1ExerciseNice", 6 "type": "application", 7 "i18n": "i18n/i18n.properties", 8 "applicationVersion": { 9 "version": "1.2.2" 10 }, 11 "dataSources": { 12 "local": { 13 "uri": "/B1SL_Summit2018/b1s/v2/", 14 "type": "OData", 15 "settings": { 16 "odataVersion": "2.0", 17 "localUri": "localService/metadata.xml" 18 } 19 } 20 } } </pre>
<p>Press Save button.</p>	
<p>Open neo-app.json file.</p> <p>Add your backend destination entry to fetch data.</p> <pre> { "path": "/yourdest", "target": { "type": "destination", "name": "yourdest" }, "description": "yourdesc" } </pre>	 <pre> 8 "version": "1.52.12", 9 "preferLocal": true, 10 "entryPath": "/resources" 11 }, 12 "description": "SAPUI5 Resources" 13 }, { 14 "path": "/test-resources", 15 "target": { 16 "type": "service", 17 "version": "1.52.12", 18 "name": "sapui5", 19 "entryPath": "/test-resources" 20 }, 21 "description": "SAPUI5 Test Resources" 22 }, 23], 24 "path": "/B1SL_Summit2018", 25 "target": { 26 "type": "destination", 27 "name": "B1SL_Summit2018" 28 }, 29 "description": "B1SL_Summit2018_PL03" 30 }], </pre>
<p>Press Save button.</p>	

v. Extra SAP Business One backend configuration steps

As at the time we have created this document SAP BUILD doesn't support yet OData v4 and SAP Business One Service Layer APIs are based on OData v4, to design our SAP Business One Build prototype we had to use a custom OData model in SAP Build to design our prototype. Therefore, the WebIDE project will not directly run after the changes done in previous steps but some extra steps will be required.

As SAP WebIDE supports OData v4 we can now replace the custom OData model we designed in SAP Build by the real SAP Business One Service Layer OData model to get SAP Business One data from our backend.

Explanation	Screenshot
<p>Open the manifest.json file.</p> <p>Change the “settings” “odataVersion” to 4.0.</p>	 <pre>manifest.json x 1 { 2 "_version": "1.8.0", 3 "sap.app": { 4 "_version": "1.3.0", 5 "id": "com.sap.build.sap.b1ExerciseNice", 6 "type": "application", 7 "i18n": "i18n/i18n.properties", 8 "applicationVersion": { 9 "version": "1.2.2" 10 }, 11 "dataSources": { 12 "local": { 13 "uri": "/B1SL_Summit2018/b1s/v2/", 14 "type": "OData", 15 "settings": { 16 "odataVersion": "4.0", 17 "localUri": "localService/metadata.xml" 18 } 19 } 20 } 21 }, 22 }</pre>
<p>Search models element inside sap.ui5</p>	 <pre>manifest.json x 64 }, 65 "models": { 66 "i18n": { 67 "type": "sap.ui.model.resource.ResourceModel", 68 "uri": "i18n/i18n.properties" 69 }, 70 "": { 71 "dataSource": "local", 72 "type": "sap.ui.model.odata.v2.ODataModel", 73 "settings": { 74 "loadMetadataAsync": false, 75 "json": true, 76 "bJSON": true, 77 "defaultBindingMode": "TwoWay", 78 "defaultCountMode": "Inline", 79 "useBatch": true, 80 "refreshAfterChange": false, 81 "disableHeadRequestForToken": true 82 } 83 } 84 }</pre>

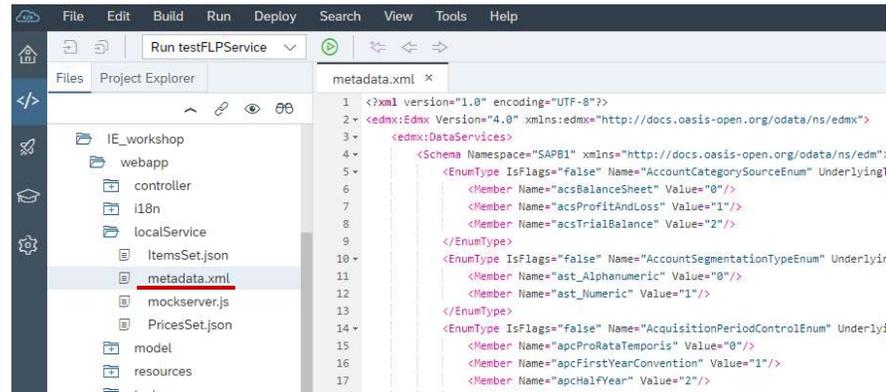
Explanation	Screenshot
<p>Replace the type of the model with empty name by sap.ui.model.odata.v4.ODataModel.</p> <p>Change the settings and add preload property true.</p> <p>Pay attention you keep the dataSource value unchanged as it matches the dataSource value defined at the beginning of the file.</p> <pre> "settings": { "operationMode": "Server", "synchronizationMode": "None", "groupId": "\$direct" }, "preload": true </pre>	<pre> "models": { "i18n": { "type": "sap.ui.model.resource.ResourceModel", "uri": "i18n/i18n.properties" }, "": { "dataSource": "local", "type": "sap.ui.model.odata.v4.ODataModel", "settings": { "operationMode": "Server", "synchronizationMode": "None", "groupId": "\$direct" }, "preload": true } }, </pre>
<p>Press the Save button.</p>	
<p>Retrieve the metadata file from SAP Business One Service Layer via Postman with the GET request https://your_b1sl_server:50000/b1s/v2/\$metadata.</p> <p>Save the response as a file named metadata.xml.</p>	

Explanation

Replace the **localService/metadata.xml** file imported from BUILD by the SAP Business One Service Layer metadata file saved in the previous step.

To avoid conflicts as the Build metadata.xml file is already there you can rename the existing file as **build_metadata.xml**.

Screenshot

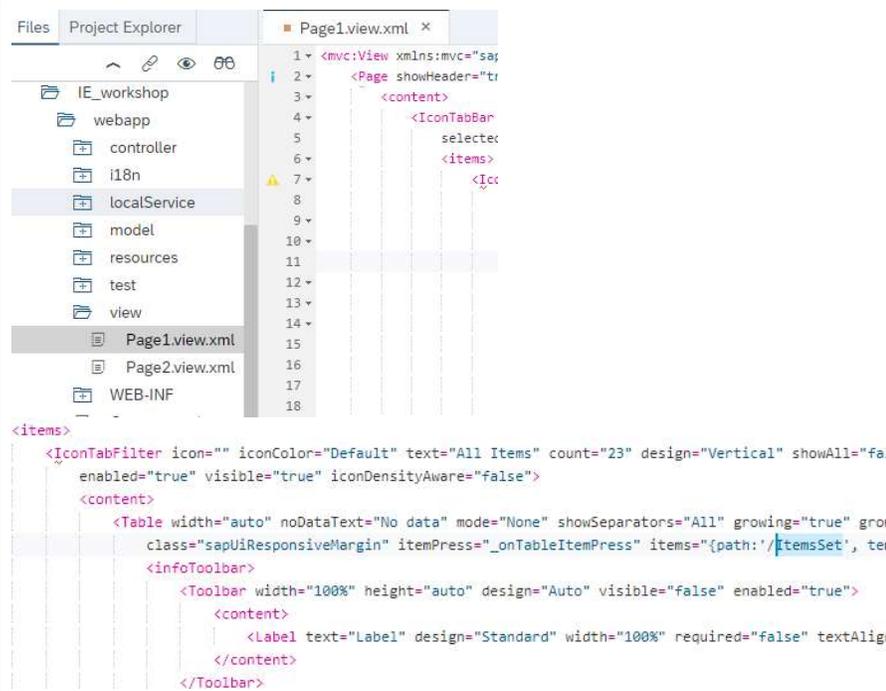


The screenshot shows the SAP Business One IDE interface. The Project Explorer on the left displays the project structure: IE_workshop > webapp > localService > metadata.xml. The main editor window shows the content of metadata.xml, which is an EDMX file. The code includes XML declarations for namespaces and a schema with several EnumType definitions and their members.

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx Version="4.0" xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
  <edmx:DataService>
    <Schema Namespace="SAP81" xmlns="http://docs.oasis-open.org/odata/ns/edm">
      <EnumType IsFlags="false" Name="AccountCategorySourceEnum" UnderlyingType="int">
        <Member Name="acsBalanceSheet" Value="0"/>
        <Member Name="acsProfitAndLoss" Value="1"/>
        <Member Name="acsTrialBalance" Value="2"/>
      </EnumType>
      <EnumType IsFlags="false" Name="AccountSegmentationTypeEnum" UnderlyingType="int">
        <Member Name="ast_Alphanumeric" Value="0"/>
        <Member Name="ast_Numeric" Value="1"/>
      </EnumType>
      <EnumType IsFlags="false" Name="AcquisitionPeriodControlEnum" UnderlyingType="int">
        <Member Name="apcProRataTemporis" Value="0"/>
        <Member Name="apcFirstYearConvention" Value="1"/>
        <Member Name="apcHalfYear" Value="2"/>
      </EnumType>
    </Schema>
  </DataService>
</edmx:Edmx>
```

Open the **Page1.view.xml** file, search for **ItemsSet** and replace it by **Items**.

In the model we created in BUILD entities have the suffix Set, while in SAP Business One Service Layer we don't have it, we need to fix it to be able to directly connect to Service Layer.



The screenshot shows the SAP Business One IDE interface. The Project Explorer on the left displays the project structure: IE_workshop > webapp > localService > Page1.view.xml. The main editor window shows the content of Page1.view.xml, which is an XML view definition. The code includes XML declarations for namespaces and a view definition with a content area containing an IconTabBar, a select control, and an items control. The items control is highlighted in red.

```
<mvc:View xmlns:mvc="sap.ui.core.mvc" showHeader="true">
  <Page showHeader="true">
    <content>
      <IconTabBar>
        <select>
          <items>
            <IconTabItem icon="" iconColor="Default" text="All Items" count="23" design="Vertical" showAll="false" enabled="true" visible="true" iconDensityAware="false">
              <content>
                <Table width="auto" noDataText="No data" mode="None" showSeparators="All" growing="true" growable="true" class="sapUiResponsiveMargin" itemPress="_onTableItemPress" items="{path: '/ItemsSet', type: 'json'}">
                  <infoToolbar>
                    <Toolbar width="100%" height="auto" design="Auto" visible="false" enabled="true">
                      <content>
                        <Label text="Label" design="Standard" width="100%" required="false" textAlign="center">
                          </content>
                        </Label>
                      </content>
                    </Toolbar>
                  </Table>
                </content>
              </IconTabItem>
            </items>
          </select>
        </IconTabBar>
      </content>
    </Page>
  </mvc:View>
```

Open **Component.js** file.

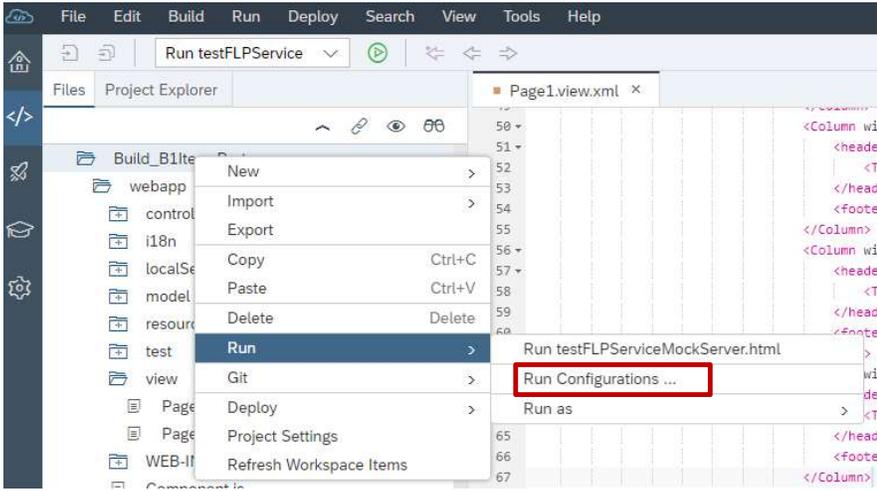
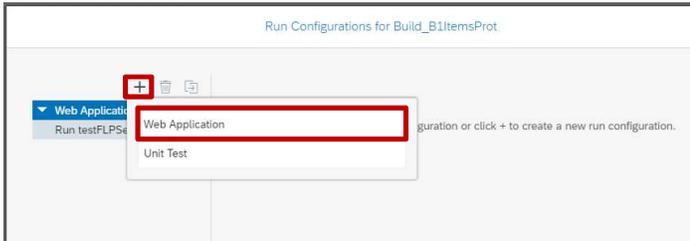
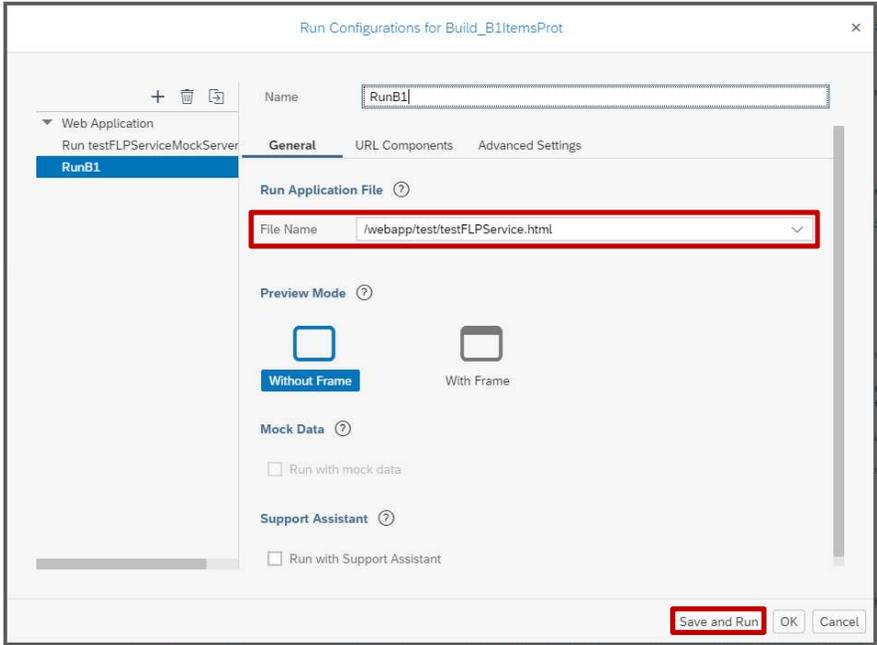
Replace **ItemsSet** by **Items** in the navigationWithContext definition.



The screenshot shows the SAP Business One IDE interface. The Project Explorer on the left displays the project structure: IE_workshop > webapp > localService > Component.js. The main editor window shows the content of Component.js, which is a JavaScript file. The code includes the sap.ui.define function and a navigationWithContext object. The ItemsSet property in the navigationWithContext object is highlighted in red.

```
sap.ui.define([
  "sap/ui/core/UIComponent",
  "sap/ui/Device",
  "com/sap/build/sap/blExerciseNice/model/models",
  "./model/errorHandling"
], function (UIComponent, Device, models, errorHandling) {
  "use strict";

  var navigationWithContext = {
    "ItemsSet": {
      "Page2": ""
    }
  };
});
```

Explanation	Screenshot
<p>To run the application this time connecting to your real B1 backend right click on your project and choose Run -> Run Configurations...</p>	
<p>Press +. Select Web Application.</p>	
<p>Give a Name to the new configuration. Select testFLPService.html as File Name. Press Save and Run.</p>	

Explanation | **Screenshot**

Now the data shown comes from B1 and not anymore from the Build sample data.

The screenshot shows a SAP WebIDE interface for a 'Shoe Store' prototype. At the top, it says 'B1 Exercise Nice' and 'Shoe Store'. Below that, there are two tabs: '23 All Items' and '50 Matching Items'. The main content is a table titled 'Items (3)' with the following data:

Item Code	Description	Quantity	Price	Currency
I00002	Blu-Ray DL Disc 10-Pack	1,537		
R00001	Printer Paper A4 White	33,703		
D00002	Portable Hard Disk 2TB	806		
C00006	Gigabit Network Card	1,483		
P20001	4GB Memory Server	547		
A00005	Rainbow Color Printer 7.5	1,564		

You can also check the details page containing the different prices depending on their price list for a specific Item by clicking on one of the rows.

The screenshot shows a SAP WebIDE interface for an 'Untitled Prototype'. It features a header with 'Title' and 'Subtitle', and three sub-section titles. The main content is a table titled 'Items (3)' with the following data:

Price List No	Price
1	2 GBP
2	1 GBP
3	1.5 GBP
4	2.5 GBP
5	3 GBP
6	3.5 GBP

Congratulations! You have imported a Build prototype to your WebIDE development environment and connected to your real SAP Business One backend server.

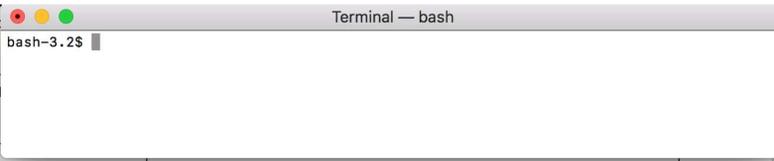
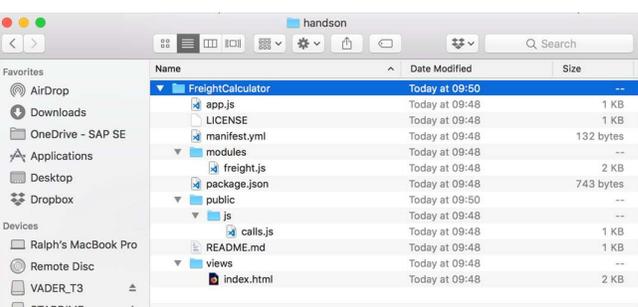
STEP 3: CLONE A NODEJS APP

In this step we are going to deploy the backend of our application.

The application we are going to deploy is based on the SMB Marketplace proof of concept we shared in the [Digital Transformation for SMBs – the Intelligent Enterprise blog](#).

It will contain the business logic required to call SAP Leonardo services and get Item details from SAP Business One and SAP Business ByDesign erps.

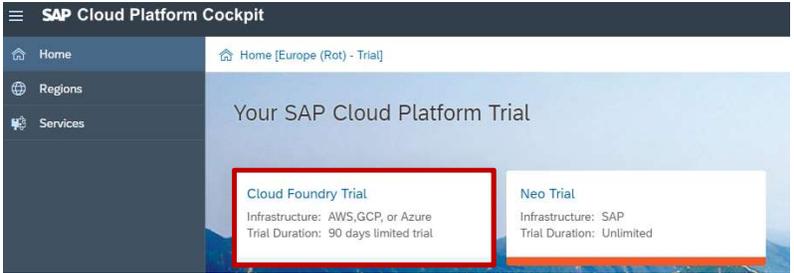
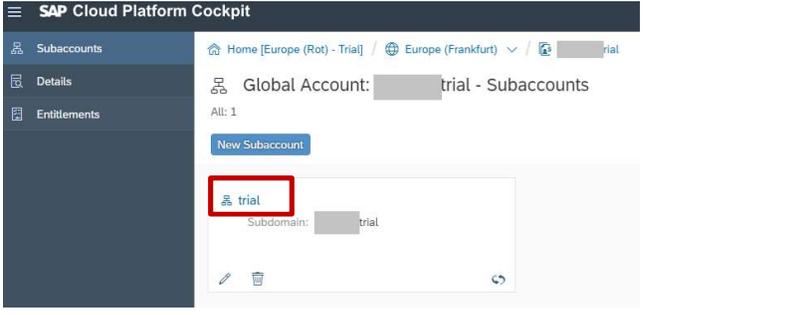
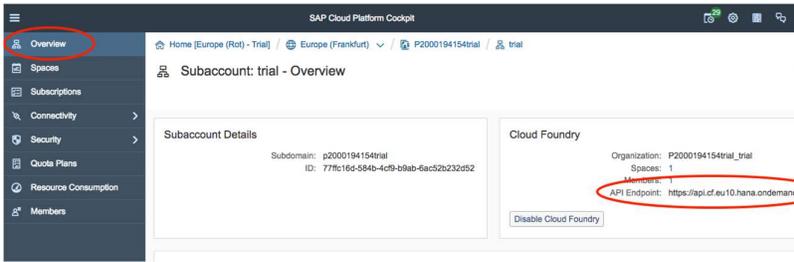
The application is written in NodeJS and the source code is available on GitHub.

Explanation	Screenshot
<p>Once git is installed (according to the pre requisites), open your system terminal (cmd, bash..)</p> <p>Navigate to a specific folder where you will download the sample application.</p> <p>Pay attention what folder is it, we will access it later.</p>	
<p>Execute the following command to clone our solution backend repository:</p> <pre>\$ git clone https://github.com/B1SA/exercise_BuildToWebIDECF/tree/master/Backend_smbmkt</pre>	
<p>You can see the app code on your file explorer:</p>	
<p>You can change the name of the app in manifest.yml file and set a unique name for your application.</p> <p>It is not a mandatory operation as we can generate a random url for our application to avoid conflicts with other accounts running the same app name, it will be shown in the next step.</p> <p>E.g <code>smbmkt<Your Initials></code></p>	

STEP 4: DEPLOY THE NODEJS APP INTO SAP CLOUD FOUNDRY

In this step, we are going to deploy our SMB Marketplace app to SAP Cloud Platform Cloud Foundry.

i. SAP Cloud Platform Cloud Foundry Environment

<p>Go to your Cloud Foundry account.</p>	 <p>The screenshot shows the SAP Cloud Platform Cockpit interface. The left sidebar has 'Home', 'Regions', and 'Services'. The main content area displays 'Your SAP Cloud Platform Trial' with two options: 'Cloud Foundry Trial' (Infrastructure: AWS, GCP, or Azure; Trial Duration: 90 days limited trial) and 'Neo Trial' (Infrastructure: SAP; Trial Duration: Unlimited). The 'Cloud Foundry Trial' option is highlighted with a red box.</p>
<p>Select your trial subaccount. Click on the trial link.</p>	 <p>The screenshot shows the 'Subaccounts' page in SAP Cloud Platform Cockpit. The left sidebar has 'Subaccounts', 'Details', and 'Entitlements'. The main content area shows 'Global Account: trial - Subaccounts' and a list of subaccounts. One subaccount named 'trial' is highlighted with a red box.</p>
<p>Open the Overview option in the menu</p> <p>Select and copy your API Endpoint. E.g. https://api.cf.eu10.hana.ondemand.com</p>	 <p>The screenshot shows the 'Overview' page for a subaccount in SAP Cloud Platform Cockpit. The left sidebar has 'Overview', 'Spaces', 'Subscriptions', 'Connectivity', 'Security', 'Quota Plans', 'Resource Consumption', and 'Members'. The 'Overview' option is highlighted with a red circle. The main content area shows 'Subaccount Details' and 'Cloud Foundry' information. The 'API Endpoint' is highlighted with a red circle and is https://api.cf.eu10.hana.ondemand.com.</p>

With the CLI installed (according to the pre-requisites), open your system **terminal** and navigate to the folder of the backend app cloned on STEP 3 of this guide

```
PS C:\[redacted]\smbmkt\smbmkt> ls

Directory: C:\[redacted]\smbmkt\smbmkt

Mode                LastWriteTime         Length Name
----                -
d-----            28/06/2018    15:29         files
d-----            28/06/2018    15:29         models
d-----            05/07/2018    15:05         modules
d-----            28/06/2018    15:29         public
d-----            28/06/2018    15:29         views
-a----            28/06/2018    15:29          954 .gitignore
-a----            05/07/2018    15:05     3923 app.js
-a----            29/06/2018    11:16      811 ChangesForWorkshop.txt
-a----            28/06/2018    17:23      658 manifest.yml
-a----            29/06/2018    15:06      931 package.json
```

From that folder, login to Cloud foundry using the command

```
cf login -a <API
ENDPOINT>
```

e.g.

```
$ cf login -a
api.cf.eu10.hana.ondeman
d.com
```

When prompted provide your SAP Cloud Platform **email** and **password**

```
PS C:\[redacted]\smbmkt\smbmkt> cf login
API endpoint: https://api.cf.eu10.hana.ondemand.com

Email> [redacted]@sap.com

Password>
Authenticating...
OK

Targeted org [redacted]_trial

Targeted space dev

API endpoint: https://api.cf.eu10.hana.ondemand.com (API version: 2.114.0)
User: [redacted]@sap.com
Org: [redacted]_trial
Space: dev
```

ii. Create the backing services

This app uses 2 [backing services](#) from SAP Cloud Platform. [Redis](#) for storing B1 Service Layer Sessions ID in cache and [PostgreSQL](#) to store [SAP Leonardo Feature Extraction Vectors](#). Here are the steps to create them:

Explanation	Screenshot
<p>Using the command terminal, navigate to the smbmkt root directory, which you downloaded or cloned previously.</p> <p>Execute the following commands to create the Redis and PostgreSQL services:</p> <pre>cf create-service redis v3.0-dev cachedb</pre>	<pre>PS C:\[redacted]\smbmkt\smbmkt> cf create-service redis v3.0-dev cachedb Creating service instance cachedb in org [redacted]_trial / space dev as [redacted]@sap.com... OK PS C:\[redacted]\smbmkt\smbmkt> cf create-service postgresql v9.6-dev smbktddb Creating service instance smbktddb in org [redacted]_trial / space dev as [redacted]@sap.com...</pre>

Explanation	Screenshot
<pre>cf create-service postgresql v9.6-dev smbmkt db</pre> <p>PS: When using a trial account some limitations apply. If you already had a postgresql or redis service you will not be able to create a second one, just reuse the one you have or delete your old one.</p>	
<p>You can check which services are active and the bound apps with the command:</p> <pre>cf services</pre> <p>(your services might not be bound to any app if just created now)</p>	 <pre>PS C:\Users\i029162\Documents\Git\Public_Trinidad\G\smbmkt> cf services Getting services in org [redacted]_trial / space dev as [redacted]@sap.com... OK name service plan bound apps last operation itemdb postgresql v9.4-dev smbmktdb, cfdemosummit18 create succeeded cachedb redis v3.0-dev smbmktdb create succeeded PS C:\Users\i029162\Documents\Git\Public_Trinidad\G\smbmkt></pre>

iii. Deploy the smbmktdb app

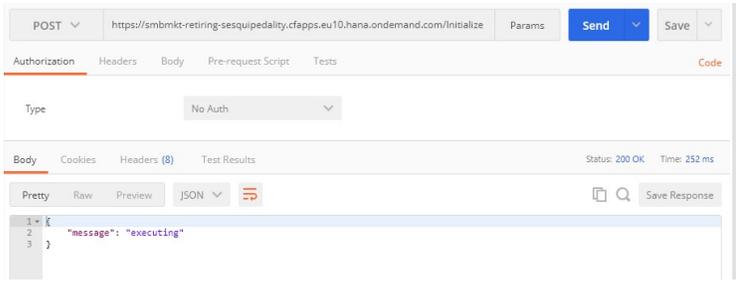
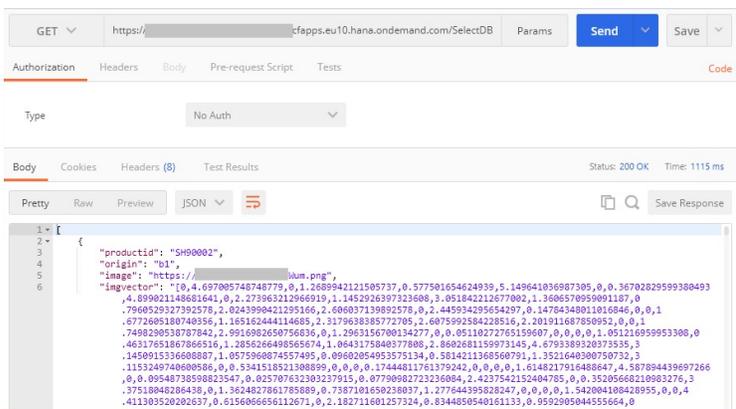
Explanation	Screenshot
<p>This app has 2 microservices (bot and smbmktdb) that can be deployed at once or separately. Their specifications are detailed in the manifest.yml.</p> <p>In this exercise we will only work with the smbmktdb microservice as the other service is the one related to Facebook Messenger that is not used in this exercise.</p> <p>From the same terminal of the previous step go to your smbmktdb/smbmktdb app folder and execute:</p> <pre>cf push --random-route</pre> <p><i>--random-route avoids name collisions with other accounts that might deploy the same app on SCP. You can choose your own app name by changing the application names in the manifest.yml.</i></p>	 <pre>PS C:\Users\i029162\Documents\Git\Public_Trinidad\G\smbmkt> cd smbmktdb PS C:\Users\i029162\Documents\Git\Public_Trinidad\G\smbmkt\smbmktdb> cf push --random-route Using manifest file C:\Users\i029162\Documents\Git\Public_Trinidad\G\smbmkt\smbmktdb\manifest.yml Updating app smbmktdb in org [redacted]_trial / space dev as [redacted]@sap.com... OK Uploading smbmktdb... Uploading app files from C:\Users\i029162\Documents\Git\Public_Trinidad\G\smbmkt\smbmktdb: Uploading 32.1K, 40 files Done uploading OK Binding service cachedb to app smbmktdb in org [redacted]_trial / space dev as [redacted]@sap.com... OK Binding service itemdb to app smbmktdb in org [redacted]_trial / space dev as [redacted]@sap.com... OK</pre>

Explanation	Screenshot
<p>At the end of the process your smb app must be running.</p> <p>You can check your apps with the command:</p> <pre>cf apps</pre>	

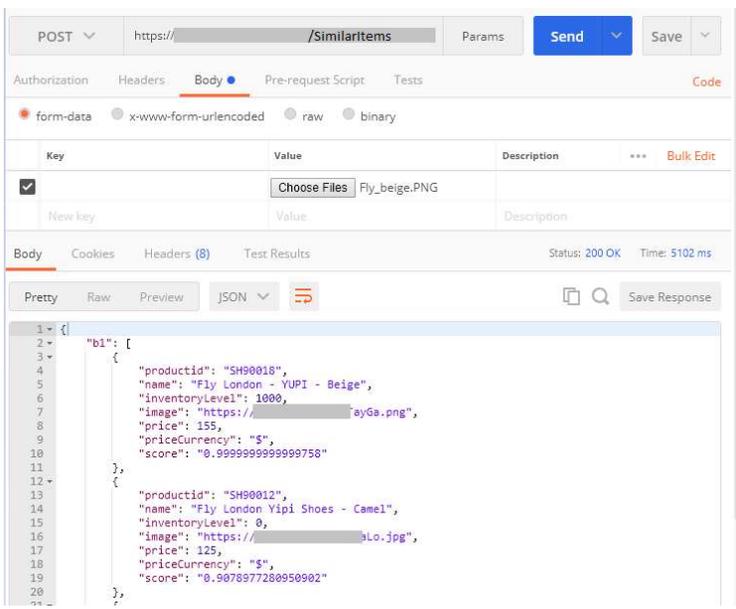
iv. Configure the SMB Mktplace backend

Explanation	Screenshot
<p>Set the following <u>Environment Variables</u> so the app can work properly.</p> <p>If you don't have a B1 or ByD system available you can then skip their corresponding environment variables set. Please note that you need at least one ERP system from both to be able to retrieve items data.</p>	<pre>B1_COMP_ENV: <SAP Business One Company Name> B1_DEFAULT_BP: <A Business Partner Code for the B1 Sales Order> B1_USER_ENV: <B1 User to login the Service Layer> B1_PASS_ENV: <Password for the B1 User> B1_SERVER_ENV: <SAP Business One server URL> B1_SLPPATH_ENV: /b1s/v1 B1_SLPORT_ENV: <SAP Business One Service Layer Server Port> BYD_AUTH: <[Base64 Encoded] user:password> BYD_DEFAULT_BP: <A Business Partner Code for the ByD Sales Order> BYD_PATH: /sap/byd/odata/cust/v1 BYD_PORT: "" BYD_SERVER: <SAP Business ByDesign server URL> FILE_SEP: - _ - LEO_API_KEY: <SAP Leonardo API Key> TEMP_DIR: files/tmp VECTOR_DIR: files/vectors</pre>
<p>Set one by one the environment variables with the command:</p> <pre>cf set-env smbmkt B1_COMP_ENV SBODEMOUS</pre>	
<p>Restart your application so it can get the new environment variables with the following command:</p> <pre>cf restart smbmkt</pre>	

v. Initialize the SMB Mktplace backend

Explanation	Screenshot
<p>To initialize the Postgresql database with the existing items from B1 and ByDesign as well as the vectors for each item please call the following API with Postman for example:</p> <p>POST <code><your backend url>/Initialize</code></p>	
<p>After initialization you can check the Postgresql items table content with the following API:</p> <p>GET <code><your backend url>/SelectDB</code></p> <p>If the Initialize command runs successfully an entry should be available for each one of your items containing the productid, origin, image and imgVector properties.</p>	

vi. Test the SMB Mktplace backend /SimilarItems API

Explanation	Screenshot
<p>With Postman call the /SimilarItems API:</p> <p>POST <code><your backend url>/SimilarItem</code></p> <p>In the body select "form-data" and choose a file containing the image of a shoe.</p>	

Congratulations! You have implemented and deployed your first Cloud Foundry application on SAP Cloud Platform!

STEP 5: CONSUME THE NODEJS APP FROM THE SAP FIORI APP

Until now our SAP Fiori application hasn't been modified and reflects exactly what was designed in BUILD. In this step we are going to modify the tab "Matching Items" in order to consume the services provided by our NodeJS backend.

i. Create a destination pointing to your smbmk backend

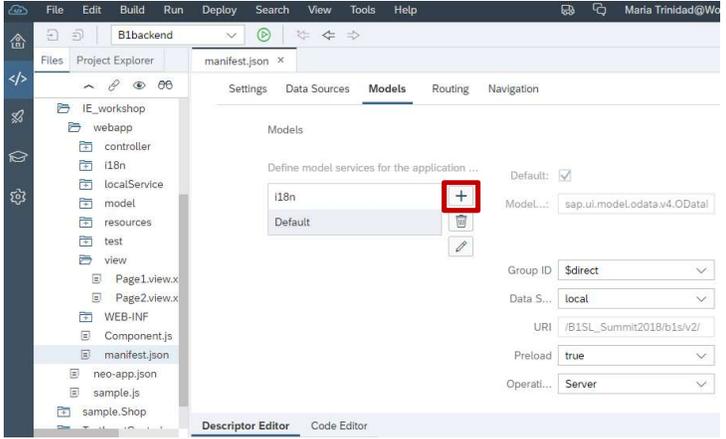
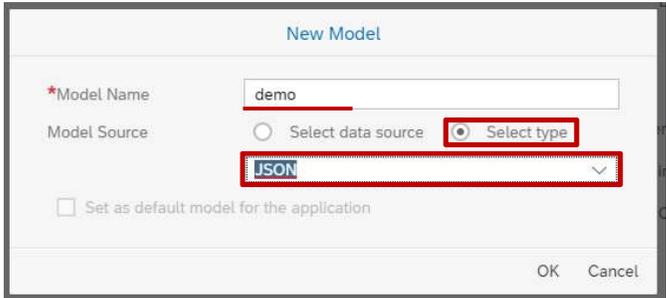
Your destination in your SAP Cloud Platform cockpit -> Connectivity -> Destinations should look like the one here, just replace the URL with your smbmk url.

Check the following tutorial [Create a Destination on SAP Cloud Platform](#) to learn more details about destinations.

The screenshot shows the 'Destination Configuration' interface. On the left, there are input fields for: *Name: smbmk_workshopNice, Type: HTTP, Description: smbmk_workshopNice, *URL: https://smbmk-...cfag, Proxy Type: Internet, and Authentication: NoAuthentication. On the right, under 'Additional Properties', there are four key-value pairs: TrustAll: true, WebIDEEnabled: true, WebIDESystem: smbmk_workNice, and WebIDEUsage: ui5_execute,odata_gen. At the bottom, there are buttons for Edit, Clone, Export, Delete, and Check Connection.

Explanation	Screenshot
<p>Open neo-app.json file.</p> <p>Add your backend destination entry to fetch data.</p> <pre>{ "path": "/yourdest", "target": { "type": "destination", "name": "yourdest" }, "description": "yourdesc" }</pre>	<p>The screenshot shows the VS Code editor with the 'neo-app.json' file open. The file content is as follows:</p> <pre>10 "entryPath": "/resources" 11 }, 12 "description": "SAPUI5 Resources" 13 }, { 14 "path": "/test-resources", 15 "target": { 16 "type": "service", 17 "version": "1.52.12", 18 "name": "sapui5", 19 "entryPath": "/test-resources" 20 }, 21 "description": "SAPUI5 Test Resources" 22 }, { 23 "path": "/B1SL_workshopNice", 24 "target": { 25 "type": "destination", 26 "name": "B1SL_workshopNice" 27 }, 28 "description": "B1SL_workshopNice" 29 }, { 30 "path": "/smbmk_workshopNice", 31 "target": { 32 "type": "destination", 33 "name": "smbmk_workshopNice" 34 }, 35 "description": "smbmk_workshopNice" 36 },</pre> <p>The entry for "/smbmk_workshopNice" is highlighted with a red box.</p>
<p>Press Save button.</p>	<p>The screenshot shows the VS Code editor's top-left toolbar. The 'Save' button (represented by a floppy disk icon) is highlighted with a red box.</p>

ii. Create a new JSON model

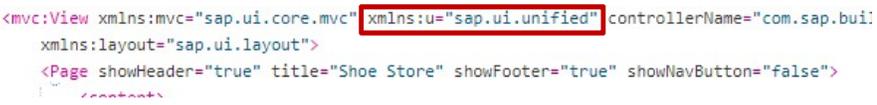
Explanation	Screenshot
<p>Open the manifest.json file with the Descriptor Editor.</p> <p>Go to the Models tab.</p> <p>Press the  button to add a new model.</p>	
<p>Enter the Model Name of the model demo if you want to avoid changing the references in the following steps.</p> <p>Choose Select type as Model Source.</p> <p>Select JSON as type.</p> <p>Press OK.</p>	

iii. Change the Image control in the Page1.view.xml file.

Explanation	Screenshot
<p>Open the Page1.view.xml file with the Code Editor.</p> <p>Search for the Image control and replace it with the following code:</p> <pre data-bbox="219 1577 487 1774"><Image id="img" tooltip="image" class="sapUiLargeMargin" n" src="{demo}/fileURL}"/ ></pre>	<pre data-bbox="544 1373 1396 1444"><IconTabFilter icon="" iconColor="Default" text="Matching Items" count="50" design="Vertical" s <content> <Image id="img" tooltip="image" class="sapUiLargeMargin" src="{demo}/fileURL}"/></pre>

iv. Create a FileUploader control.

In BUILD we added a SearchField control as the FileUploader control was not available. We will now replace it with a FileUploader.

Explanation	Screenshot
<p>Open the Page1.view.xml file with the Code Editor.</p> <p>Search the SearchField control and replace it by the following code.</p> <p>We use the <code>smbmkt</code> destination created in a previous step to get the <code>SimilarItems</code> url.</p> <p>Replace <code>smbmkt_destination</code> with your <code>smbmkt</code> destination name.</p>	 <pre> <Image id="img" tooltip="image" class="sapUiLargeMargin" src="{demo}/fileURL" /> <u:FileUploader id="fileUploader" name="files" uploadUrl="/smbmkt_workshopNice/SimilarItems" useMultiPart="true" > <u:headerParameters > <u:FileUploaderParameter name="Accept" value="application/json" /> </u:headerParameters > </u:FileUploader > <u:FileUploader id="fileUploader" name="files" uploadUrl="/smbmkt_destination/SimilarItems" useMultiPart="true" sendXHR="true" uploadOnChange="true" tooltip="Upload your file to the local server" fileType="jpg,png,gif" mimeType="application/x-zip- compressed,application/zip,application/octet- stream,image/png,image/jpg,image/jpeg,image/bmp,image/tiff" change="fileUploadChange" uploadStart="fileUploadStart" uploadComplete="fileUploadComplete"> <u:headerParameters> <u:FileUploaderParameter name="Accept" value="application/json"/> </u:headerParameters> </u:FileUploader> </pre>
<p>Add the prefix <code>xmlns:u="sap.ui.unified"</code>, required by the FileUploader control, at the beginning of the Page1.view.xml file.</p>	 <pre> <mvc:View xmlns:mvc="sap.ui.core.mvc" xmlns:u="sap.ui.unified" controllerName="com.sap.bui: xmlns:layout="sap.ui.layout"> <Page showHeader="true" title="Shoe Store" showFooter="true" showNavButton="false"> <content> </pre>

v. Bind the Matching Items Table to our backend properties

Let's define first the IDs of our Table and ColumnListItem controls, we will need them to further bind them to our backend response.

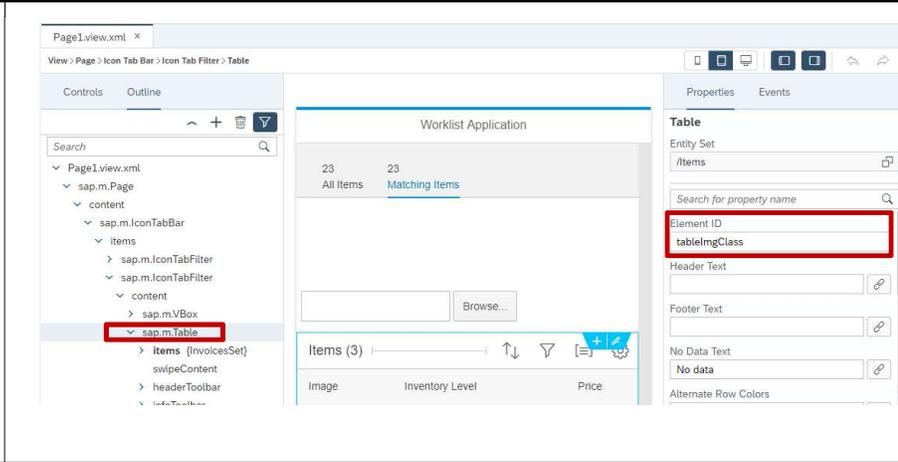
Explanation

Open the **Page1.view.xml** file with the Layout Editor.

In the Outline tab (left of the screen) open the second **IconTabFilter** content and select the **sap.m.Table**.

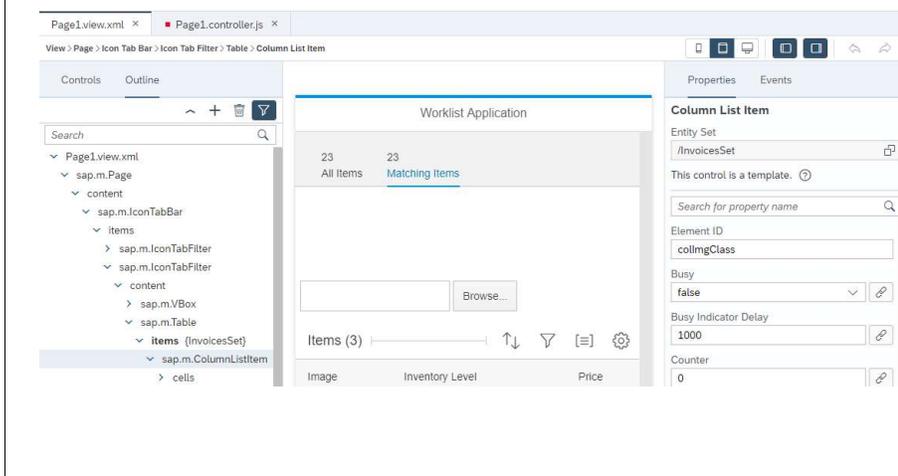
In the Properties tab (right of the screen) enter **tablemgClass** as **Element ID**.

Screenshot



In the Outline tab (left of the screen), inside the **sap.m.Table** we selected in previous step now select the **Items -> sap.m.ColumnListItem** element.

In the Properties tab (right of the screen) enter **collmgClass** as **Element ID**.



Now let's map each column in the Table to our backend response properties.

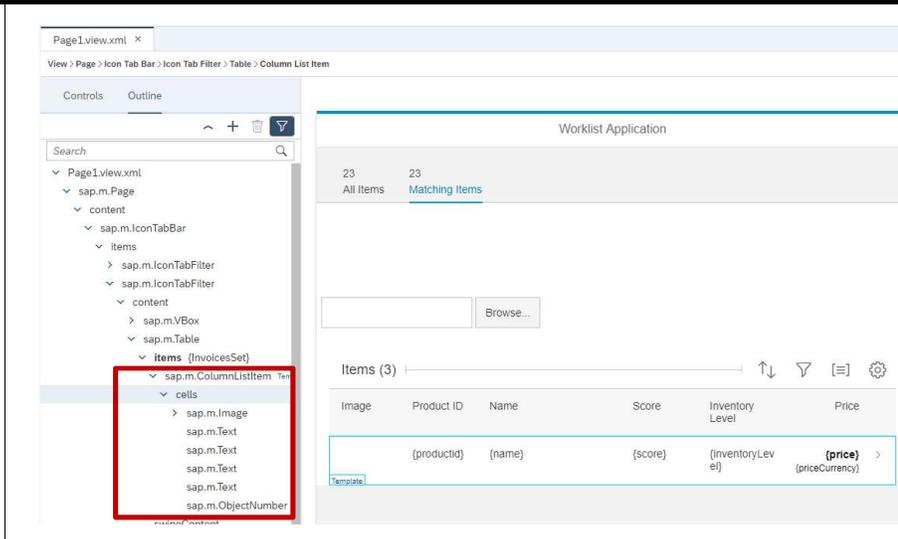
Explanation

Open the **Page1.view.xml** file with the Layout Editor.

Open the Outline tab (left of the screen) and select the **sap.m.ColumnListItem -> cells** element.

Make sure you get 6 cells defined with the types as marked in the screen capture. Maybe the easier is to delete them and recreate them in the right order.

Screenshot



Explanation

For the **sap.m.Image** open the **detailBox** and enter **{image}** in the **Src** property.

Screenshot

The screenshot shows the SAP Web IDE interface. The Outline pane on the left displays the project structure, with the path `Page1.view.xml > sap.m.Page > content > sap.m.IconTabBar > items > sap.m.IconTabFilter > content > sap.m.VBox > sap.m.Table > items (InvoicesSet) > cells > sap.m.Image > detailBox` highlighted. The Properties pane on the right shows the configuration for the selected `sap.m.Image` control. The `Src` property is highlighted with a red box and contains the value `{image}`. The main preview area shows a 'Worklist Application' with a table containing one item with ID 23.

For the cells of type **sap.m.Text** go over them and set the **Text** property to the different properties names returned by the `smbmkt` backend: `{productid}`, `{name}`, `{score}`, `{inventoryLevel}`

The screenshot shows the SAP Web IDE interface. The Outline pane on the left displays the path `Page1.view.xml > sap.m.Page > content > sap.m.IconTabBar > items > sap.m.IconTabFilter > content > sap.m.VBox > sap.m.Table > items (InvoicesSet) > cells > sap.m.Text` highlighted. The Properties pane on the right shows the configuration for the selected `sap.m.Text` control. The `Text` property is highlighted with a red box and contains the value `{productid}`. The main preview area shows the same 'Worklist Application' table.

For the last cell of type **sap.m.ObjectNumber** set the property **Number** to `{price}` and **Unit** to `{priceCurrency}`.

The screenshot shows the SAP Web IDE interface. The Outline pane on the left displays the path `Page1.view.xml > sap.m.Page > content > sap.m.IconTabBar > items > sap.m.IconTabFilter > content > sap.m.VBox > sap.m.Table > items (InvoicesSet) > cells > sap.m.ObjectNumber` highlighted. The Properties pane on the right shows the configuration for the selected `sap.m.ObjectNumber` control. The `Number` and `Unit` properties are highlighted with a red box and contain the values `{price}` and `{priceCurrency}` respectively. The main preview area shows the 'Worklist Application' table with the price column highlighted.

Open the **Page1.view.xml** file with the Code Editor.

Search for the **Table** with id **"tableImgClass"** we updated in previous steps.

Add the following property to indicate the

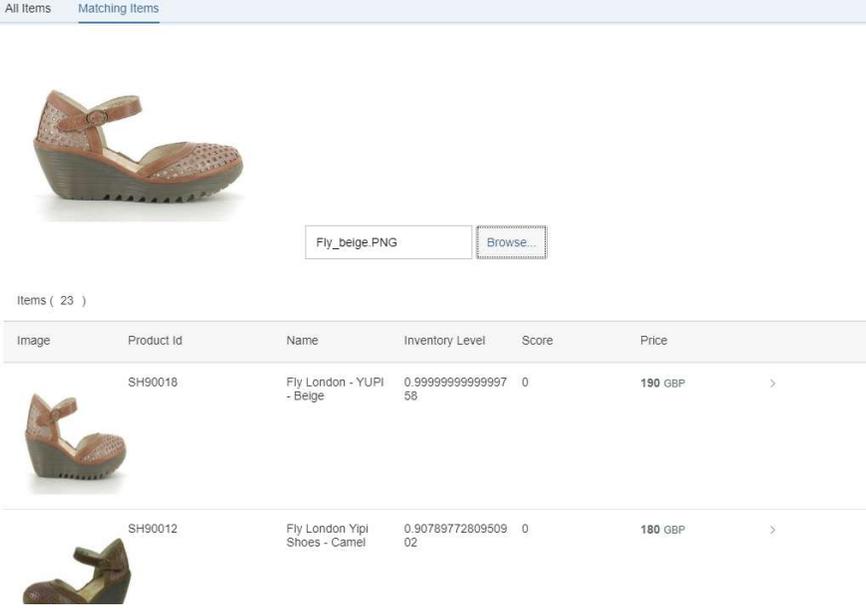
```
</u:FileUploader>
<Table id="tableImgClass" items="{path : '/result'}" width="auto" noDi
  <infoToolbar/>
  <headerToolbar
    <OverflowToolbar width="auto" height="auto" design="Transpare
  <content>
    <Text text="Items (" width="auto" maxLines="1" wrappi
    <Text text="23" width="17px" maxLines="1" wrapping="fi
```

Explanation	Screenshot
<p>path to the results from SimilarItems: items="{path : '/result'}"</p>	

vi. Implement the Page1.controller.js.

Explanation	Screenshot
<p>Open the Page1.controller.js file.</p> <p>Implement the fileUploadChange function.</p> <p>This function will be called when a file has been selected.</p>	<pre> fileUploadChange: function(oControlEvent) { // init the src file, name & url this.srcFileURL = null; this.srcFileName = null; this.srcFile = null; // keep a reference of the uploaded file name and create a url when is an image this.srcFile = oControlEvent.getParameters().files[0]; this.srcFileName = this.srcFile.name; if (this.srcFile.type.match("image.*")) { this.srcFileURL = URL.createObjectURL(this.srcFile); } }, </pre>
<p>You can get the code from the following link: https://github.com/B1SA/exercise_BuildToWeb1/DECF/blob/master/extras/STEP%205/Page1.controller.js_ext.txt</p>	<pre> fileUploadChange: function (oControlEvent) { // init the src file, name & url this.srcFileURL = null; this.srcFileName = null; this.srcFile = null; // keep a reference of the uploaded file name and create a url out of that when this is an image this.srcFile = oControlEvent.getParameters().files[0]; this.srcFileName = this.srcFile.name; if (this.srcFile.type.match("image.*")) { this.srcFileURL = URL.createObjectURL(this.srcFile); } }, </pre>

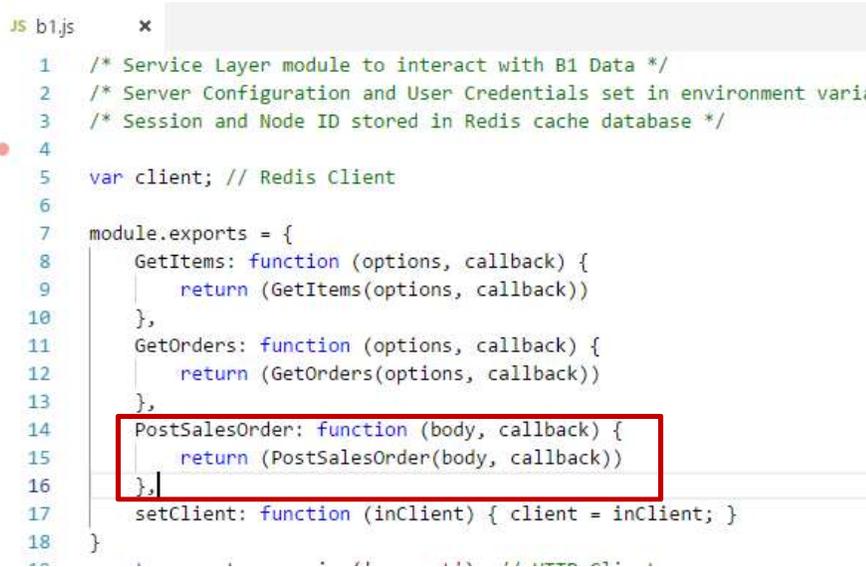
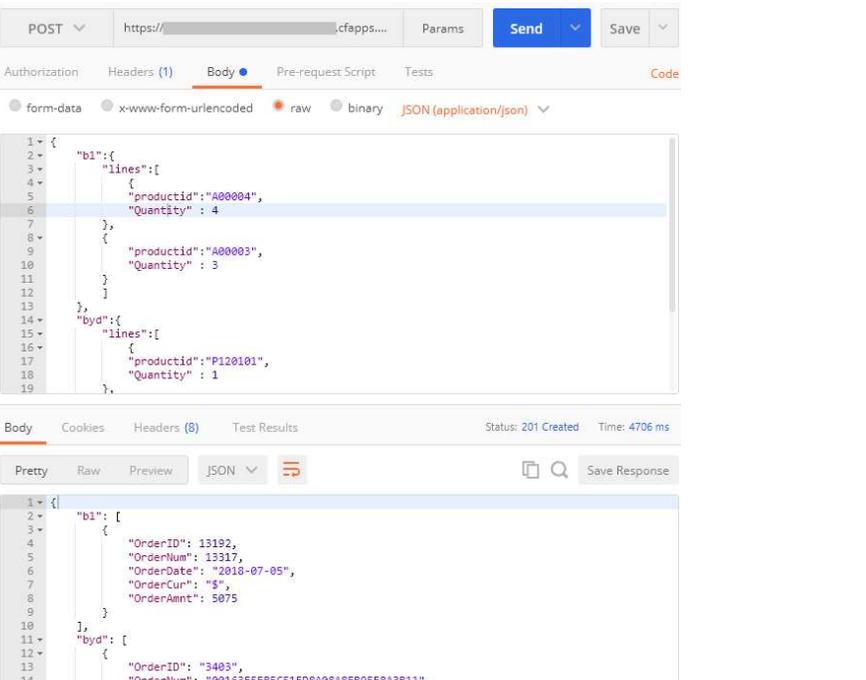
Explanation	Screenshot
<p>Now let's implement the fileUploadComplete function.</p> <p>This function will be called after the fileUploader uploadUrl ({demo>/url}) has been called and a response returned.</p>	<pre>fileUploadComplete: function (oControlEvent) { // get the current view var oView = this.getView(); // smbmt backend // clear previous results from the model oView.getModel("demo").setProperty("/result", null); var processResult = function (oController, data) { oView = oController.getView(); // merge with existing results - working with B1 only on this case var result = oView.getModel("demo").getProperty("/result"); if (result) { result.push.apply(result, data.b1); } else { result = data.b1; } oView.getModel("demo").setProperty("/result", result); oView.getModel("demo").setProperty("/fileURL", oController.srcFileURL); // Set Model to Table var oTable = oView.byId("tableImgClass"); oTable.setModel(oView.getModel("demo")); }; if (oControlEvent.getParameters().status === 200) { // get the response as JSON and process the results processResult(this, JSON.parse(oControlEvent.getParameters().responseRaw)); } else { MessageToast.show("Error " + oControlEvent.getParameters().status + " : " + JSON.parse(oControlEvent } }</pre>
<p>You can get the code from the following link:</p> <p>https://github.com/B1SA/exercise_BuildToWebIDE/CF/blob/master/extras/STEP%205/Page1.controller.js_ext.txt</p>	<pre>fileUploadComplete: function (oControlEvent) { // get the current view var oView = this.getView(); // smbmt backend // clear previous results from the model oView.getModel("demo").setProperty("/result", null); var processResult = function (oController, data) { oView = oController.getView(); // merge with existing results - working with B1 only on this case var result = oView.getModel("demo").getProperty("/result"); if (result) { result.push.apply(result, data.b1); } else { result = data.b1; } }; oView.getModel("demo").setProperty("/result", result); oView.getModel("demo").setProperty("/fileURL", oController.srcFileURL); // Set Model to Table var oTable = oView.byId("tableImgClass"); oTable.setModel(oView.getModel("demo")); }; if (oControlEvent.getParameters().status === 200) { // get the response as JSON and process the results processResult(this, JSON.parse(oControlEvent.getParameters().responseRaw)); } else {</pre>

Explanation	Screenshot																					
	<pre> MessageToast.show("Error " + oControlEvent.getParameters().status + " : " + JSON.parse(oControlEvent.getParameters().responseRaw).error_description); } } </pre>																					
<p>Add the MessageToast definition at the beginning of the file.</p> <pre> "sap/m/MessageToast", , MessageToast </pre>	<pre> sap.ui.define(["sap/ui/core/mvc/Controller", "sap/m/MessageBox", "sap/m/MessageToast", "./utilities", "sap/ui/core/routing/History"], function (BaseController, MessageBox, MessageToast, Utilities, History) { "use strict"; </pre>																					
<p>Now the code should be complete.</p> <p>Run your SAP Fiori application to check all the new features.</p> <p>Press Browse and choose an image containing a shoe.</p> <p>The SimilarItems backed API is called and similar items result shown in the table.</p>	 <p>The screenshot shows a user interface with a selected image of a shoe (Fly_beige.PNG) and a 'Browse...' button. Below the image is a table of similar items:</p> <table border="1"> <thead> <tr> <th>Image</th> <th>Product Id</th> <th>Name</th> <th>Inventory Level</th> <th>Score</th> <th>Price</th> <th></th> </tr> </thead> <tbody> <tr> <td></td> <td>SH90018</td> <td>Fly London - YUPI - Beige</td> <td>0.999999999999997 58</td> <td>0</td> <td>190 GBP</td> <td>></td> </tr> <tr> <td></td> <td>SH90012</td> <td>Fly London Yipi Shoes - Camel</td> <td>0.90789772809509 02</td> <td>0</td> <td>180 GBP</td> <td>></td> </tr> </tbody> </table>	Image	Product Id	Name	Inventory Level	Score	Price			SH90018	Fly London - YUPI - Beige	0.999999999999997 58	0	190 GBP	>		SH90012	Fly London Yipi Shoes - Camel	0.90789772809509 02	0	180 GBP	>
Image	Product Id	Name	Inventory Level	Score	Price																	
	SH90018	Fly London - YUPI - Beige	0.999999999999997 58	0	190 GBP	>																
	SH90012	Fly London Yipi Shoes - Camel	0.90789772809509 02	0	180 GBP	>																

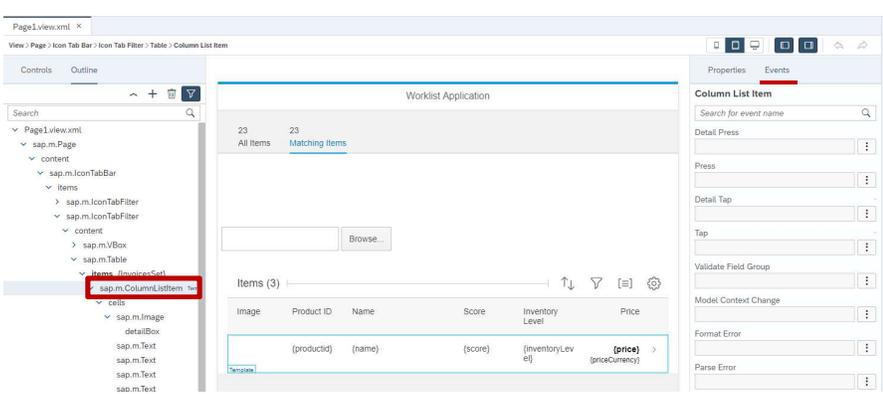
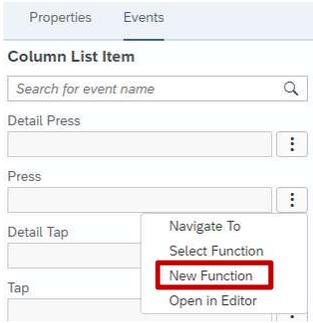
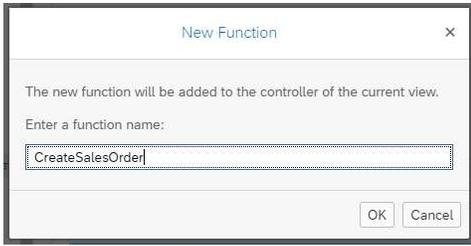
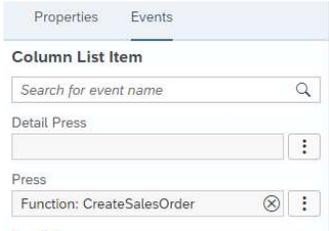
STEP 6: ADD A NEW SERVICE TO THE NODEJS APPLICATION

Let's add a new service to the NodeJS app that will create Sales Orders in the ERP system.

Explanation	Screenshot
<p>Go to the <code>smbmkt</code> folder you deployed before in Cloud Foundry.</p> <p>Open the <code>app.js</code> file with a Java Script editor (Visual Studio Code is an option).</p> <p>Add a post service called <code>/SalesOrders</code>.</p> <p>This service will call a function in the <code>biz</code> module.</p> <p>You can get the code from the following link: https://github.com/B1SA/exercise_BuildToWebIDECF/blob/master/extras/STEP%206/app_ext.txt</p>	<pre>app.post('/SalesOrders', function (req, res) { console.log("REQUEST: Create Sales Order") biz.CreateSalesOrder(req.body, function (response) { res.setHeader('Content-Type', 'application/json') res.status(201) res.send(response) }) });</pre>
<p>Open the <code>modules/biz.js</code> file.</p> <p>Add a function called <code>CreateSalesOrder</code>.</p> <p>You can get the code from the following link: https://github.com/B1SA/exercise_BuildToWebIDECF/blob/master/extras/STEP%206/biz_ext.txt</p>	<pre>function CreateSalesOrder(body, callback) { /* Receives a body with all items from each erp */ var fResp = {}; call = 0; for (key in body) { var re = PostErpSalesOrder(key, body[key]).then(function (salesOrder) { fResp[Object.keys(salesOrder)] = salesOrder[Object.keys(salesOrder)].values; call++; if (call == Object.keys(body).length) { callback(fResp) } }) } }</pre>

Explanation	Screenshot
<p>In the <code>erp/b1.js</code> file.</p> <p>Declare in <code>module.exports</code> the <code>PostSalesOrder</code> function.</p> <p>You can get the code from the following link: https://github.com/B1SA/exercise_BuildToWebIDE/commit/206/b1_ext.txt</p>	 <pre> 1 /* Service Layer module to interact with B1 Data */ 2 /* Server Configuration and User Credentials set in environment varia 3 /* Session and Node ID stored in Redis cache database */ 4 5 var client; // Redis Client 6 7 module.exports = { 8 GetItems: function (options, callback) { 9 return (GetItems(options, callback)) 10 }, 11 GetOrders: function (options, callback) { 12 return (GetOrders(options, callback)) 13 }, 14 PostSalesOrder: function (body, callback) { 15 return (PostSalesOrder(body, callback)) 16 }, 17 setClient: function (inClient) { client = inClient; } 18 } </pre>
<p>Go to your cmd line window.</p> <p>Run the <code>cf</code> command <code>cf push</code></p> <p>To upload the changes you did to your app to Cloud Foundry.</p>	 <pre> PS C:\... \smbmkt\smbmkt> cf push Using manifest file C:\... \smbmkt\smbmkt\manifest.yml Updating app smbmkt in org ..._trial / space dev as ...@sap.com... OK Uploading smbmkt... Uploading app files from: C:\... \smbmkt\smbmkt Uploading 32.1K, 40 files Done uploading </pre>
<p>You can test now your new service with Postman with the url of your app as follows:</p> <p>https://smbmkt-YOURAPP/SalesOrders</p>	 <p>POST https://.../cfapps... Params Send Save</p> <p>Authorization Headers (1) Body Pre-request Script Tests Code</p> <p>form-data x-www-form-urlencoded raw binary JSON (application/json)</p> <pre> 1 { 2 "b1": { 3 "lines": [4 { 5 "productid": "A00004", 6 "Quantity": 4 7 }, 8 { 9 "productid": "A00003", 10 "Quantity": 3 11 } 12] 13 }, 14 "byd": { 15 "lines": [16 { 17 "productid": "P120101", 18 "Quantity": 1 19 } 20] 21 } 22 } </pre> <p>Body Cookies Headers (8) Test Results Status: 201 Created Time: 4706 ms</p> <p>Pretty Raw Preview JSON Save Response</p> <pre> 1 { 2 "b1": [3 { 4 "OrderID": 13192, 5 "OrderNum": 13317, 6 "OrderDate": "2018-07-05", 7 "OrderCur": "\$", 8 "OrderAmt": 5075 9 } 10], 11 "byd": [12 { 13 "OrderID": "3403", 14 "OrderNum": "00163E5EB5C51ED&A08A8EB95E8A3B11", </pre>

STEP 7: CALL THE NEW NODEJS SERVICE FROM YOUR SAP FIORI APP

Explanation	Screenshot
<p>Open the Page1.view.xml file with the Layout Editor.</p> <p>Open the Outline tab and select sap.m.ColumnListItem control we worked on previously.</p> <p>Select the Events tab (right side).</p> <p>Click on the  button corresponding to the Press event.</p>	
<p>Choose the New Function option.</p>	
<p>Enter CreateSalesOrder as function name.</p> <p>Press OK.</p>	
<p>The new function will be indicated inside the Press event.</p>	

Explanation	Screenshot
<p>Open the Page1.controller.js file.</p> <p>A new empty function has been automatically created based on our last step.</p>	<pre> CreateSalesOrder: function (oEvent) { //This code was generated by the layout editor. } </pre>
<p>Let's implement this function to call our smbmk backend nodejs /SalesOrder service.</p> <p>We use here the destination pointing to our smbmk backend.</p>	<pre> CreateSalesOrder: function (oEvent) { // Get Data from ODataModel V4 /Orders var body = { "b1": { "lines": [{ "productid": oEvent.getSource().getBindingContext().getObject().productid, "Quantity": 1 }] } }; \$.ajax({ url: "/smbmk_workshopNice/SalesOrders", type: "POST", data: JSON.stringify(body), contentType: "application/json", success: function (data) { MessageToast.show("B1 SalesOrder number " + data.b1[0].OrderNum + " created."); }, error: function (jqXHR, textStatus, errorThrown) { MessageToast.show("POST SalesOrders error: " + JSON.stringify(jqXHR.responseJSON)); } }); } </pre>
<p>You can get the code from the following link:</p> <p>https://github.com/B1SA/exercise_BuildToWebIDE/CF/blob/master/extras/STEP%207/Page1.controller.js_ext.txt</p> <p>Replace smbmk_destination with your smbmk destination name.</p>	<pre> CreateSalesOrder: function (oEvent) { // Get Data from ODataModel V4 /Orders var body = { "b1": { "lines": [{ "productid": oEvent.getSource().getBindingContext().getObject().productid, "Quantity": 1 }] } }; \$.ajax({ url: "/smbmk_destination/SalesOrders", type: "POST", data: JSON.stringify(body), contentType: "application/json", success: function (data) { MessageToast.show("B1 SalesOrder number " + data.b1[0].OrderNum + " created."); }, error: function (jqXHR, textStatus, errorThrown) { MessageToast.show("POST SalesOrders error: " + JSON.stringify(jqXHR.responseJSON)); } }); } </pre>

Congratulations! You have just implemented your first full stack loosely coupled extension!

www.sap.com/contactsap

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See <http://www.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.

