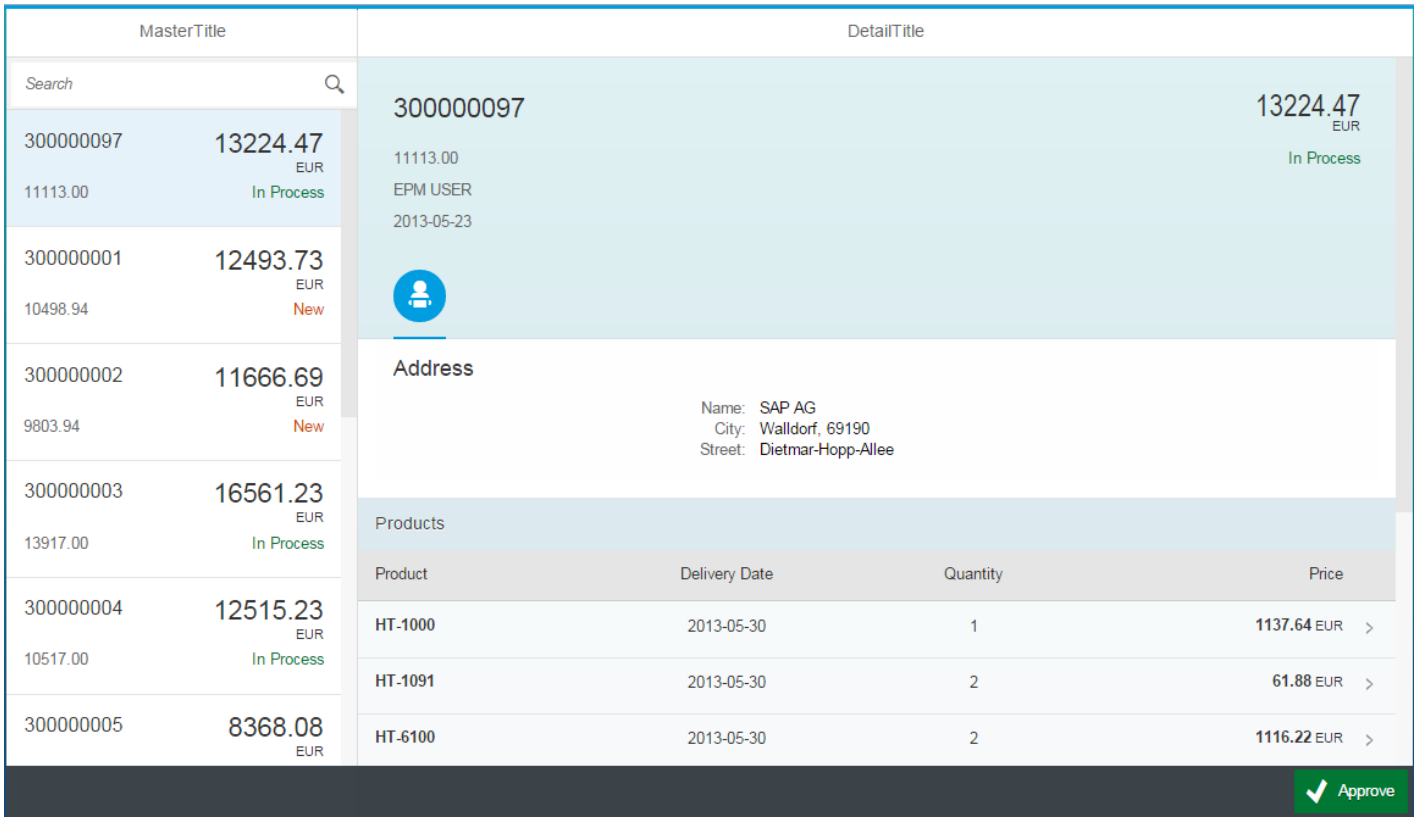
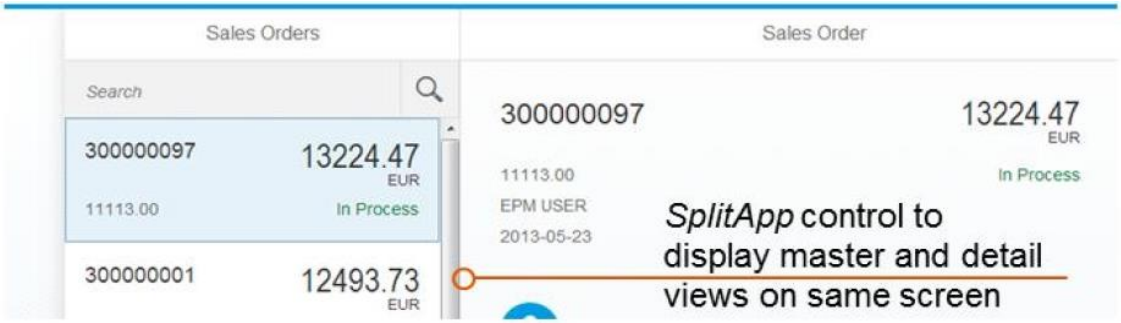
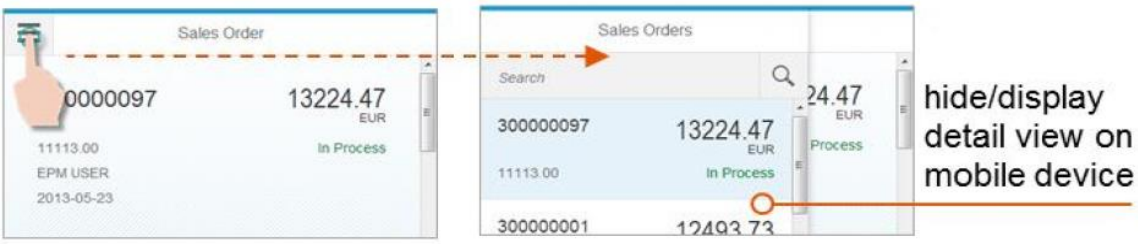
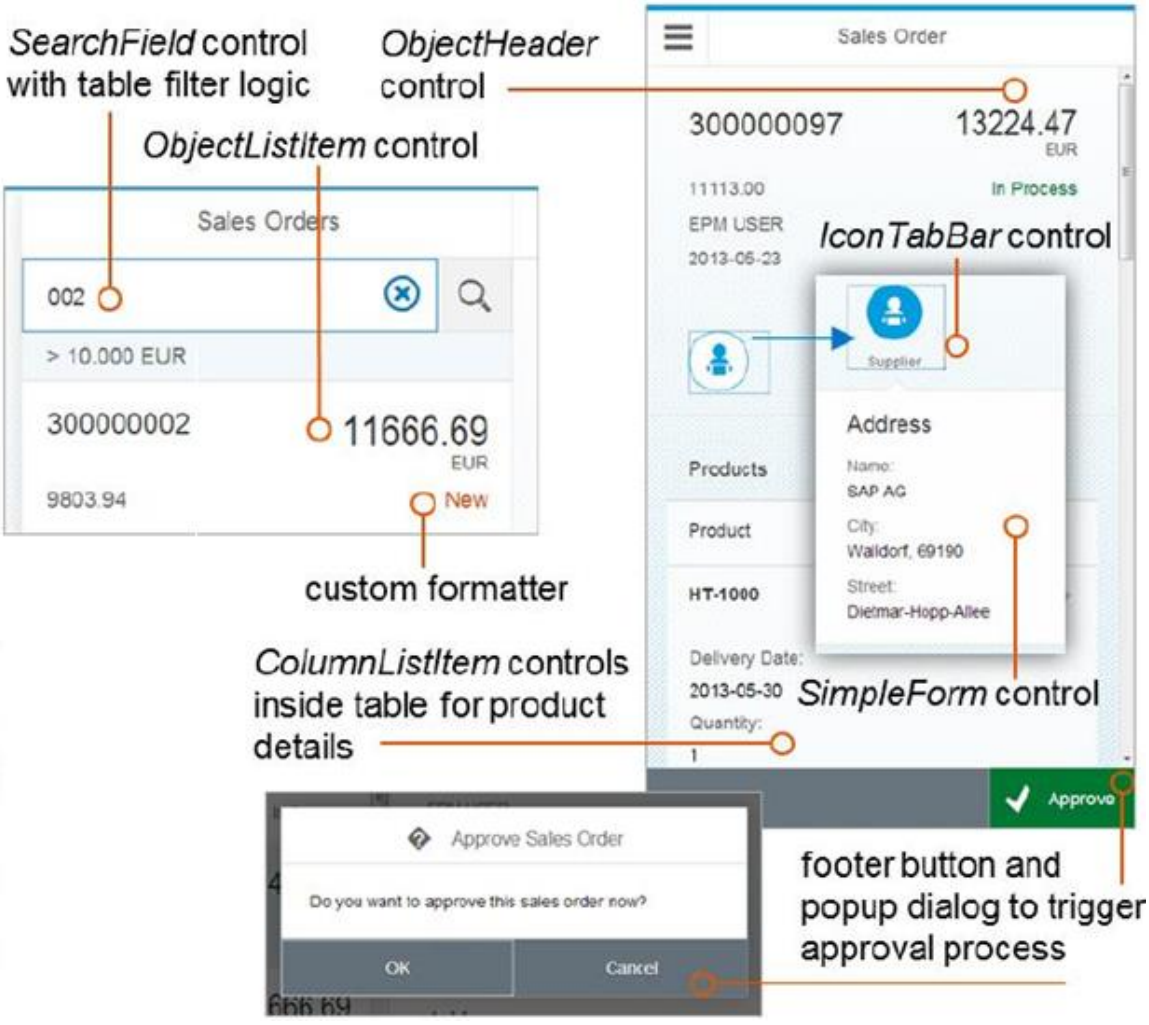
SAP FIORI-LIKE APPLICATION UI

The User Interface of the final Fiori-like SAPUI5 application described in this document looks this:




Sales Orders
Sales Order
Search
300000097
13224.47
EUR
11113.00
In Process
EPM USER
2013-05-23
300000001
12493.73
EUR
SplitApp control to display master and detail views on same screen


Sales Order
Sales Orders
Search
300000097
13224.47
EUR
11113.00
In Process
EPM USER
2013-05-23
300000001
hide/display detail view on mobile device

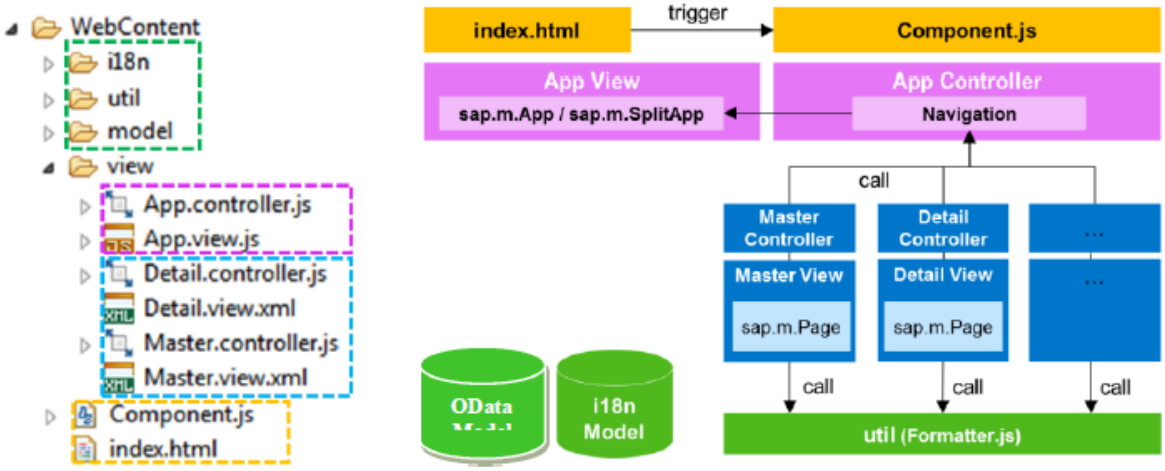

SearchField control with table filter logic
ObjectHeader control
ObjectListItem control
Sales Orders
002
> 10.000 EUR
300000002
11666.69
EUR
9803.94
New
custom formatter
Sales Order
300000097
13224.47
EUR
11113.00
In Process
EPM USER
2013-05-23
IconTabBar control
Supplier
Address
Name:
SAP AG
City:
Walldorf, 69190
Street:
Dietmar-Hopp-Allee
Products
Product
HT-1000
Delivery Date:
2013-05-30
Quantity:
1
SimpleForm control
ColumnListItem controls inside table for product details
Approve
Approve Sales Order
Do you want to approve this sales order now?
OK
Cancel
footer button and popup dialog to trigger approval process

SAP FIORI-LIKE APPLICATION ARCHITECTURE

Following the MVC design principle the Approve Sales Order application consists of the following main pieces:

- **Component.js:** Acts as a root or component container of the whole application. Implements logic to create the application's root view (*App* view) and used model instances.
- **Views with related controllers**: *App*, *Master,Detai and LineItem*. *App* is our top-level view, containing the *Master* and *Detail* views. In the *App* view we use a *SplitApp* control to contain the *Master* and *Detail* views via the *App* control's 'pages' aggregation.
- **Models:** *i18n* for locale-dependent texts, JSON model with mock data to be replaces with an OData model in real applications. A device model for device specific data needed at runtime.

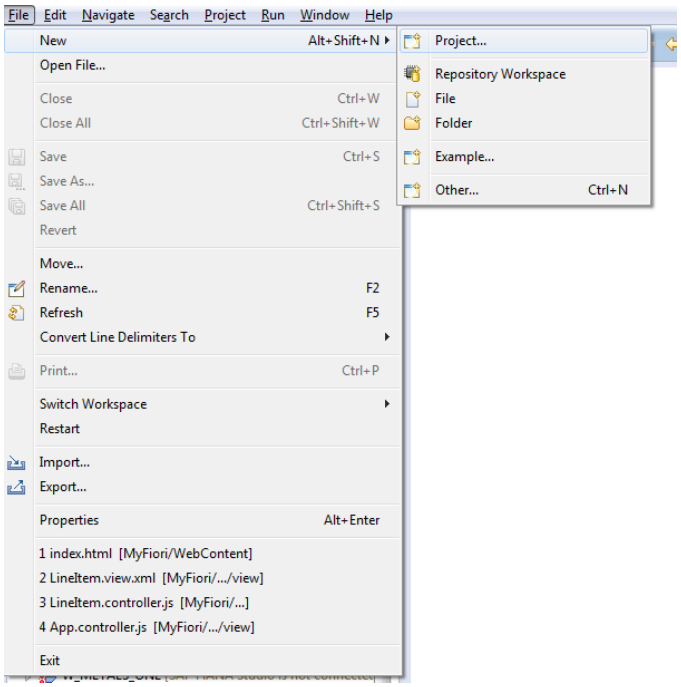
Artifacts and package structure of the final SAP-Fiori like SAPUI5 application, which we incrementally build in the following 10 exercises, are displayed in this diagram:

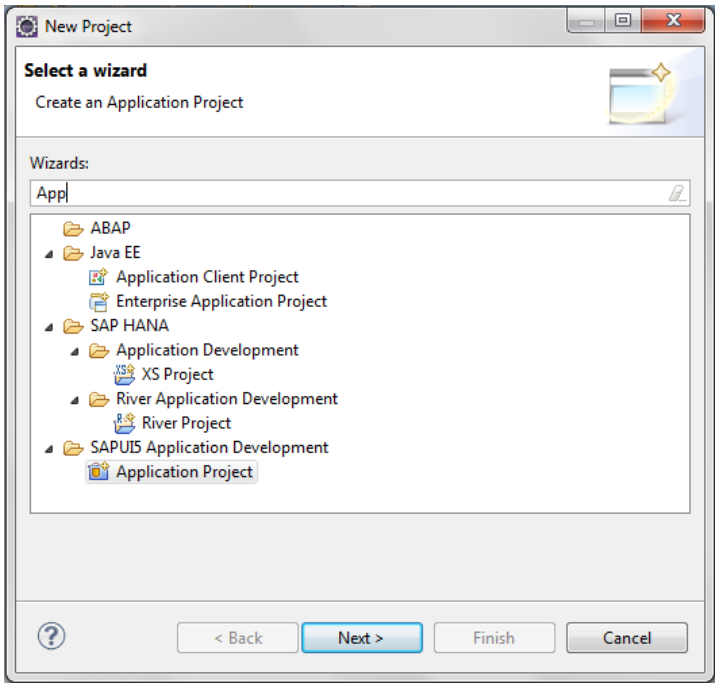


EXERCISE 0 – GETTING STARTED

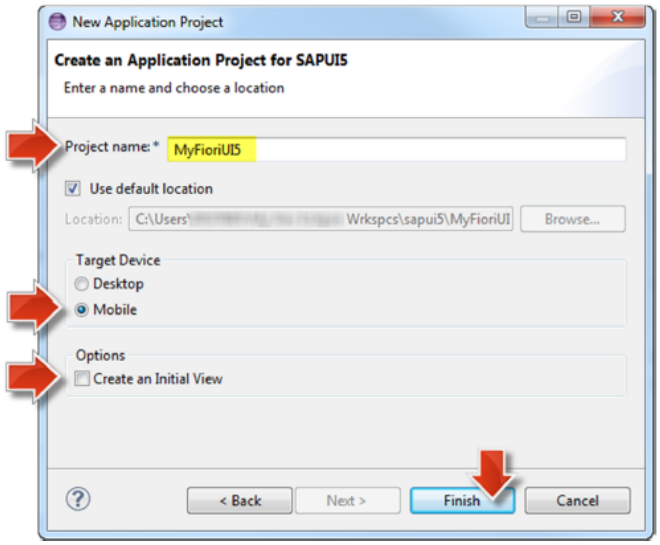
Set up the SAPUI5 development environment:

1. Download and install the SAPUI5 developer tools (Eclipse Java EE-based design time environment for the UI Development Toolkit for HTML5) from https://tools.hana.ondemand.com/#sapui5
2. The Eclipse with the requited plugins is present on the desktop.
3. The basic file required for creating the above Fiori App is also placed on the desktop.
4. Go to File ->New -> Project search for SAPUI5 Application project.

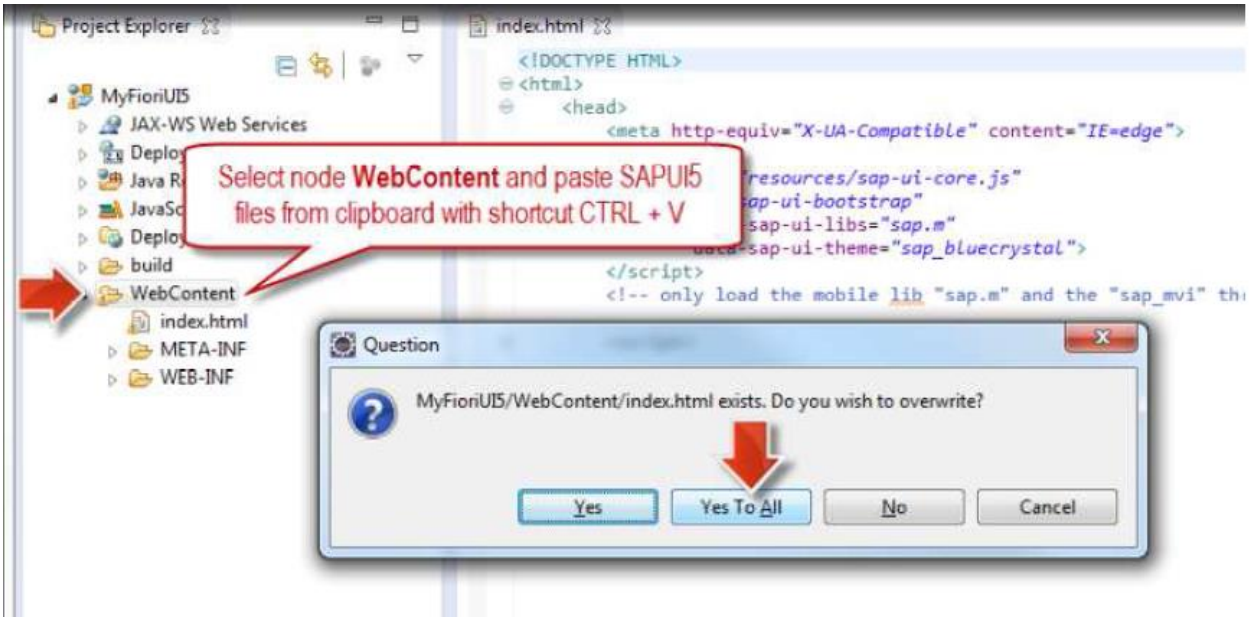




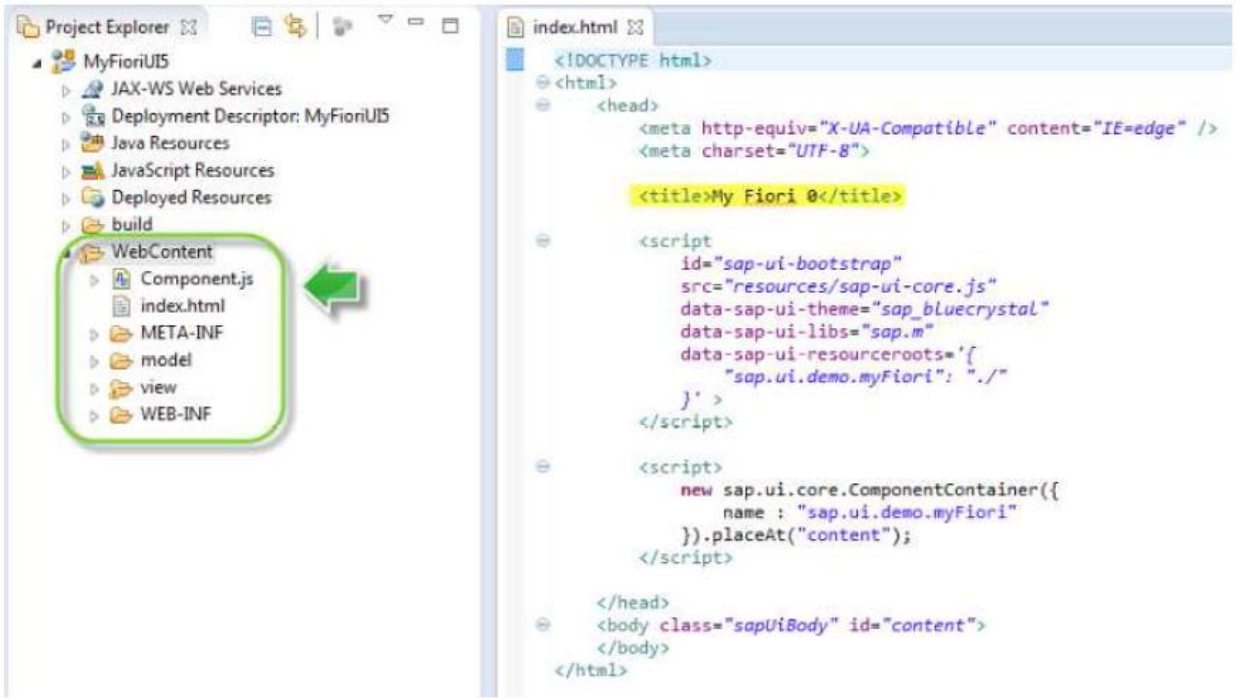
5. Enter your project name MyFioriUI5, chose “mobile” as the target device, uncheck “Create an Initial View” and click on Finish



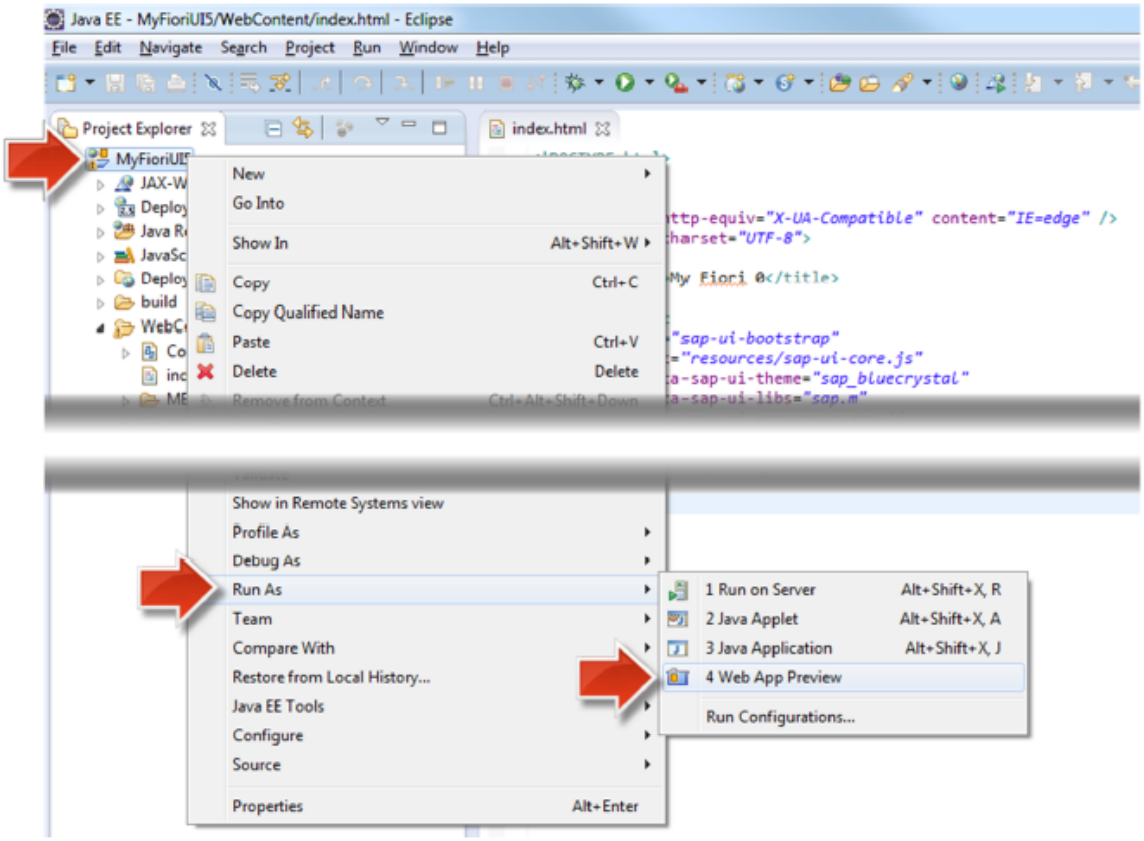
6. In Eclipse, select the folder “WebContent” of your SAPUI5 Application project on your desktop and paste the sample files . Confirm with “Yes to All”.



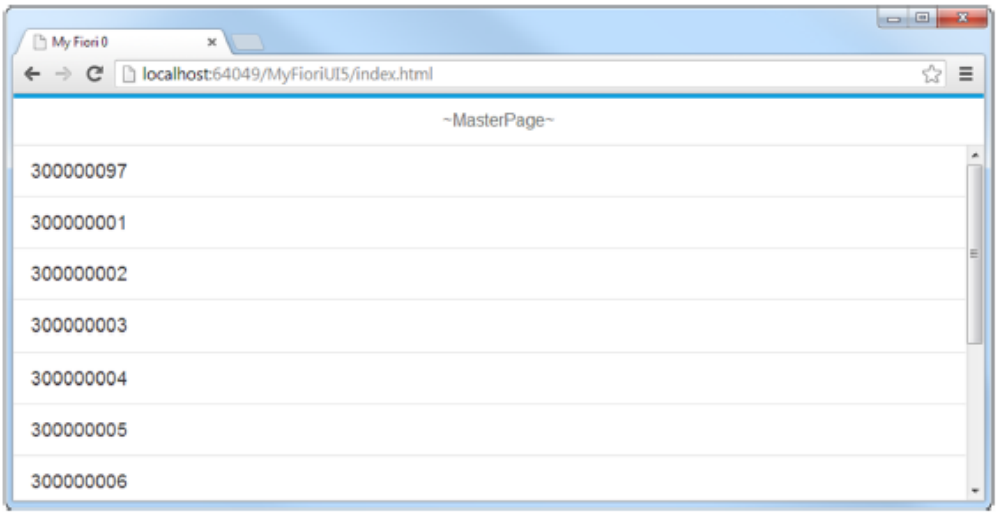
The imported SAPUI5 sources from the initial application template ‘WebContent – 0’ are displayed in the
Project Explorer under node MyFioriUI5 > Web Content



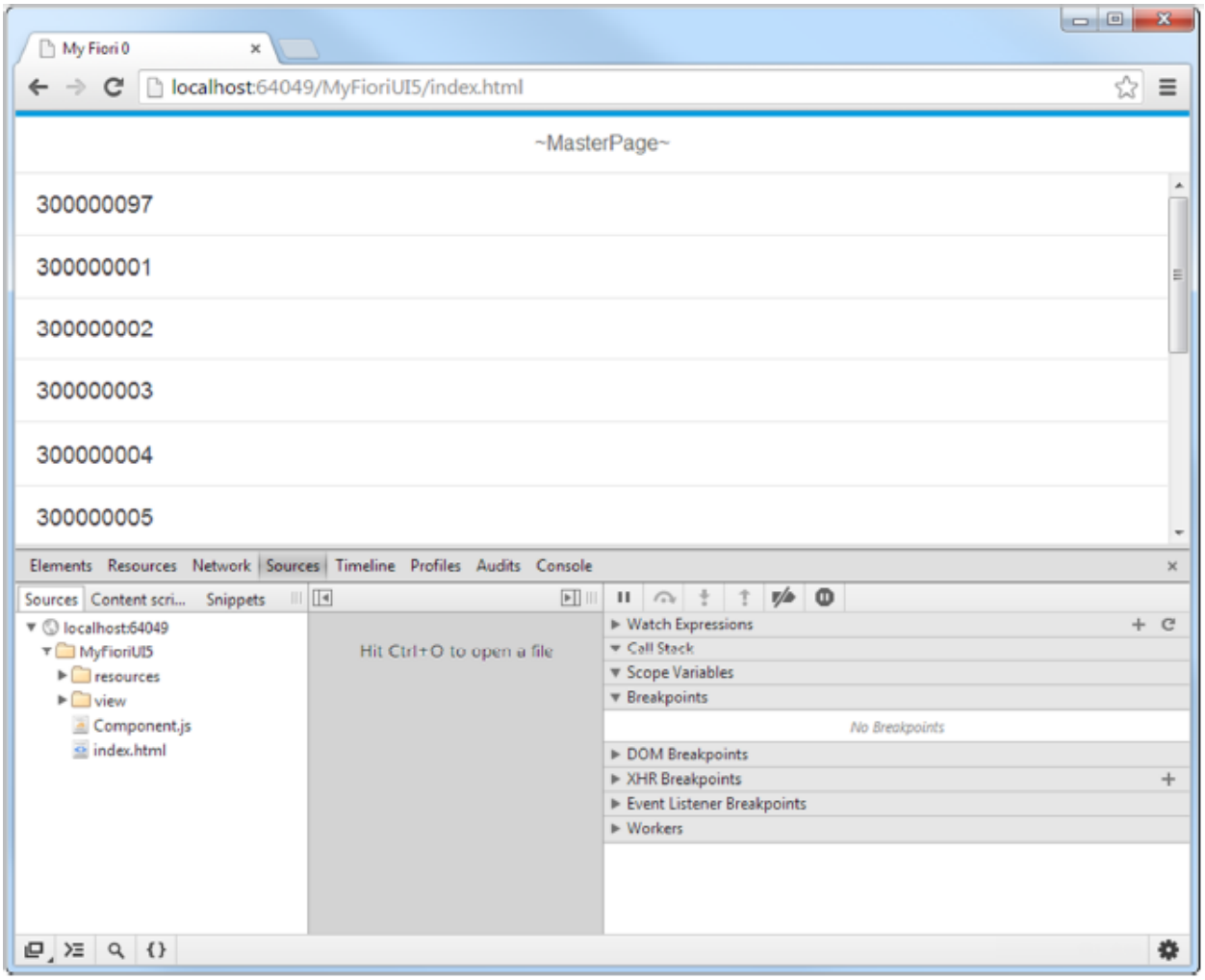
7. Right-click on the project node MyFioirUI5 and select context menu item “Run As”->”Web App
Preview”.



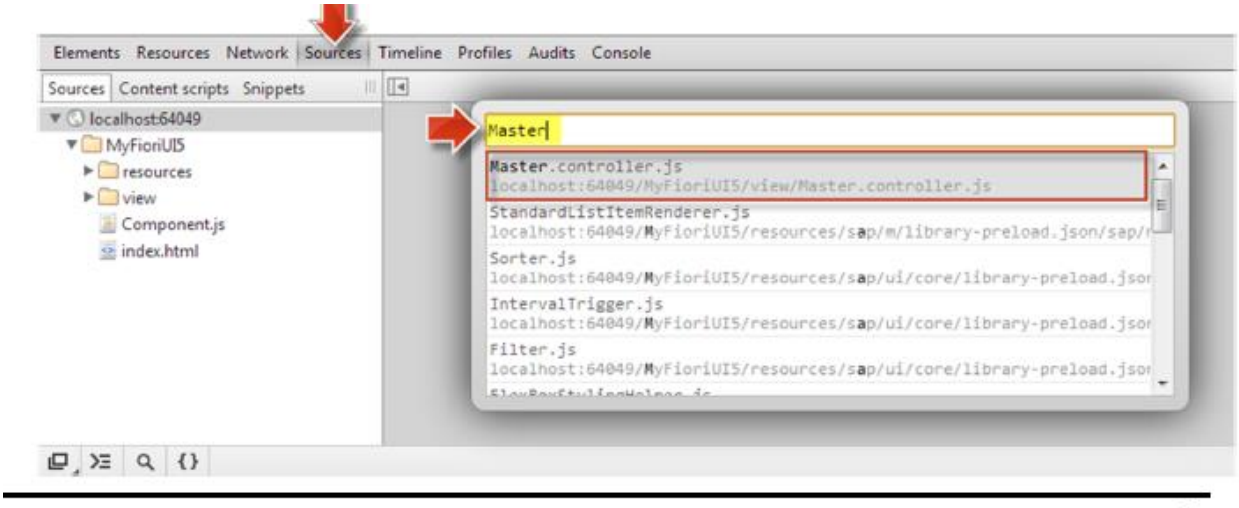
8. The application will be started and the preview will appear in eclipse. Copy the application URL to the
And paste it in Chrome Browser.



9. Press F12 to start the Chrome Developer Tools

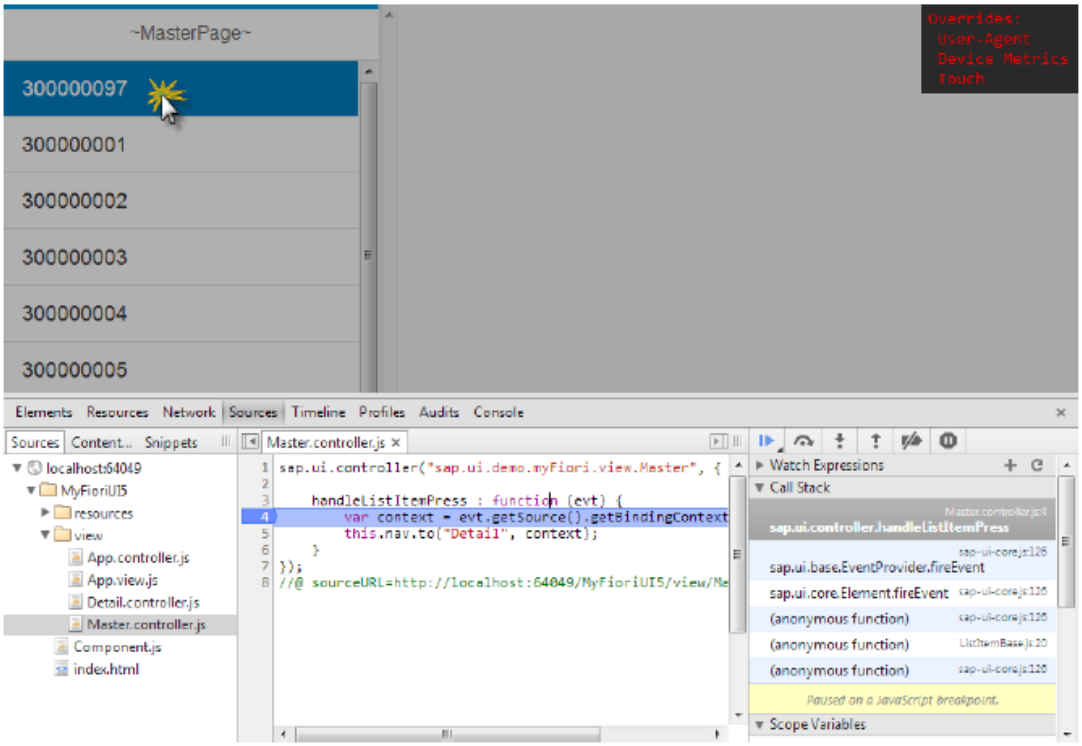


10. Open the tab “Sources”. Press CTRL-O and enter “Master” in the search field. Now the file “Master.controller.js” is selected and you press the Enter key.

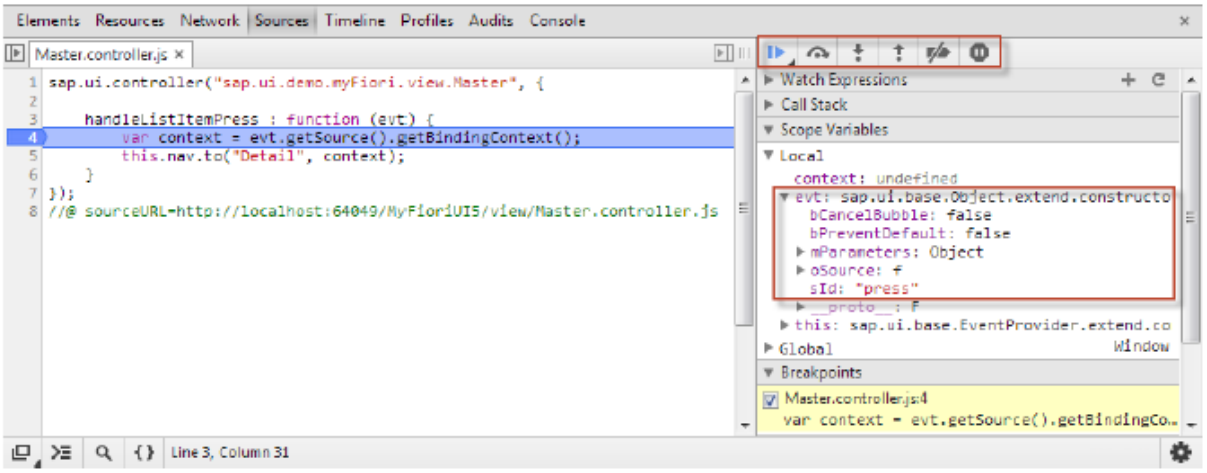


11. Set a breakpoint in line 4 by clicking on the line number until it is highlighted in blue. Notice the listing of the breakpoint in the right panel.

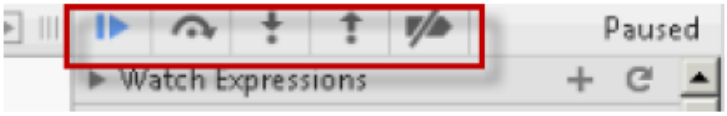
12. Now click on a line item in the running application. This causes the application to stop at the breakpoint.



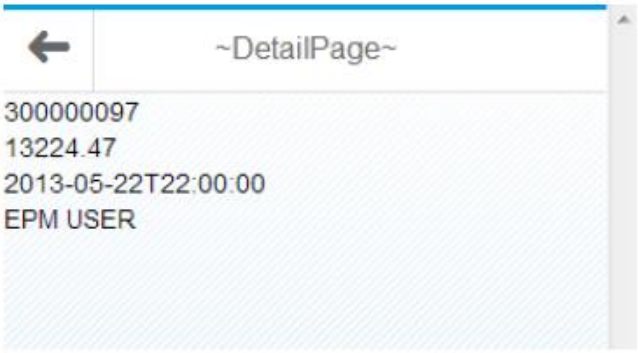
13. Collapse the panel “Call Stack” and open “Scope Variables”. Investigate the event parameter evt in the right panel. With this you can understand the current state at runtime.



14. Click on the “Play” button (blue) to resume the application execution.



15. The application now displays the DetailPage.



NOTE: Keep the Google Chrome browser open during all exercises. Test the results of your exercise procedures by reloading the My Fiori 0 page via ‘Reload’ toolbar icon or via keyboard shortcut F5.

EXERCISE 1 – RESOURCE MODEL

Objective

Set proper titles to master and detail pages by implementing a resource model (aka i18n model, i18n stands for internationalization).

Preview

Before:

After:

Description

What we're going to do in this exercise is to replace the hardcoded texts in the views with references to texts in a separate properties file. This is done via a resource model, which is used as a wrapper for resource bundles and has a one-time binding mode. Once the texts are abstracted into separate files, they can then be maintained, and translated, independently. So we'll modify the Master and Detail views, and create the properties file with the text contents.

Changes

1. Add new folder **i18n** and add new file **messageBundle.properties** and put the below content there and save the file.

   MasterTitle=Sales Orders
   DetailTitle=Sales Order

Component.js

2. The message bundle is loaded with the help of a Resource Model.
3. The Resource Model is made available as global model under the name "i18n"

```
createContent : function() {

// create root view

var oView = sap.ui.view({
id : "app",
viewName : "sap.ui.demo.myFiori.view.App",
type : "JS",
viewData : { component : this }
});

// set i18n model

var i18nModel = new sap.ui.model.resource.ResourceModel({
bundleUrl : "i18n/messageBundle.properties"
});
oView.setModel(i18nModel, "i18n");

// set data model on root view

Var oModel = new sap.ui.model.json.JSONModel("model/mock.json");
oView.setModel(oModel);

// done
return oView;

}
```

Master.view.xml

4. Switch the title to point to the “i18n” model and there to the text “MasterTitle”
5. Save the modified Master.view.xml file with keyboard shortcut CTRL+S

```
<core:View controllerName="sap.ui.demo.myFiori.view.Master"
 xmlns="sap.m"
xmlns:core="sap.ui.core" >

<Page
title="{i18n>MasterTitle}" >
……..
```

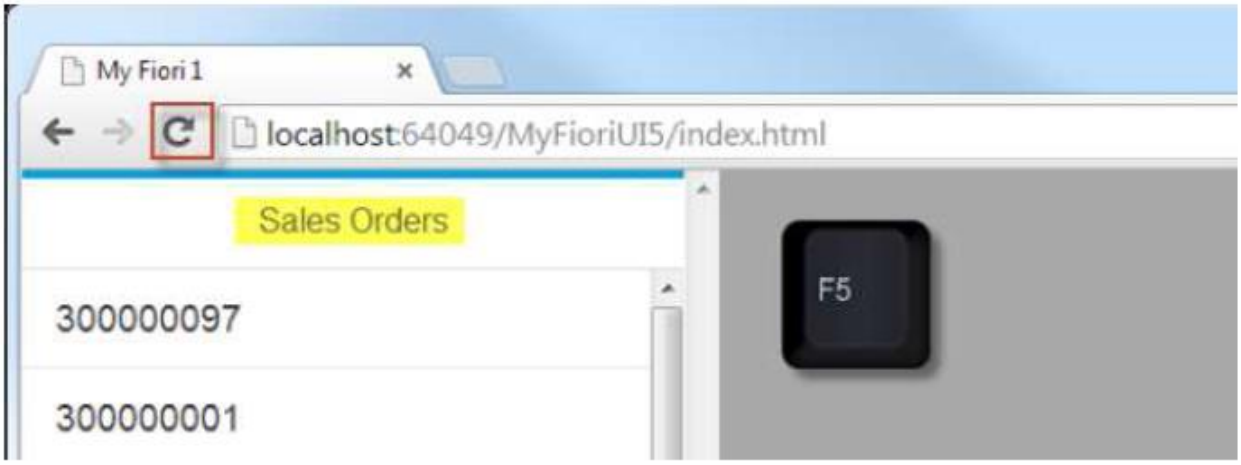
Detail.view.xml

6. Also adjust the title of the detail view.
7. Save the modified Detail.view.xml file.

```
<core:View controllerName="sap.ui.demo.myFiori.view.Detail"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
title="{i18n>DetailTitle}"
showNavButton="true"
navButtonPress="handleNavButtonPress" >
……
```

Google Chrome browser

Open the (already started) Google Chrome browser window and reload the index.html via toolbar icon or keyboard shortcut F5.



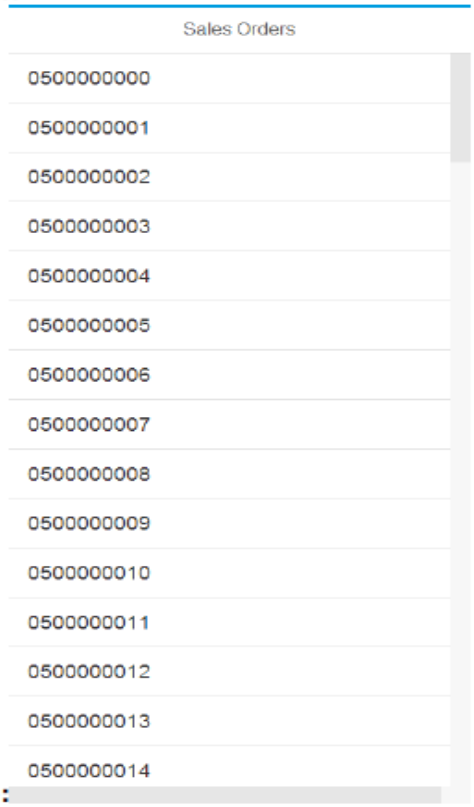
EXERCISE 2 – OBJECT CONTROLS

Objective

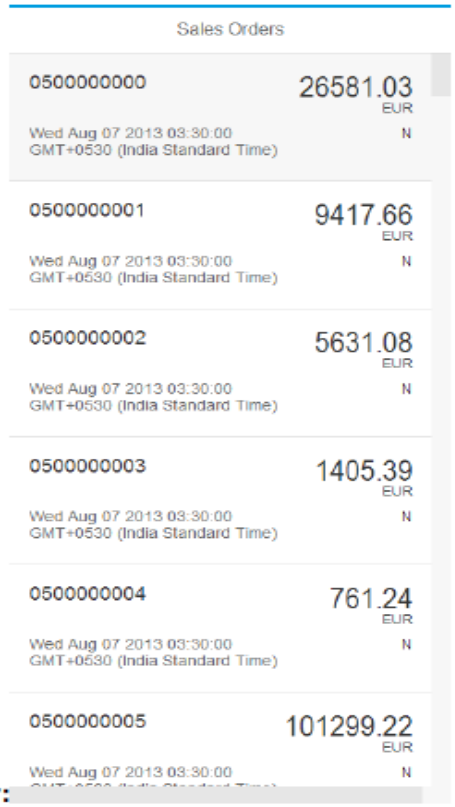
Make the UI of the master list and the detail page more beautiful by using the SAPUI5 controls sap.m.ObjectListItem and sap.m.ObjectHeader.

Preview

Before:

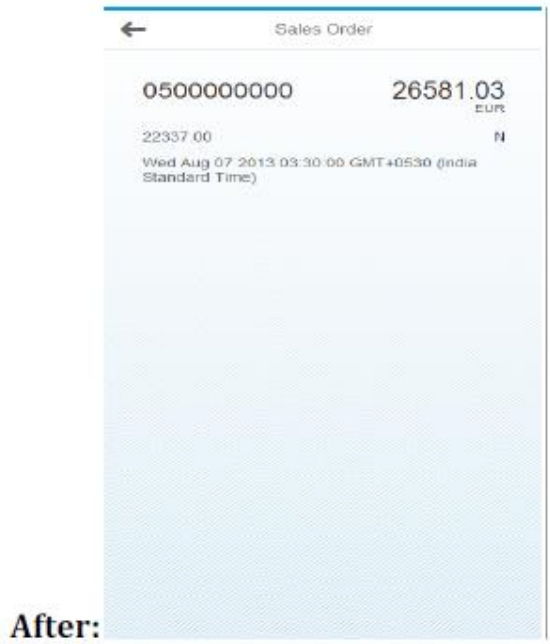


After:



Before:

After:



Description

In this exercise we will replace a couple of controls in the Master view & in the Detail view. In the Master view, rather than the simple flat list item style presented by the StandardListItem control that is in use currently, we'll present the overview of the sales orders in a more readable and useful way by using the ObjectListItem control instead.
In the Detail view, we'll make a similar change, replacing the simple layout (currently afforded by the VBox control) with a more readable display thanks to the ObjectHeader control. Along the way we'll add a few more properties from the data model, such as CurrencyCode.

Changes

Master.view.xml

1. Replace the StandardListItem control with the more powerful ObjectListItem
2. Attributes and statuses are defined by own objects
3. Save the modified Master.view.xml file.

```
<core:View
controllerName="sap.ui.demo.myFiori.view.Master"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
title="{i18n>MasterTitle}" >
<List
items="{/SalesOrderCollection}" >
<ObjectListItem
type="Active"
press="handleListItemPress"
title="{SoId}"
number="{GrossAmount}"
numberUnit="{CurrencyCode}" >
<attributes>
<ObjectAttribute text="{BuyerName}" />
</attributes>
<firstStatus>
<ObjectStatus text="{LifecycleStatus}" />
</firstStatus>
</ObjectListItem>
</List>
</Page>
</core:View>
```

Detail.view.xml

1. Replace the texts with the more beautiful ObjectHeader control (which has almost the same API as the ObjectListItem control but utilizes the space in a different way).
2. Save the modified Detail.view.xml file

```
<core:View
controllerName="sap.ui.demo.myFiori.view.Detail"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
title="Sales Order"
showNavButton="true"
navButtonPress="handleNavButtonPress" >
<ObjectHeader
title="{SoId}"
number="{GrossAmount}"
numberUnit="{CurrencyCode}" >
<attributes>
<ObjectAttribute text="{BuyerName}" />
<ObjectAttribute text="{CreatedByBp}" />
<ObjectAttribute text="{CreatedAt}" />
</attributes>
<firstStatus>
<ObjectStatus text="{LifecycleStatus}" />
</firstStatus>
</ObjectHeader>
</Page>
</core:View>
```

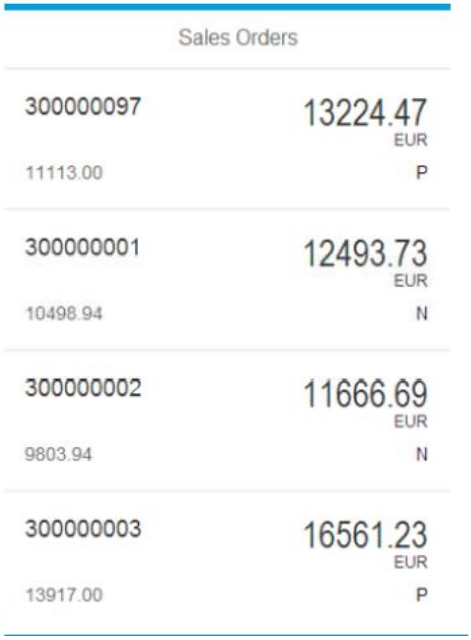
EXERCISE 3 – FORMATTER

Objective

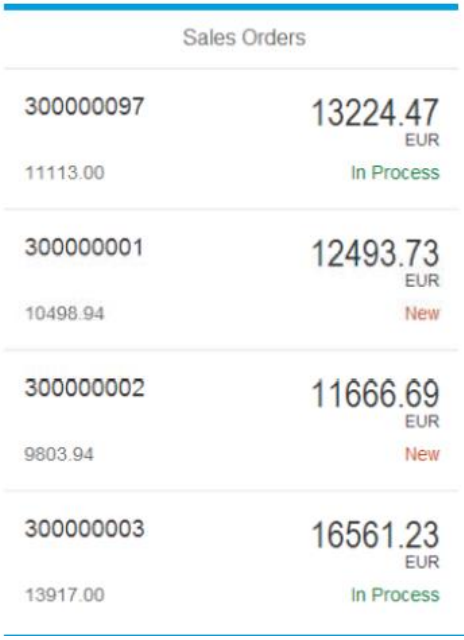
Format status, color and date properly by implementing custom formatters that are used in data binding.

Preview

Before:

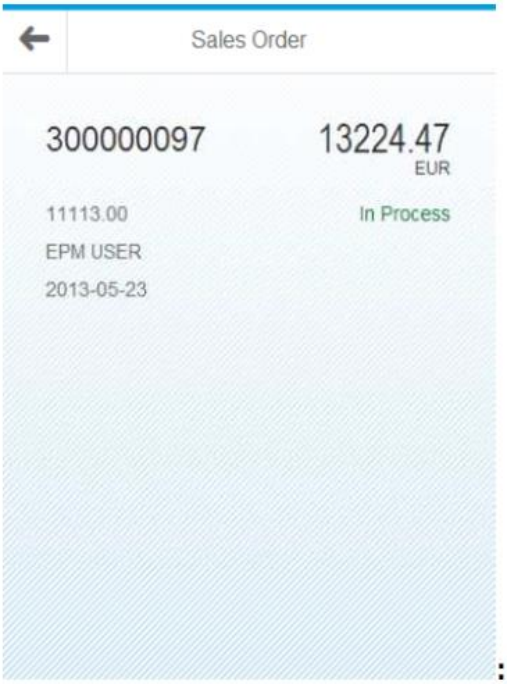


After:



Before:

After:



Description

In this exercise we will introduce a couple of formatting functions and use them in the application. They are custom functions so we put them in a module file 'Formatter.js' in a separate folder (in this case we've chosen the folder name 'util'). One of the functions uses an SAPUI5 static class for date formatting so we specify that requirement (for sap.ui.core.format.DateFormat) before defining our functions We then use the formatting functions in the Detail and Master views; in order to do this, we need to 'require' the new module in the respective controllers. To execute the formatting on the property paths from the data model (such as 'CreatedAt' or 'LifecycleStatus') we need a different binding syntax and for that we have to add a bindingSyntax parameter in the SAPUI5 bootstrap.

Changes

i18n/messageBundle.properties

1. Add two new texts to the properties file that are used to display the status

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
```

Formatter.js

2. This file contains functions to format dates, status text and status colors.
3. add new folder util dd new file formatter.js

```
jQuery.sap.declare("sap.ui.demo.myFiori.util.Formatter");
jQuery.sap.require("sap.ui.core.format.DateFormat");

sap.ui.demo.myFiori.util.Formatter = {
_statusStateMap : {
"P" : "Success",
"N" : "Warning"
},

statusText : function (value) {
var bundle = this.getModel("i18n").getResourceBundle();
return bundle.getText("StatusText" + value, "?");
},

statusState : function (value) {
var map = sap.ui.demo.myFiori.util.Formatter._statusStateMap;
return (value && map[value]) ? map[value] : "None";
},

date : function (value) {
if (value) {
var oDateFormat = sap.ui.core.format.DateFormat.getDateTimeInstance
({pattern: "yyyy-MM-dd"});
return oDateFormat.format(new Date(value));
} else {
return value;
}
}
};
```

index.html

4. Open the “index.html” with the HTML editor of eclipse by right clicking on the file and chosing “Open With > HTML Editor”
5. For the formatting we want to use the “complex” binding syntax of SAPUI5. This we enable in the bootstrap script tag.

```
<!DOCTYPE html>
<html>
…
<script
id="sap-ui-bootstrap"
src="../../resources/sap-ui-core.js"
data-sap-ui-theme="sap_bluecrystal"
data-sap-ui-libs="sap.m"
data-sap-ui-xx-bindingSyntax="complex"
"sap.ui.demo.myFiori": "./"
}' >
</script>
```

Master.view.xml

6. Use a complex binding with a formatter for the text field.
7. Use a complex binding with a formatter for the text and state field (which controls the semantical color of the status text).

```
controllerName="sap.ui.demo.myFiori.view.Master"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
title="{i18n>MasterTitle}" >
<List
items="{/SalesOrderCollection}" >
<ObjectListItem
type="Active"
press="handleListItemPress"
title="{SoId}"
number="{GrossAmount}"
numberUnit="{CurrencyCode}" >
<attributes>
<ObjectAttribute text="{BuyerName}" />
</attributes>
<firstStatus>
<ObjectStatus
text="{
path: 'LifecycleStatus',
formatter: 'sap.ui.demo.myFiori.util.Formatter.statusText'
}"
state="{
path: 'LifecycleStatus',
formatter: 'sap.ui.demo.myFiori.util.Formatter.statusState'
}" />
</firstStatus>
</ObjectListItem>
</List>
</Page>
</core:View>
```

Master.controller.js

8. Require the formatter file in the controller of the view

```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");
sap.ui.controller("sap.ui.demo.myFiori.view.Master", {
...
```

Detail.view.xml

```
<core:View controllerName="sap.ui.demo.myFiori.view.Detail"
        xmlns="sap.m" xmlns:core="sap.ui.core">
        <Page title="{i18n>
                DetailTitle}" showNavButton="true"
                navButtonPress="handleNavButtonPress">
                <ObjectHeader title="{SoId}" number="{GrossAmount}"
                        numberUnit="{CurrencyCode}">
                        <attributes>
                                <ObjectAttribute text="{BuyerName}" />
                                <ObjectAttribute text="{CreatedByBp}" />
                                <ObjectAttribute text="{
                                path: 'CreatedAt',
                        formatter: 'sap.ui.demo.myFiori.util.Formatter.date'
                                        }" />
                        </attributes>
                        <firstStatus>
                                <ObjectStatus text="{
                        path: 'LifecycleStatus',
                    formatter:'sap.ui.demo.myFiori.util.Formatter.statusText'
                                        }"
                                                state="{
                        path: 'LifecycleStatus',
                formatter:'sap.ui.demo.myFiori.util.Formatter.statusState'
                                        }" />
                        </firstStatus>
                </ObjectHeader>
        </Page>
</core:View>
```

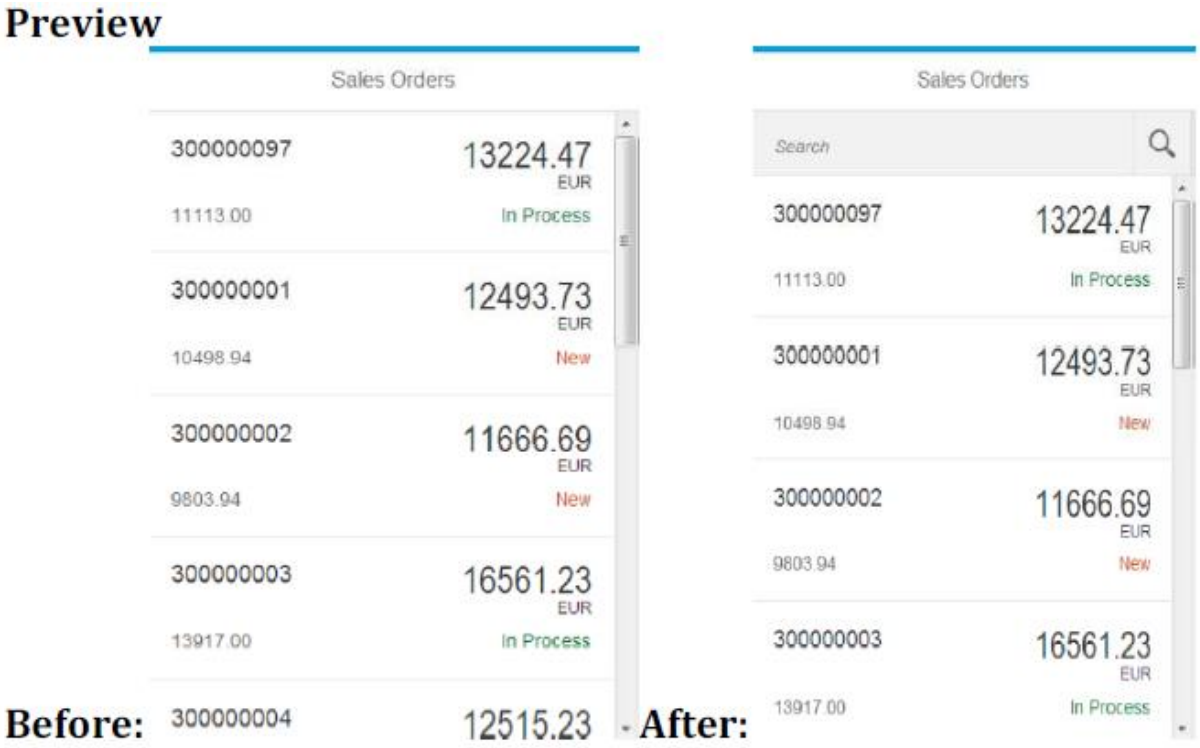
Detail.controller.js

```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");
sap.ui.controller("sap.ui.demo.myFiori.view.Detail", {
                ……..
                ……..
```

EXERCISE 4 – SEARCH

Objective

Implement a search on the master list by using ***sap.m.SearchField.***



Description

Now we're going to add a SearchField control to the initial page of the application. We'll add it as a child within the Page's 'subHeader' aggregation which expects a Bar (sap.m.Bar) control. To handle the search, we'll specify a handler for the SearchField's 'search' event. This handler 'handleSearch' is defined in the view's controller, and the search effect is achieved by adding a 'contains string' filter to the binding of the List control's items aggregation.

Changes

Master.view.xml

1. The search field is put to a bar that is placed in the sub header of the page.
2. Set the search field to 100% width to utilize all the space
3. Do not forget to add an "id" to the list in order to access the list later on in the controller

```
<core:View
controllerName="sap.ui.demo.myFiori.view.Master"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
title="{i18n>MasterTitle}" >
<subHeader>
<Bar>
<contentLeft>
<SearchField
search="handleSearch"
width="100%" >
</SearchField>
</contentLeft>
</Bar>
</subHeader>
<List
id="list"
items="{/SalesOrderCollection}" >
```

Master.controller.js

4. Implement a new handler function on the view controller. Make sure to separate the function from the other handler function with a “,”
5. Access the “query” as a parameter of the event object

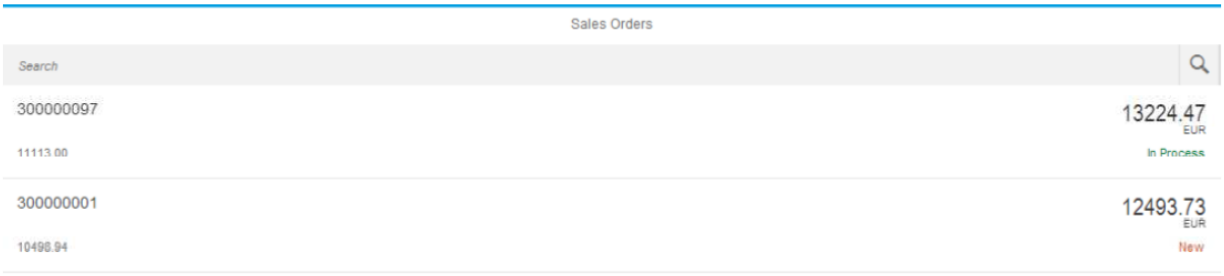
```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");
sap.ui.controller("sap.ui.demo.myFiori.view.Master", {
handleListItemPress : function (evt) {
var context = evt.getSource().getBindingContext();
this.nav.to("Detail", context);
},

handleSearch : function (evt) {
// create model filter
var filters = [];
var query = evt.getParameter("query");
if (query && query.length > 0) {
var filter = new
sap.ui.model.Filter("SoId",sap.ui.model.FilterOperator.Contains, query);
filters.push(filter);
}
// update list binding
var list = this.getView().byId("list");
var binding = list.getBinding("items");
binding.filter(filters);
}
});
```

EXERCISE 5 – SPLIT APP & SHELL

Objective

Utilize the additional space by using the sap.m.SplitApp control which shows the master and detail view next to each other. Wrap the split app in a shell that fills the remaining space on the desktop.

Preview
Before:



After:

Description

So far we've had 3 views in our application – *App*, *Master* and *Detail*. *App* is our top-level view, containing the *Master* and *Detail* views. In the *App* view we used an *App* control to contain the *Master* and *Detail* views via the *App* control's 'pages' aggregation. This is a typical scenario for an app designed primarily for a smartphone-sized screen. But if the screen size is larger (e.g. on a tablet or desktop) we want to automatically utilize the extra space and for that we will switch from the *App* control to the *SplitApp* control. Alongside swapping out the control, we'll add new view '*Empty*' which will be shown in the detail part of the *SplitApp* –straightaway, if there is enough space. Finally, for optimal utilization of space on larger devices such as desktops, we will wrap the whole thing in a *Shell* control.

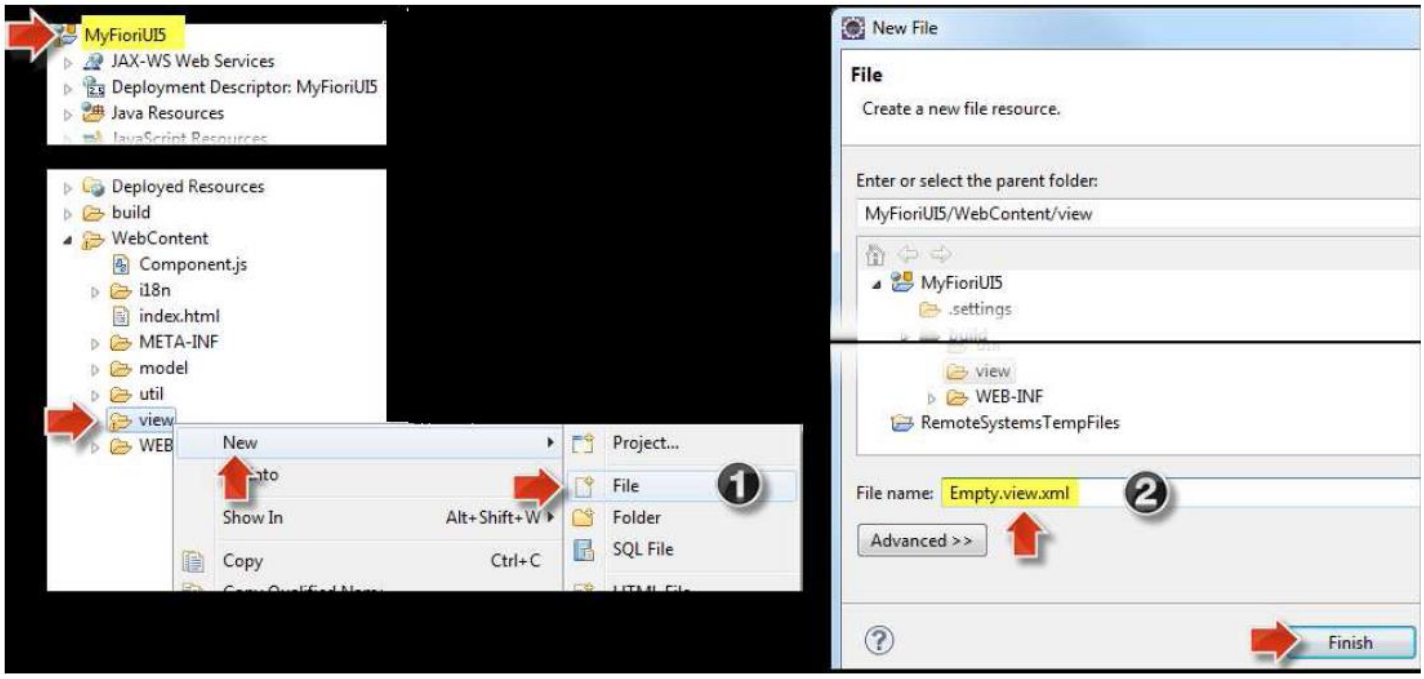
Changes

Empty.view.xml

1. create NEW XML view called Empty.view.xml .This is only a very empty page

```
<core:View
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page>
</Page>
</core:View>
```

NOTE: *to create a new* ***Empty.view.xml*** *file select node item* ***MyFioriUI5 > WebContent > view*** *in the Project Explorer (1). Click context menu* ***"New > File"****, enter file name* ***Empty.view.xml*** *in the 'New File' popup dialog (2) and press Finish.*



App.view.js

2. *Load the empty view instead of the detail view*

```
sap.ui.jsview("sap.ui.demo.myFiori.view.App", {
getControllerName: function () {
return "sap.ui.demo.myFiori.view.App";
},
createContent: function (oController) {
// to avoid scroll bars on desktop the root view must be set to block display
this.setDisplayBlock(true);

// create app
this.app = new sap.m.SplitApp();

// load the master page
var master =sap.ui.xmlview("Master","sap.ui.demo.myFiori.view.Master");
master.getController().nav = this.getController();
this.app.addPage(master, true);

// load the empty page
var empty = sap.ui.xmlview("Empty","sap.ui.demo.myFiori.view.Empty");
this.app.addPage(empty, false);

return this.app;
}
});
```

index.html

3. Wrap the split app in a shell control using the title defined before.
4. **Why in the index.html?** This is done outside of the component because if you would plug a component in the SAP Fiori Launchpad this already renders the shell.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta charset="UTF-8">
<title>My Fiori 5</title>
<script
id="sap-ui-bootstrap"
src="../../resources/sap-ui-core.js"
data-sap-ui-theme="sap_bluecrystal"
data-sap-ui-libs="sap.m"
data-sap-ui-xx-bindingSyntax="complex"
data-sap-ui-resourceroots='{
"sap.ui.demo.myFiori": "./"
}' >
</script>
<script>
new sap.m.Shell({
app : new sap.ui.core.ComponentContainer({
name : "sap.ui.demo.myFiori"
})
}).placeAt("content");
</script>
</head>
<body class="sapUiBody" id="content">
</body>
</html>
```

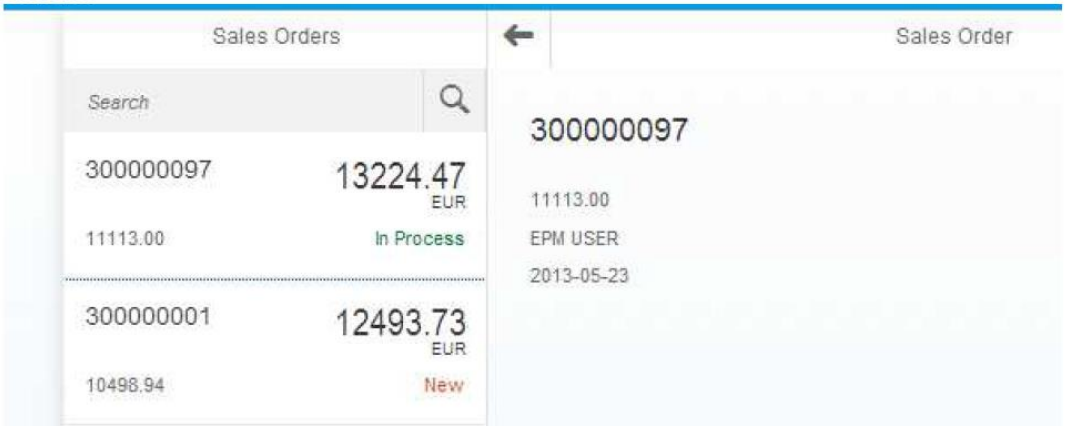
EXERCISE 6 – ADDITIONAL DEVICE ADAPTATION

Objective

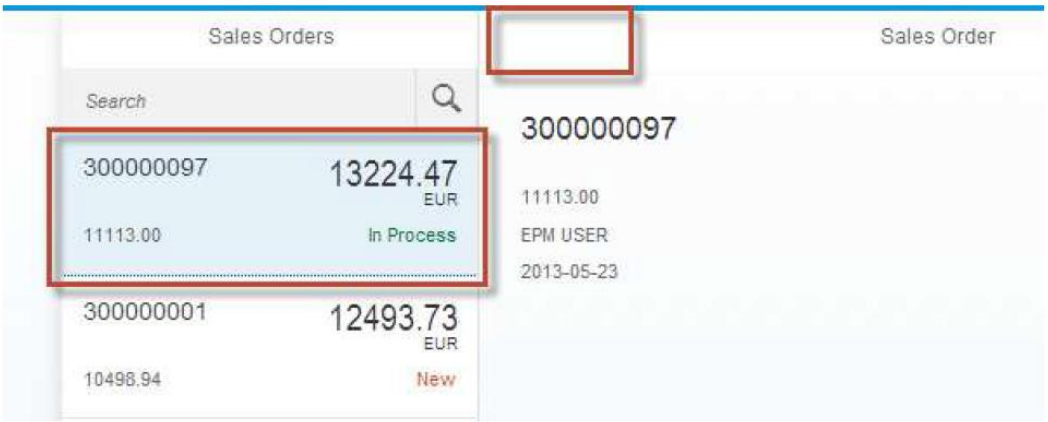
Adapt the controls to phone/tablet/desktop devices

- Show the back button in the detail page only on the phone.
- Switch the list to selection mode on the tablet and desktop.

Preview
Before:



After:



Description

If the user can see both the master and detail section of the SplitApp at the same time because, say, they're using a tablet, there's not much point in showing a back button on the detail section – it's only really relevant on smaller screen sizes where either one or the other section is visible. So we will set the visibility of the back button (referred to as the 'navigation button' in the control) to be device dependent. Also, depending on the device, we will set different list selection modes. Notice that we do the device determination up front when the application starts (in Component.js) setting the results of the determination in a one-way bound named data model, data from which can then be used in property path bindings in the Detail and Master views.

Changes

Component.js

1. Set a global model named “device”
2. Set isPhone and with the help of the “device API”.

```
jQuery.sap.declare("sap.ui.demo.myFiori.Component");
sap.ui.core.UIComponent.extend("sap.ui.demo.myFiori.Component", {
createContent : function() {
…
// set device model
var deviceModel = new sap.ui.model.json.JSONModel({
isPhone : jQuery.device.is.phone,
isNoPhone : ! jQuery.device.is.phone,
});
deviceModel.setDefaultBindingMode("OneWay");
oView.setModel(deviceModel, "device");

// done
return oView;
}
});
```

Detail.view.xml

3. Bind the showNavButton property to the device model

```
<core:View
controllerName="sap.ui.demo.myFiori.view.Detail"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
title="{i18n>DetailTitle}"
showNavButton="{device>/isPhone}"
navButtonPress="handleNavButtonPress" >
```

Master.view.xml

4. Bind the “list” mode to the device model
5. Add a select event to the list

```
<List
id="list"
select="handleListSelect"
items="{/SalesOrderCollection}" >
<ObjectListItem
press="handleListItemPress"
title="{SoId}"
number="{GrossAmount}"
numberUnit="{CurrencyCode}" >
```

Master.controller.js

6. Implement the select event in the view's controller
7. Implement Select First item Function.

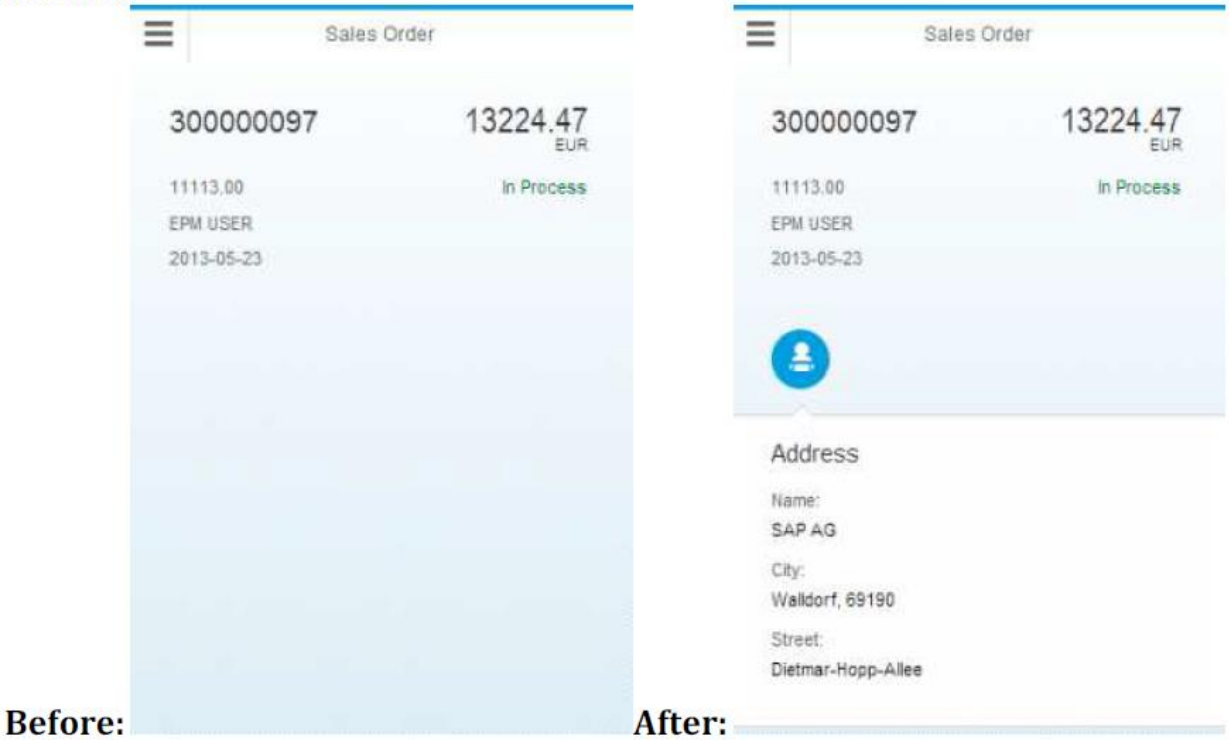
```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");
sap.ui.controller("sap.ui.demo.myFiori.view.Master", {

onInit : function(){
    //Sets the first item on initial load
     if (! jQuery.device.is.phone)
    this.getView().byId("list").attachUpdateFinished(this.setFirstItem);
},

setFirstItem: function(oEvent){
    if(oEvent.getParameters().reason === "Refresh"){
            var item = { listItem : this.getItems()[0]};
            this.getItems()[0].setSelected(false);
            this.fireSelect(item);
    }
},

……..
……..

handleSearch : function (evt) {
…
},
handleListSelect : function (evt) {
var context = evt.getParameter("listItem").getBindingContext();
this.nav.to("Detail", context);
}
});
```

EXERCISE 7 – SUPPLIER TAB

Objective

Add an info tab to the detail page that shows a little form with data of the business partner of the sales order.

Preview



Description

In this exercise we will enhance the display of the sales order detail view with a section showing the supplier name and address. In the Detail view, we'll use an IconTabBar control to introduce the information visually, and a SimpleForm control to display the information. The SimpleForm control is from the sap.ui.layout library, so we need to add this to the SAPUI5 bootstrap in the index.html too.

Changes

index.html

1. Load the additional UI library "sap.ui.layout"

```
<!DOCTYPE html>
...
<script
id="sap-ui-bootstrap"
src="../../resources/sap-ui-core.js"
data-sap-ui-theme="sap_bluecrystal"
data-sap-ui-libs="sap.m, sap.ui.layout"
data-sap-ui-xx-bindingSyntax="complex"
data-sap-ui-resourceroots='{
"sap.ui.demo.myFiori": "./"
}' >
</script>
```

Detail.view.xml

2. Set xml namespaces for the new package (form) Implement a sap.m.IconTabBar
3. Implement a sap.ui.layout.SimpleForm and bind the data. The data source will be connected in the next step.

```
controllerName="sap.ui.demo.myFiori.view.Detail"
xmlns="sap.m"
xmlns:form="sap.ui.layout.form"
xmlns:core="sap.ui.core" >
<Page
…….
<ObjectHeader
...
</ObjectHeader>
<IconTabBar
expanded="{device>/isNoPhone}" >
<items>
<IconTabFilter
icon="sap-icon://supplier">
<form:SimpleForm id="SupplierForm" minWidth="1024" >
<core:Title text="Address" />
<Label text="Name"/>
<Text text="{CompanyName}" />
<Label text="City"/>
<Text text="{City}, {PostalCode}" />
<Label text="Street"/>
<Text text="{Street}" />
</form:SimpleForm>
</IconTabFilter>
</items>
</IconTabBar>
</Page>
</core:View>
```

Detail.Controller.js

4. Bind the supplier form we just created to the data of the structure “BusinessPartner”

```
sap.ui.controller("sap.ui.demo.myFiori.view.Detail", {

handleNavButtonPress : function (evt) {
this.nav.back("Master");
},

onBeforeRendering:function(){
this.byId("SupplierForm").bindElement("BusinessPartner");
}
});
```

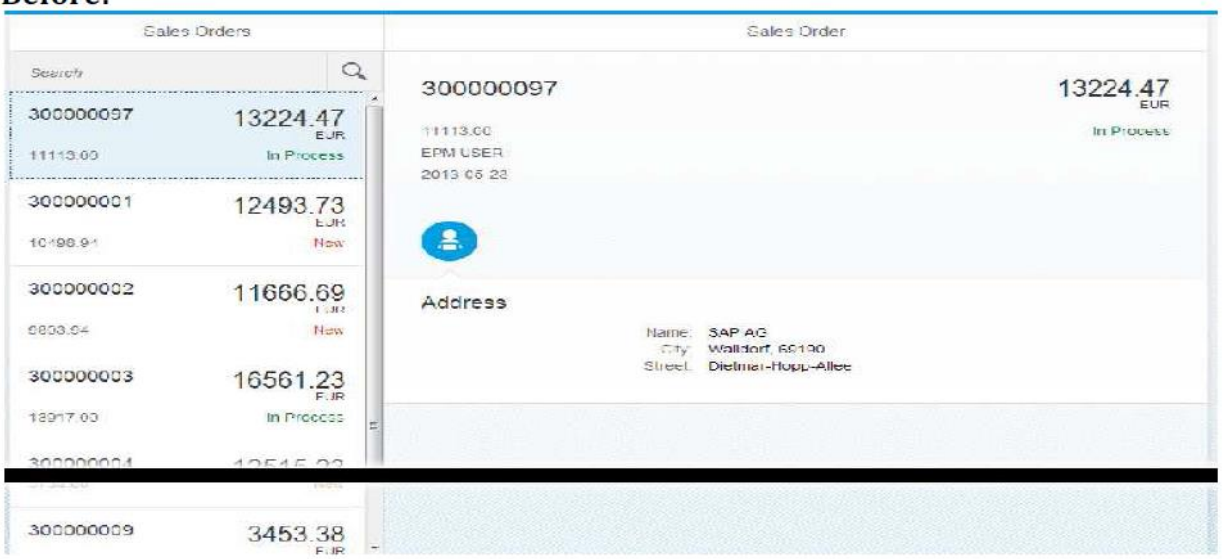
EXERCISE 8 – APPROVAL PROCESS

Objective

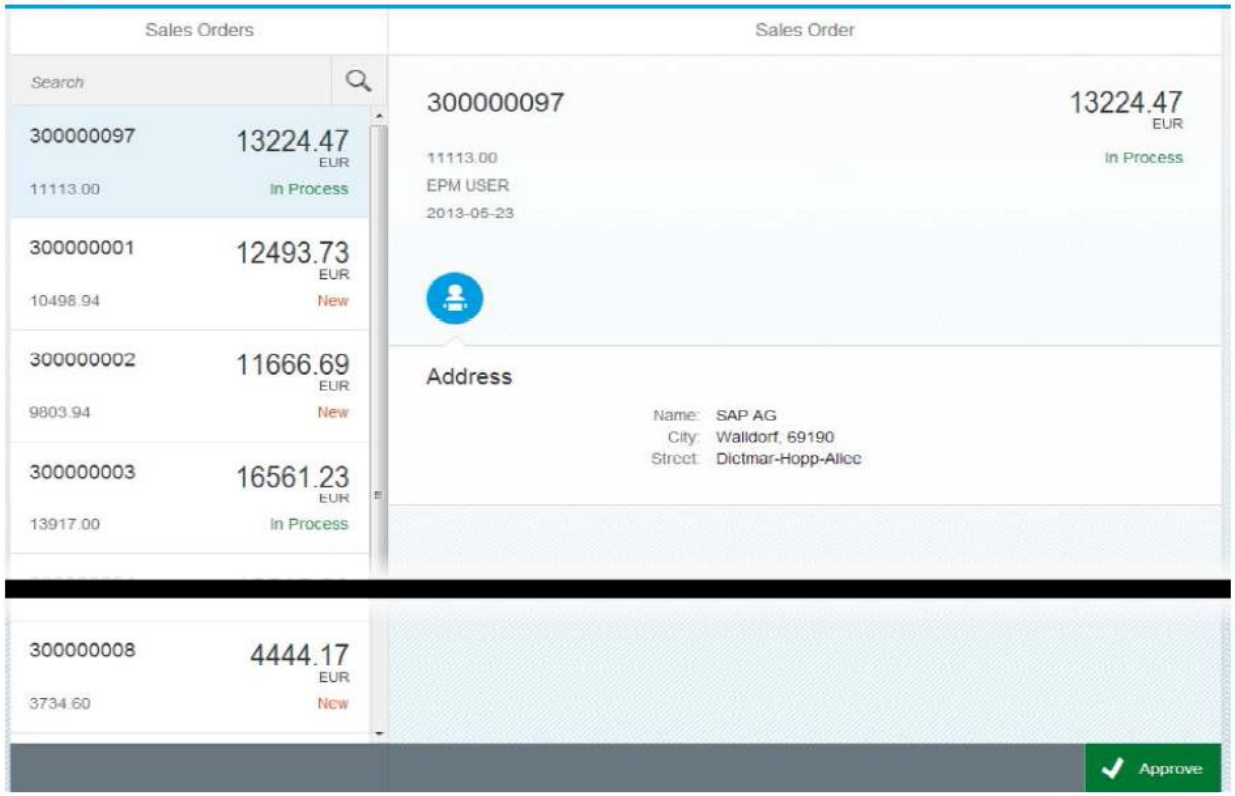
Add button to the footer of the detail page to trigger the approval of a sales order. When the user presses the button a confirmation dialog is shown. If the user confirms the dialog the sales order is deleted from the model and a confirmation message is shown.

Preview

Before:



After:



Description

To achieve the aim of this exercise, we'll be making small changes to lots of the files in the project. We need to add a footer bar (a Bar control within the footer aggregation of the Page) to each of the views (Detail, Empty and Master) to keep things visually nice and consistent. We'll add a Button control to the right side of the footer bar in the Detail view, and in the corresponding controller we'll define the function to be called ('handleApprove') when the Button's 'press' event is fired. We'll just simulate the approval process by displaying a MessageBox popup. control and then showing a MessageToast. For this we'll need to show some texts, so we'll add them to the same properties file we set up earlier in relation to the resource model.

Changes

i18n/messageBundle.properties

1. Add more texts for the approve button and dialog.

```
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
```

Detail.view.xml

2. Add a footer to the Detail page which holds the button to trigger the approval

```
< IconTabBar >
…
</IconTabBar>
<footer>
<Bar>
<contentRight>
<Button
text="{i18n>ApproveButtonText}"
type="Accept"
icon="sap-icon://accept"
press="handleApprove" />
</contentRight>
</Bar>
</footer>
</Page>
</core:View>
```

Detail.controller.js

3. First we need to register 2 more classes used to work with dialogs (MessageBox and MessageToast)
4. On handling the approve event we first show a confirmation dialog (MessageBox)
5. If the user confirms we only show a success message (MessageToast).
6. Calling a real service is not part of this exercise.

```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");
jQuery.sap.require("sap.m.MessageBox");
jQuery.sap.require("sap.m.MessageToast");
sap.ui.controller("view.Detail", {

handleNavButtonPress : function (evt) {
this.nav.back("Master");
},

handleApprove : function (evt) {
// show confirmation dialog
var bundle = this.getView().getModel("i18n").getResourceBundle();
sap.m.MessageBox.confirm(
bundle.getText("ApproveDialogMsg"),
function (oAction) {
if (sap.m.MessageBox.Action.OK === oAction) {
// notify user
var successMsg = bundle.getText("ApproveDialogSuccessMsg");
sap.m.MessageToast.show(successMsg);
// TODO call proper service method and update model (not part of this
session)
}
},
bundle.getText("ApproveDialogTitle")
);
}
});
```

Empty.view.xml

7. We now need footers in all pages for symmetry

```
<core:View
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page>
<footer>
<Bar>
</Bar>
</footer>
</Page>
</core:View>
```

Master.view.xml

8. We now need footers in all pages for symmetry

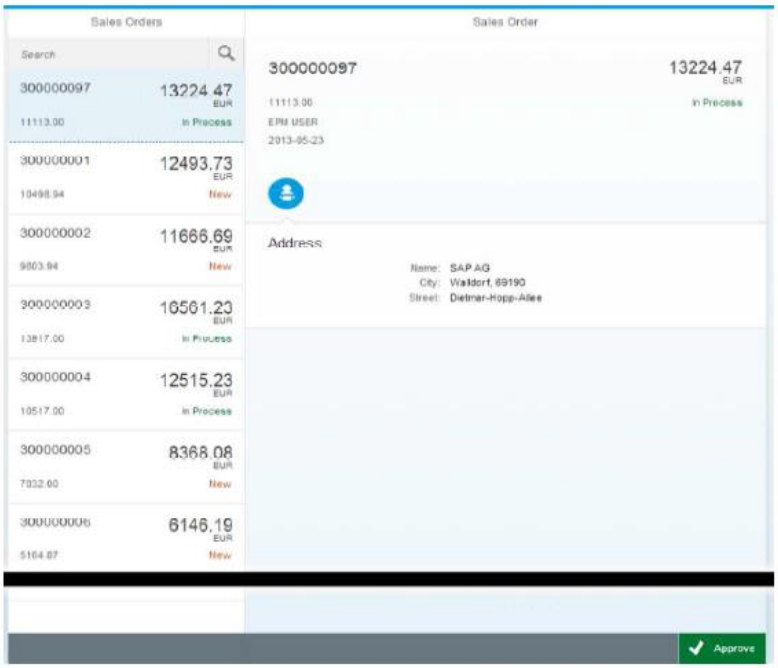
```
...
</ObjectListItem>
</List>
<footer>
<Bar>
</Bar>
</footer>
</Page>
</core:View>
```

EXERCISE 9 – LINE ITEM

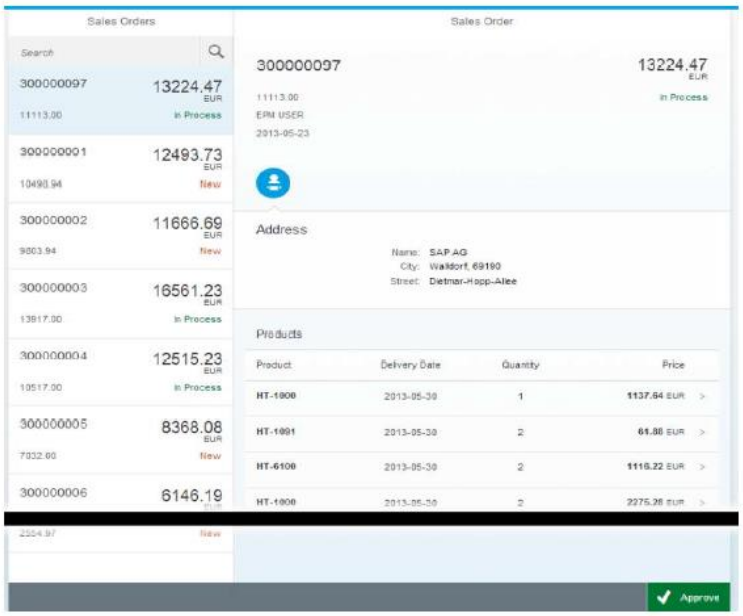
Objective

Extend the detail page with a table that shows the line items of the sales order. The rows are active and allow navigating to the new line item page.

Preview
Before:



After:



Description

In this exercise we're going to add some more details to the existing Detail view, specifically a new Table control containing the line items from the selected order. We'll put the Table control underneath the IconTabBar that we introduced in an earlier exercise. To format each order item's quantity, we'll add a further function called 'quantity' to the Formatter.js module we already have. This will then be used in the complex binding definition of the respective 'quantity' text in the table ColumnListItem's cells aggregation. We'll handle the selection of a line in the line items table with a 'handleLineItemsPress' function in the Detail view's controller. This is bound to the press event of the Table's Column ListItem as you can see in the Detail view XML below. On selection, we want to navigate to a new view, LineItem, passing the context of the selected item. So we'll create a new LineItem view, also containing a Page control with a Bar in the footer aggregation, like all the other views, and display line item details. When the navigation button is pressed we transition back to the Detail view with a simple handler 'handleNavBack' in the LineItem controller.

Changes

i18n/messageBundle.properties

1. Add more message texts

```
LineItemTableHeader=Products
LineItemTitle=Product
```

util/Formatter.js

2. We need a new formatter for quantities that removes the trailing zeros from the number

```
jQuery.sap.declare("sap.ui.demo.myFiori.util.Formatter");
jQuery.sap.require("sap.ui.core.format.DateFormat");
sap.ui.demo.myFiori.util.Formatter = {
…
},
quantity : function (value) {
try {
return (value) ? parseFloat(value).toFixed(0) : value;
} catch (err) {
return "Not-A-Number";
}
}
};
```

Detail.view.xml

3. We set a CSS class on the page control that will set proper margins on the table control in this page.
4. There is quite a bit of change to implement the table with the help of a list.

```
<core:View
controllerName="sap.ui.demo.myFiori.view.Detail"
xmlns="sap.m" xmlns:form="sap.ui.layout.form" xmlns:core="sap.ui.core" >
<Page title="{i18n>DetailTitle}"
class="sapUiFioriObjectPage"
showNavButton="{device>/isPhone}"
navButtonPress="handleNavButtonPress" >
...
</IconTabBar>
<Table
headerText="{i18n>LineItemTableHeader}"
items="{LineItems}" >
<columns>
    <Column>
    <header><Label text="Product" /></header>
    </Column>
    <Column
    minScreenWidth="Tablet"
    demandPopin="true"
    hAlign="Center" >
    <header><Label text="Delivery Date" /></header>
    </Column>
    <Column
    minScreenWidth="Tablet"
    demandPopin="true"
    hAlign="Center" >
    <header><Label text="Quantity" /></header>
    </Column>
    <Column
    hAlign="Right" >
    <header><Label text="Price" /></header>
    </Column>
</columns>
<ColumnListItem
type="Navigation"
press="handleLineItemPress" >
```

```
<cells>
<ObjectIdentifier
title="{ProductId}" />
<Text
text="{
path:'DeliveryDate',
formatter:'sap.ui.demo.myFiori.util.Formatter.date'
}"/>
<Text
text="{
path:'Quantity',
formatter:'sap.ui.demo.myFiori.util.Formatter.quantity'
}"/>
<ObjectNumber
number="{GrossAmount}"
numberUnit="{CurrencyCode}" />
</cells>
</ColumnListItem>
</Table>
<footer>
…
</footer>
</Page>
</core:View>
```

Detail.controller.js

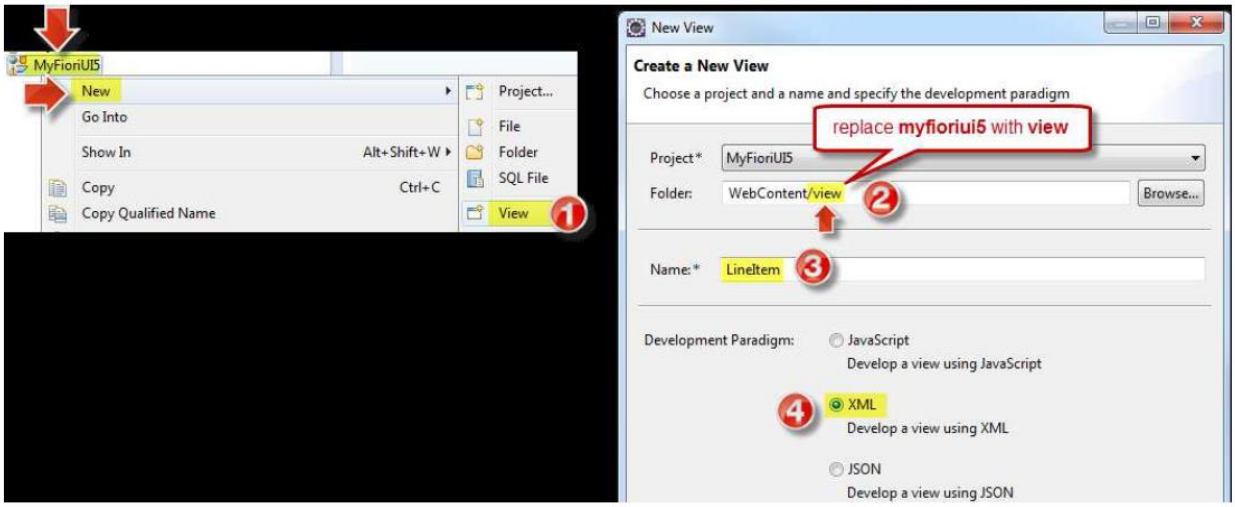
9. When a line item is pressed we navigate to the new line item page

```
…
handleApprove : function (evt) {
…
},
handleLineItemPress : function (evt) {
var context = evt.getSource().getBindingContext();
this.nav.to("LineItem", context);
}
});
```

view/LineItem.view.xml

10. Add new XML view LineItem, REPLACE all initial code
11. For the sake of simplicity we only put an object header to the line item page.

NOTE: To add the new XML view LineItem select node item MyFioriUI5 in the Project Explorer. Select context menu item “New > View” (1). In the ‘New View’ popup dialog change folder name to WebContent/view (2), enter name LineItem (3), select Development Paradigm XML (4) and press Finish. The “New View” function creates two files LineItem.controller.js and LineItem.view.xml with initial JS/XML-code inside. In both files, select all code with CTRL+A and delete it before adding the highlighted code of this exercise.




```
<core:View
controllerName="sap.ui.demo.myFiori.view.LineItem"
xmlns="sap.m"
xmlns:core="sap.ui.core" >
<Page
id="page"
title=" {i18n>LineItemTitle}"
showNavButton="true"
navButtonPress="handleNavBack" >
<footer>
<Bar>
</Bar>
</footer>
<content>
<ObjectHeader
title="{ProductId}"
number="{GrossAmount}"
numberUnit="{CurrencyCode}" >
```

```
<attributes>
<ObjectAttribute text="{
path:'DeliveryDate',
formatter:'sap.ui.demo.myFiori.util.Formatter.date'
}" />
<ObjectAttribute text="{
path:'Quantity',
formatter:'sap.ui.demo.myFiori.util.Formatter.quantity'
}" />
</attributes>
</ObjectHeader>
</content>
</Page>
</core:View>
```

LineItem.controller.js

12. open newly added with view/LineItem.view.xml, REPLACE all initial code
13. We only need to handle the back navigation to the Detail page

```
sap.ui.controller("sap.ui.demo.myFiori.view.LineItem", {
handleNavBack : function (evt) {
this.nav.back("Detail");
}
});
```