Develop SAP Business One extensions on the SAP Cloud Platform

Run Simple

TABLE OF CONTENTS

PREREQUISITES **4**

i. Download and Install Development Tools **4**

ii. Create a SAP Cloud platform trial account **5**

iii. Activate Web IDE Full Stack service **6**

iv. SAP API Business Hub **9**

STEP 1: CREATE A SAP FIORI APP CONNECTING TO SAP BUSINESS ONE SERVICE LAYER VIA SAP API BUSINESS HUB **10**

i. Create a SAPUI5 Application **11**

ii. Add a Data Source to the SAPUI5 Application **15**

iii. Create a Model **17**

iv. Add controls to the View1 view. **18**

Add a sap.m.Table control ***18***

Add a Search Field control to the sap.m.Table ***23***

v. Add a second view called Details **27**

Create a Details view. ***27***

Add an Object Header control ***28***

vi. Define navigation between View1 and Details. **31**

STEP 2: CREATE A NODEJS APP **33**

STEP 3: DEPLOY THE NODEJS APP INTO SAP CLOUD FOUNDRY **34**

STEP 4: INTEGRATE THE NODEJS APP INTO THE SAP FIORI APP **38**

i. Add a Button "Add Freight" to the Details view. **38**

i. Implement the Button business logic calling the NodeJS server side and Service Layer **40**

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 Fiori application connecting to SAP Business One Service Layer via SAP API Business Hub
- Step 2: Implement a server side NodeJS application
- Step 3: Deploy the NodeJS application to SAP Cloud Foundry
- Step 4: Integrate the server side NodeJS application into the Fiori application created in Step 1

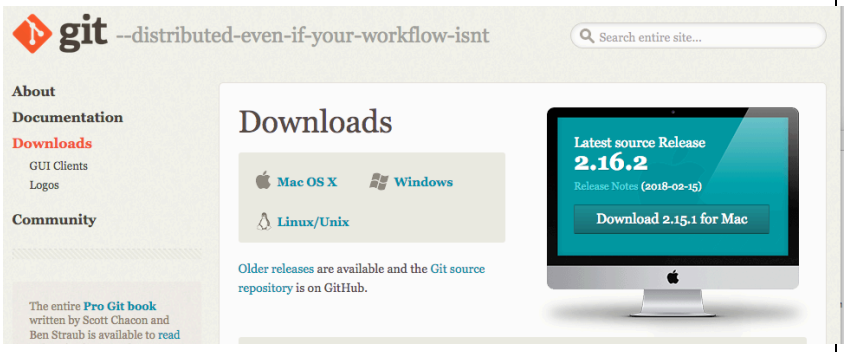
This hands-on exercise will require several steps, please follow them in the proposed order as each step is counting on the precedent steps.

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/cloudfou
# ...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

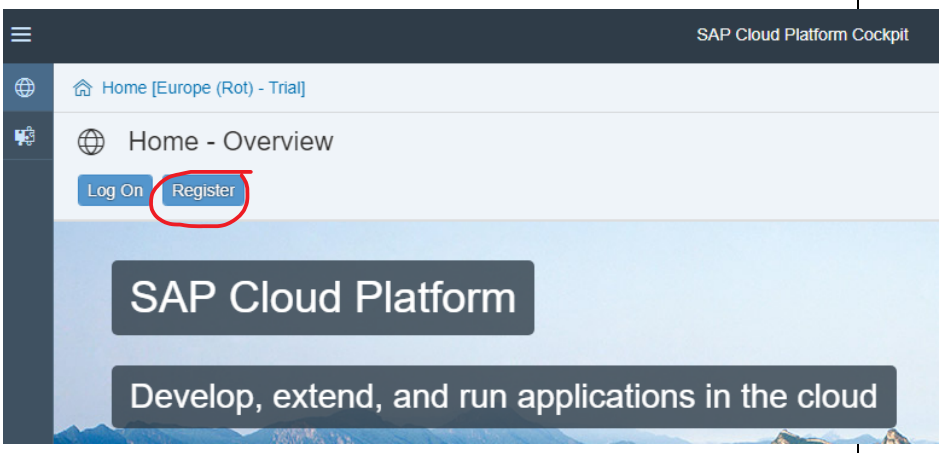
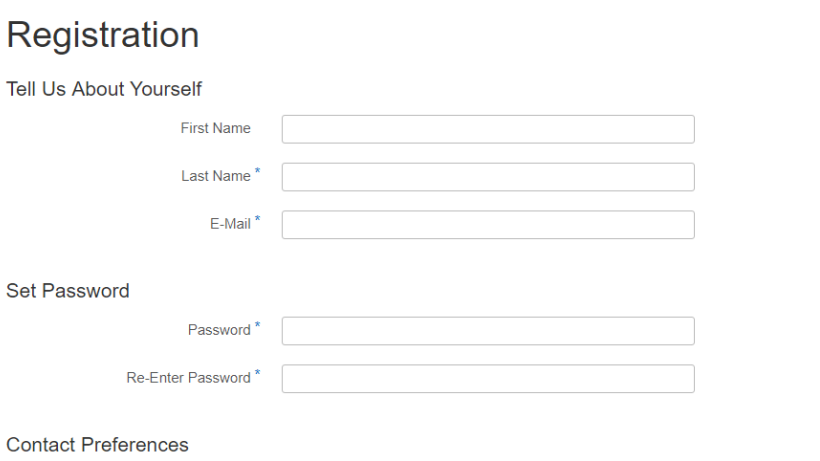
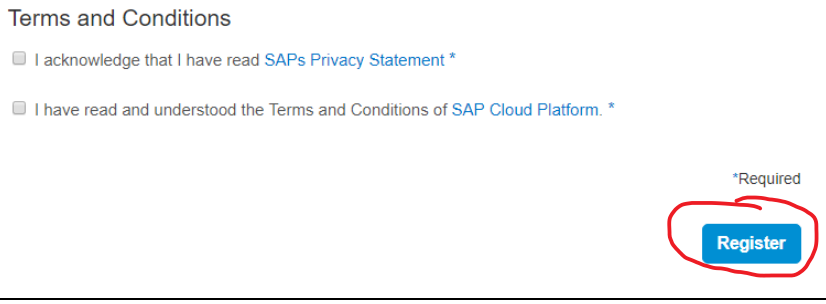
	Mac OS X 64 bit	Windows 64 bit	Linux 64 bit
Installers	pkg	zip	rpm / deb
Binaries	tgz	zip	tgz

ii. Create a SAP Cloud platform 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.

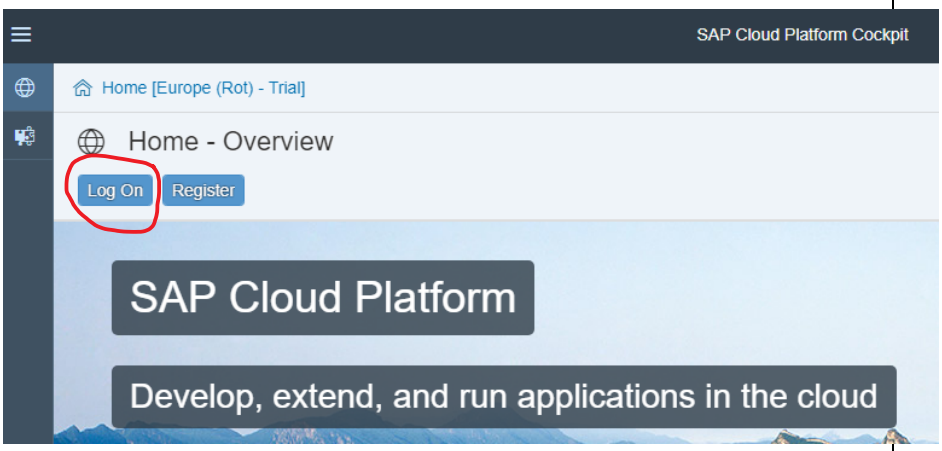
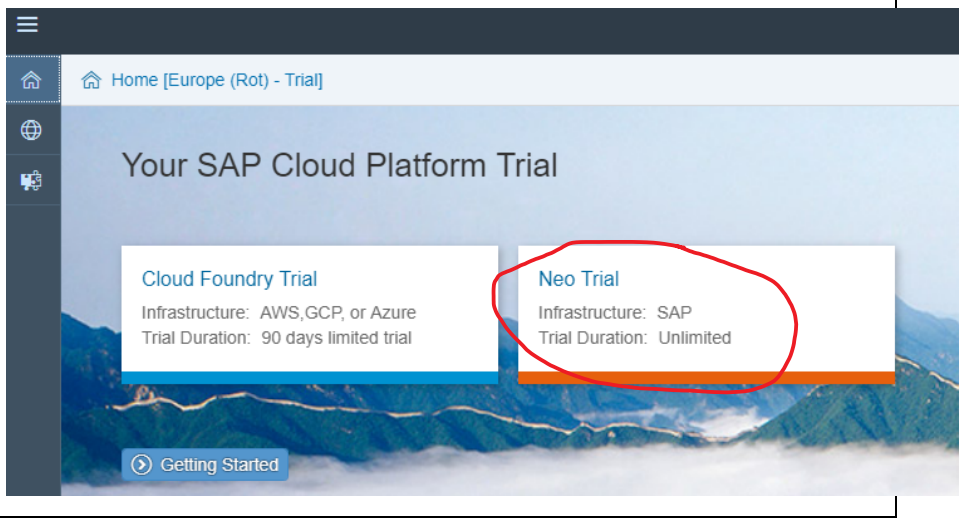
To create a trial SAP Cloud Platform account, go to the following link:

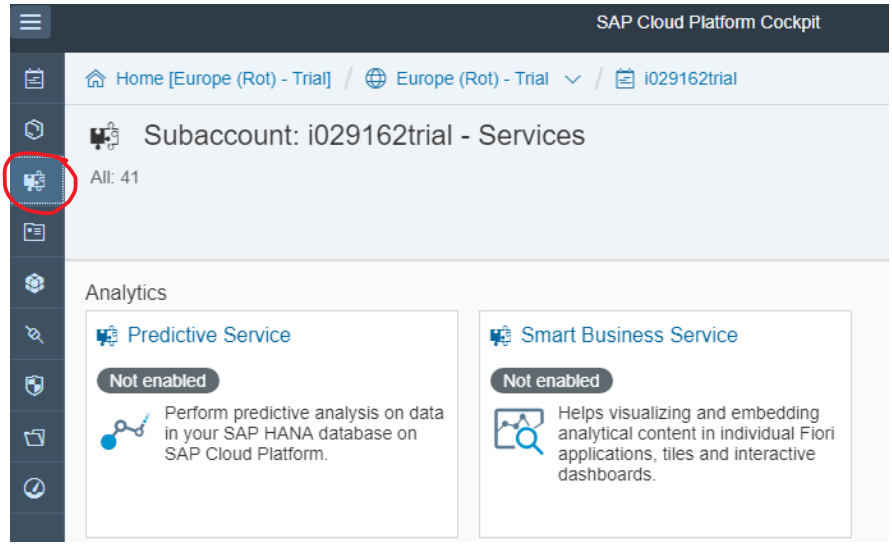
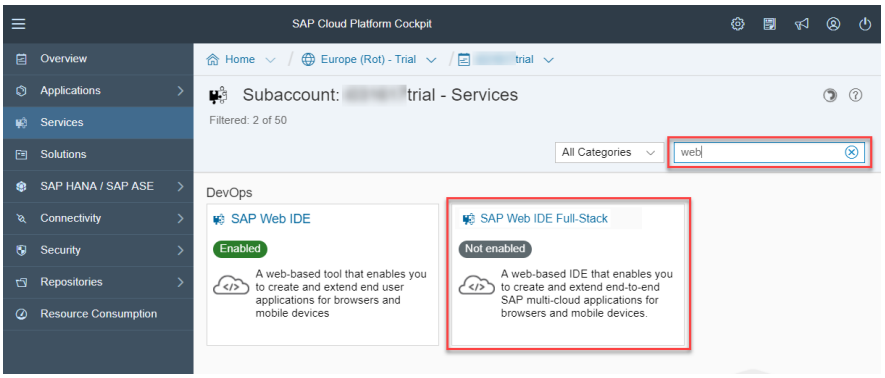
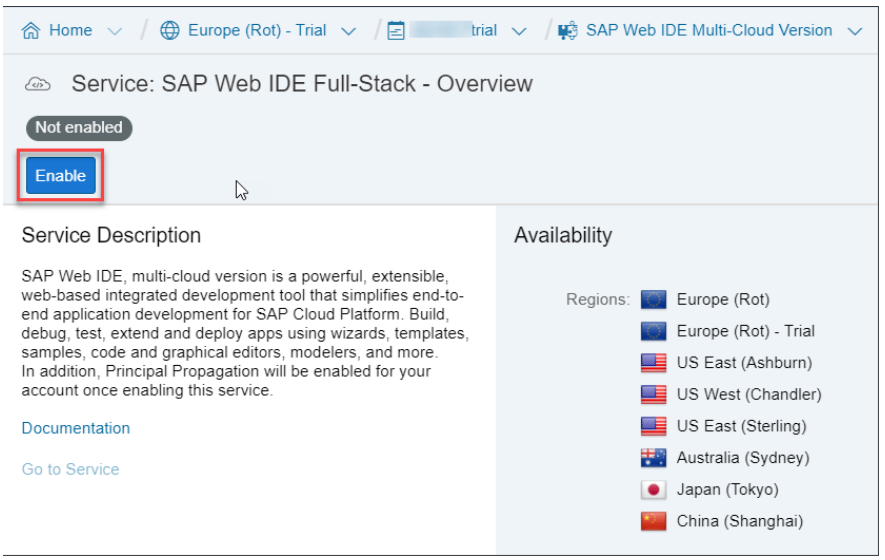
Explanation	Screenshot
To create a trial SAP Cloud Platform account, go to the following link: https://account.hanatrial.ondemand.com Press the Register button	
Enter all your details	
Accept the terms and conditions by checking both check boxes and press “Register”.	

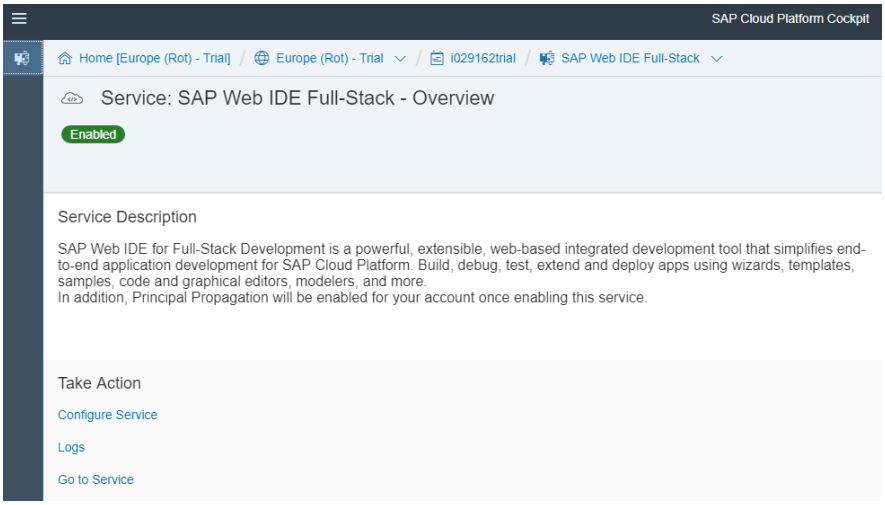
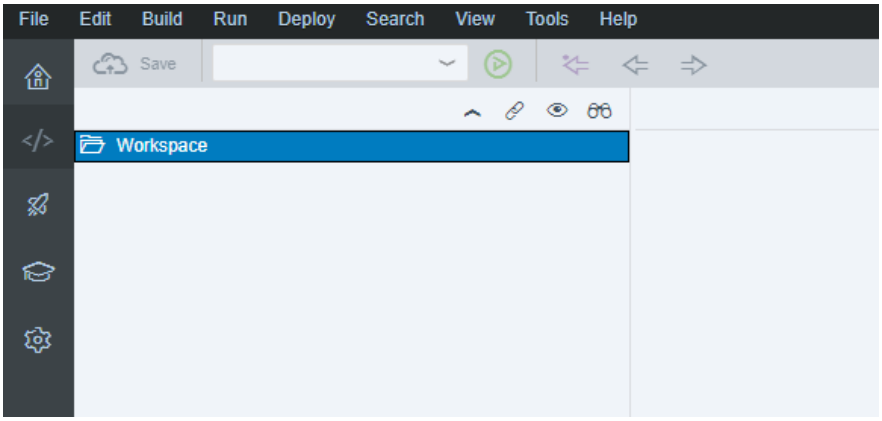
iii. Activate Web IDE Full Stack service

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
Open your trial SAP Cloud Platform account from the following link: https://account.hanatrial.ondemand.com	
Press the Log On button if you are not automatically logged in	
After login if you are proposed between Cloud Foundry Trial and Neo Trial please choose Neo Trial.	

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
Once Enabled select the link **Go to Service** to open Web IDE Full Stack.	
Web IDE opens with an empty Workspace unless you already developed applications with Web IDE in the past.	

iv. SAP API Business Hub

With SAP API Business Hub, you can easily build sample apps, extensions, and integrations. Discover and consume digital content packages with APIs, pre-packaged integrations, and sample apps from SAP and select partners.

Please go to the following link for more details:
https://api.sap.com

STEP 1: CREATE A SAP FIORI APP CONNECTING TO SAP BUSINESS ONE SERVICE LAYER VIA SAP API BUSINESS HUB

The objective of this first exercise is to develop a SAP Fiori app using the **SAP UI5** template.

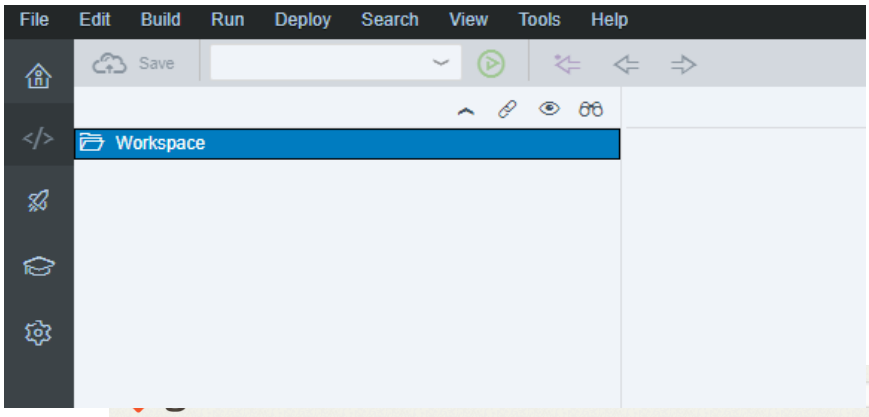
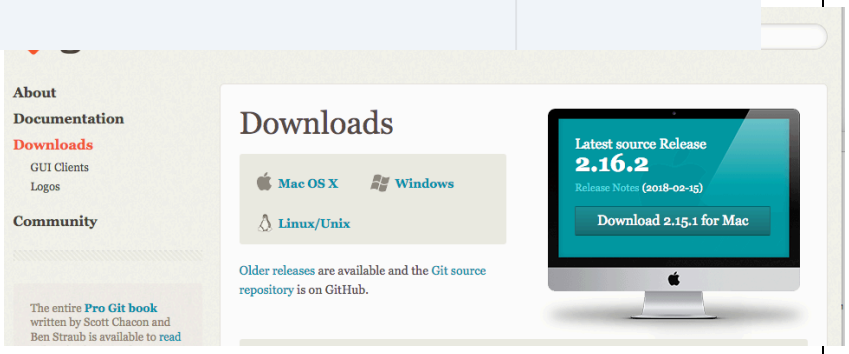
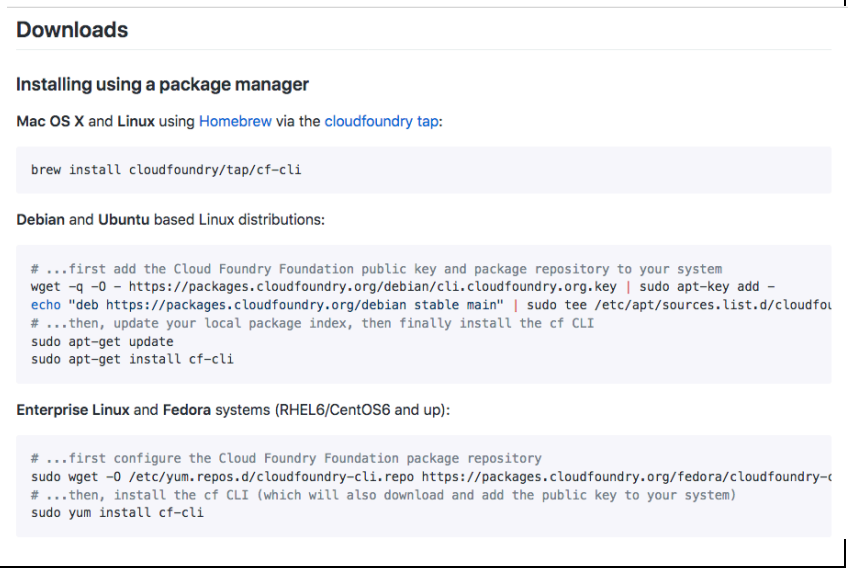
Service Layer provides OData v4 support since SAP Business One 9.3 PL04 version for SAP HANA.

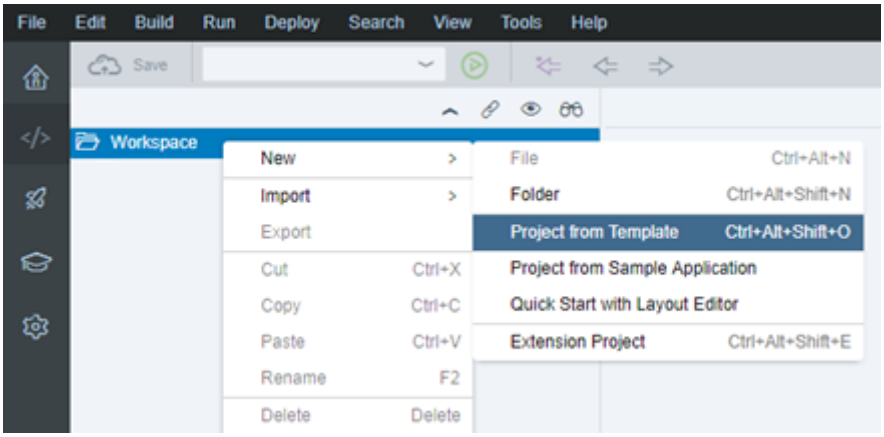
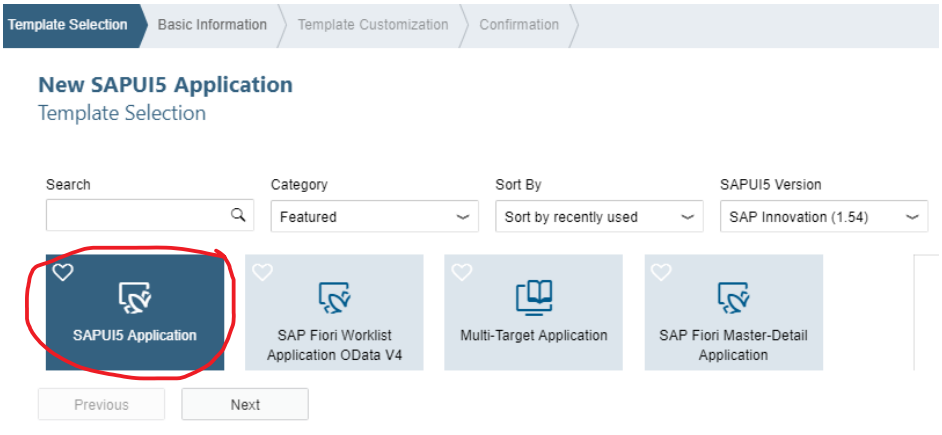
In this exercise, we will use the Service Layer APIs exposed via the SAP API Business Hub, please refer to the prerequisites section

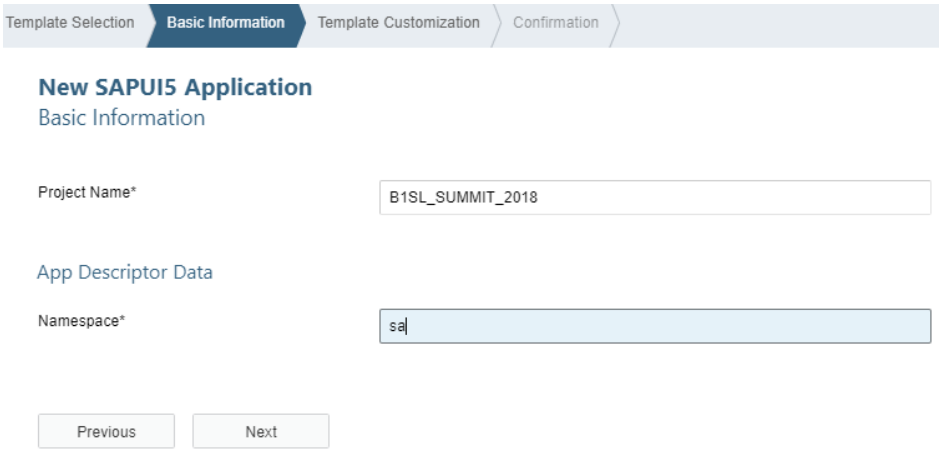
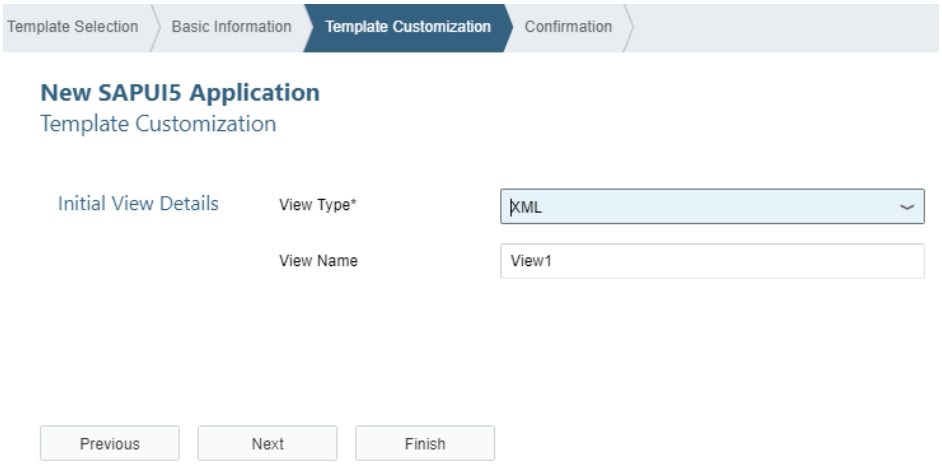
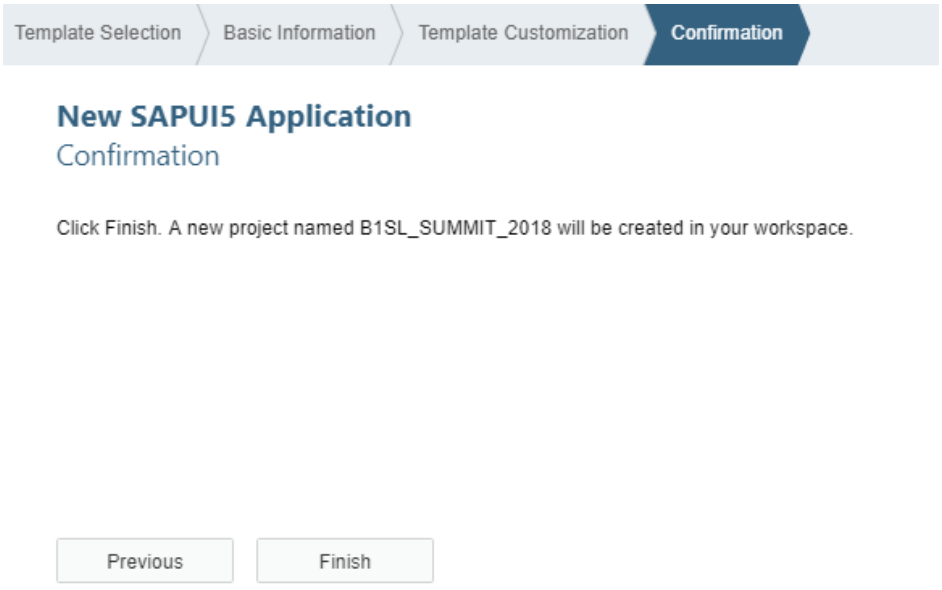
SAP API Business Hub for more details.

Web IDE supports OData v4 on some templates like SAP Fiori Worklist Application OData v4 and SAPUI5. More templates will support OData v4 gradually. In this exercise, we will use the SAPUI5 template.

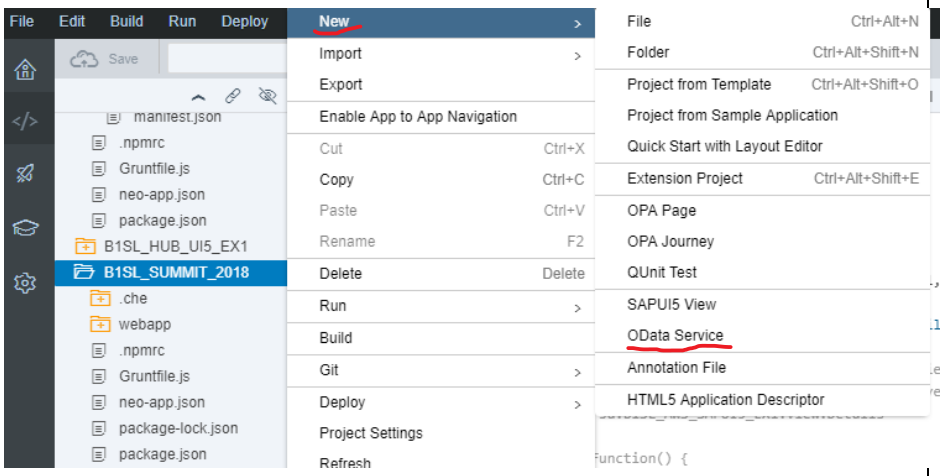
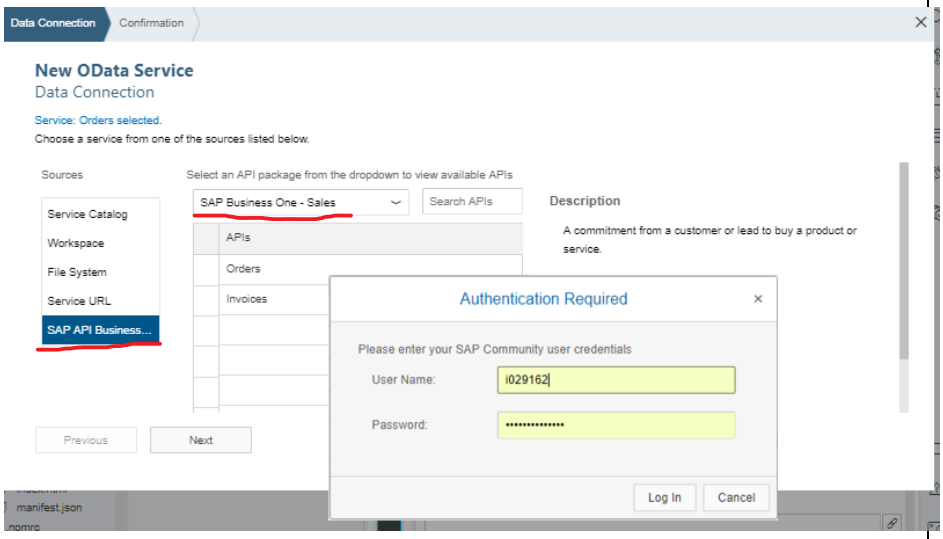
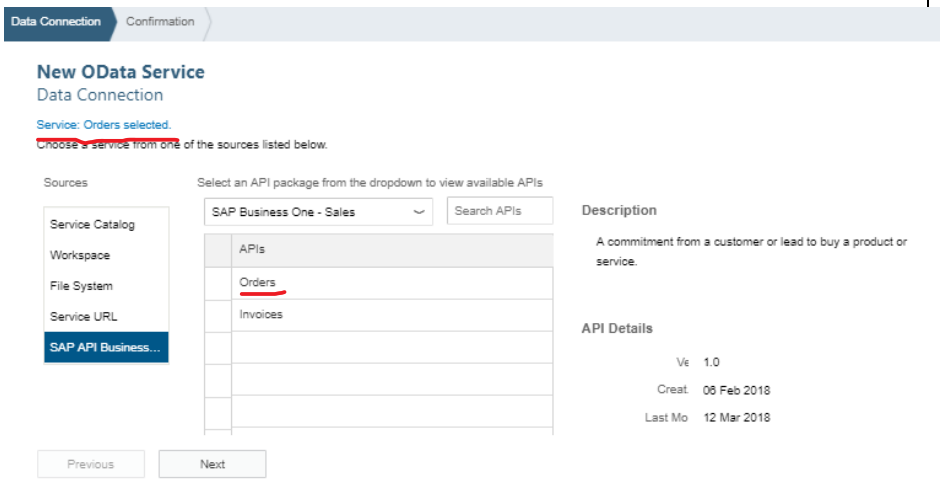
i. Create a SAPUI5 Application

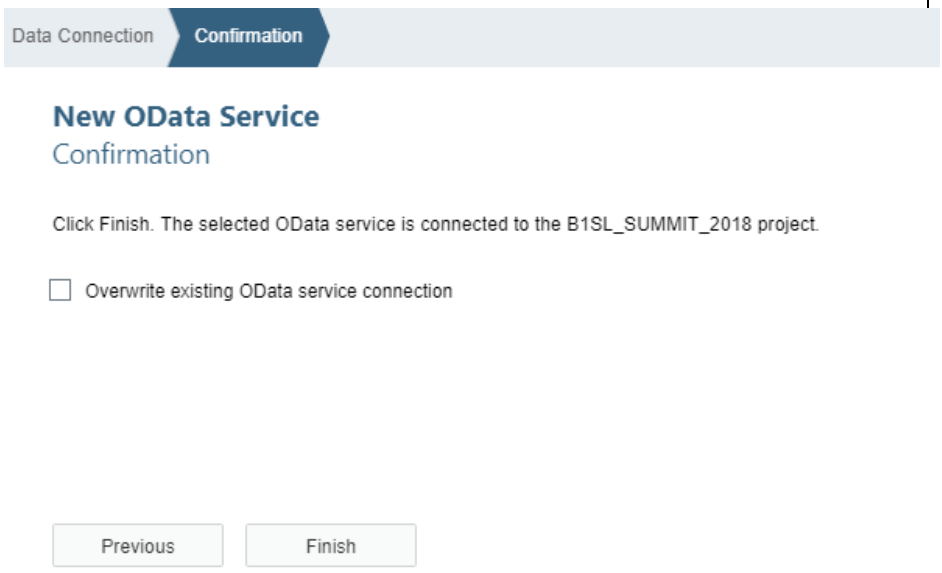
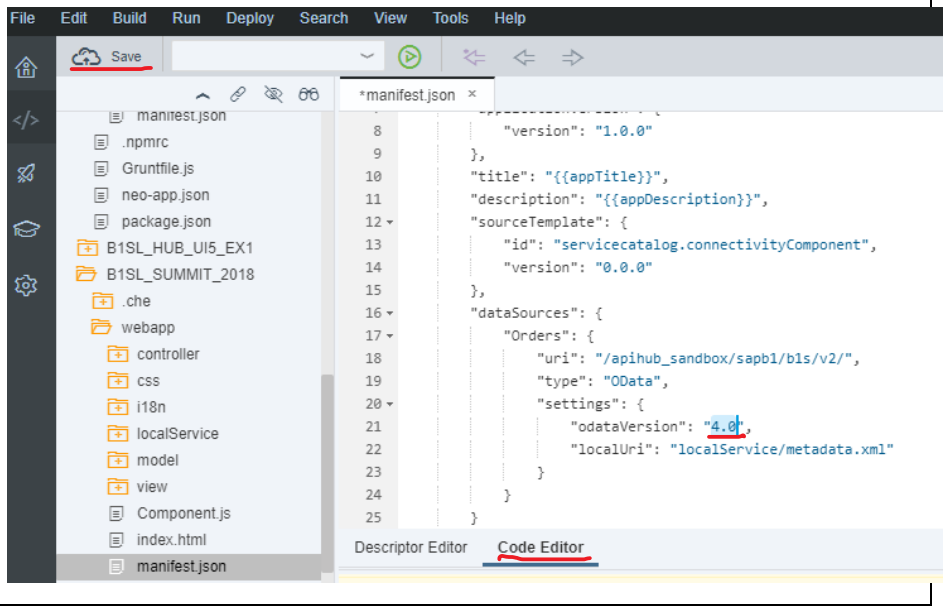
Explanation	Screenshot
Open SAP Web IDE Full Stack. **ii. Check the prerequisites sections Download and Install Development Tools**	
Download and install git v control on your system fr following link https://git-scm.com/down	
We will also make use of Cloud Platform Cloud Fo Environment. To do so, we need the C Foundry command line ir (CLI) You can download it and if the CF CLI for your ope system on. https://github.com foundry/cli#downlo	

Explanation	Screenshot
	Installers and compressed binaries | | Mac OS X 64 bit | Windows 64 bit | Linux 64 bit | | --- | --- | --- | --- | | Installers | pkg | zip | rpm / deb | | Binaries | tgz | zip | tgz |
Create a SAP Cloud platform trial account and Activate Web IDE Full Stack service if you don't know how to open Web IDE Full Stack.	
Right click on your Workspace and select **New -> Project from Template**.	
Select the **SAPUI5 Application** template. Press **Next**. If you don't see this template, change the Category to **All categories**.	

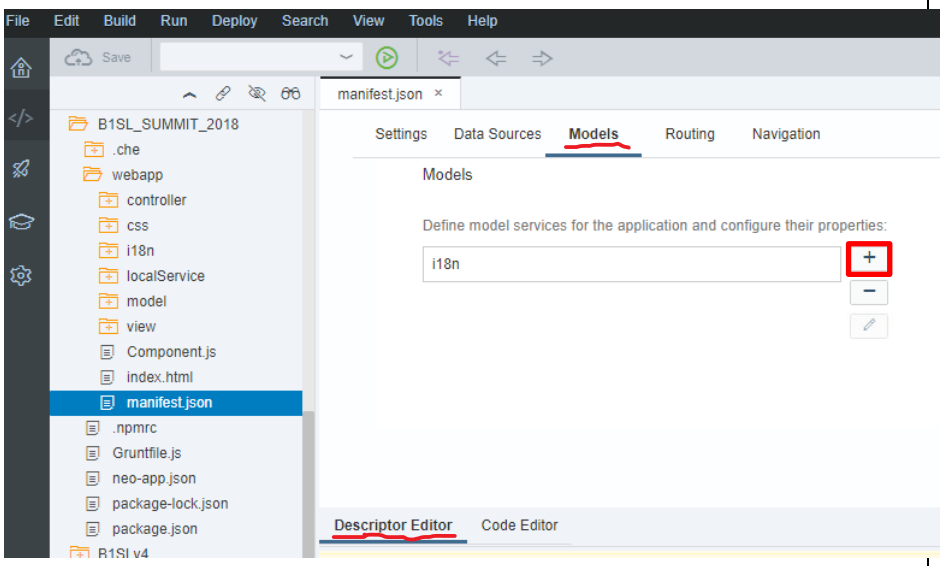
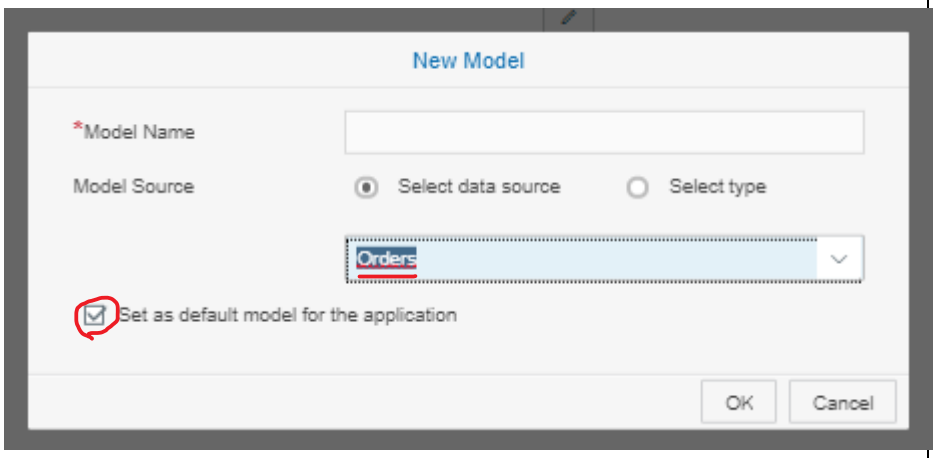
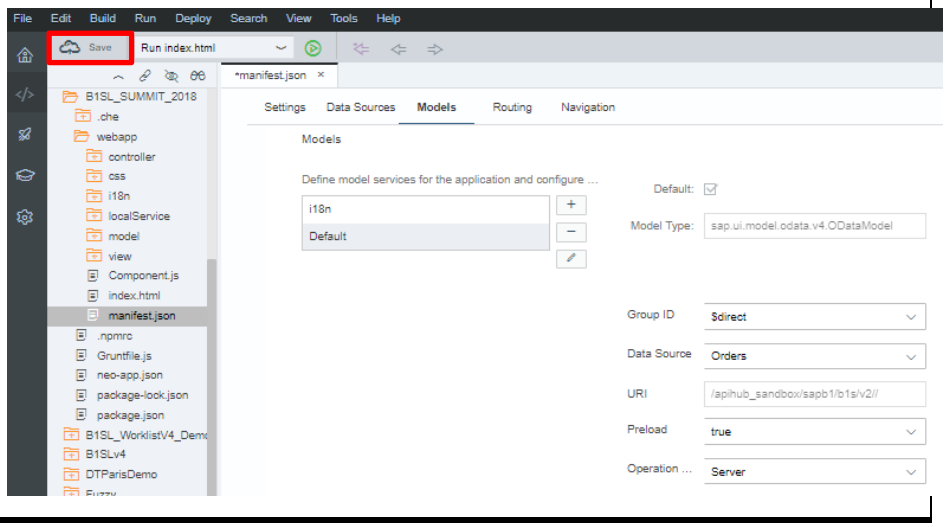
Explanation	Screenshot
Enter a **Project Name**. Enter a **Namespace**. Press **Next**.	
Keep the **Initial View Details** with the **default** values. Press **Next**.	
Press **Finish** to confirm the creation of the SAPUI5 template.	

iii. Add a Data Source to the SAPUI5 Application

Explanation	Screenshot
In Web IDE select your project and right click to get the menus. Select **New -> OData Service**.	
Select **SAP API Business Hub** in the Sources (left side of the screen). Select the "**SAP Business One – Sales**" API package from the dropdown control. You might be prompted to enter your SAP Community User Name and Password. Enter your credentials and press Log In.	
From the available APIs presented select **Orders** and make sure the "**Service: Orders selected**" blue message is shown at the top of the Data Connection tab. Press **Next**.	

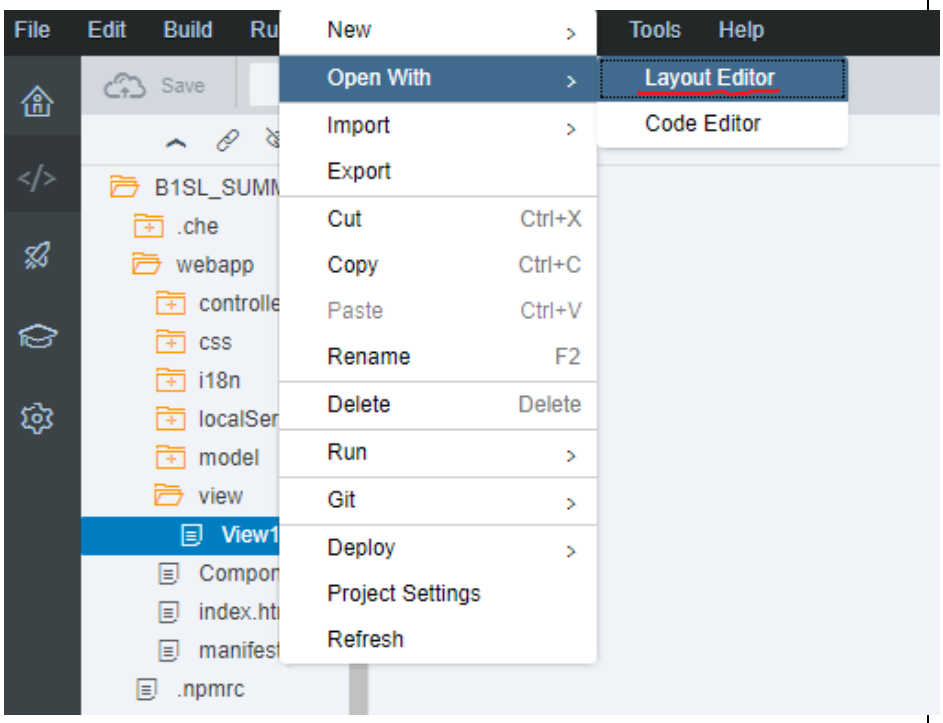
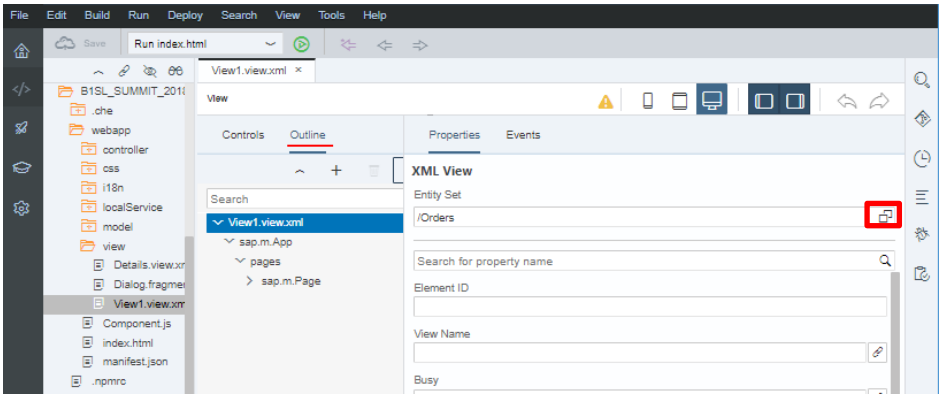
Explanation	Screenshot
Press **Finish**.	
Open the file "**webap/manifest.json**" inside your application and select the **Code Editor**. Search for the element "dataSources"/"Orders" and change the "odataVersion" value inside "settings" to "4.0". **Save** your changes.	

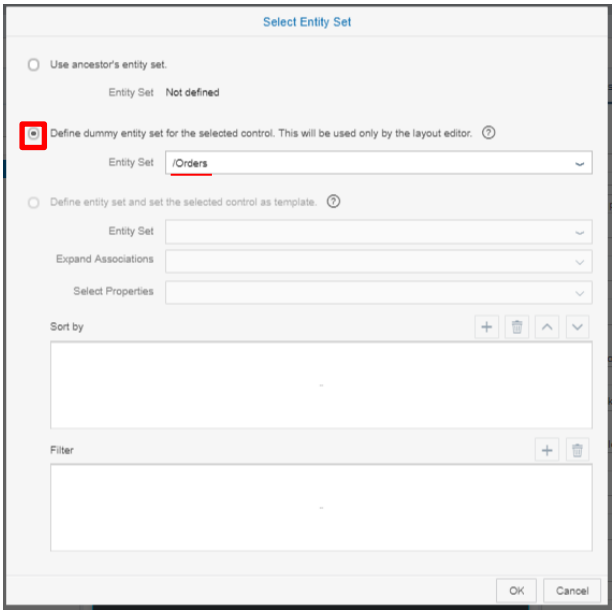
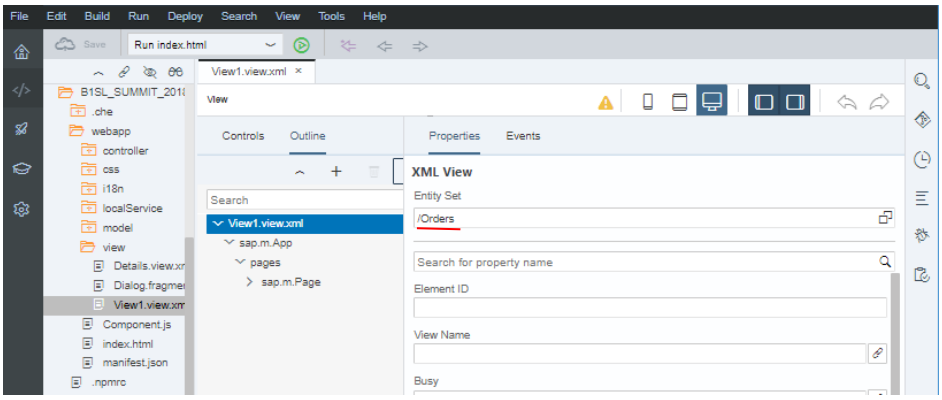
iv. Create a Model

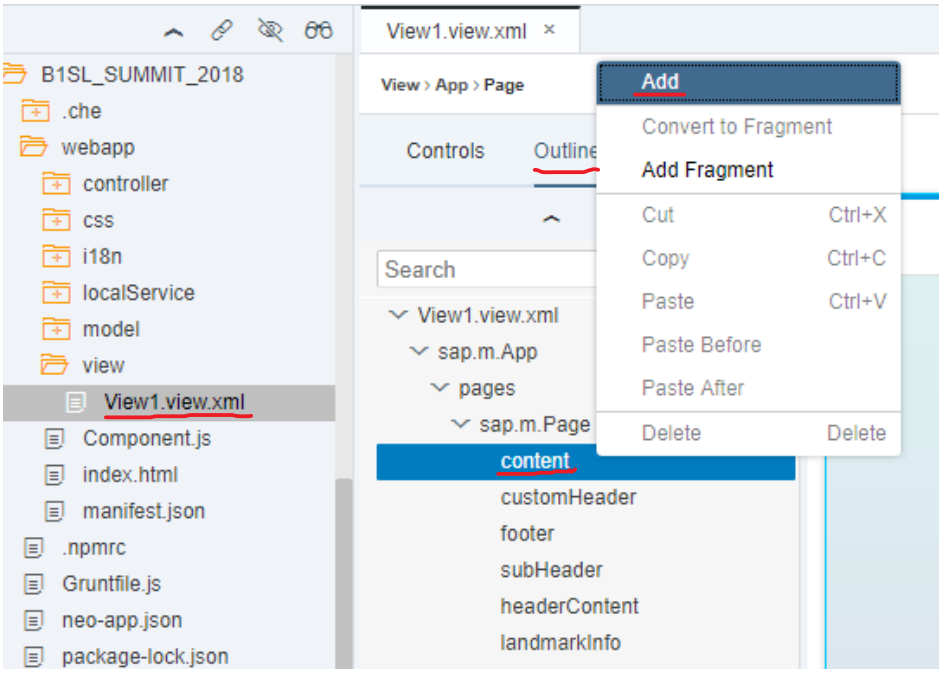
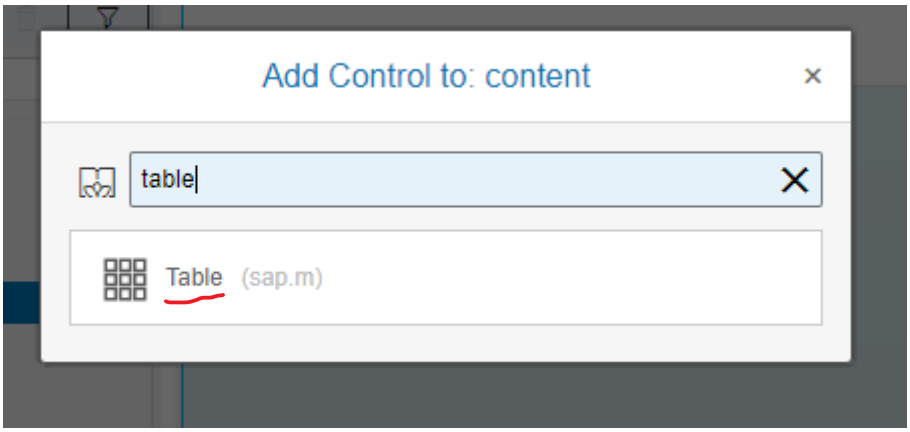
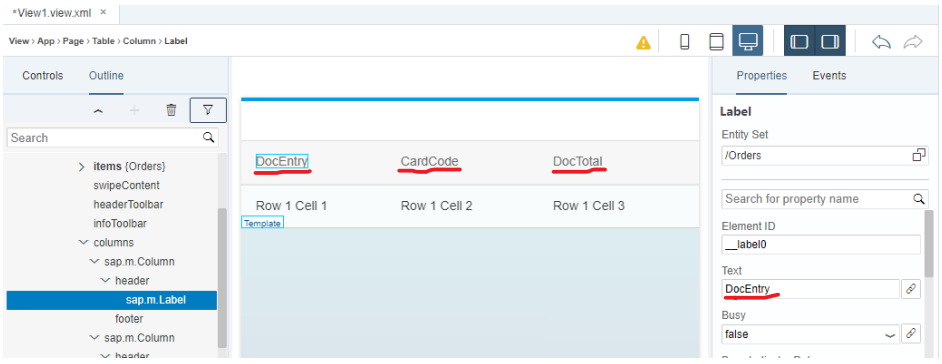
Explanation	Screenshot
Open the **manifest.json** file, select **Descriptor Editor** (bottom of the screen) and **Models tab**. Press the **+** button to add a new model.	
Check the box "**Set as default model for the application**". There is no need to enter a Model Name as we checked the default model option. Select **Orders** as the **Model Source**. Press **OK**.	
A new **Default** model will be shown in the Models tab pointing to the API Hub URI for OData v4 Data Source. Press **Save** button to your changes.	

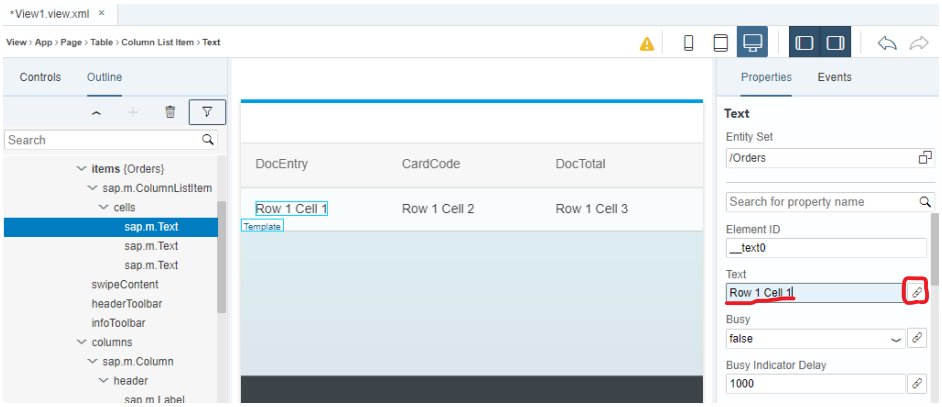
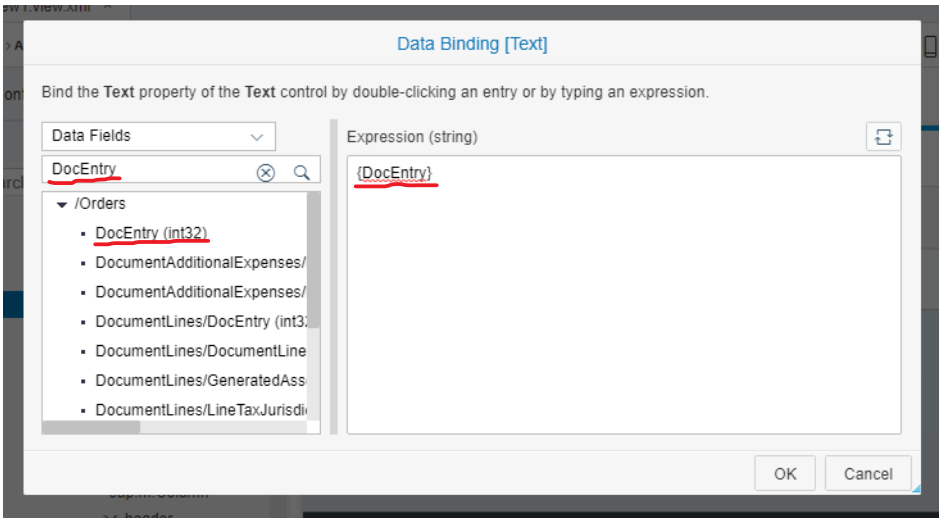
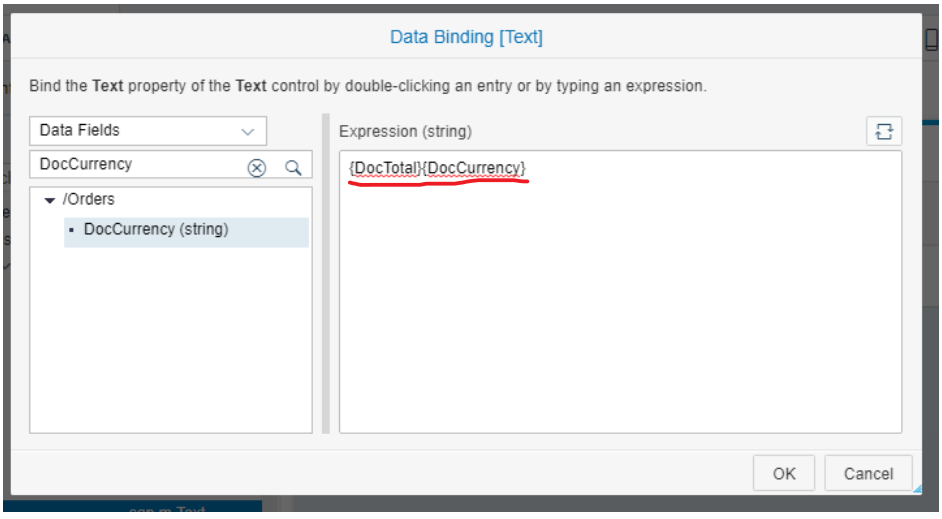
v. Add controls to the View1 view.

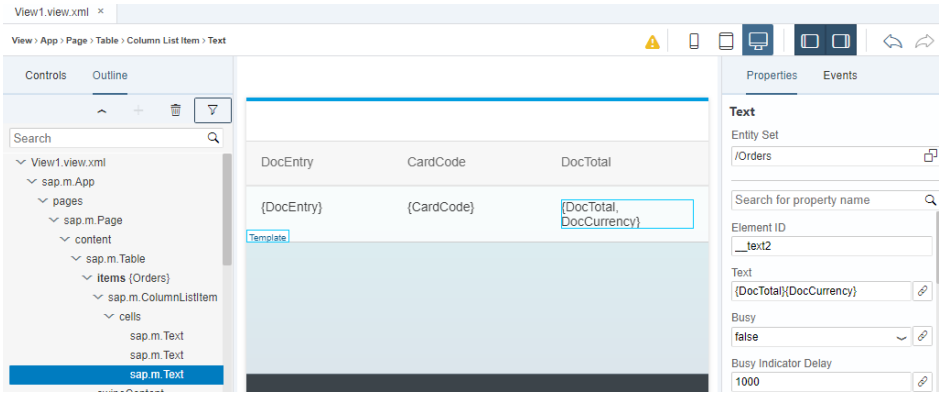
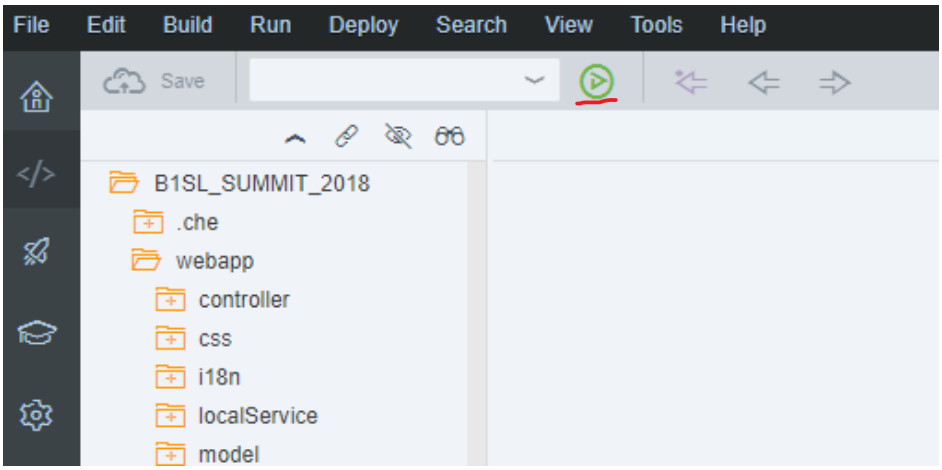
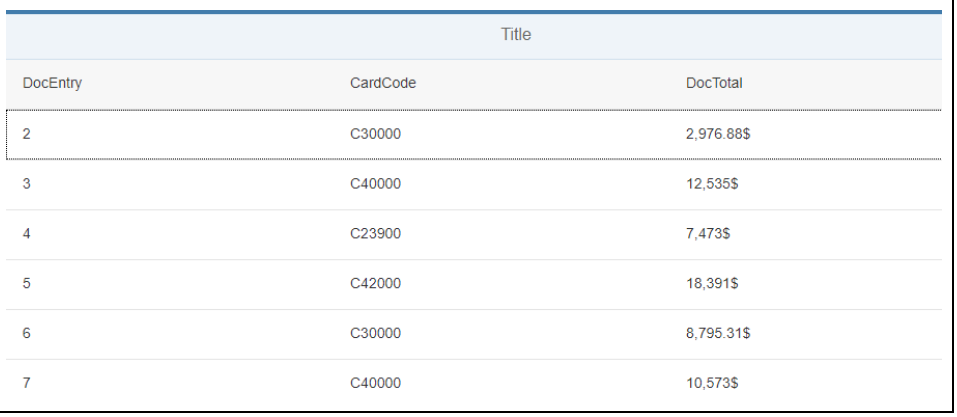
Add a sap.m.Table control

Explanation	Screenshot
Open the **view folder**. Open the **View1.view.xml** file with the **Layout Editor**.	
Select the **Outline** tab. Select **View.view.xml**. Click on the  button to select the **Entity Set** associated to our view.	

Explanation	Screenshot
Check the second option **Define dummy entity set…** and choose the **/Orders** Entity Set. Press **OK**.	
Once back to the Layout Editor the **/Orders** entity should be shown. Press **Save** button.	

Explanation	Screenshot
Expand **sap.m.Page**. **Right click** on **content** and select **Add**.	
Enter **table** to filter the controls shown in the list. **Click** on the **Table** control so it will be added to our view.	
On the table created edit each **sap.m.Column -> header -> sap.m.Label Text** Property: - Change Header 1 by **DocEntry**. - Change Header 2 by **CardCode**. - Change Header 3 by **DocTotal**.	

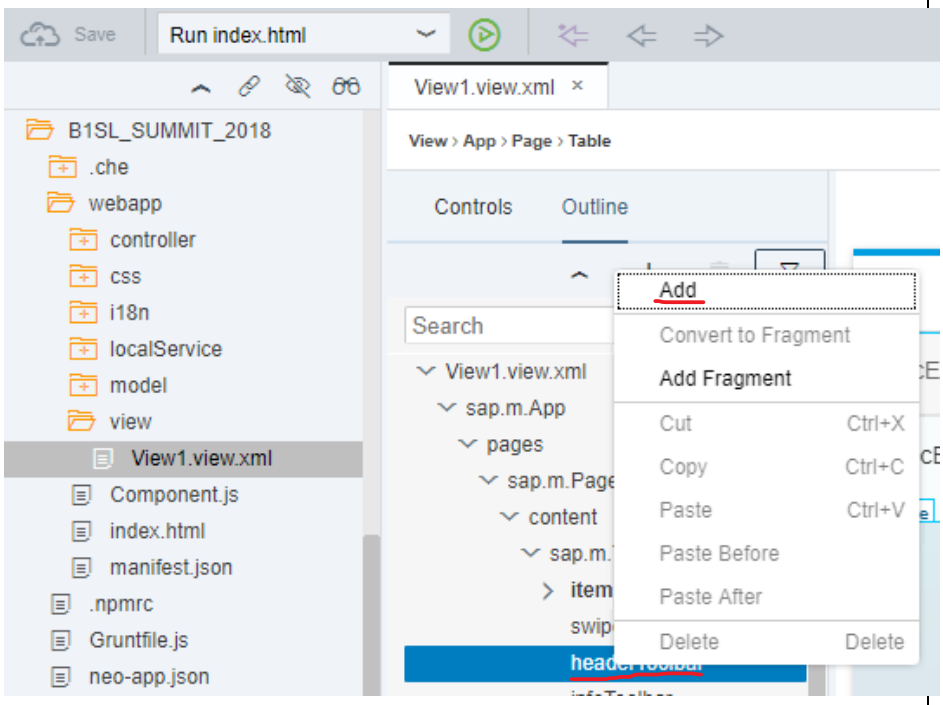
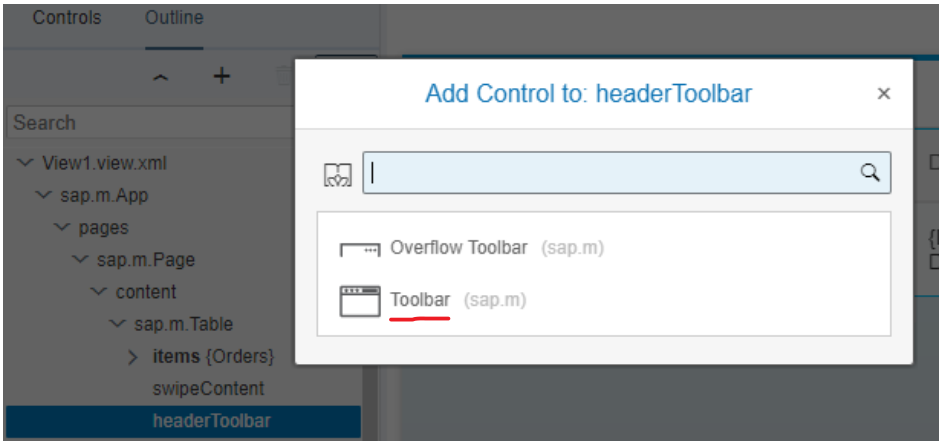
Explanation	Screenshot
Select each one of the **sap.m.ColumnListItem -> sap.m.Text** cells to bind them to their corresponding Orders properties. Then on the **Text property** click the **Binding button** on the right side of the Text property.	
Delete the default Expression string. In the Search for data field enter the name of the Orders property **DocEntry**. **Double click on the field name** so it will be copied to the Expression string between curly brackets. **Repeat this step for CardCode column**.	
For the **DocTotal** column we will add 2 properties: - **DocTotal** - **DocCurrency** We will then search and add first the **DocTotal** and afterwards search and add the **DocCurrency**. Press **Save** button.	

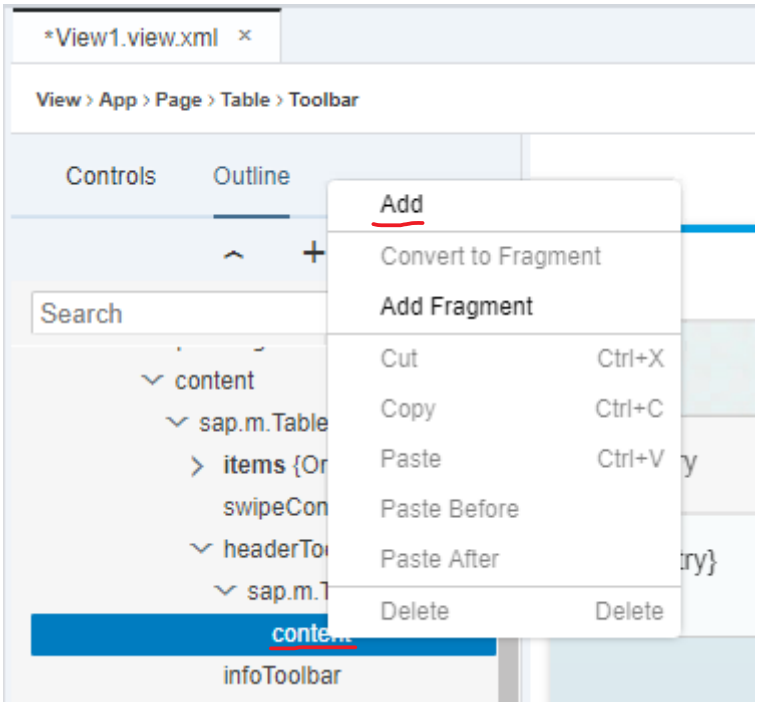
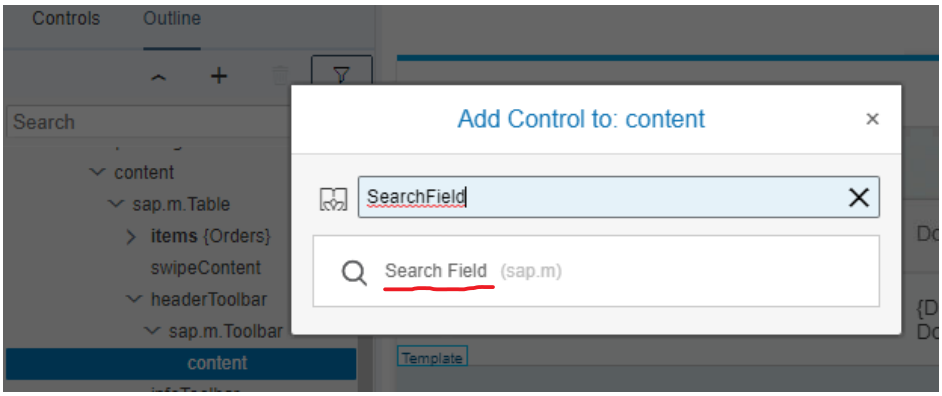
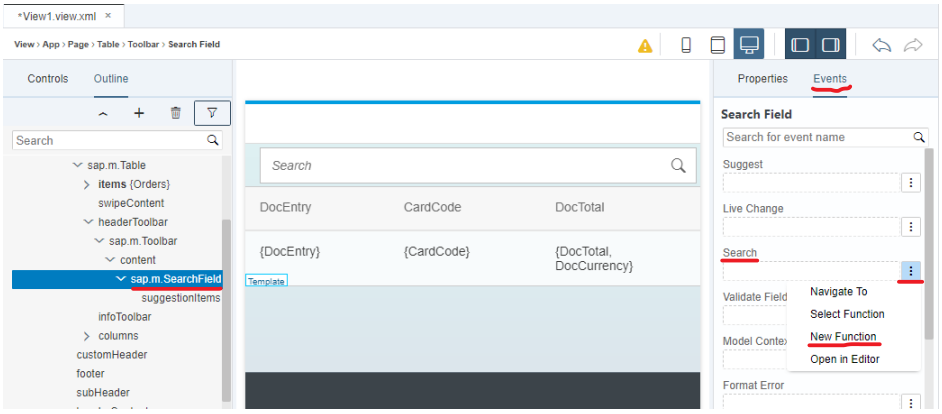
Explanation	Screenshot
Your **View1.view.xml** layout should look like this at this point.	
Click on the green arrow to run your application.	
A new tab opens showing your application. You can see the Table is automatically filled with the list of Orders available in the API Business Hub sandbox running SAP Business One.	

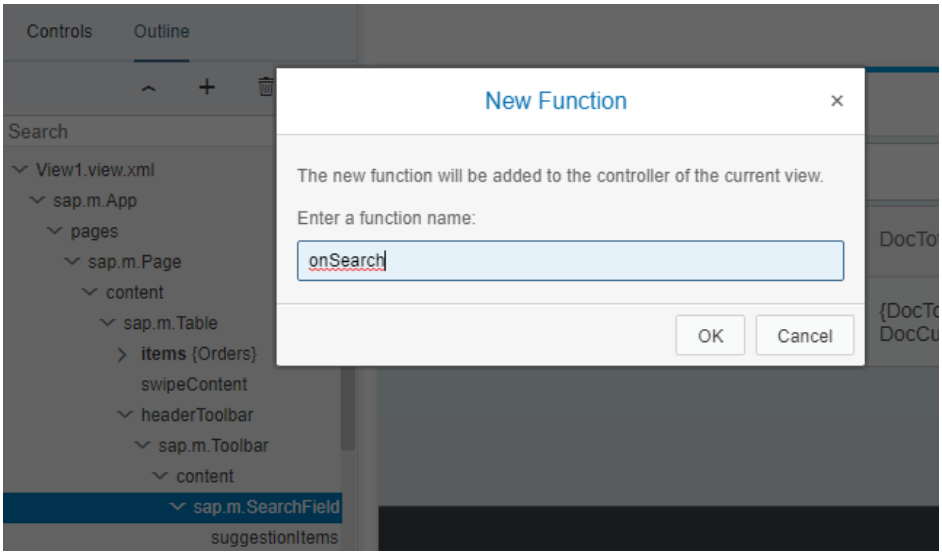
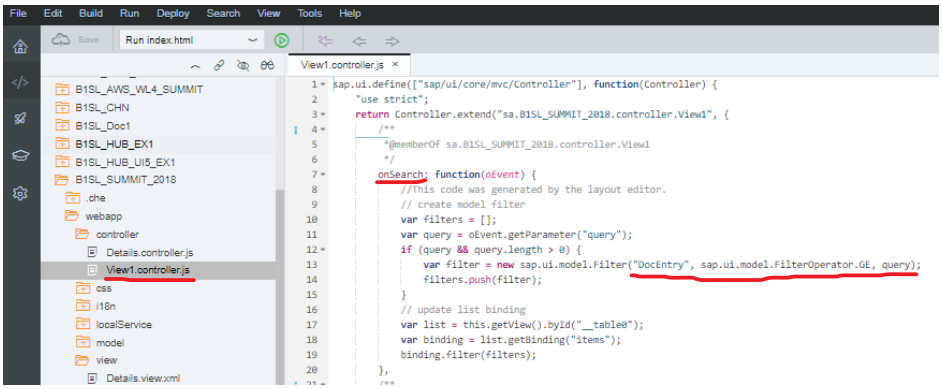
Title

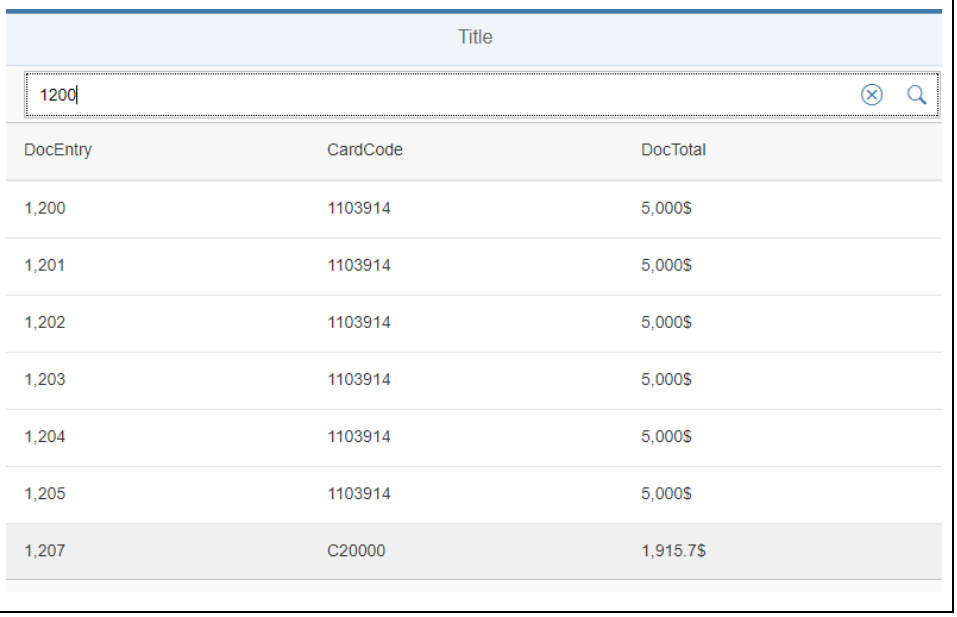
DocEntry	CardCode	DocTotal
2	C30000	2,976.88$
3	C40000	12,535$
4	C23900	7,473$
5	C42000	18,391$
6	C30000	8,795.31$
7	C40000	10,573$

Add a Search Field control to the sap.m.Table

Explanation	Screenshot
Open the **View.view.xml** file with the Layout Editor. Go to the **Outline** tab and select **sap.m.Table -> headerToolbar**. **Right click** to get the context menus and select **Add**.	
Double click on the **Toolbar** proposed control.	

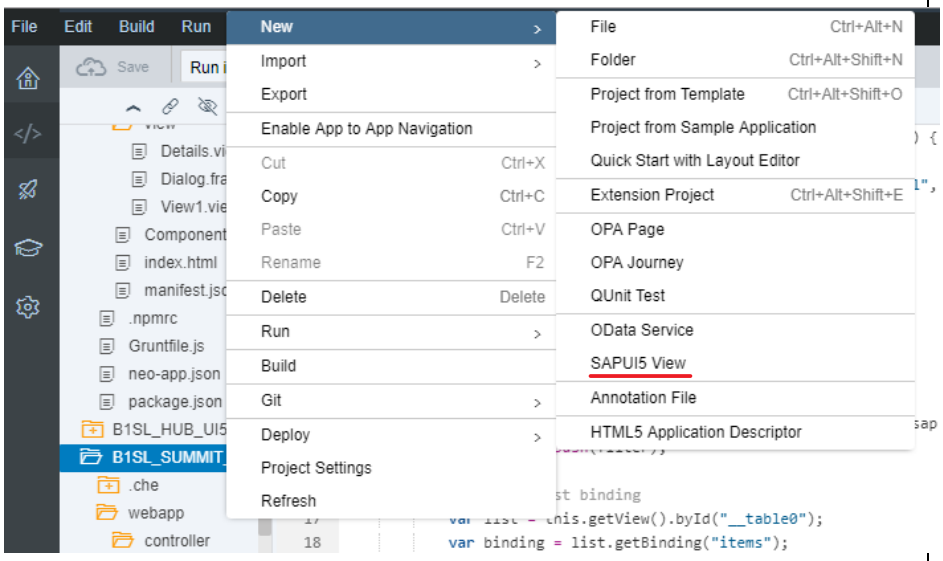
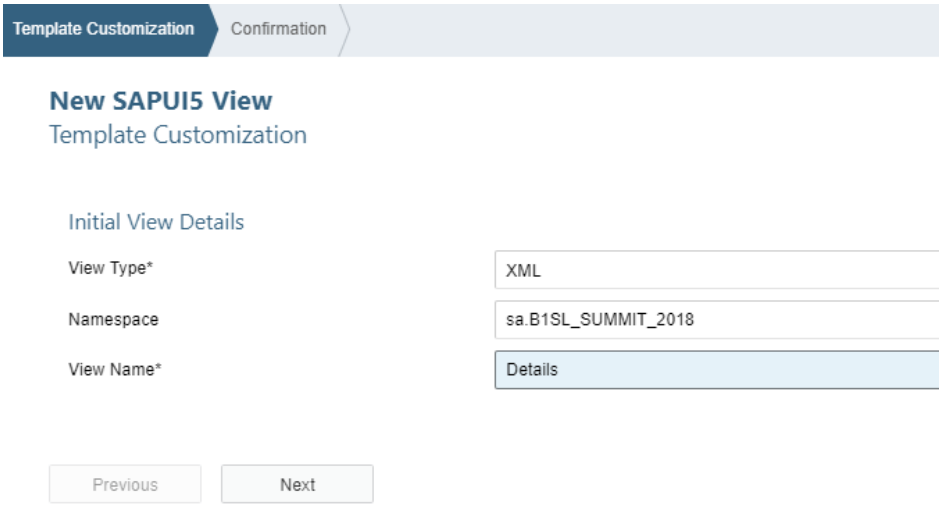
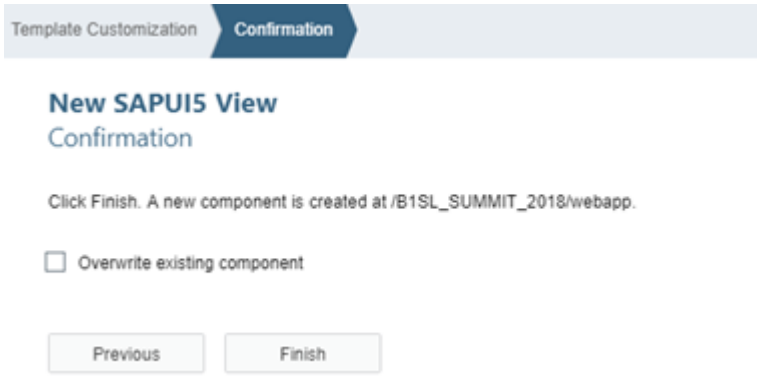
Explanation	Screenshot
Go to the created **sap.m.Toolbar -> content** element and **right click** to select the **Add** menu.	
Enter **SearchField** to filter the controls. **Click** on the **Search Field** proposed control.	
Select the **sap.m.SearchField** control in the Outline tab. Select the **Events** tab in the right side of the screen. On the **Search** event click on the ⋮ button and select **New Function** option.	

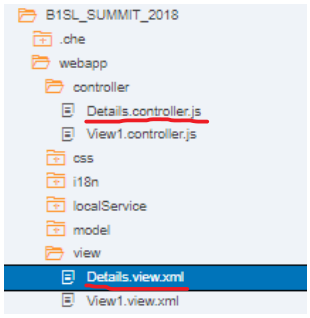
Explanation	Screenshot
Enter **onSearch** as function name that will be called when the Search button will be pressed. Press **OK**. Press **Save**. A new function with that name will be created in the View.Controller.js.	
Open the **View1.controller.js** file. Copy the code from https://github.com/B1SA/B1_SCP_HandsOn/blob/master/snippets/View1.controller.js_onSearch.js into your **onSearch** method so it looks like here. This code filters the Orders with DocEntry greater or equal than the value entered in the SearchField. Press **Save**.	
Run your application. Check that the SearchField filters the table with Orders having a DocEntry higher than the entered value.	

Explanation	Screenshot
	

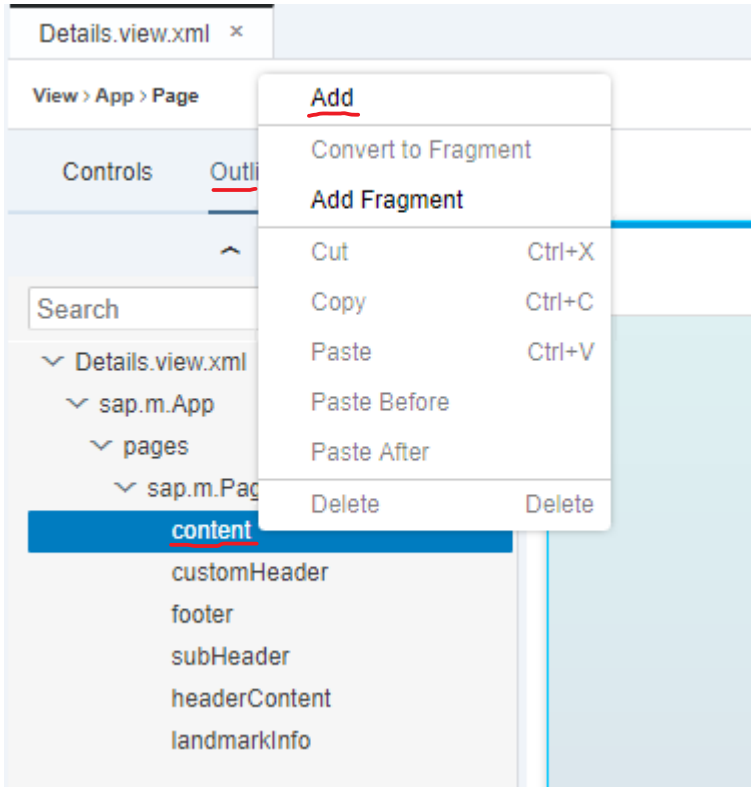
vi. Add a second view called Details

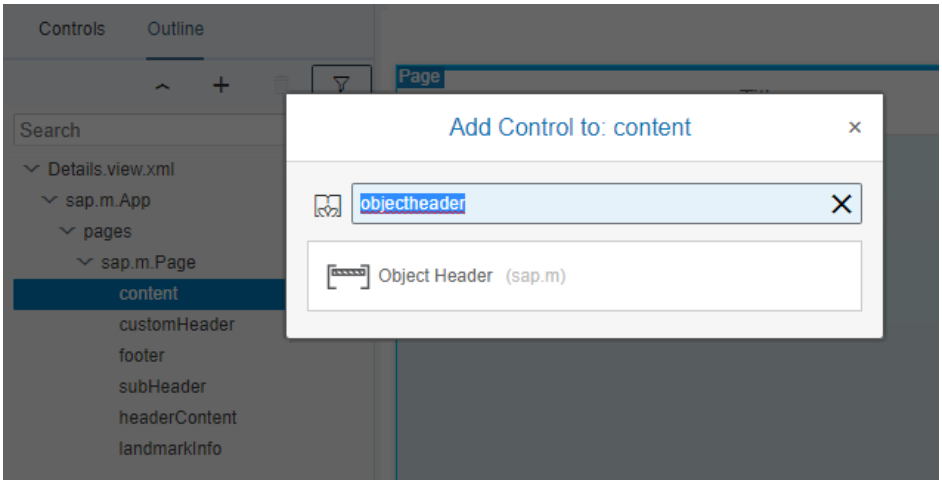
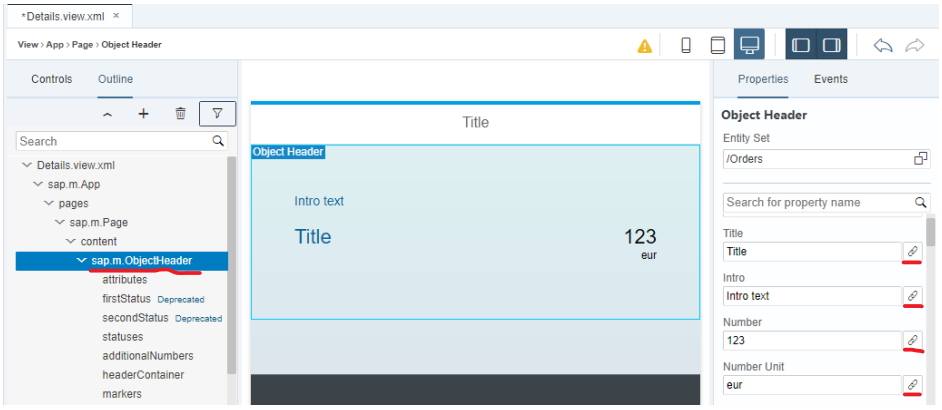
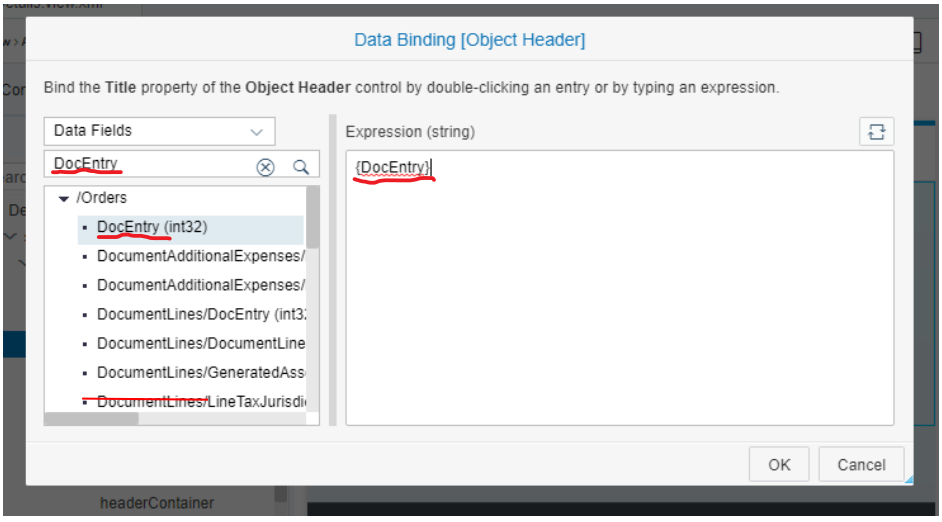
Create a Details view.

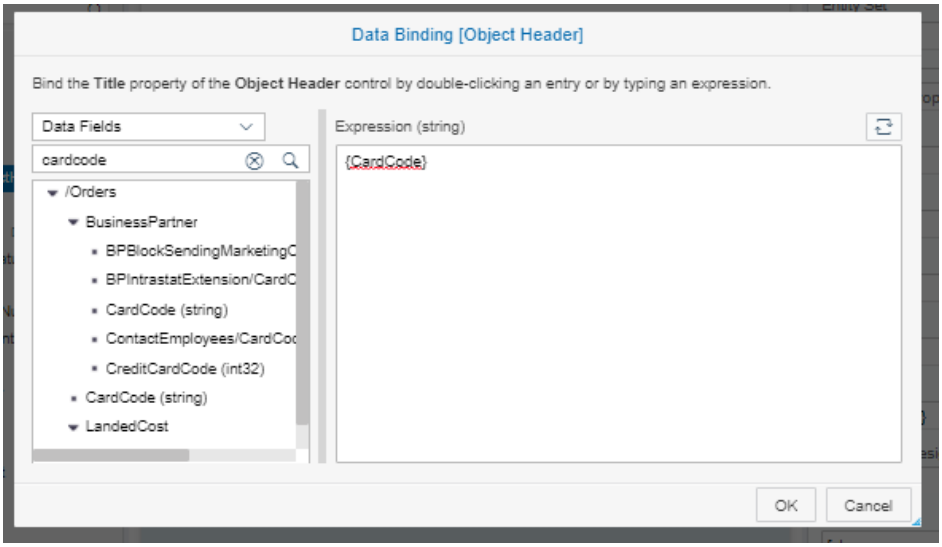
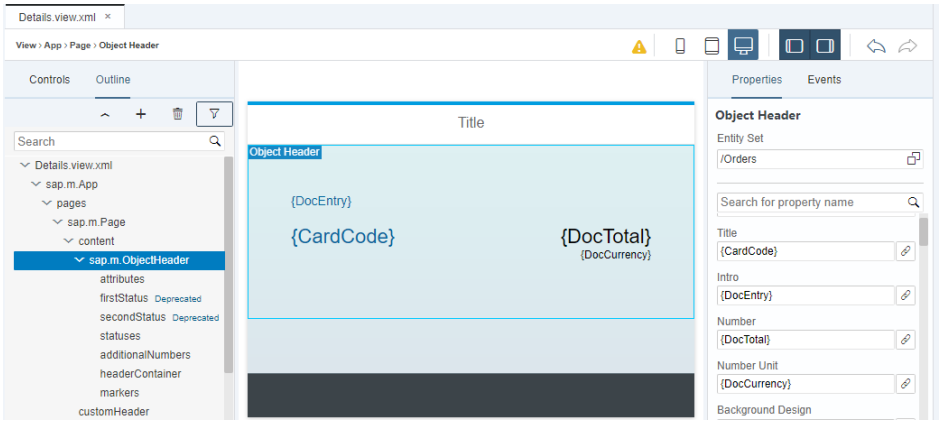
Explanation	Screenshot
Right click on your application and select **New -> SAPUI5** View.	
Keep the default values for View Type and Namespace. Enter the name of the new View: **Details**. Press **Next**.	
Press Finish to confirm the creation of the **Details** view.	

Explanation	Screenshot
Two files are created: - Details.view.xml - Details.controller.js.	

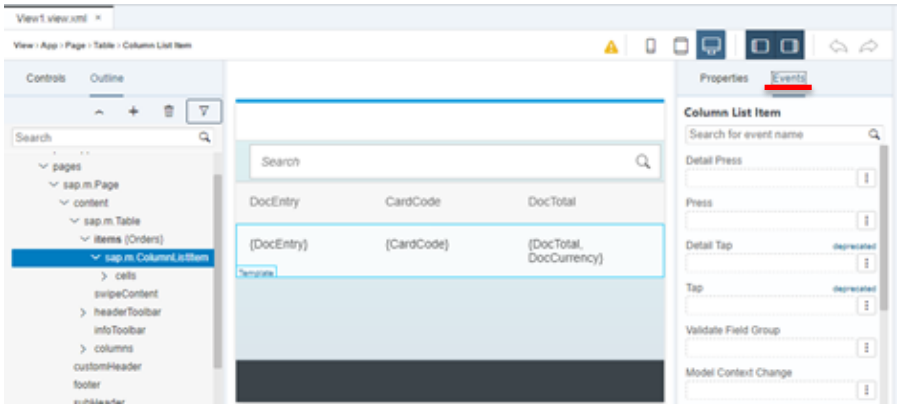
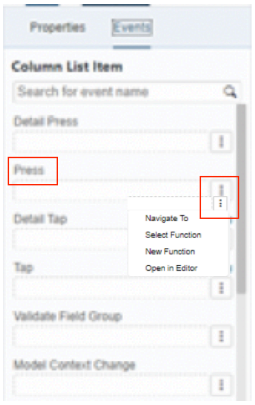
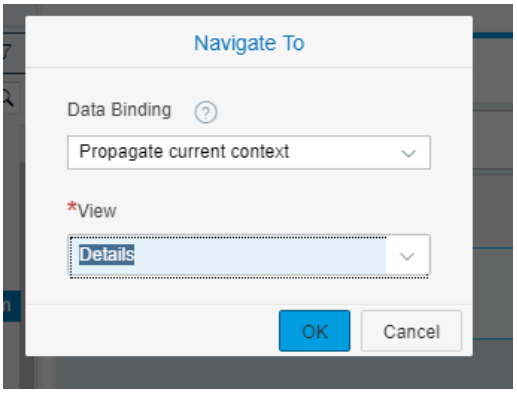
Add an Object Header control.

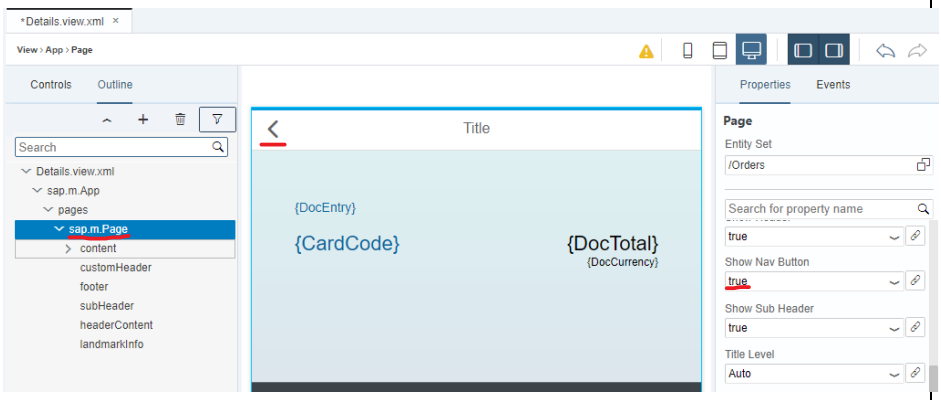
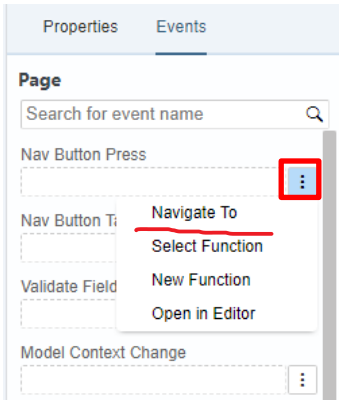
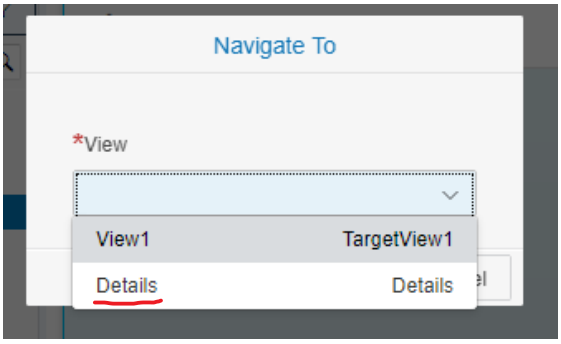
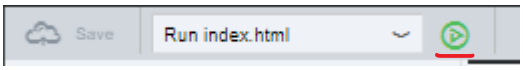
Explanation	Screenshot
Open the **Details.view.xml** file with the Layout editor. Go to the **Outline** tab. Right click on **Details.view.xml -> sap.m.App -> sap.m.Page -> content** and select **Add**.	

Explanation	Screenshot
Enter **objectheader** to search for the Object Header control. Click on the **Object Header** proposed control.	
Select the created **sap.m.ObjectHeader** element in the Outline tree. Go to the **Properties tab** in the right side of the screen. **Click** one by one on the **Data Binding icon** for the properties: - **Title** - **Intro** - **Number** - **Number Unit**	
A new dialog will be presented for each one of the properties. Search for the Orders Data Field to be assigned to each property: - Title -> CardCode - Intro -> DocEntry - Number -> DocTotal - Number Unit -> DocCurrency and double click on the proposed list to get it into the Expression string.	

Explanation	Screenshot
Note: **Do not select BusinessPartners/CardCode** for the CardCode property. Press **Save** button.	
The sap.m.ObjectHeader should look like this at this point.	

vii. Define navigation between View1 and Details.

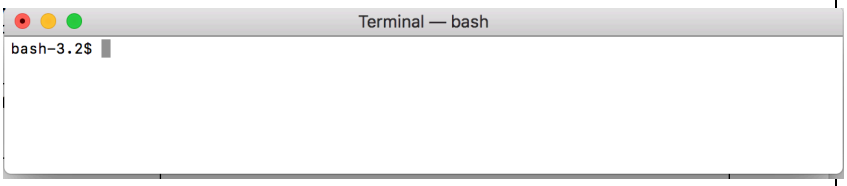
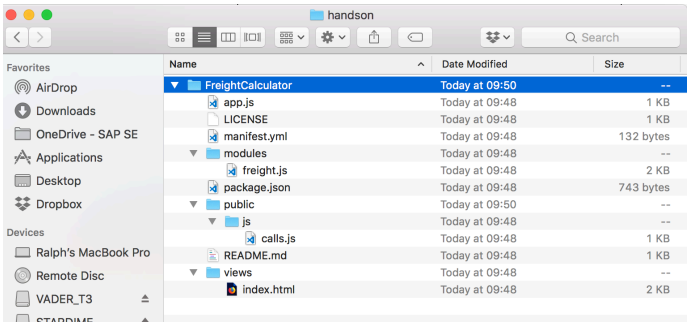
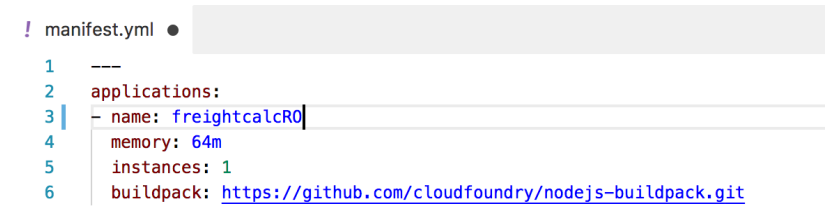
Explanation	Screenshot
From our main View1 table we want to navigate to Details view when the user clicks on a row. Open **View1.view.xml** file with Layout Editor. Select **Outline** tab then **sap.m.ColumnsListItem**. Select the **Events** tab.	
Click on the **Press** icon. Select **Navigate To** menu.	
Select **Propagate current context** on the Data Binding combo box. Select **Details** on the View combo box. Press **OK**. Press **Save** to save the View1.view.xml changes. The selected Order data from the View1 will be bound to the Details view.	

Explanation	Screenshot
From our Details view we want to navigate back to our main View1 when the user press on the Navigate back button. Open the **Details.view.xml** file with the Layout Editor. Go to the **Outline** tab and select **sap.m.Page**. On the **Properties** tab scroll down to see the **Show Nav Button** property and change its value to **true**.	
Go to the **Events** tab. On the **Nav Button Press** event click on the icon  . Select the option **Navigate To**.	
Select **View1** on the combo box. We will navigate back to View one when the Nav back button will be pressed. Click on the **Save** button.	
Test your app to see navigation is working fine between View1 and Details view.	

Congratulations! You have created your first SAP UI5 application connecting to SAP API Business Hub

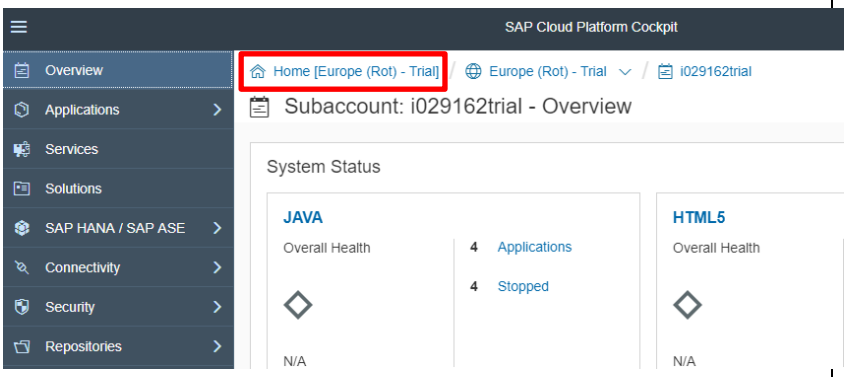
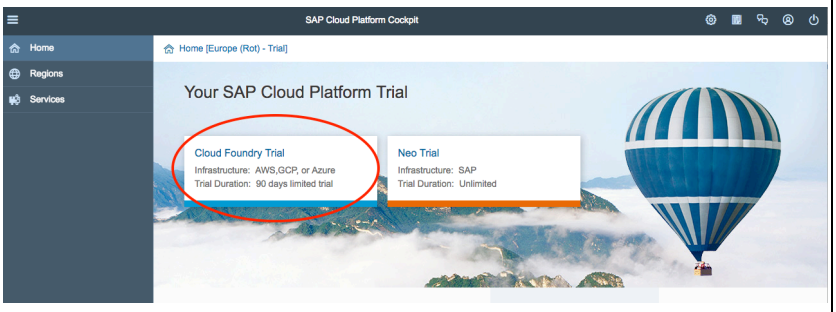
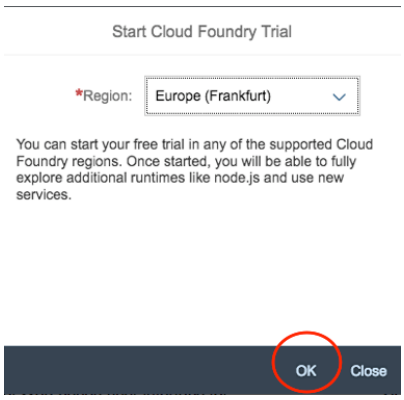
STEP 2: CREATE A NODEJS APP

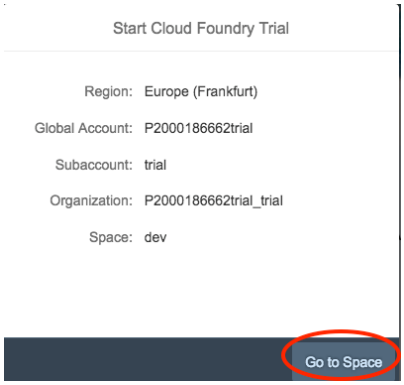
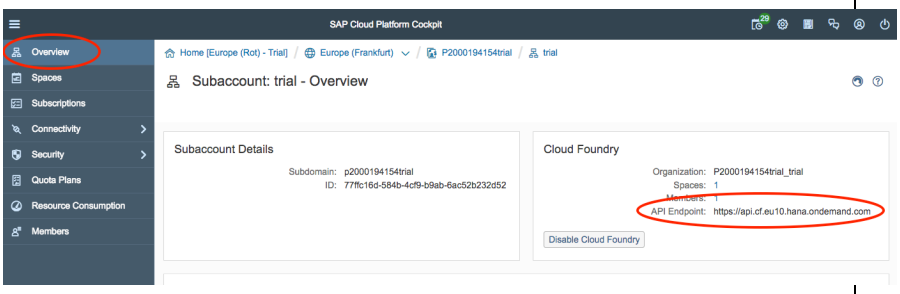
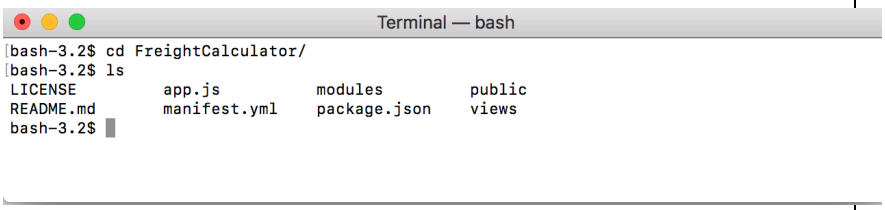
In this step we are going to implement the backend of our application. It will contain a simple business logic to calculate freight costs from different providers and consume 3rd Party services. The application is written in NodeJS and the source code is available on GitHub.

Explanation	Screenshot
Once **git** is installed (according to the pre requisites), open your system **terminal** (cmd, bash..) **Navigate** to a specific folder where you will download a sample application. **Pay attention** what folder is it, we will access it later	
Execute the following command to clone our solution backend repository: `$ git clone https://github.com/B1SA/FreightCalculator.git`	
You can see the app code on your file explorer:	
Edit the File **manifest.yml** and set a unique name for your application. E.eg FreightCalc<Your Initials>	

STEP 3: DEPLOY THE NODEJS APP INTO SAP CLOUD FOUNDRY

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

Explanation	Screenshot
First, we need to activate the SAP Cloud Platform Cloud Foundry Environment. On the SAP Cloud Platform Dashboard click on the **HOME** option	
First, we need to activate the SAP Cloud Platform Cloud Foundry Environment. From the SAP CP Home Screen, click on **Cloud Foundry Trial**	
Select the Trial Region that most suits you. And Click on **OK**	

Explanation	Screenshot
This will initialize your Cloud Foundry Trial and create a DEV space (where the solutions will be deployed). Go Ahead and access your space.	
Your space is created and ready to deploy your app. We now need your environment endpoint to be able to push our app. Click on the **trial link**	
And then on **Overview option** in the menu **Select and copy your API Endpoint**. E.g. `https://api.cf.eu10.hana.ondemand.com`	
With the CLI installed (according to the pre requisites), open your system **terminal** and navigate to the folder of the backend app cloned on Congratulations! You have created your first SAP UI5 application connecting to SAP API Business Hub STEP 2: Create a NodeJS app of this guide	

Explanation	Screenshot

From that folder, login to Cloud foundry using the command
 	
`cf login -a <API ENDPOINT>`
 	
e g.
 	
`$ cf login -a api.cf.eu10.hana.ondemand.com`
 	
When prompted provide your SAP Cloud Platform **email** and **password**
	Terminal — bash
	bash-3.2$ cf login -a api.cf.eu10.hana.ondemand.com
	API endpoint: api.cf.eu10.hana.ondemand.com
	
	Email> sipemsbh@sharklasers.com
	
	Password>
	Authenticating...
	OK
	
	Targeted org P2000194154trial_trial
	
	Targeted space dev
	
	API endpoint: https://api.cf.eu10.hana.ondemand.com (API version: 2.101.0)
	User: sipemsbh@sharklasers.com
	Org: P2000194154trial_trial
	Space: dev
	bash-3.2$

Explanation	Screenshot
Now all you have to do is push your app to the SAP Cloud Platform Cloud Foundry by using the **command**: `cf push`	Terminal — cf push bash-3.2$ cf push Using manifest file /Users/Ralph/OneDrive - SAP SE/handson/FreightCalculator/manifest.yml Creating app freightcalcRO in org P2000194154trial_trial / space dev as sipemsbh@sharklasers.com... OK Creating route freightcalcro.cfapps.eu10.hana.ondemand.com... OK Binding freightcalcro.cfapps.eu10.hana.ondemand.com to freightcalcRO... OK Uploading freightcalcRO... Uploading app files from: /Users/Ralph/OneDrive - SAP SE/handson/FreightCalculator Uploading 5.3K, 11 files Done uploading OK Starting app freightcalcRO in org P2000194154trial_trial / space dev as sipemsbh@sharklasers.com
This process will read the **manifest.yml** to name your application and also upload and deploy all the artefacts in a container in the Cloud Foundry Environment. Once the Process finishes, you can see your app URL:	**requested state:** started **instances:** 1/1 **usage:** 64M x 1 instances **urls:** freightcalcro.cfapps.eu10.hana.ondemand.com **last uploaded:** Thu Mar 1 11:54:08 UTC 2018 **stack:** cflinuxfs2 **buildpack:** https://github.com/cloudfoundry/nodejs-buildpack.git **state** **since** **cpu** **memory** **disk** **details** #0 running 2018-03-01 11:54:39 AM 0.0% 44.7M of 64M 55.3M of 1G bash-3.2$

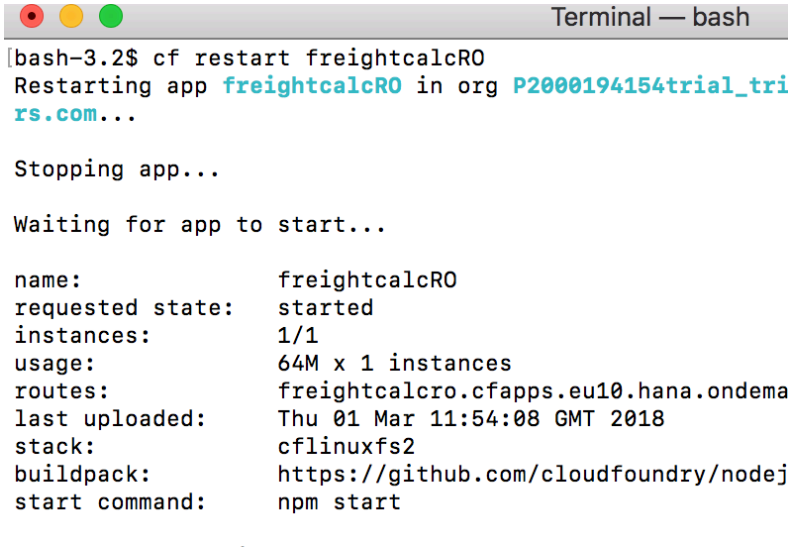
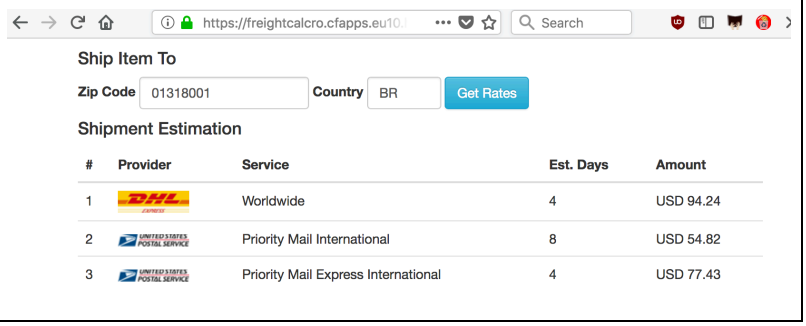
This application makes use of a 3rd Party service called Shippo to calculate shipping costs. In order to consume their services, we need an API KEY.

The Instructors of this hands-on session will provide a set of keys you can use in the next step. However, you can also get your own FREE test key on their website https://goshippo.com/

Best practices of cloud development (see https://12factor.net/config) suggests that any kind of configuration (such as keys) should not be part of the codebase but set as environment variables. And that is easily done with Cloud Foundry

Explanation	Screenshot

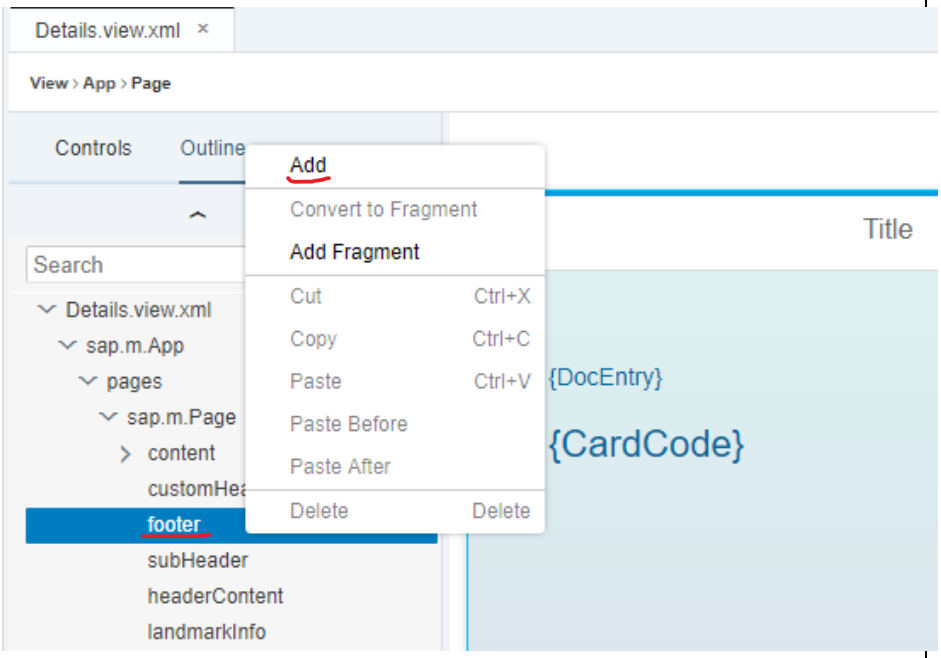
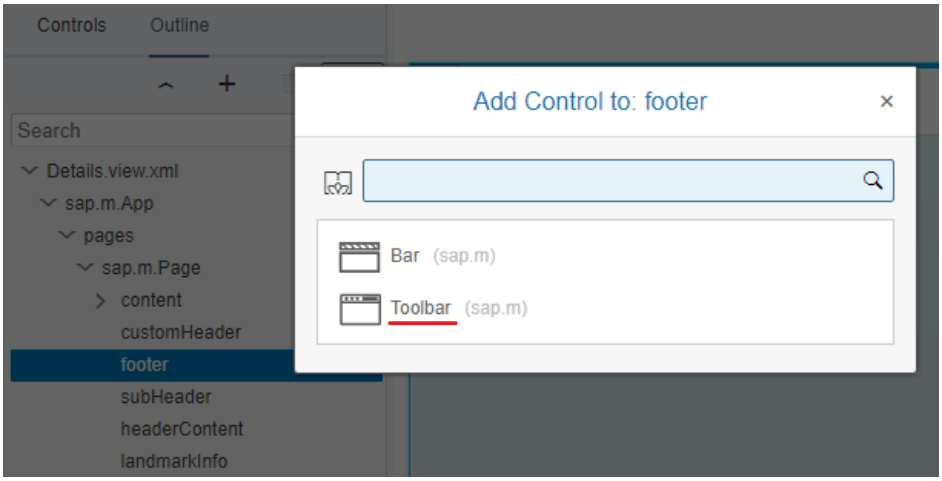
Back to the terminal, set the Environment Variable **SHIPPO_KEY** to a valid API Key with the following command
 	
`cf set-env <appname> <variable name> <variable value`
e.g.
`cf set-env freightcalcRO SHIPPO_KEY shippo_test_1234`
	

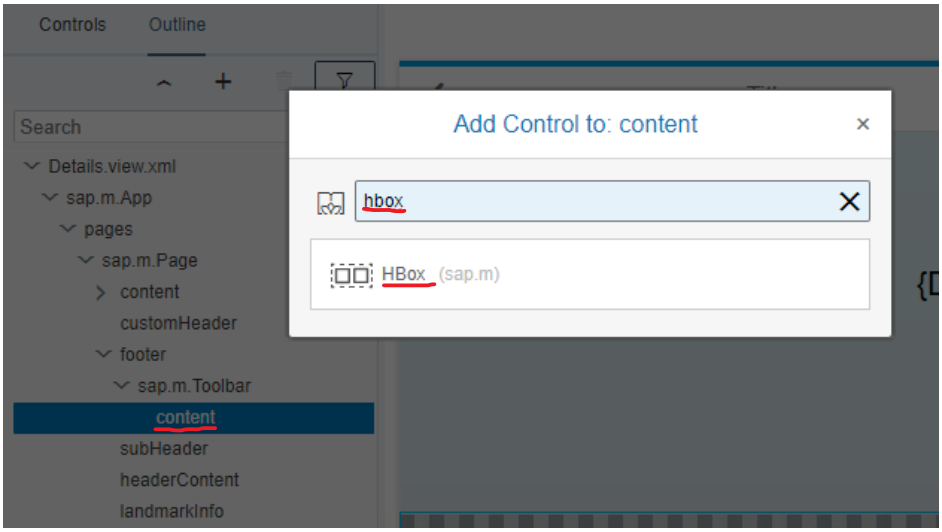
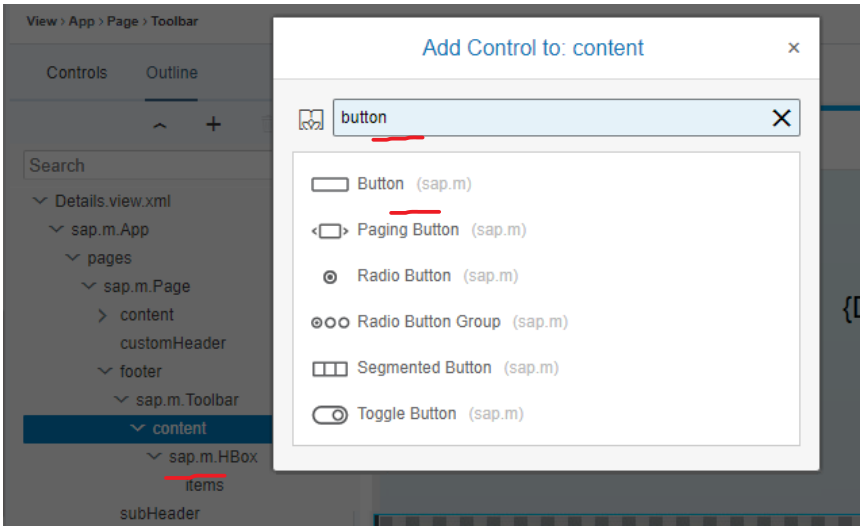
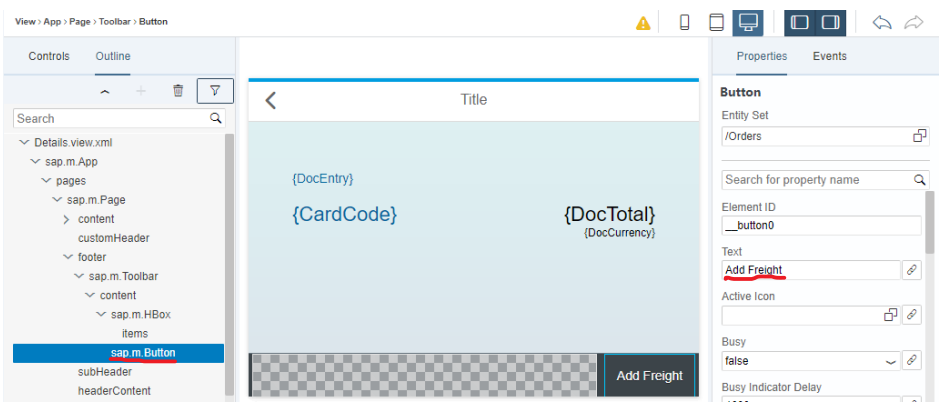
Up next, restart your app `cf restart <appname>`	
You can test the app with a test page on the app URL	

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

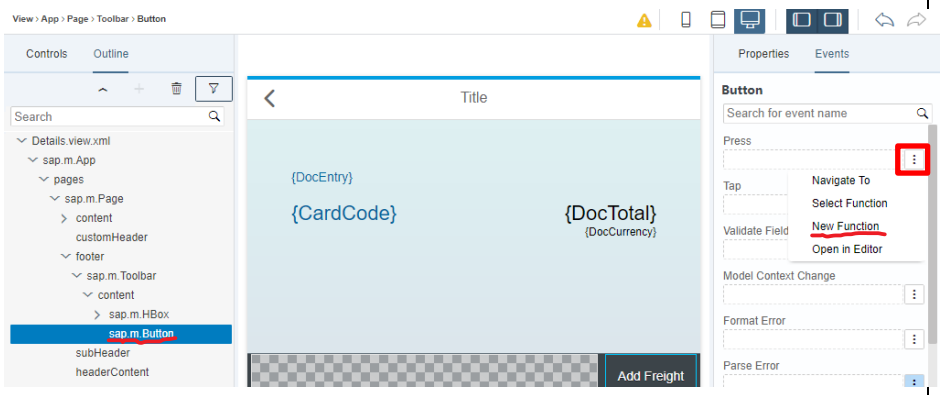
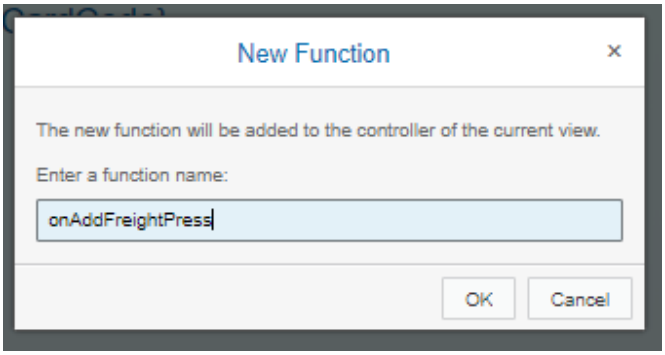
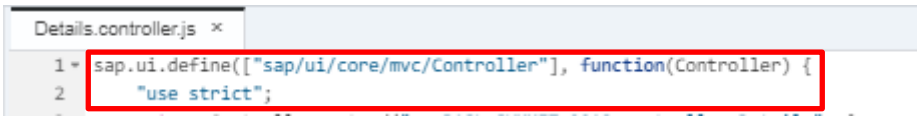
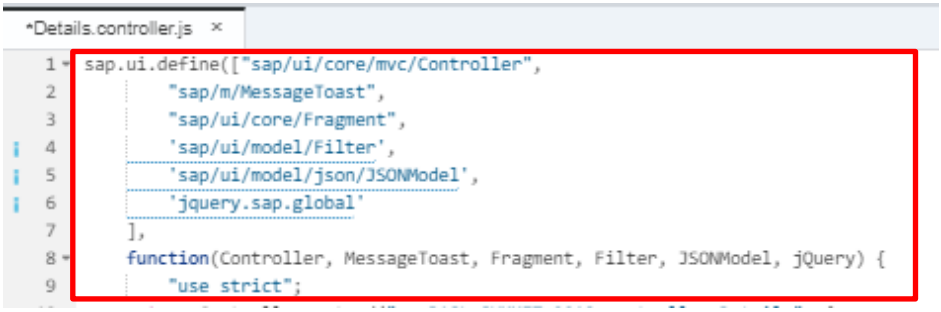
STEP 4: INTEGRATE THE NODEJS APP INTO THE SAP FIORI APP

i. Add a Button “Add Freight” to the Details view.

Explanation	Screenshot
Open the **Details.view.xml** file with the Layout Editor. On the **Outline** tab right click on the **sap.m.App -> pages -> sap.m.Page -> footer** element and select **Add**.	
Select the **Toolbar** control proposed.	

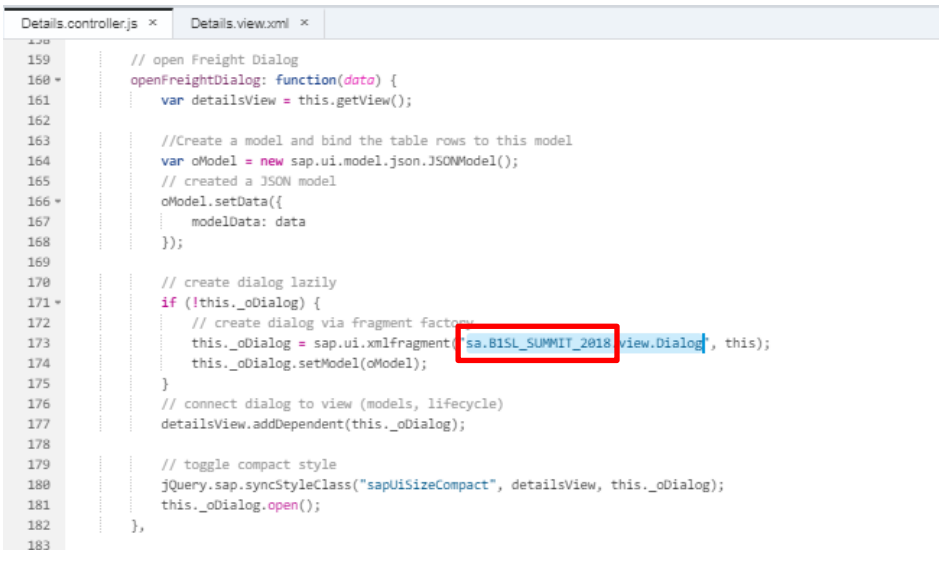
Explanation	Screenshot
On the Outline tab right click on **sap.m.Toolbar -> content** and select **Add**. Enter **hbox** in the Add Control to: content new dialog. Select the proposed **HBox** control.	
On the Outline tab right click on **sap.m.Toolbar -> content** and select **Add**. Enter **button** in the Add Control to: content dialog. Select the **Button** control.	
On the **Outline** tab select the **sap.m.Button**. On the **Properties** tab change the **Text** property by **Add Freight**.	

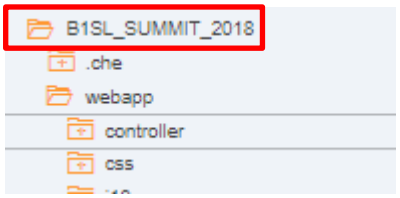
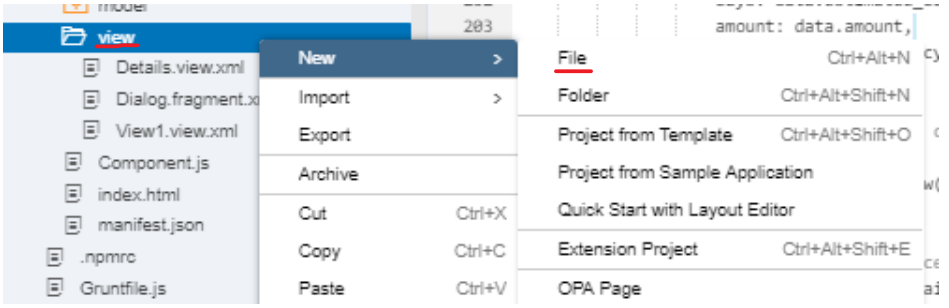
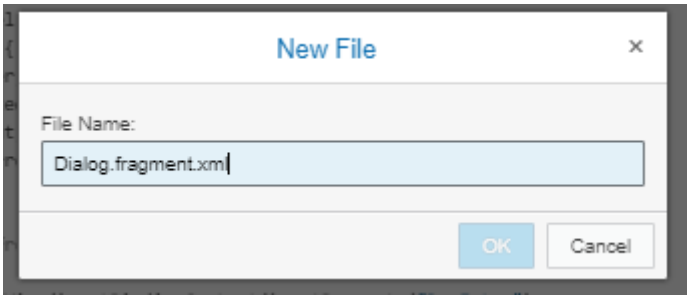
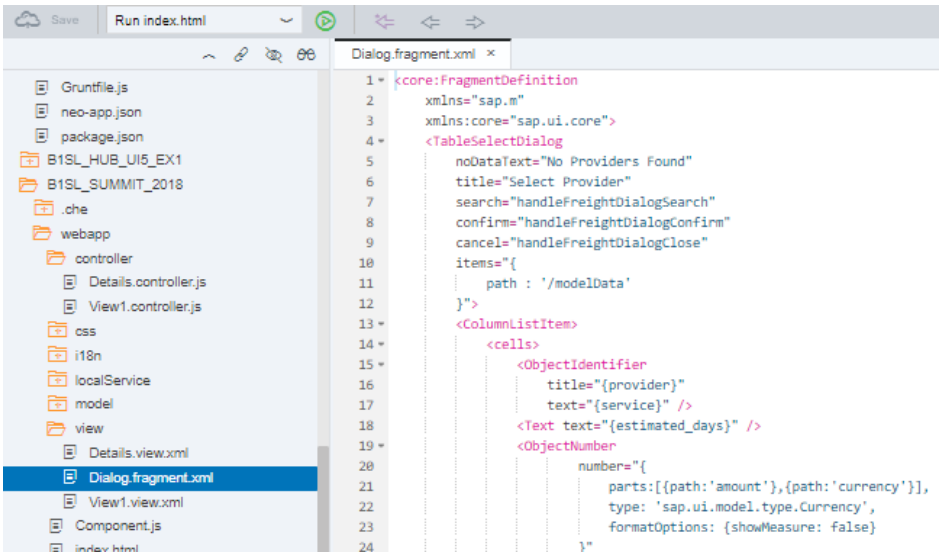
i. Implement the Button business logic calling the NodeJS server side and Service Layer.

Explanation	Screenshot
Select **Outline** tab, **sap.m.Button**. Open **Events** tab. Click on the **Press** event  icon and select **New Function**.	
Enter **onAddFreightPress** as the function name. This function will be called when the event Press is fired on the AddFreight button.	
Open the **Details.controller.js** file and scroll down to the bottom of the file. A new empty function called **onAddFreightPress** has been automatically created.	
Before implementing this function lets add some definitions to the Details.controller.js file. Open the **Details.controller.js** file. Remove the code at the beginning of the file until “use strict”; as shown here in the first screen. **Replace** that code from by the one shared at https://github.com/B1SA/B1_SCP_HandsOn/blob/master/snippets/Details.controller.js_definitions.js as	 

Explanation	Screenshot
shown in the second screen.	

Let now implement the function **onAddFreightPress**.
 	
When the user clicks on the AddFreight button we call the server side NodeJS to get freight options.
 	
Replace the URL to the freight calculator by **your backend application URL** (code marked in red in the screen capture).
 	
In case of success we call the function openFreightDialog.
 	
In case of error we simply show a MessageToast.
 	
You can get this code from https://github.com/B1SA/B1_SCP_HandsOn/blob/master/snippets/Details.controller.js_onAddFreight.js.
PS: The code contains more functions than this one, please copy the full set of functions. We will explain each one of the functions in the coming steps.
	
	`onAddFreightPress: function(oEvent) {`
	`  //This code was generated by the layout editor.`
	
	`  // Get Data from ODataModel V4 /Orders`
	`  var body = {`
	`  "to": {`
	`  "zip": this.getView().getBindingContext().getProperty("AddressExtension/ShipToZipCode"),`
	`  "country": this.getView().getBindingContext().getProperty("AddressExtension/ShipToCountry")`
	`  }`
	`  };`
	`  // open Freight view`
	`  // from Freight view selection we will get back to Object view`
	`  var oThis = this;`
	`  $.ajax({`
	`  url: "https://freightcalc.cfapps.eu10.hana.ondemand.com/Rates",`
	`  type: "POST",`
	`  data: JSON.stringify(body),`
	`  contentType: "application/json",`
	`  success: function(data) {`
	`  oThis.openFreightDialog(data);`
	`  },`
	`  error: function(e) {`
	`  MessageToast.show("POST Freight error: " + e);`
	`  }`
	`  });`
	`},`

On the Details.controller.js scroll and search for the function **openFreightDialog**. This function opens a new Dialog showing the freight options returned by the server side.
 	
This function will use an xml fragment to design the freight options dialog.
 	
Edit this function and **change** the pointer to the **xmlfragment** to match the name of your namespace and SAPUI5 application (you can get this information from the beginning of your Details.controller.js file. In my case it is
	
	Details.controller.js × Details.view.xml ×
	`// open Freight Dialog`
	`openFreightDialog: function(data) {`
	`  var detailsView = this.getView();`
	
	`  //Create a model and bind the table rows to this model`
	`  var oModel = new sap.ui.model.json.JSONModel();`
	`  // created a JSON model`
	`  oModel.setData({`
	`  modelData: data`
	`  });`
	
	`  // create dialog lazily`
	`  if (!this._oDialog) {`
	`  // create dialog via fragment factory`
	`  this._oDialog = sap.ui.xmlfragment("sa.B1SL_SUMMIT_2018.view.Dialog", this);`
	`  this._oDialog.setModel(oModel);`
	`  }`
	`  // connect dialog to view (models, lifecycle)`
	`  detailsView.addDependent(this._oDialog);`
	
	`  // toggle compact style`
	`  jQuery.sap.syncStyleClass("sapUiSizeCompact", detailsView, this._oDialog);`
	`  this._oDialog.open();`
	`},`

Explanation	Screenshot
"sa.**B1SL_SUMMIT_2018**.view.Dialog".	
Let's add the Fragment required to show the Dialog. Right click on **view folder** and select **New->File** menu.	
Give the name **Dialog.fragment.xml** to the new file. Press **OK**.	
Download the Dialog.fragment.xml file from https://github.com/B1SA/B1_SCP_HandsOn/blob/master/snippets/Dialog.fragment.xml. Copy the code inside your **Dialog.fragment.xml** file. You can notice in that file contains the controls of the Dialog. Also, that file contains 3 callback functions will be called for the events: - search - confirm - cancel We will implement those functions in our Details.controller.js file.	

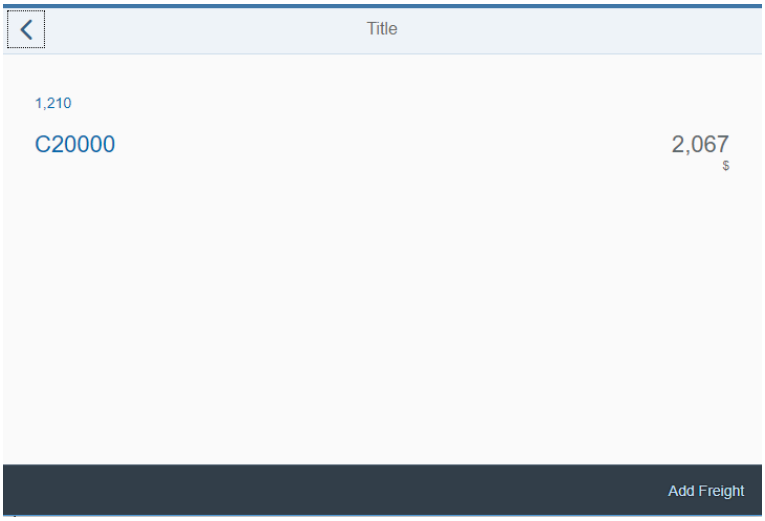
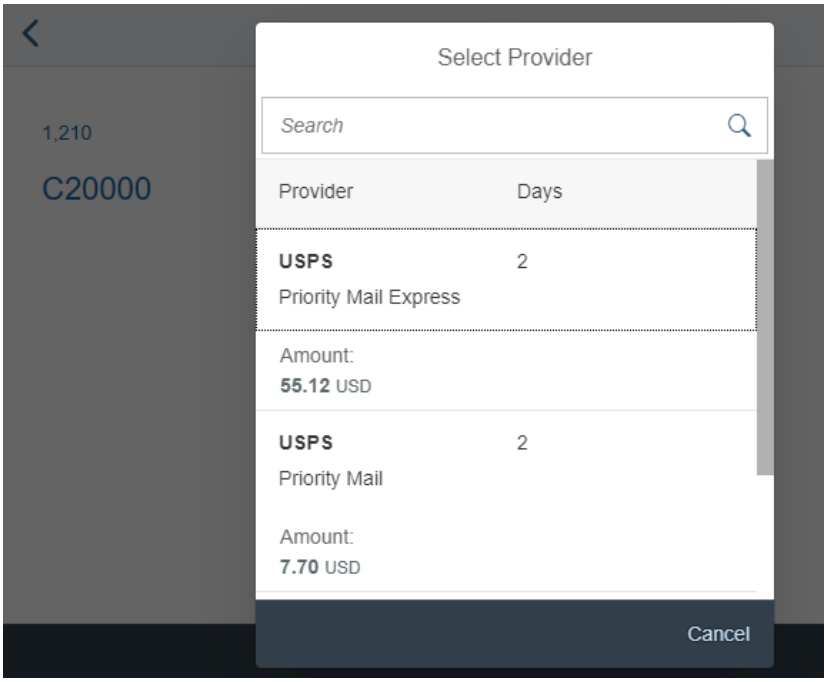
Explanation	Screenshot

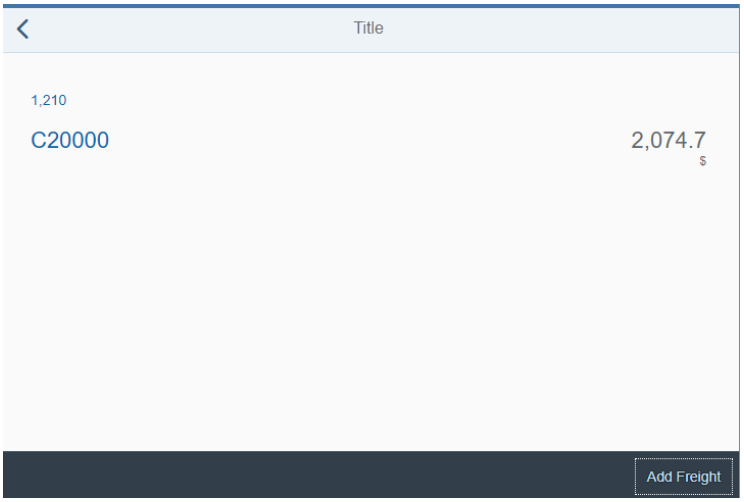
Open the **Details.controller.js** file.
 	
The **handleFreightDialogSearch** (code copied already earlier) is called when the user enters data to filter the list of freight options. The filter will be based on the provider property.
 	
The **handleFreightDialogConfirm** function is called when the user selects one of the freight options proposed by the Dialog.
This function calls the function UpdateSO to update the Order via Service Layer API.
 	
The **handleFreightDialogClose** function is called when the user closes the Dialog without selecting a freight option. A MessageToast will show to the user the message “No Provider selected.”
	
	handleFreightDialogSearch: function(oEvent) {
	var sValue = oEvent.getParameter("value");
	var oFilter = new Filter("provider", sap.ui.model.FilterOperator.Contains, sValue);
	var oBinding = oEvent.getSource().getBinding("items");
	oBinding.filter([oFilter]);
	},
	
	handleFreightDialogConfirm: function(oEvent) {
	
	// Get SelectedItem details
	var oSelectedItem = oEvent.getParameter("selectedItem");
	var ctx = oSelectedItem.getBindingContext();
	var data = ctx.getModel().getProperty(ctx.getPath());
	var providerDetails = {
	name: data.provider,
	days: data.estimated_days,
	amount: data.amount,
	currency: data.currency
	};
	
	// Get Order details from current view binding context
	var orderDetails = {
	DocEntry: this.getView().getBindingContext().getProperty("DocEntry")
	};
	
	// Update Order via Service Layer
	this.updateSO(providerDetails, orderDetails);
	
	console.log("You have chosen " + providerDetails.name + ", " + providerDetails.days + " Days, " + providerDetails.amount +
	providerDetails.currency + ".");
	},
	
	handleFreightDialogClose: function(oEvent) {
	MessageToast.show("No Provider selected.");
	},

Explanation	Screenshot

The **updateSO** function calls SAP Business One Service Layer via the SAP API Business Hub sandbox.
 	
We do an ajax call to the destination /apihub_sandbox/ automatically created when we added the Data Source pointing to the SAP API Business Hub. Then we specify the DocEntry of the Order to be updated with a PATCH request.
 	
In case of success we show a MessageToast message “SL success”.
 	
In case of error we show a MessageToast message “SL error”.
	Details.controller.js × Details.view.xml ×
	218 },
	219
	220 // Update Order via Service Layer
	221 updateSO: function(providerDetails, orderDetails) {
	222
	223 var oThis = this;
	224
	225 var body = {
	226 "DocEntry": orderDetails.DocEntry,
	227 "DocumentAdditionalExpenses": [{
	228 "ExpenseCode": 1,
	229 "LineTotal": providerDetails.amount,
	230 "Remarks": providerDetails.name
	231 }]
	232 };
	233
	234 $.ajax({
	235 url: "/apihub_sandbox/sapb1/b1s/v2/Orders(" + orderDetails.DocEntry + ")",
	236 type: "PATCH",
	237 data: JSON.stringify(body),
	238 contentType: "application/json",
	239 success: function(data) {
	240 oThis.soUpdateSuccess(data);
	241 },
	242 error: function(e) {
	243 oThis.soUpdateError(e);
	244 }
	245 });
	246 },
	247
	248 soUpdateSuccess: function(data) {
	249 // Refresh the Model to update the Order DocTotal we have changed
	250 this.getView().getBindingContext().getModel().refresh();
	251 MessageToast.show("Success. Check the DocTotal amount has changed.");
	252 },
	253 soUpdateError: function(e) {
	254 MessageToast.show("SL error: " + e);
	255 }
	256 });
	257 });

Test your application.	Save Run index.html
Filter the Orders table with **DocEntry higher than 1211**. Please follow directions from instructors so you only update a specific Order DocEntry to avoid conflicts between hands-on participants.	Title 1211 DocEntry CardCode DocTotal 1,211 C20000 2,067$ 1,213 C20000 2,067$ 1,215 C20000 2,067$

Explanation	Screenshot
Check the Order DocTotal before adding the freight costs.	
Select one of the providers.	

Explanation	Screenshot
Check the Order DocTotal amount is updated after you select a freight provider based on the provider costs. You will also see the success message if the update of the Order is successful.	

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

www.sap.com/contactsap