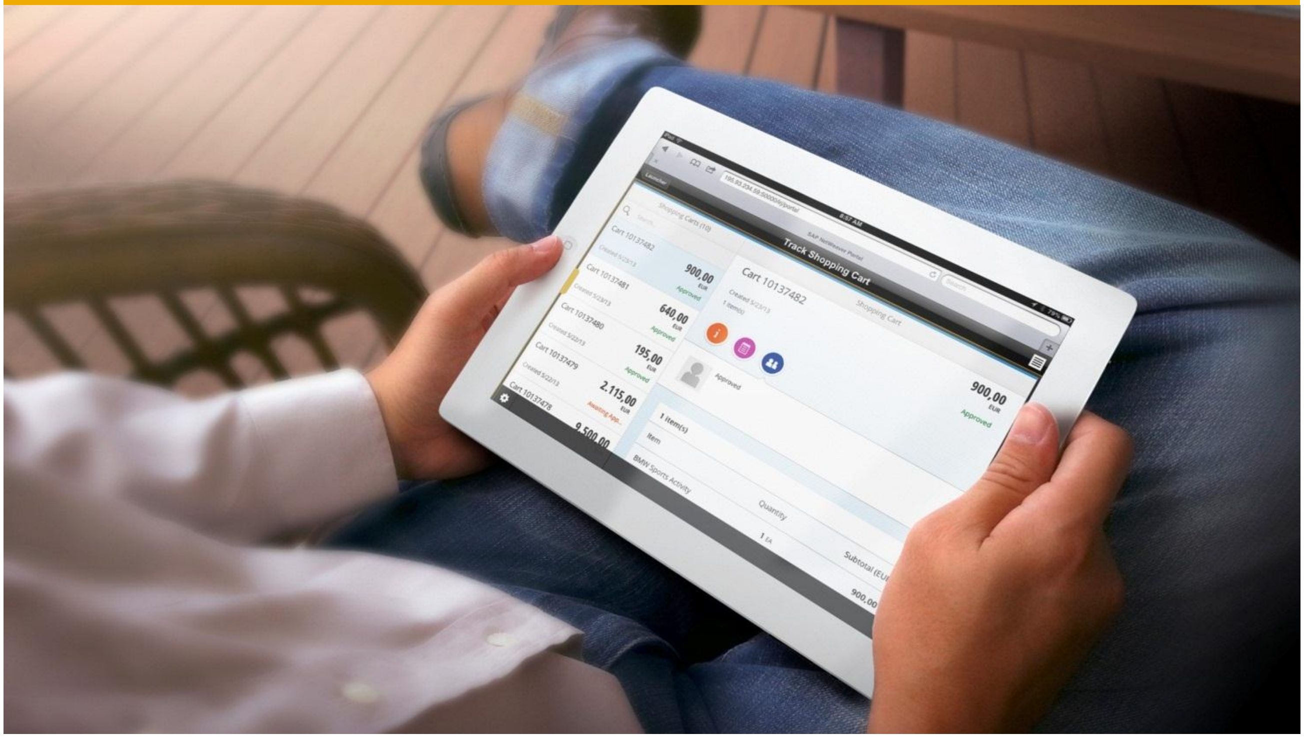


Building SAP Fiori-like UIs with SAPUI5

Exercises / Solutions

Thomas Marz, Frederic Berg, Bertram Ganz, Oliver Graeff / SAP AG
DJ Adams / Bluefin Solutions

January 2014

TABLE OF CONTENTS

RELATED RESOURCES

Building SAP Fiori-like UIs with SAPUI5

- SCN Blog “Building SAP-Fiori-like UIs with SAPUI5 in 10 exercises”, Bertram Ganz, Jan 2014: teaser blog for this tutorial document with download links and community discussion
- SCN Blog “Building SAP Fiori-like UIs with SAPUI5” by DJ Adams, SAP Mentor, Oct 2013: links to screencasts that describe and showcase the Fiori-like SAPUI5 application documented in this tutorial.
- YouTube video “SAPUI5/Fiori - Exploration of an App” by DJ Adams, SAP Mentor
- YouTube video “SAP CodeTalk: SAPUI5 & Fiori - Parts #1 and #2 ” by DJ Adams, SAP Mentor

UI development toolkit for HTML5 (SAPUI5)

- SCN blog “Get to Know the UI Development Toolkit for HTML5 (aka SAPUI5)”, Bertram Ganz: Overview introduction to SAPUI5, links to related information at one place
- UI development toolkit for HTML5 Developer Center on SCN
- SAPUI5 Demo Kit (Documentation, API-Reference, ...)

APPLIES TO

The exercises described in this document are based on the **SAPUI5 Runtime version 1.16.4** that is comprised in the following SAP platform releases:

- SAP NetWeaver AS ABAP 7.0/7.01/7.02/7.03/7.31: UI add-on 1.0 for SAP NetWeaver SPS 06
- SAP NetWeaver AS ABAP 7.40 SPS 6
- SAP NetWeaver AS Java 7.31 SPS 10
- SAP HANA Platform SPS 07: SAP HANA Extended Application Services (SAP HANA XS)
- OpenUI5 1.16.7 (see http://sap.github.io/openui5/)
- Evaluation package for UI development toolkit for HTML5 1.16.3 (available on UI5 Dev Center on SCN)

NOTE: You **do not need any SAP backend** to build and run the SAPUI5 application sample described in this document. The Fiori-like sample runs locally on a web server that is provided with the SAPUI5 tools in Eclipse IDE. The application's business data is retrieved from a **mock resource** (via JSON model) so that no backend service is required.

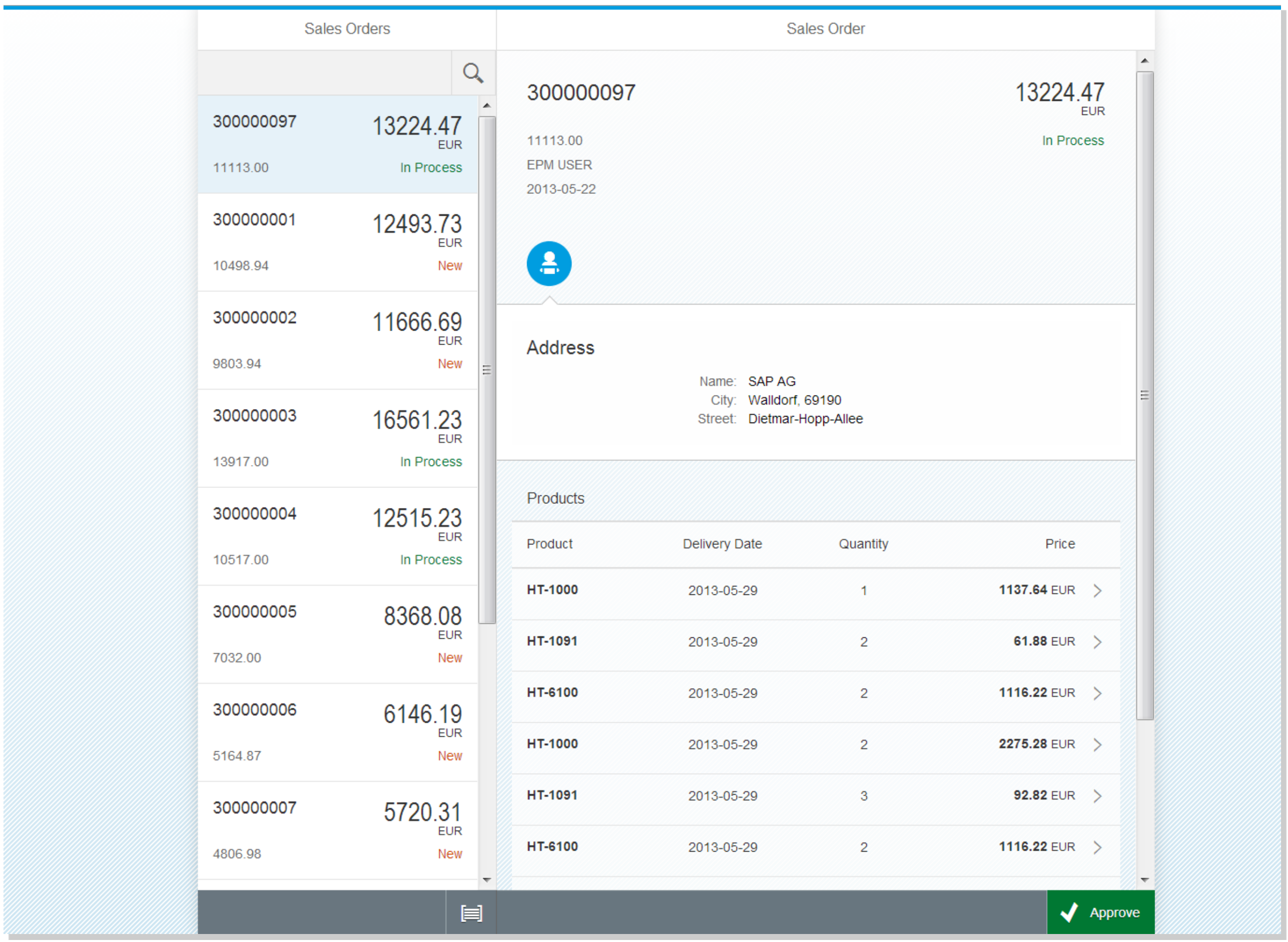
SAPUI5 PROJECT SOURCES

You can download all SAPUI5 project sources that are described in this document on SCN here …
BuildingSAPFiori-likeUIsWithSAPUI5_Projects.zip

- **myFiori0**: initial SAPUI5 project you need as template to start exercise 0.
- **myFiori1-myFiori9**: SAPUI5 projects with "incremental" source code for exercises 1-9.
- **myFiori10**: SAPUI5 project with the final SAP Fiori-like UI5 application after completion of exercise 10.

SAP FIORI-LIKE APPLICATION UI

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



300000097	13224.47 EUR	11113.00	In Process
300000001	12493.73 EUR	10498.94	New
300000002	11666.69 EUR	9803.94	New
300000003	16561.23 EUR	13917.00	In Process
300000004	12515.23 EUR	10517.00	In Process
300000005	8368.08 EUR	7032.00	New
300000006	6146.19 EUR	5164.87	New
300000007	5720.31 EUR	4806.98	New

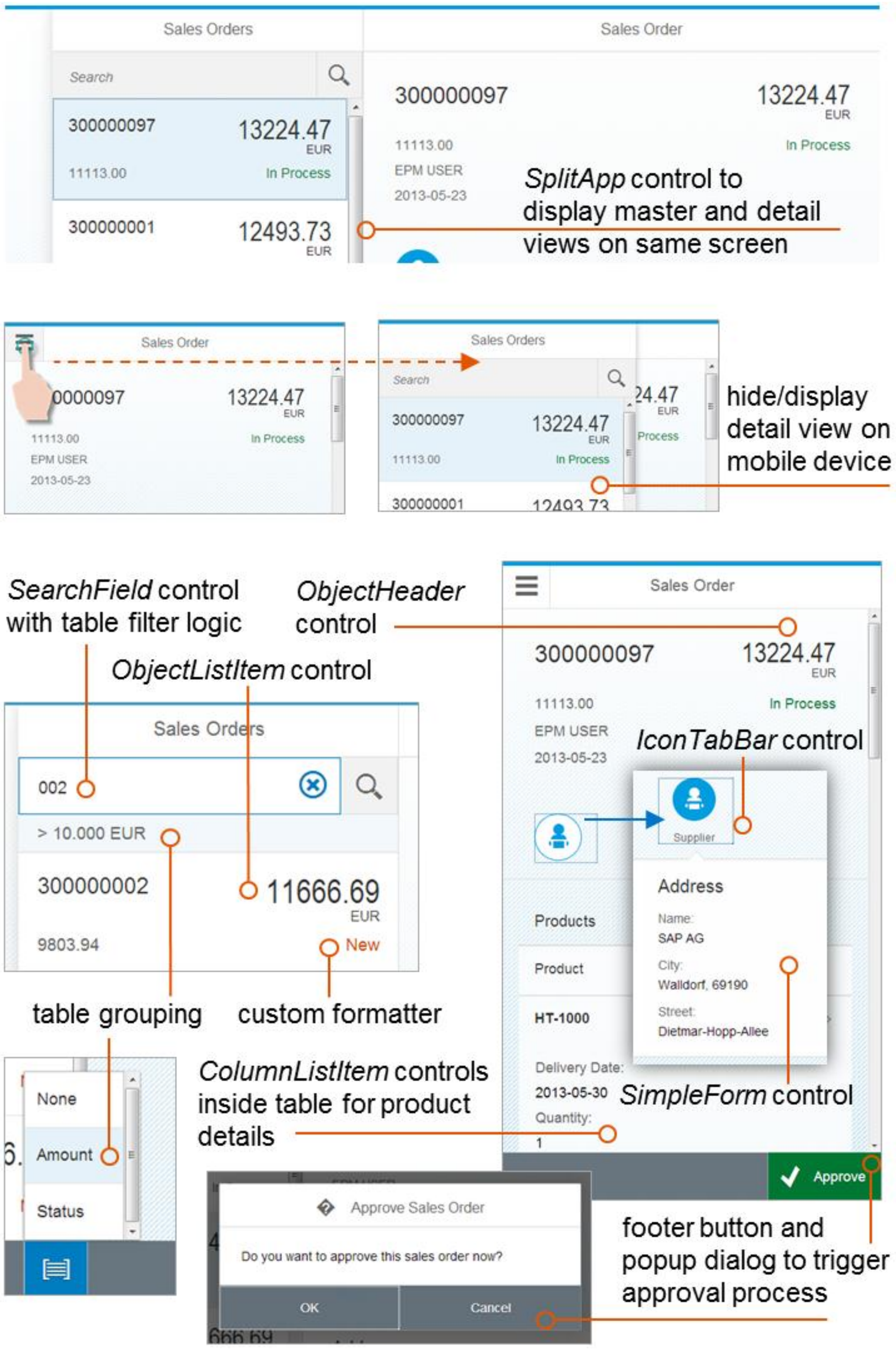
300000097 13224.47 EUR
11113.00
EPM USER
2013-05-22
In Process

Address

Name: SAP AG
City: Walldorf, 69190
Street: Dietmar-Hopp-Allee

Products

Product	Delivery Date	Quantity	Price
HT-1000	2013-05-29	1	1137.64 EUR
HT-1091	2013-05-29	2	61.88 EUR
HT-6100	2013-05-29	2	1116.22 EUR
HT-1000	2013-05-29	2	2275.28 EUR
HT-1091	2013-05-29	3	92.82 EUR
HT-6100	2013-05-29	2	1116.22 EUR

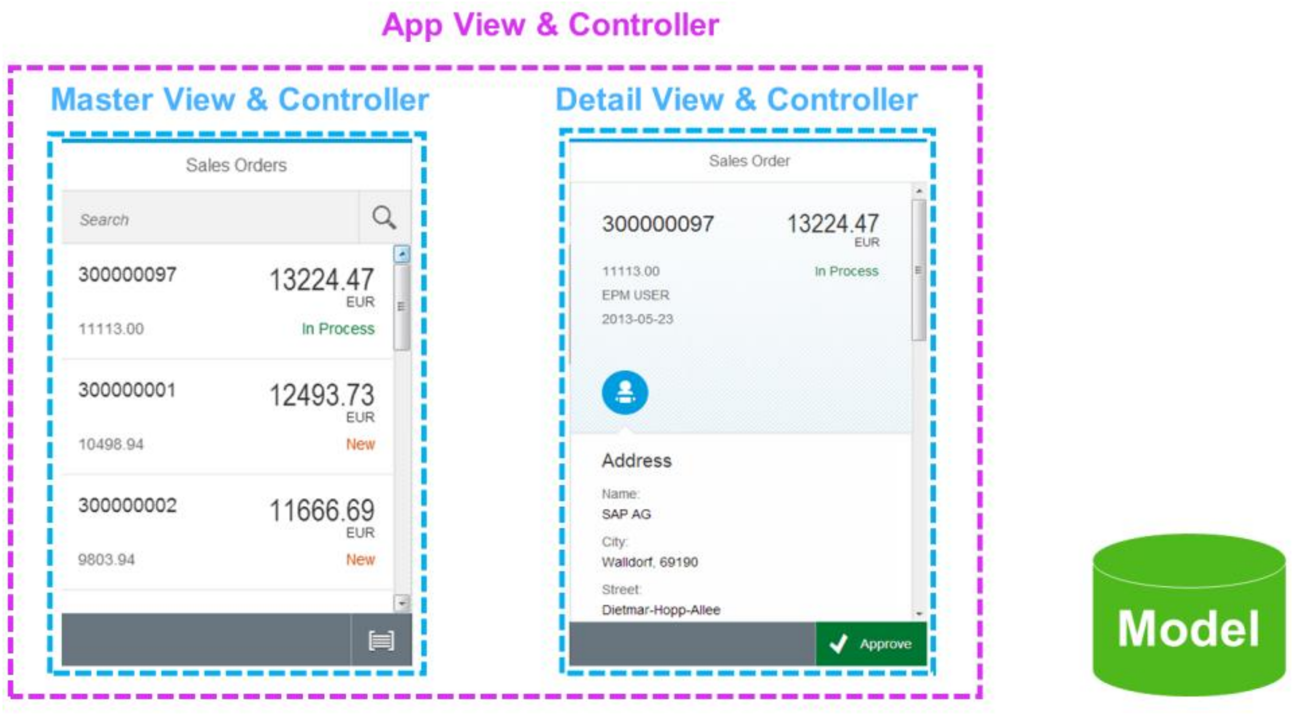


Do you want to approve this sales order now?

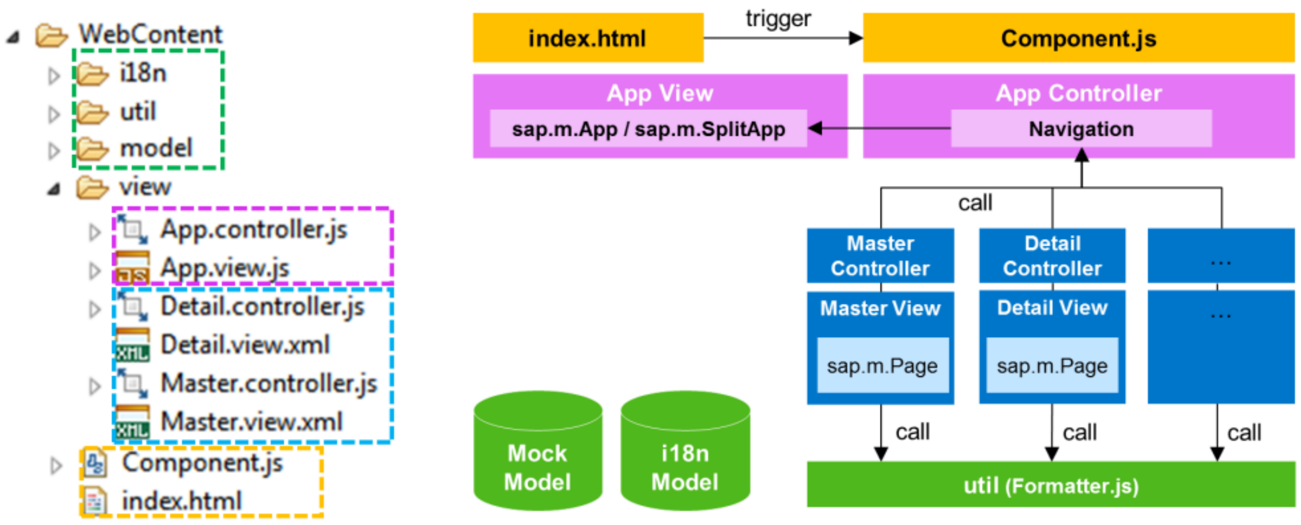
SAP FIORI-LIKE APLICATION ARCHITECTURE

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* and *Detail*. *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-dependant texts, JSON model with mock data (to be replaced with e.g. 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, that we incrementally build in the following 11 exercises, are displayed in this diagram:



EXERCISE 0 – GETTING STARTED

Objective
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 (via the SAP HANA Cloud 90 days trial license). Or follow SAP note 1763144 - UI development toolkit for HTML5 Eclipse Tools 1.6.4 that describes how to install the productive version of the SAPUI5 developer tools.

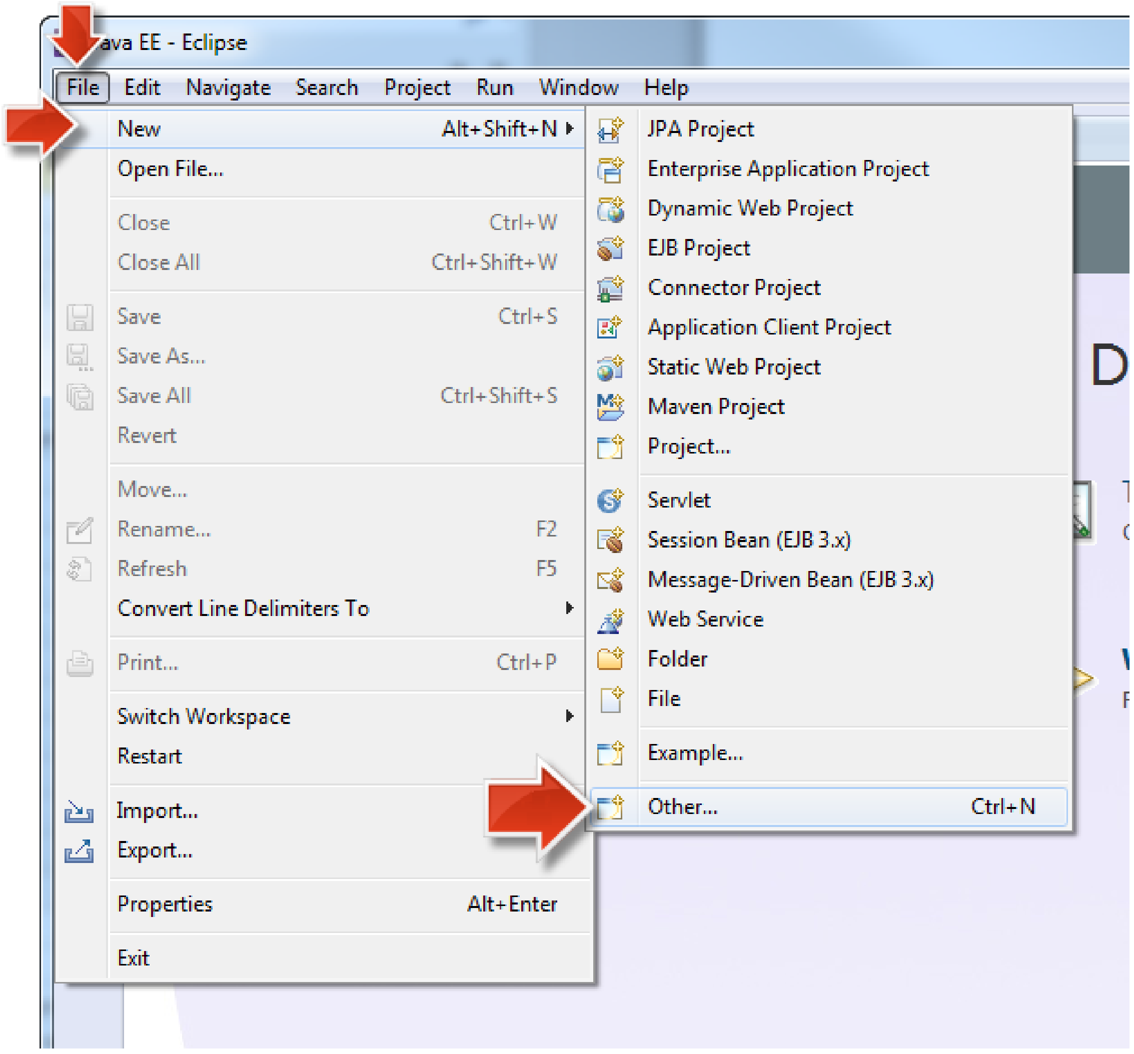
2. Download the ZIP-file containing all SAPUI5 project sources that are need for this document:

   from SAP Community Network: **BuildingSAPFiori-likeUIsWithSAPUI5_Projects.zip**

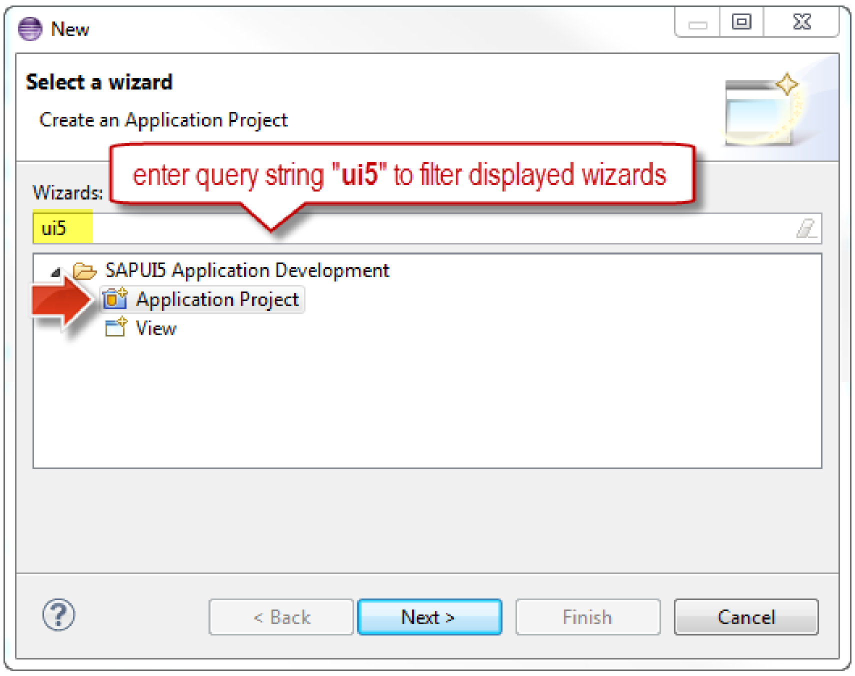
3. Extract ZIP-file to a local folder on your file system (in exercise step 8 on the initial SAPUI5 project import this folder is named ***<myUI5FioriProjectsFolder>*****).**

4. Start your Eclipse IDE with the SAPUI5 developer tools installed.

5. Go to **"File"->"New"->"Other…"** to create a new SAPUI5 Application.



6. Filter for Wizards including “**UI5**” and select “**Application Project**”. Click “**Next**”.



New

Select a wizard

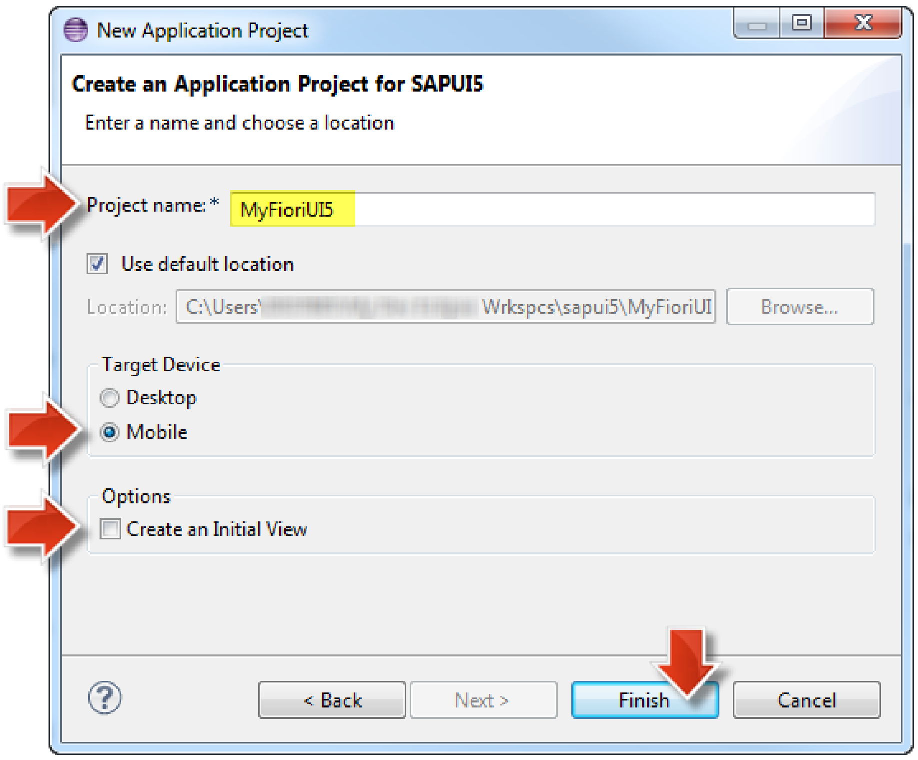
Create an Application Project

Wizards: ui5

enter query string "**ui5**" to filter displayed wizards

SAPUI5 Application Development
- Application Project
- View

7. Enter your project name **MyFioriUI5**, chose “**mobile**” as the target device, <u>uncheck</u> “**Create an Initial View**” and click on **Finish**.



New Application Project

Create an Application Project for SAPUI5

Enter a name and choose a location

Project name: * MyFioriUI5

☑ Use default location

Location: C:\Users\[redacted]Wrkspcs\sapui5\MyFioriUI

Target Device
- ○ Desktop
- ● Mobile

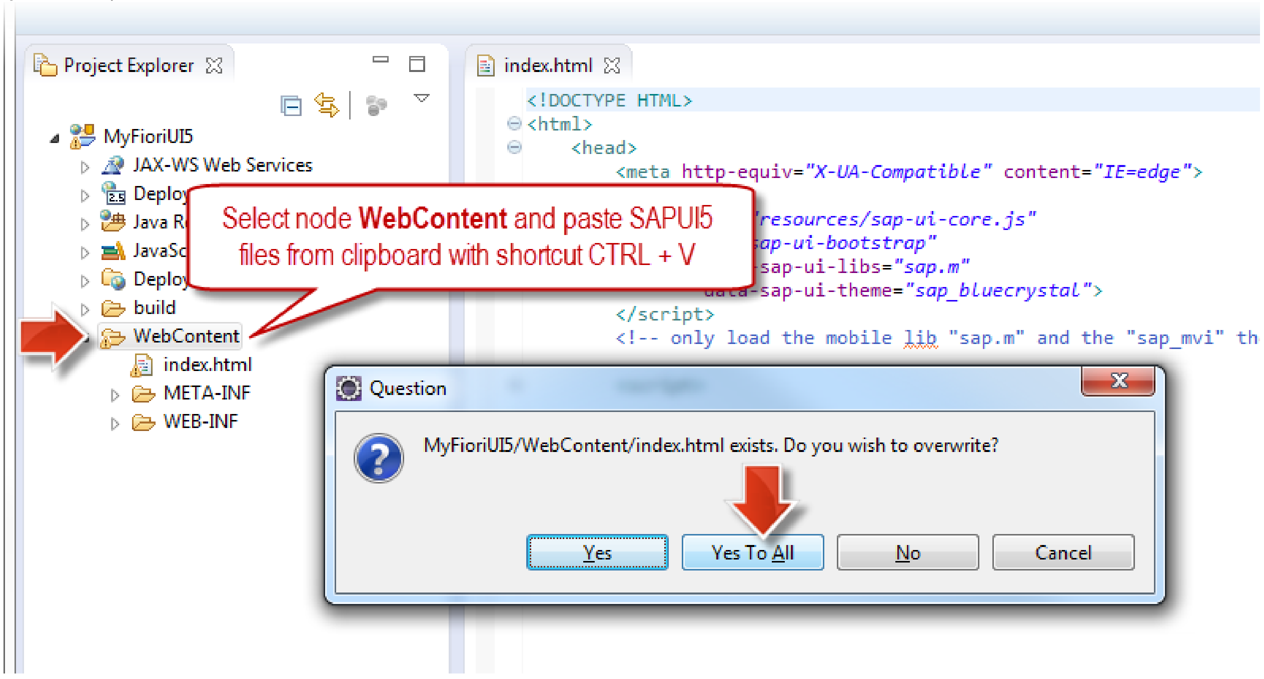
Options
- ☐ Create an Initial View

8. In a new Eclipse installation you might minimize the Eclipse *Welcome page* to see the Java EE perspective with the Project Explorer that now contains the **MyFioirUI5** project.

9. Open the local folder *<myUI5FioriProjectsFolder>* where you extracted the SAPUI5 projects in ZIP file *BuildingSAPFiori-likeUIsWithSAPUI5_Projects.zip*.

10. Open the folder “***<myUI5FioriProjectsFolder>/*myFiori0**”, select (CTRL-A) and copy the contents (CTRL-C).

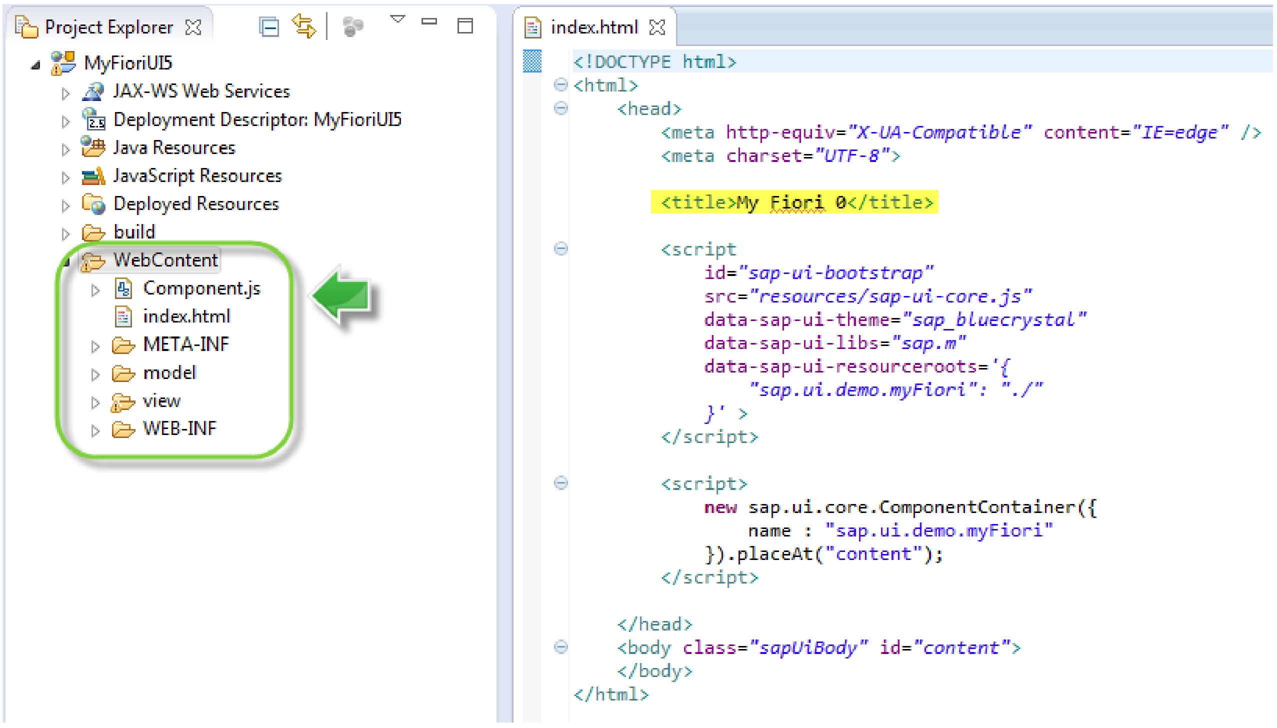
11. In Eclipse, select the folder “**WebContent**” of your SAPUI5 Application project and paste the sample files (CTRL-V). Confirm with “**Yes to All**”.



```
<!DOCTYPE HTML>
<html>
    <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
                    'resources/sap-ui-core.js"
                    ap-ui-bootstrap"
                    sap-ui-libs="sap.m"
                data-sap-ui-theme="sap_bluecrystal">
        </script>
        <!-- only load the mobile lib "sap.m" and the "sap_mvi" th
```

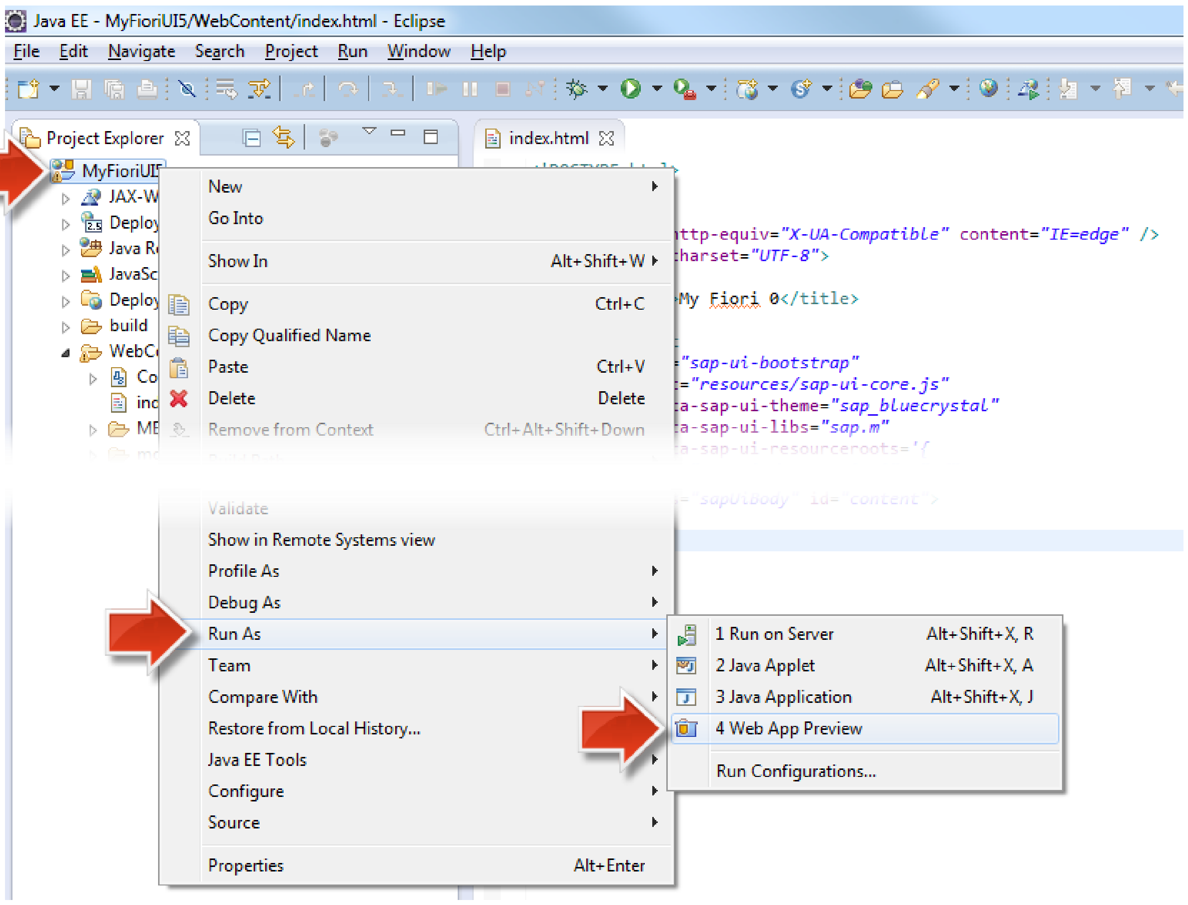
MyFioriUI5/WebContent/index.html exists. Do you wish to overwrite?

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

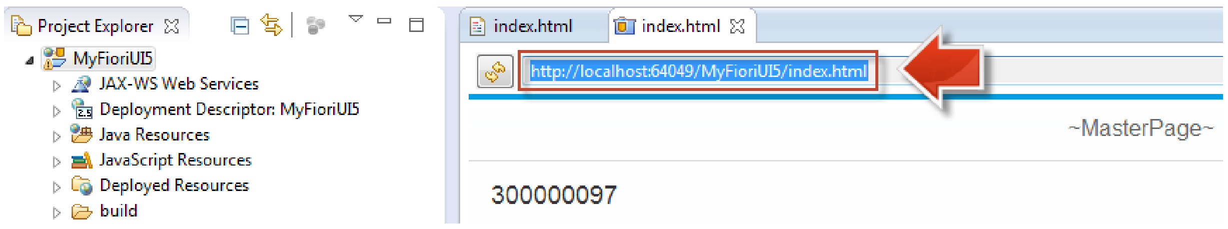


```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta charset="UTF-8">

        <title>My Fiori 0</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-resourceroots='{
                "sap.ui.demo.myFiori": "./"
            }' >
        </script>

        <script>
            new sap.ui.core.ComponentContainer({
                name : "sap.ui.demo.myFiori"
            }).placeAt("content");
        </script>

    </head>
    <body class="sapUiBody" id="content">
    </body>
</html>
```

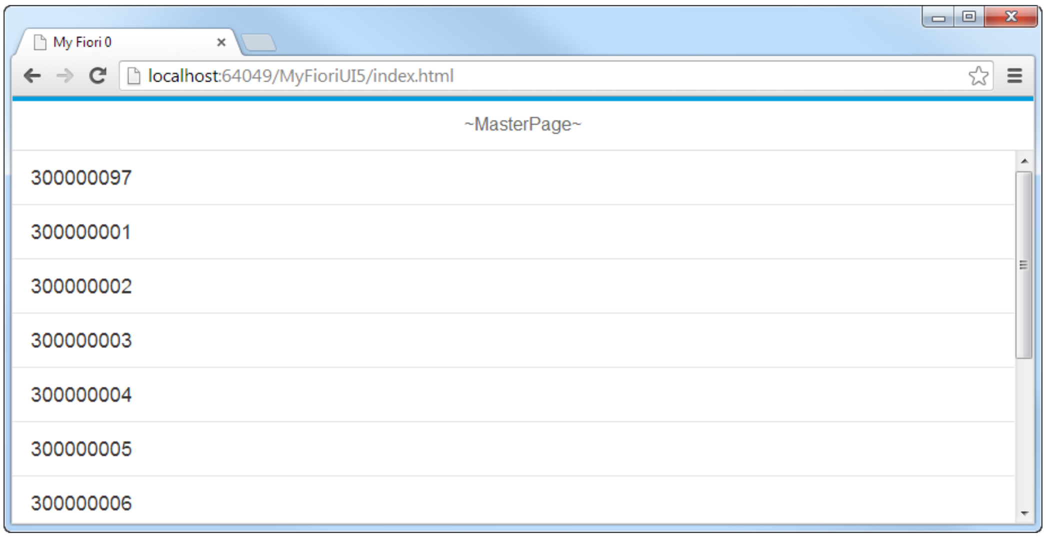
12. Right-click on the project node *MyFioirUI5* and select context menu item **“Run As”->”Web App Preview”**.


Java EE - MyFioriUI5/WebContent/index.html - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
index.html
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Validate
Show in Remote Systems view
Profile As
Debug As
Run As
Team
Compare With
Restore from Local History...
Java EE Tools
Configure
Source
Properties Alt+Enter
1 Run on Server Alt+Shift+X, R
2 Java Applet Alt+Shift+X, A
3 Java Application Alt+Shift+X, J
4 Web App Preview
Run Configurations...

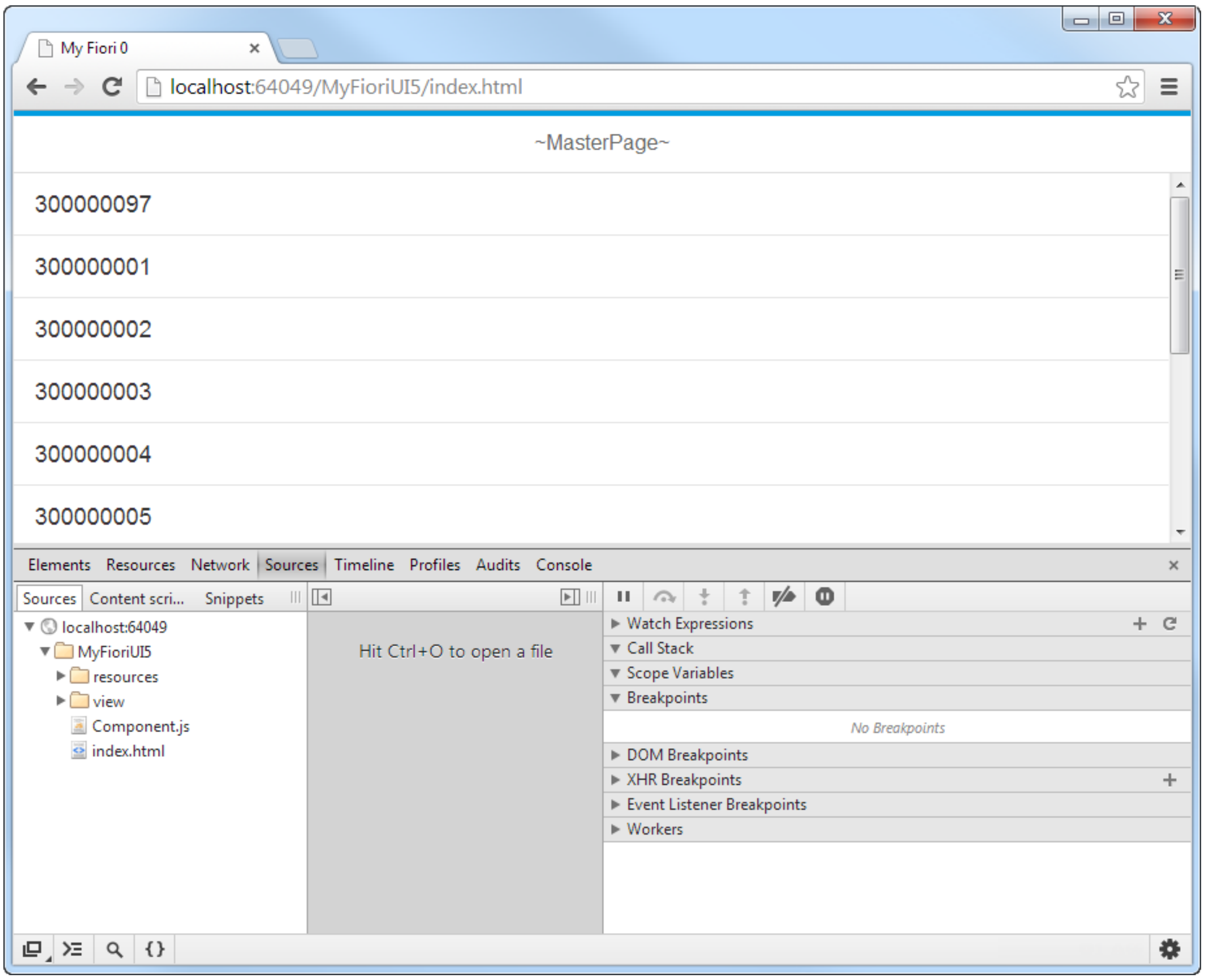
13. The application will be started and the preview will appear in eclipse. Copy the application URL to the clipboard (Ctrl+C).



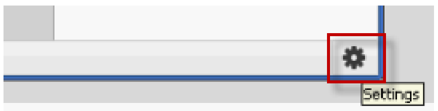
14. Start **Google Chrome** via the Shortcut on your desktop and paste the application URL.



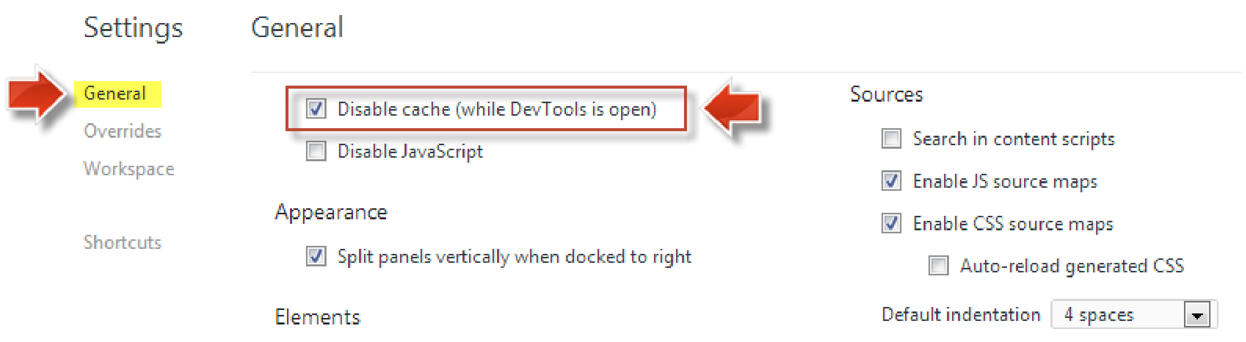
15. Press **F12** to start the Chrome Developer Tools



16. Click on the settings button in the bottom right corner.



17. Under "**Settings -> General**" chose the option **"Disable cache (while the DevTools is open)"**



Settings

General
Overrides
Workspace
Shortcuts

General

- [✓] Disable cache (while DevTools is open)
- [ ] Disable JavaScript

Appearance

- [✓] Split panels vertically when docked to right

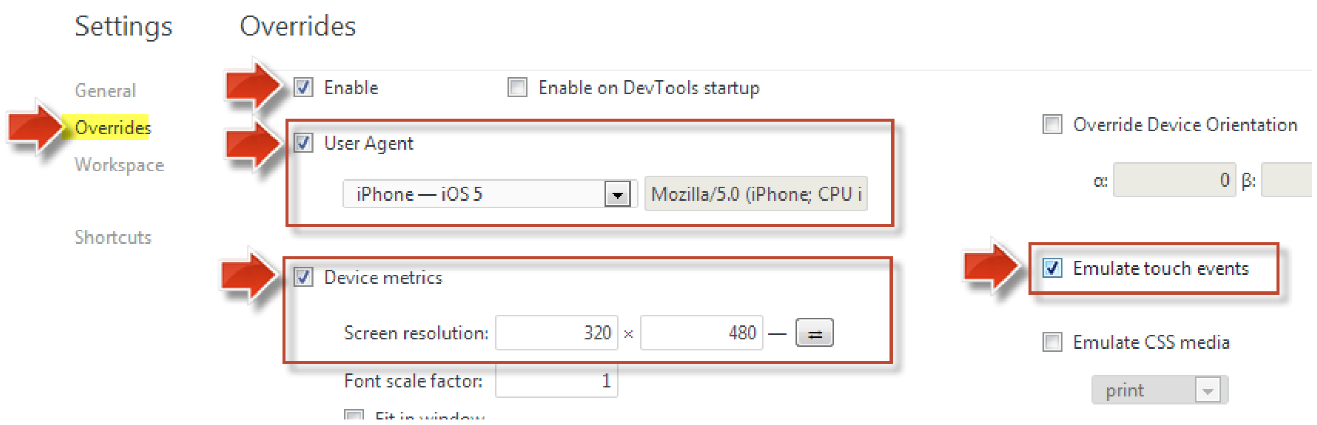
Elements

Sources

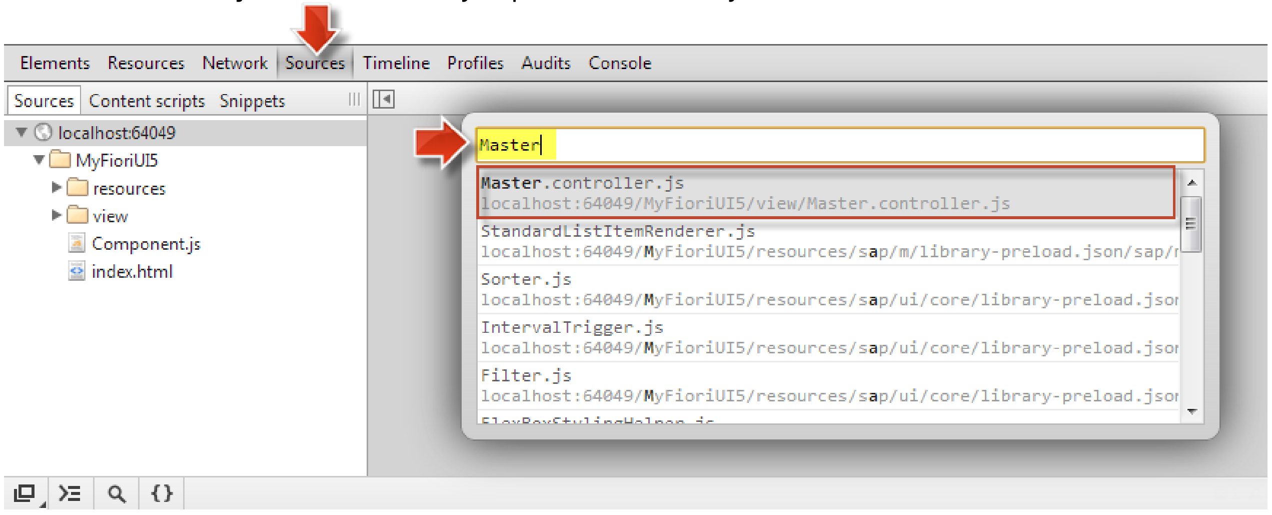
- [ ] Search in content scripts
- [✓] Enable JS source maps
- [✓] Enable CSS source maps
- [ ] Auto-reload generated CSS

Default indentation: 4 spaces

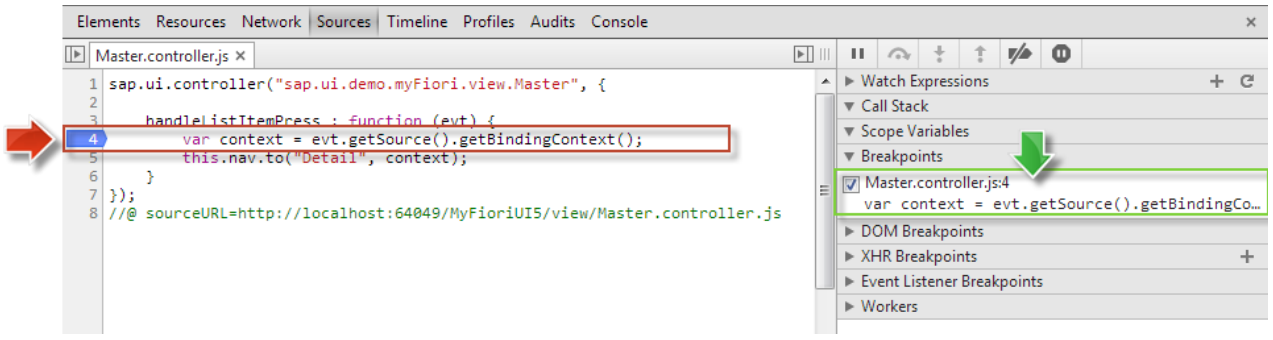
18. Under **Settings -> Overrides**: Mark check box “Enable”. Set the User Agent to “**iPhone 5**”, set the Device metrics to **320x480** and enable “**Emulate touch events**”. Close the “**Settings**” popup.



19. 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.

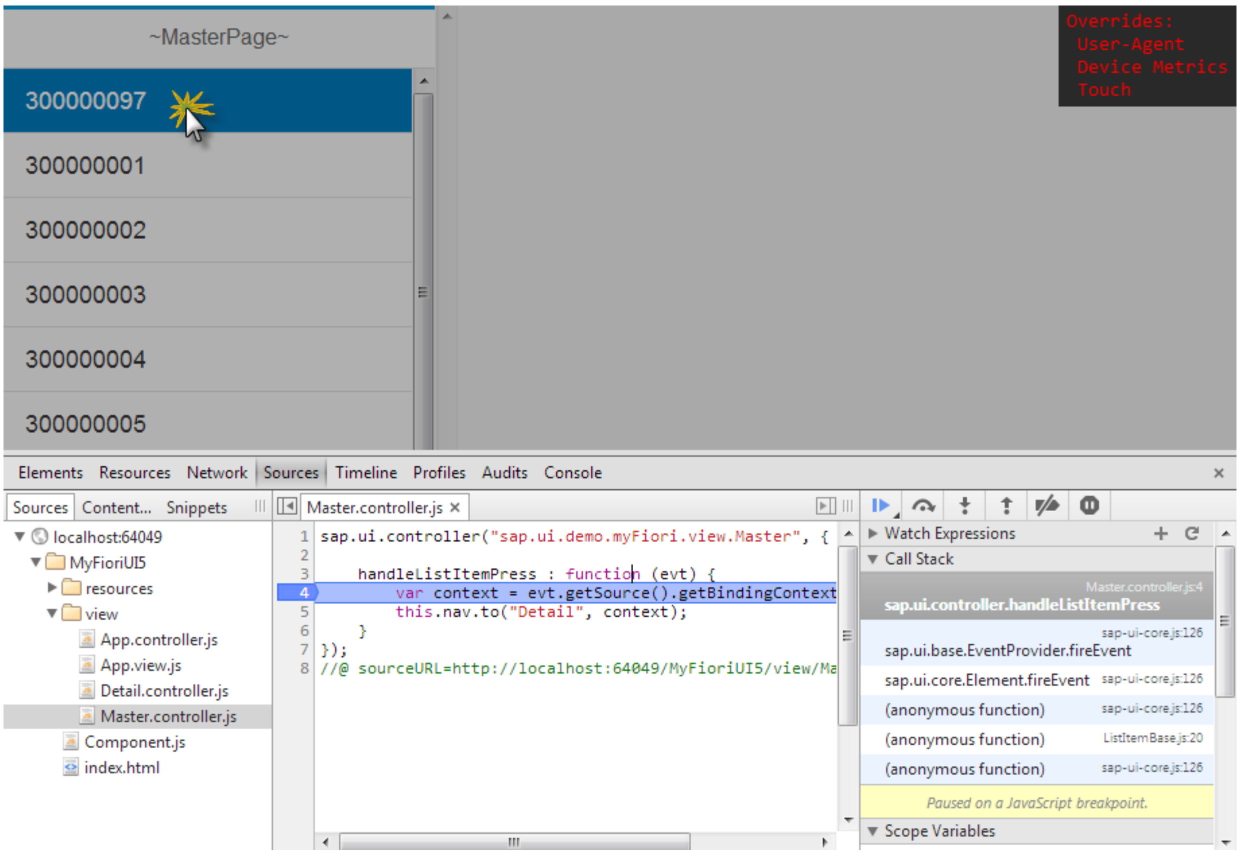

Elements Resources Network Sources Timeline Profiles Audits Console
Sources Content scripts Snippets
localhost:64049
MyFioriUI5
resources
view
Component.js
index.html
Master
Master.controller.js
localhost:64049/MyFioriUI5/view/Master.controller.js
StandardListItemRenderer.js
localhost:64049/MyFioriUI5/resources/sap/m/library-preload.json/sap/r
Sorter.js
localhost:64049/MyFioriUI5/resources/sap/ui/core/library-preload.jsor
IntervalTrigger.js
localhost:64049/MyFioriUI5/resources/sap/ui/core/library-preload.jsor
Filter.js
localhost:64049/MyFioriUI5/resources/sap/ui/core/library-preload.jsor

20. 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.


Elements Resources Network Sources Timeline Profiles Audits Console
Master.controller.js
Watch Expressions
Call Stack
Scope Variables
Breakpoints
Master.controller.js:4
var context = evt.getSource().getBindingCo…
DOM Breakpoints
XHR Breakpoints
Event Listener Breakpoints
Workers

```
1 sap.ui.controller("sap.ui.demo.myFiori.view.Master", {
2
3     handleListItemPress : function (evt) {
4         var context = evt.getSource().getBindingContext();
5         this.nav.to("Detail", context);
6     }
7 });
8 //@ sourceURL=http://localhost:64049/MyFioriUI5/view/Master.controller.js
```

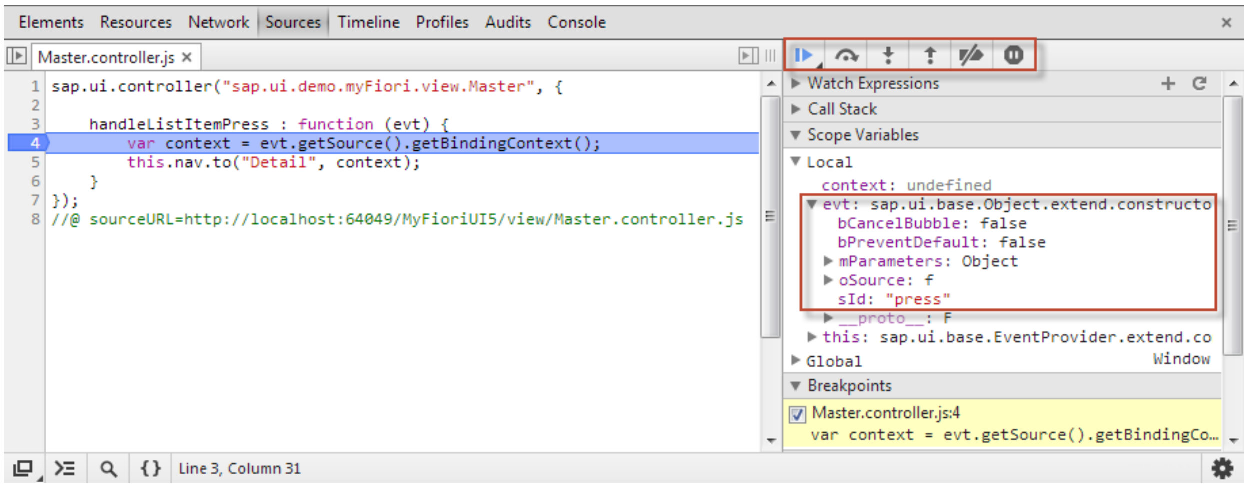
21. Now click on a line item in the running application. This causes the application to stop at the breakpoint.



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

    handleListItemPress : function (evt) {
        var context = evt.getSource().getBindingContext
        this.nav.to("Detail", context);
    }
});
//@ sourceURL=http://localhost:64049/MyFioriUI5/view/Ma
```

```
sap.ui.controller.handleListItemPress          Master.controller.js:4
sap.ui.base.EventProvider.fireEvent            sap-ui-core.js:126
sap.ui.core.Element.fireEvent                  sap-ui-core.js:126
(anonymous function)                           sap-ui-core.js:126
(anonymous function)                           ListItemBase.js:20
(anonymous function)                           sap-ui-core.js:126
Paused on a JavaScript breakpoint.
```

22. 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.

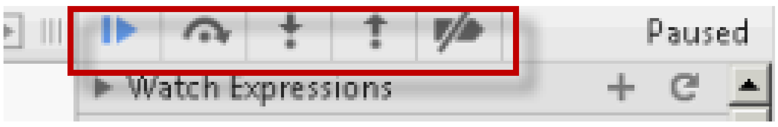


```
sap.ui.controller("sap.ui.demo.myFiori.view.Master", {

    handleListItemPress : function (evt) {
        var context = evt.getSource().getBindingContext();
        this.nav.to("Detail", context);
    }
});
//@ sourceURL=http://localhost:64049/MyFioriUI5/view/Master.controller.js
```

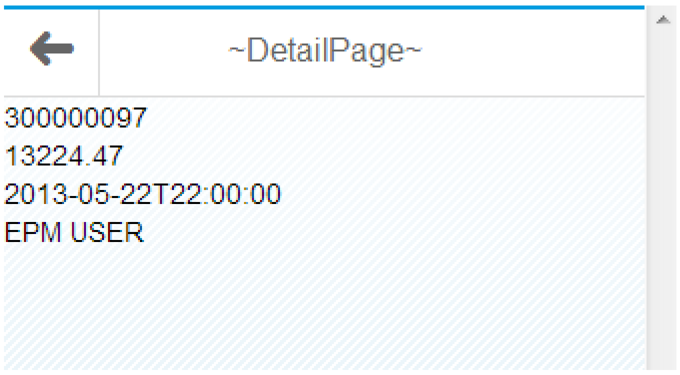
```
▼ Local
    context: undefined
  ▼ evt: sap.ui.base.Object.extend.constructo
      bCancelBubble: false
      bPreventDefault: false
    ▶ mParameters: Object
    ▶ oSource: f
      sId: "press"
    ▶ __proto__: F
  ▶ this: sap.ui.base.EventProvider.extend.co
▶ Global                                   Window
```

```
☑ Master.controller.js:4
  var context = evt.getSource().getBindingCo…
```

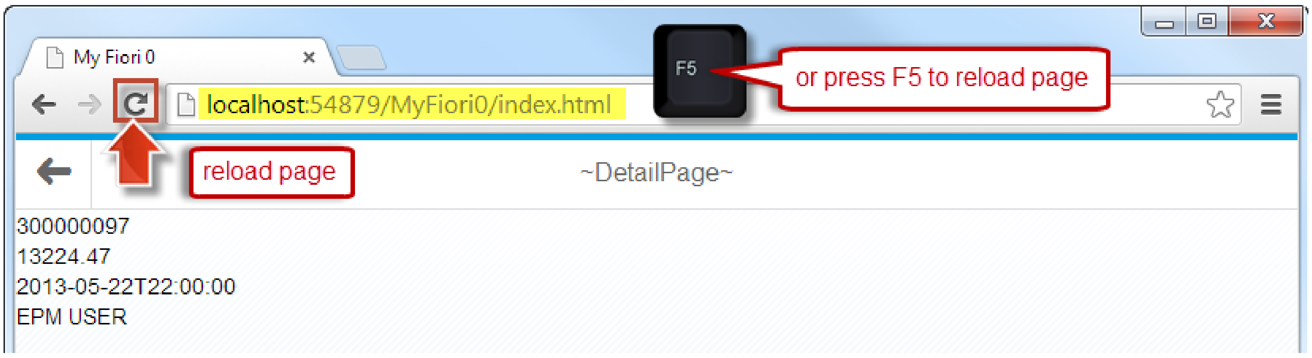
23. Click on the "**Play**" button (blue) to resume the application execution.



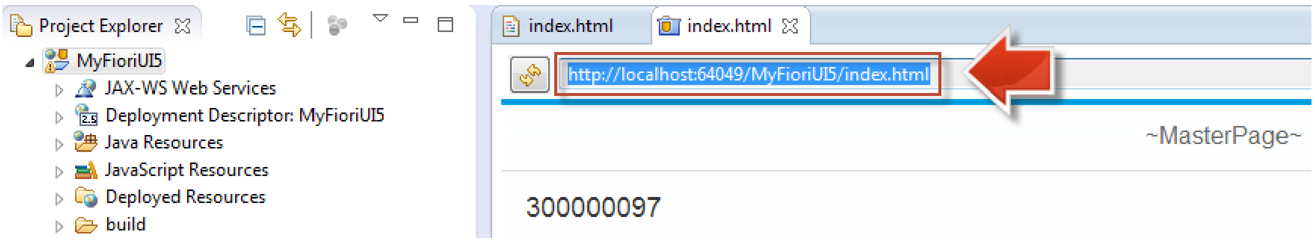
24. The application now displays the **DetailPage**


~DetailPage~
300000097
13224.47
2013-05-22T22:00:00
EPM USER

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**:


My Fiori 0
localhost:54879/MyFiori0/index.html
F5
or press F5 to reload page
reload page
~DetailPage~
300000097
13224.47
2013-05-22T22:00:00
EPM USER

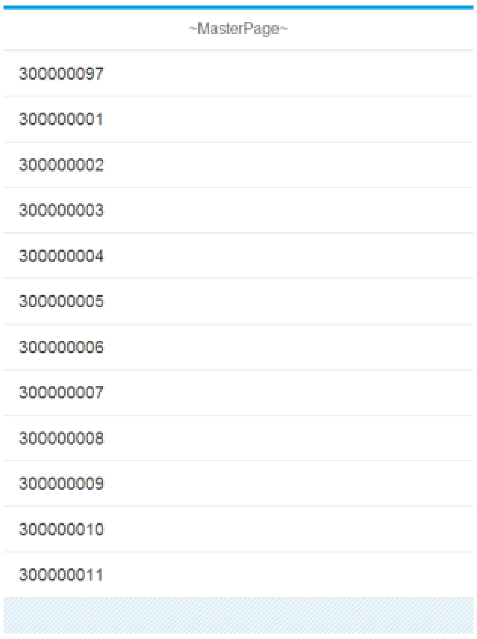
NOTE: Keep the preview in Eclipse open throughout the session. If you close the preview you cannot test the application in Chrome anymore. If you have closed the preview you need to reopen it one more time and COPY & PASTE THE NEW URL to the Chrome browser again.

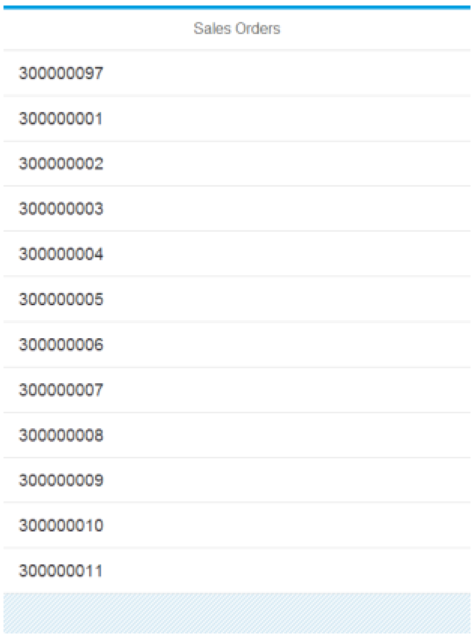


EXERCISE 1 – RESOURCE MODEL

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

Preview

Before: 
~MasterPage~
300000097
300000001
300000002
300000003
300000004
300000005
300000006
300000007
300000008
300000009
300000010
300000011

After: 
Sales Orders
300000097
300000001
300000002
300000003
300000004
300000005
300000006
300000007
300000008
300000009
300000010
300000011

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

i18n/messageBundle.properties (ADD NEW FOLDER i18n > ADD NEW FILE Formatter.js)
- Create a new folder named "**i18n**", add new file **messageBundle.properties** and put the above content there
- Make sure the file does NOT start with an empty line
- Save the new message bundle file with **CTRL+S**

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
```

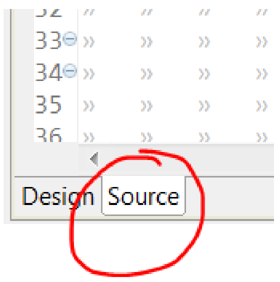
Component.js
- The message bundle is loaded with the help of a *ResourceModel*
- The *ResourceModel* is made available as global model under the name "**i18n**"

```
        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;
       }
```

view/Master.view.xml

- This file is opened with the XML editor of Eclipse. Switch to the "**Source**" tab of the XML editor to change the source code.



- Switch the title to point to the "**i18n**" model and there to the text "**MasterTitle**"
- 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}" >
       …
```

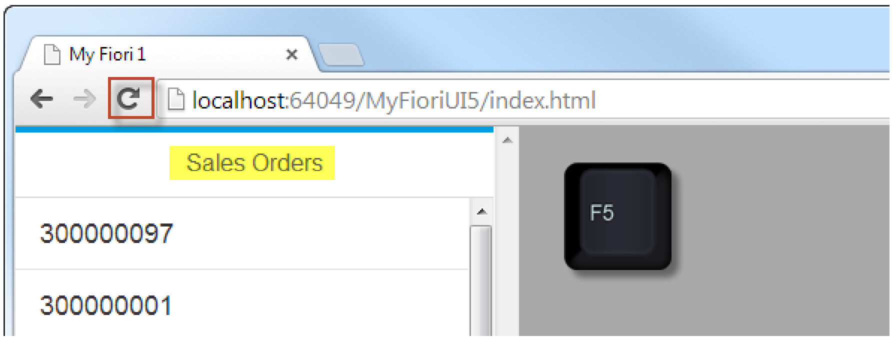
view/Detail.view.xml

- Also adjust the title of the detail view
- Save the modified **Detail.view.xml** file with shortcut **CTRL+S**

```
<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.**



Further Reading:

- ModelViewController: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/MVC.1.html
- Component Concept: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Components.html
- Databinding: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/DataBinding.html
- Localization: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/I18NinAppDev.html
- ResourceModel: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/ResourceModel.html

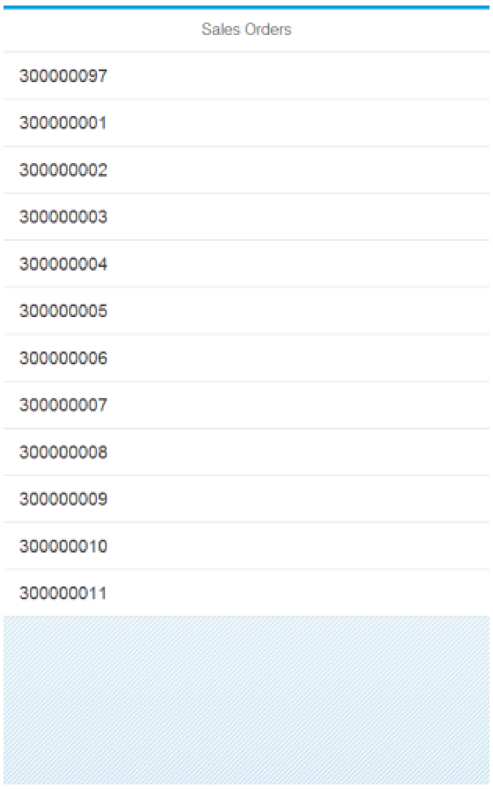
EXERCISE 2 – OBJECT CONTROLS

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

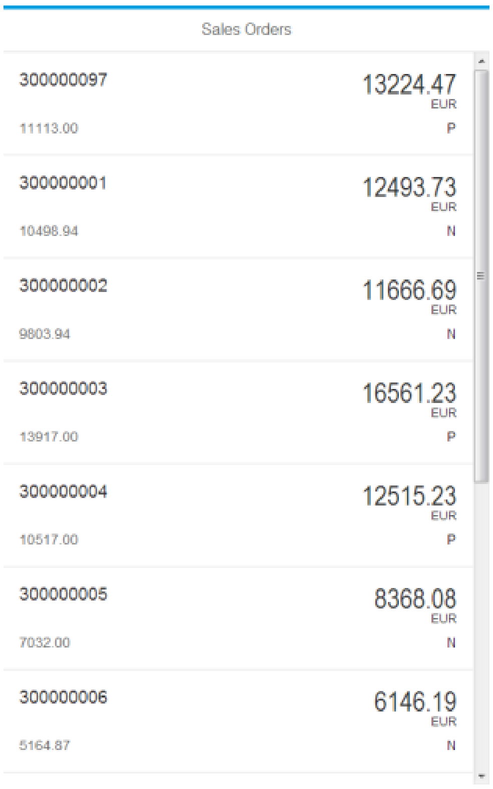
Before:



Sales Orders

300000097
300000001
300000002
300000003
300000004
300000005
300000006
300000007
300000008
300000009
300000010
300000011

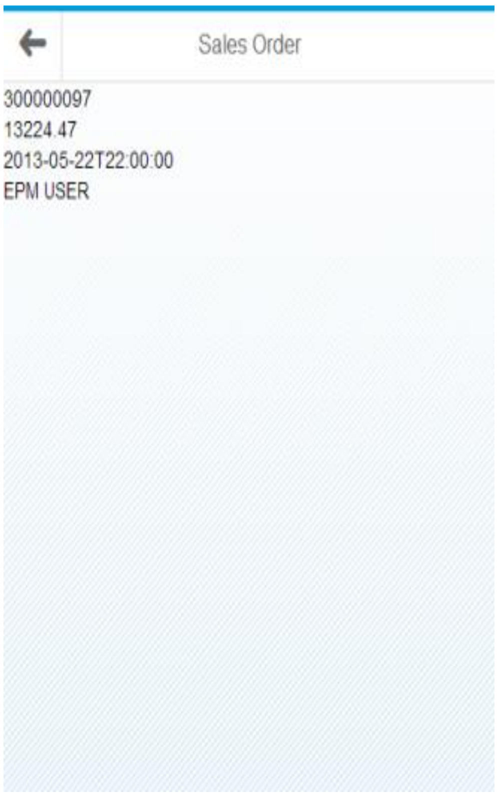
After:



Sales Orders

300000097	13224.47 EUR
11113.00	P
300000001	12493.73 EUR
10498.94	N
300000002	11666.69 EUR
9803.94	N
300000003	16561.23 EUR
13917.00	P
300000004	12515.23 EUR
10517.00	P
300000005	8368.08 EUR
7032.00	N
300000006	6146.19 EUR
5164.87	N

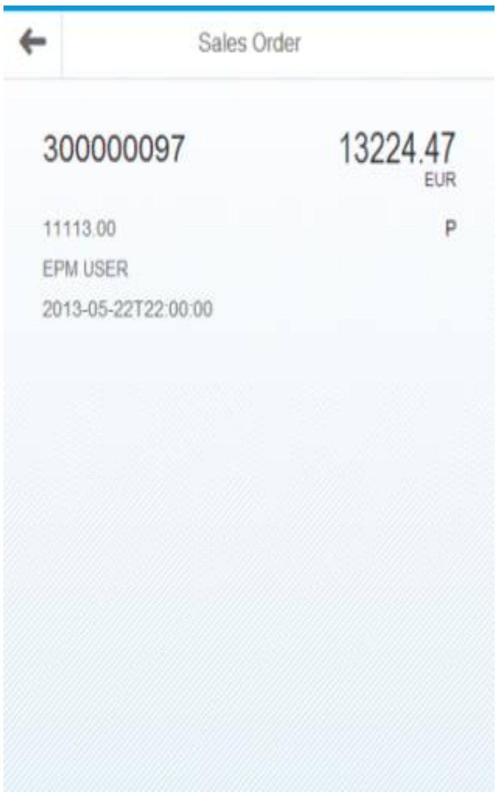
Before:



Sales Order

300000097
13224.47
2013-05-22T22:00:00
EPM USER

After:



Sales Order

300000097 13224.47 EUR
11113.00 P
EPM USER
2013-05-22T22:00:00

Description
In this exercise we will replace a couple of controls; one in the Master view and the other 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

view/Master.view.xml

- Replace the ***StandardListItem*** control with the more powerful ***ObjectListItem***
- Attributes and statuses are defined by own objects
- Save the modified **Master.view.xml** file with shortcut **CTRL+S**

```
<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>
```

view/Detail.view.xml

- 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).
- Save the modified **Detail.view.xml** file with shortcut **CTRL+S**

```
<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>
```

Further Reading:

- Working with lists: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/List.html
- ObjectHeader API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.ObjectHeader.html
- ObjectListItem API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.ObjectListItem.html

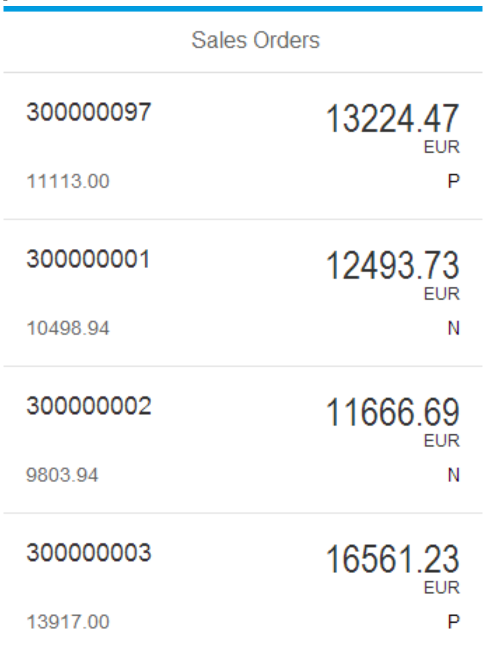
EXERCISE 3 – FORMATTER

Objective

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

Preview

Before:



Sales Orders

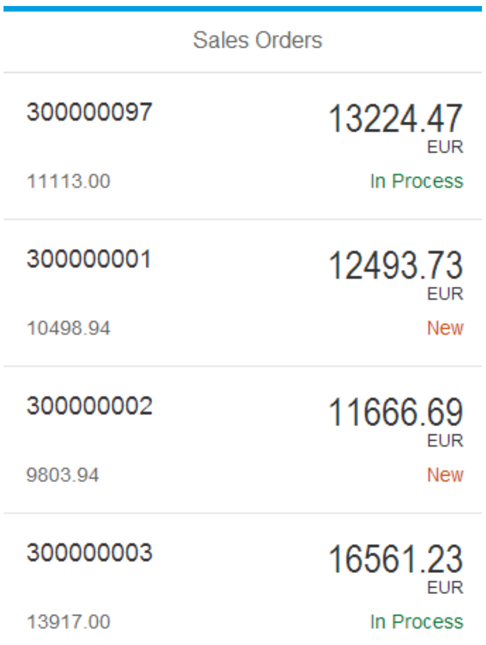
300000097 13224.47 EUR
11113.00 P

300000001 12493.73 EUR
10498.94 N

300000002 11666.69 EUR
9803.94 N

300000003 16561.23 EUR
13917.00 P

After:



Sales Orders

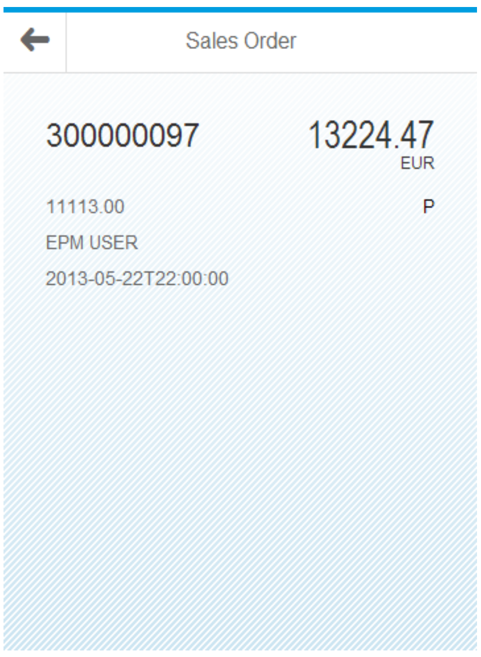
300000097 13224.47 EUR
11113.00 In Process

300000001 12493.73 EUR
10498.94 New

300000002 11666.69 EUR
9803.94 New

300000003 16561.23 EUR
13917.00 In Process

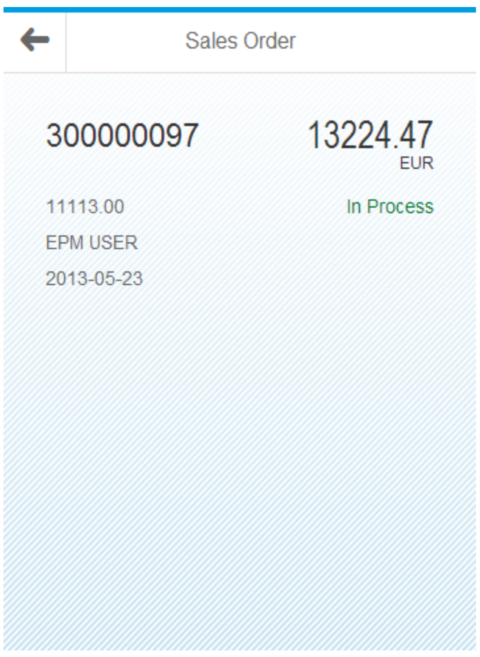
Before:



Sales Order

300000097 13224.47 EUR
11113.00 P
EPM USER
2013-05-22T22:00:00

After:



Sales Order

300000097 13224.47 EUR
11113.00 In Process
EPM USER
2013-05-23

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

- 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
```

util/Formatter.js (ADD NEW FOLDER util > ADD NEW FILE Formatter.js)

- This file contains functions to format dates, status text and status colors.

```
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

- Open the "**index.html**" with the HTML editor of eclipse by right clicking on the file and chosing "**Open With > HTML Editor**"
- 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"
                       data-sap-ui-resourceroots='{
                               "sap.ui.demo.myFiori": "./"
                       }' >
               </script>
               …
</html>
```

view/Detail.view.xml

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

```
<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>
```

view/Detail.controller.js

- 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.Detail", {

…
```

view/Master.view.xml

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

```
<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="{
                                                  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>
```

view/Master.controller.js

- 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", {

…
```

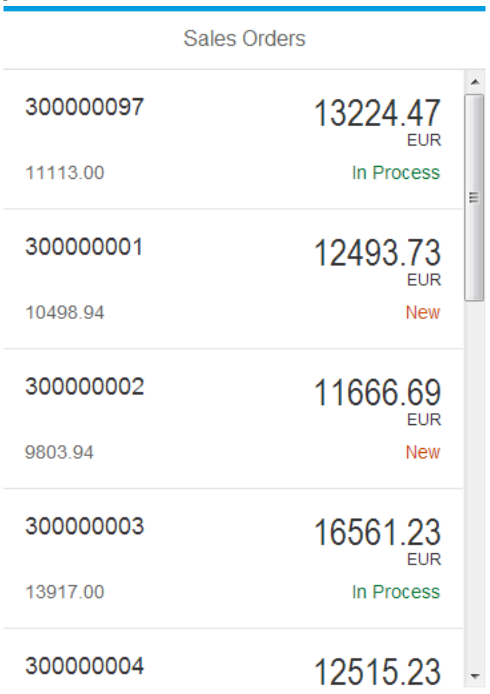
Further Reading:

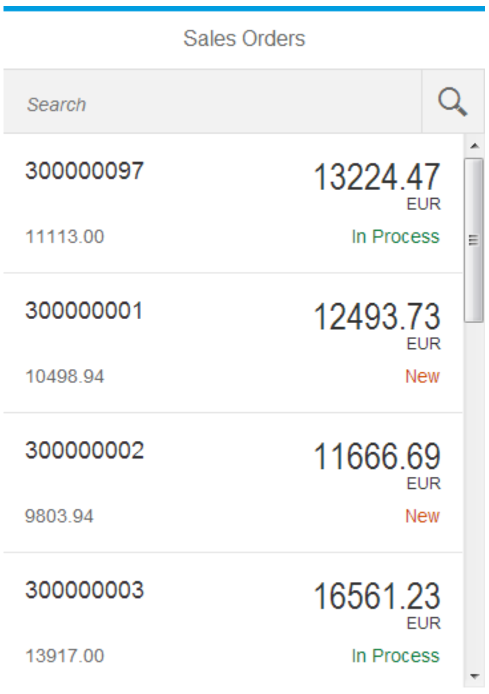
- Bootstrap Configuration Options: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Configuration.html#ListofConfigurationOptions
- Property Binding and Formatting: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/BindingProperties.html
- Modularization and Dependency Management (require/declare modules): https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/ModularizationConcept.html

EXERCISE 4 – SEARCH

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

Preview

Before:


After:


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
view/Master.view.xml

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

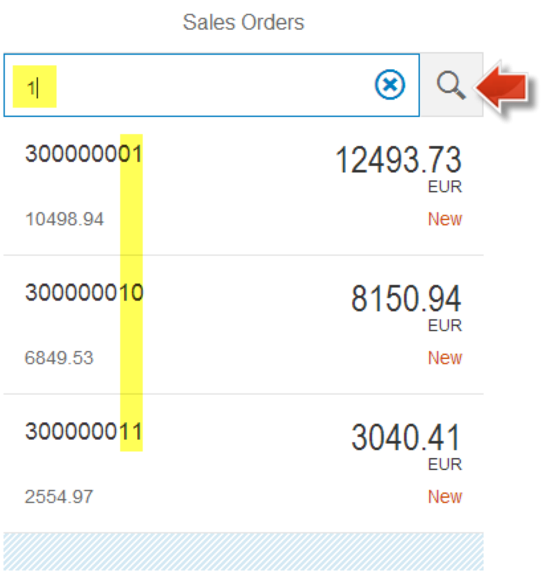
```
<core:View
       controllerName="sap.ui.demo.myFiori.view.Master"
       xmlns="sap.m"
       xmlns:core="sap.ui.core" >
       <Page
              title="{i18n>MasterTitle}" >
              <subHeader>
                     <Bar>
                            <contentLeft>
                       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

view/Master.controller.js

- Implement a new handler function on the view controller. Make sure to separate the function from the other handler function with a “**,**”
- Access the “**query**” as a parameter of the event object
- If the “**query**” is not empty add a ***FilterOperator*** to the array of filters.
- Access the list instance by calling “**byId**” on the view.
- Apply the filter array on the binding object of the list.

```
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);
       }
});
```

Google Chrome browser



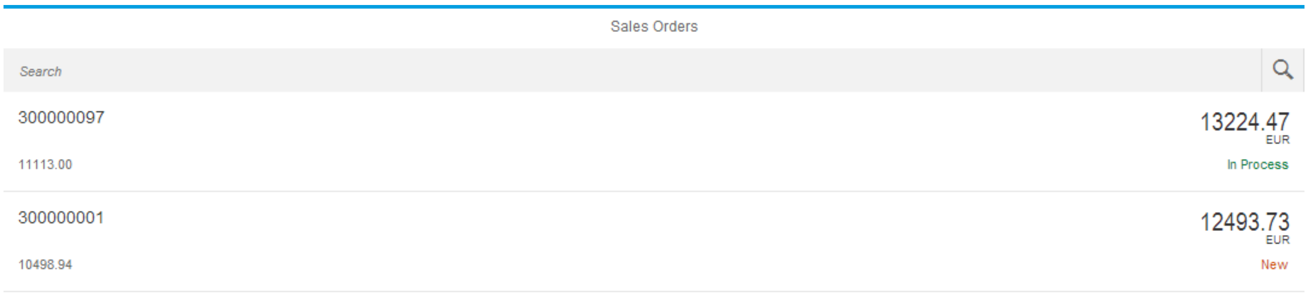
Further Reading:

- SearchField: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.SearchField.html
- Model Filter: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.ui.model.Filter.html

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:

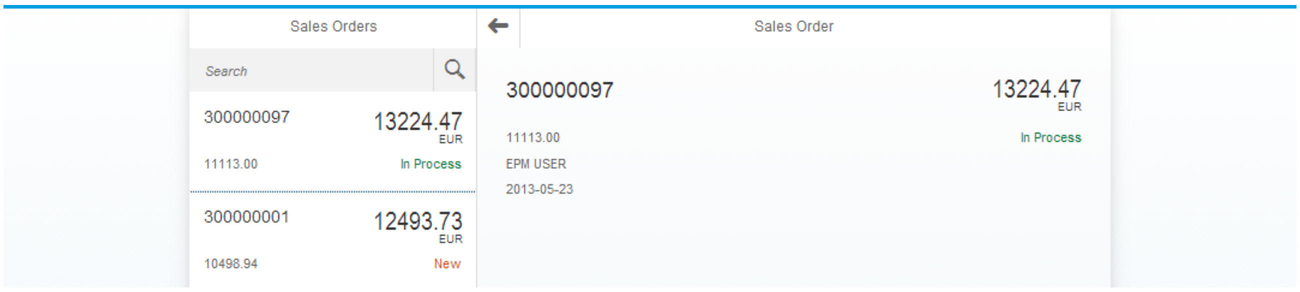


Sales Orders

Search

300000097 11113.00	13224.47 EUR In Process
300000001 10498.94	12493.73 EUR New

After:



Sales Orders

Search

300000097 11113.00	13224.47 EUR In Process
300000001 10498.94	12493.73 EUR New

Sales Order

300000097 13224.47 EUR

11113.00 In Process

EPM USER

2013-05-23

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 (yes, the same name) 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.

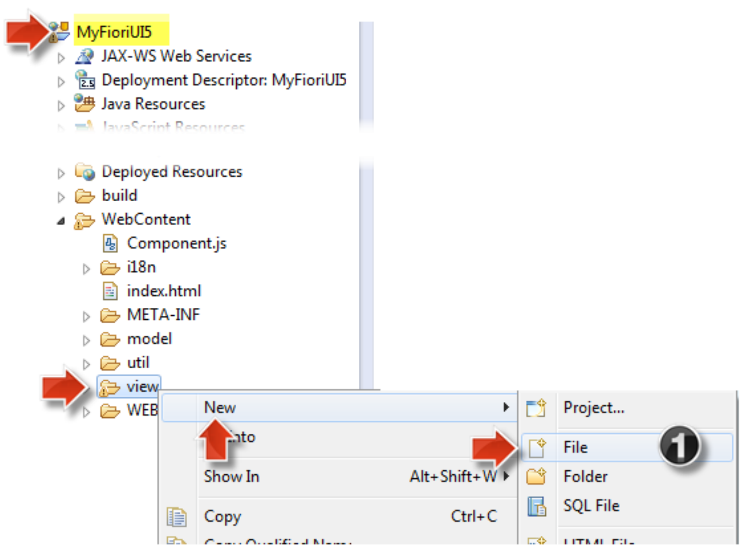
Changes

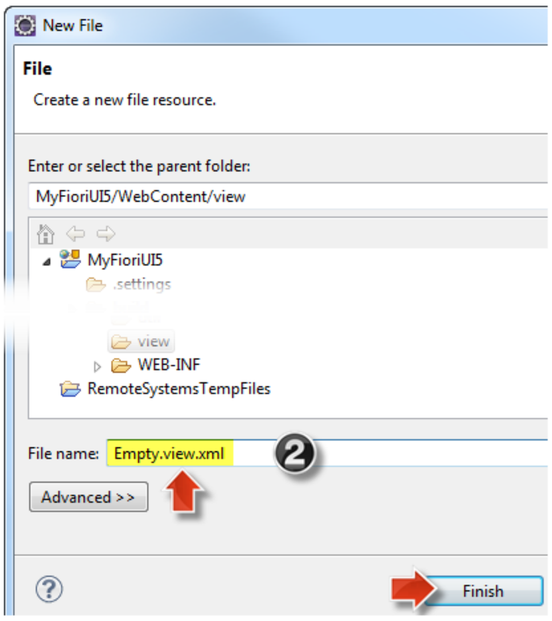
view/Empty.view.xml (create NEW XML view view/Empty.view.xml)

- 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*.





view/App.view.js

- 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;
	}
});
```

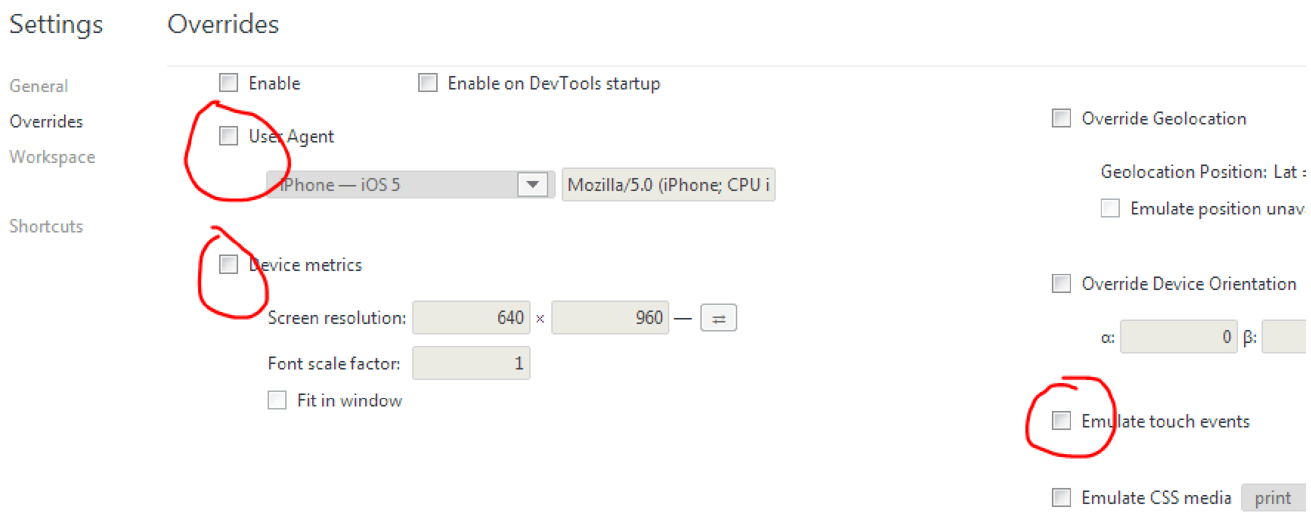
index.html

- Wrap the split app in a shell control using the title defined before.
- **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>
```

- In the Chrome Dev Tools, remove flags for User Agent and Device Metrics override. This will display the SplitApp control.


Settings
Overrides

General
Overrides
Workspace
Shortcuts

☐ Enable ☐ Enable on DevTools startup
☐ User Agent
iPhone — iOS 5 | Mozilla/5.0 (iPhone; CPU i
☐ Device metrics
Screen resolution: 640 × 960
Font scale factor: 1
☐ Fit in window
☐ Override Geolocation
Geolocation Position: Lat =
☐ Emulate position unav
☐ Override Device Orientation
α: 0 β:
☐ Emulate touch events
☐ Emulate CSS media print

Further Reading:

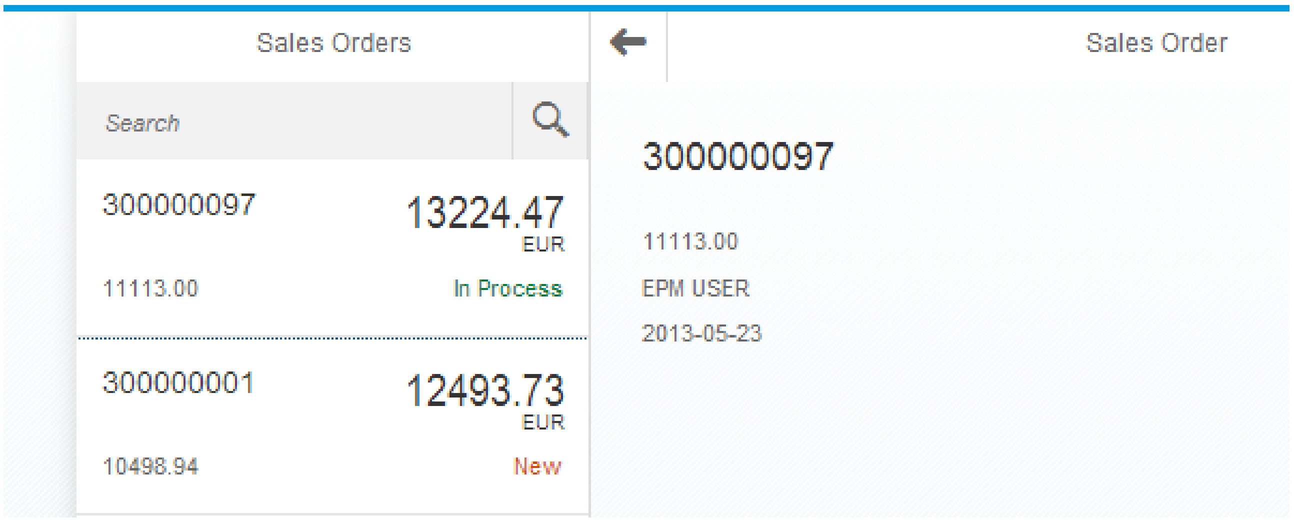
- SplitApp control: https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/SplitApp.html
- SplitApp API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.SplitApp.html
- Shell API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.Shell.html

EXERCISE 6 – ADDITIONAL DEVICE ADAPTATION

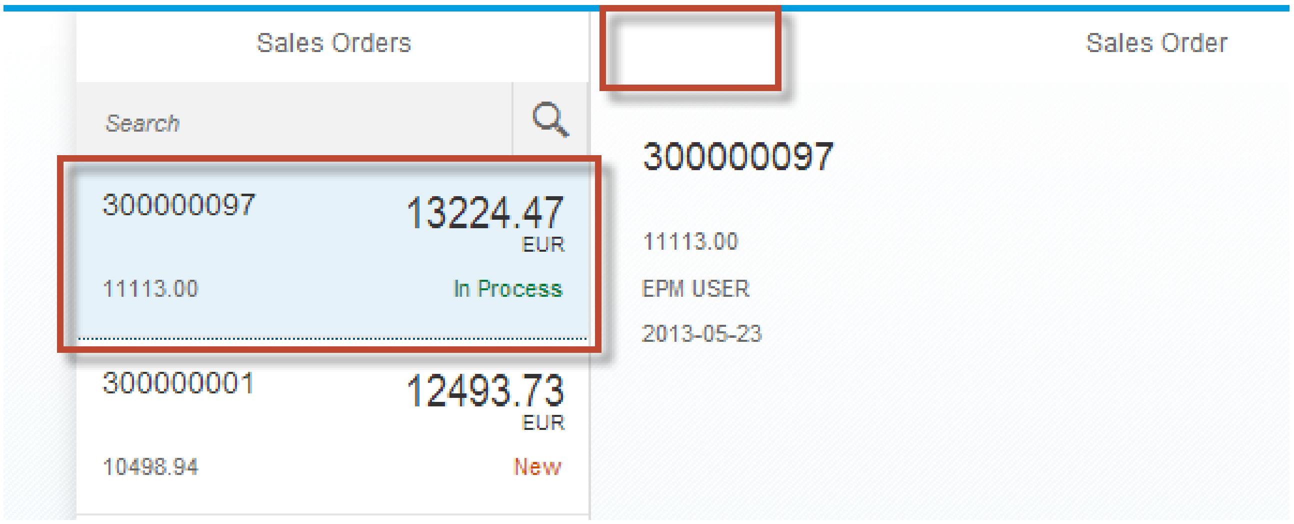
Objective
Adapt the controls to phone/tablet/desktop devices:

- 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 and item 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

- Set a global model named “**device**”
- Set **isPhone**, **listMode** and **listItemType** 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,
                        listMode : (jQuery.device.is.phone) ? "None" : "SingleSelectMaster",
                        listItemType : (jQuery.device.is.phone) ? "Active" : "Inactive"
                });
                deviceModel.setDefaultBindingMode("OneWay");
                oView.setModel(deviceModel, "device");

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

view/Detail.view.xml

- 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" >
```

view/Master.view.xml

- Bind the “**list**” mode and the “**list item type**” to the device model
- Add a select event to the list

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

view/Master.controller.js

- Implement the select event in the view’s controller

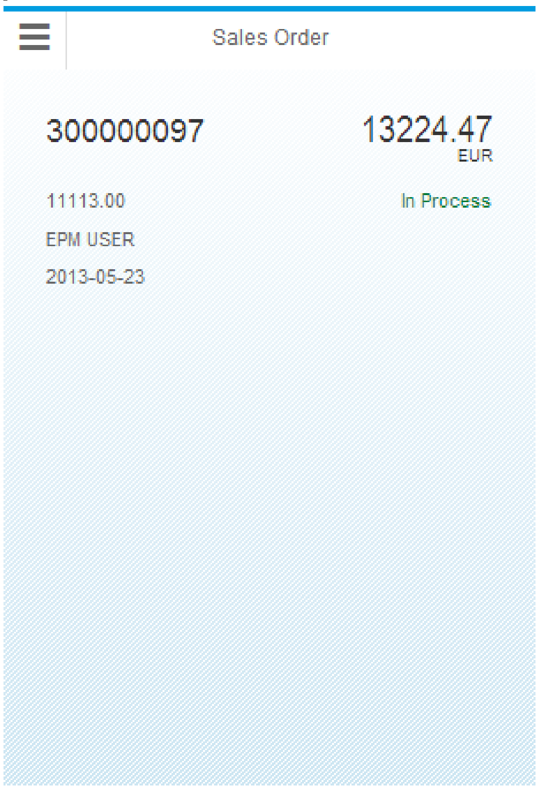
```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");

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) {
			…
		},

		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

Before: 

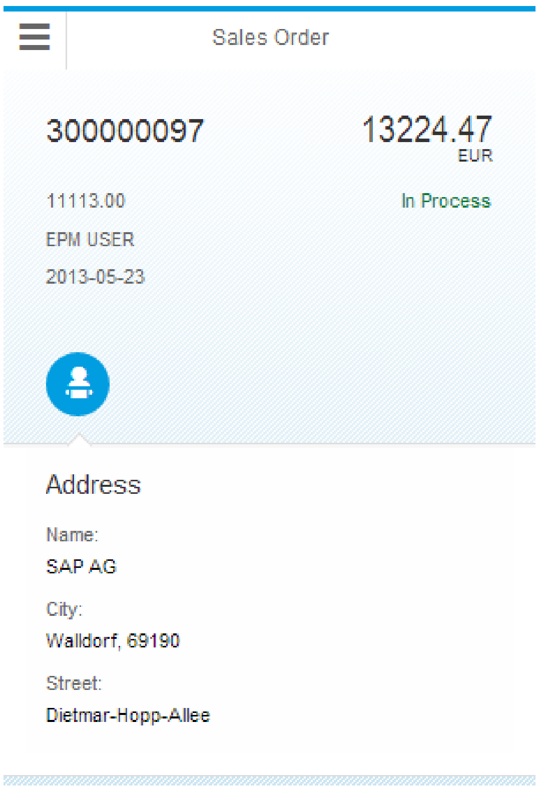
Sales Order

300000097 13224.47 EUR

11113.00 In Process

EPM USER

2013-05-23

After: 

Sales Order

300000097 13224.47 EUR

11113.00 In Process

EPM USER

2013-05-23

Address

Name:
SAP AG

City:
Walldorf, 69190

Street:
Dietmar-Hopp-Allee

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

- 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>

…
```

view/Detail.view.xml

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

```
<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}"
		showNavButton="{device>/isPhone}"
		navButtonPress="handleNavButtonPress" >
		<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>
```

- **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");
	}


});
```

Further Reading:

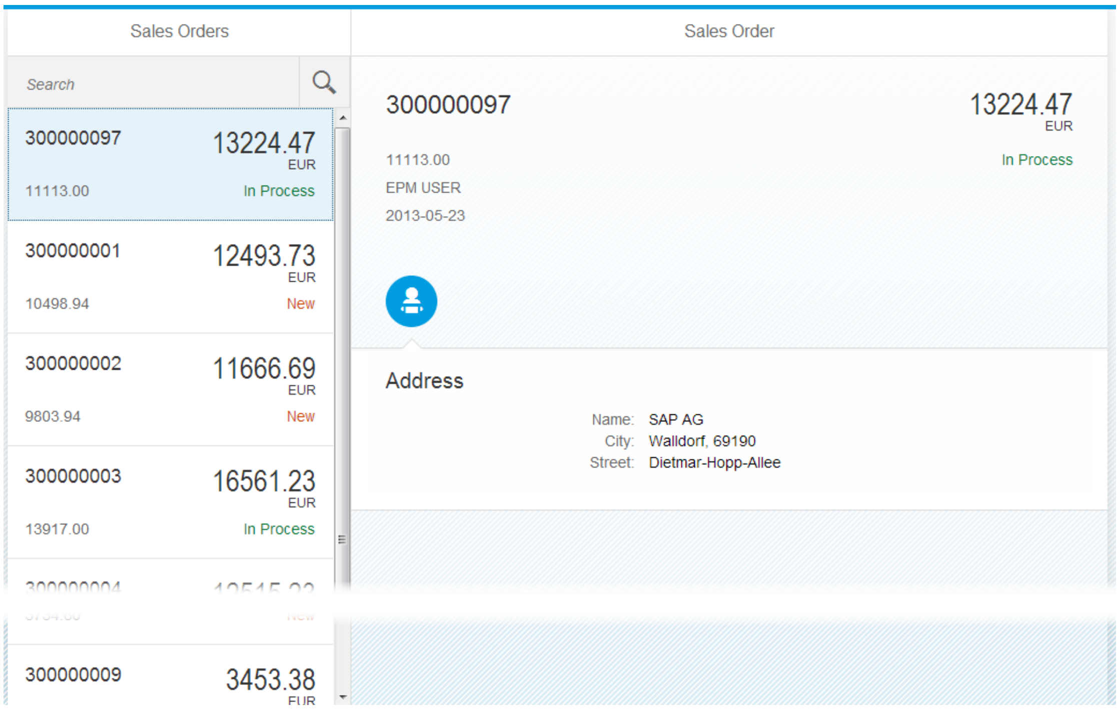
- Icon Tab Bar API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.IconTabBar.html
- Simple Form API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.ui.layout.form.SimpleForm.html

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.

Disclaimer: The server is not really called.

Preview
Before:



Sales Orders | Sales Order

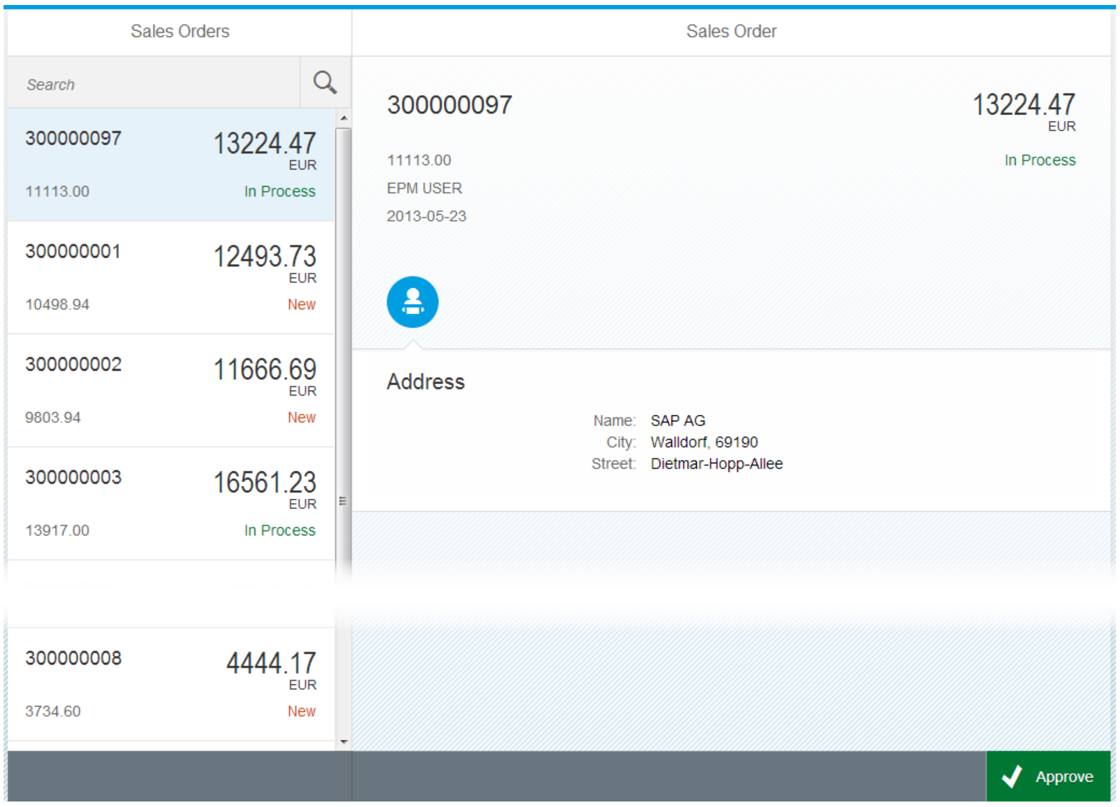
Search

300000097	13224.47 EUR	
11113.00		In Process
300000001	12493.73 EUR	
10498.94		New
300000002	11666.69 EUR	
9803.94		New
300000003	16561.23 EUR	
13917.00		In Process
300000009	3453.38 EUR	

300000097 13224.47 EUR
11113.00 In Process
EPM USER
2013-05-23

Address
Name: SAP AG
City: Walldorf, 69190
Street: Dietmar-Hopp-Allee

After:



Sales Orders | Sales Order

Search

300000097	13224.47 EUR	
11113.00		In Process
300000001	12493.73 EUR	
10498.94		New
300000002	11666.69 EUR	
9803.94		New
300000003	16561.23 EUR	
13917.00		In Process
300000008	4444.17 EUR	
3734.60		New

300000097 13224.47 EUR
11113.00 In Process
EPM USER
2013-05-23

Address
Name: SAP AG
City: Walldorf, 69190
Street: Dietmar-Hopp-Allee

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
- Add more texts for the approve button and dialog

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
```

view/Detail.view.xml
- 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>
```

view/Detail.controller.js
- First we need to register 2 more classes used to work with dialogs (*MessageBox*, MessageToast)
- On handling the approve event we first show a confirmation dialog (*MessageBox*)
- If the user confirms we only show a success message (*MessageToast*). **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")
			);
		}
});
```

view/Empty.view.xml

- 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>
```

view/Master.view.xml

- We now need footers in all pages for symmetry

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

Further Reading:

- Page API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.Page.html
- Modularization and Dependency Management (require/declare modules): https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/ModularizationConcept.html

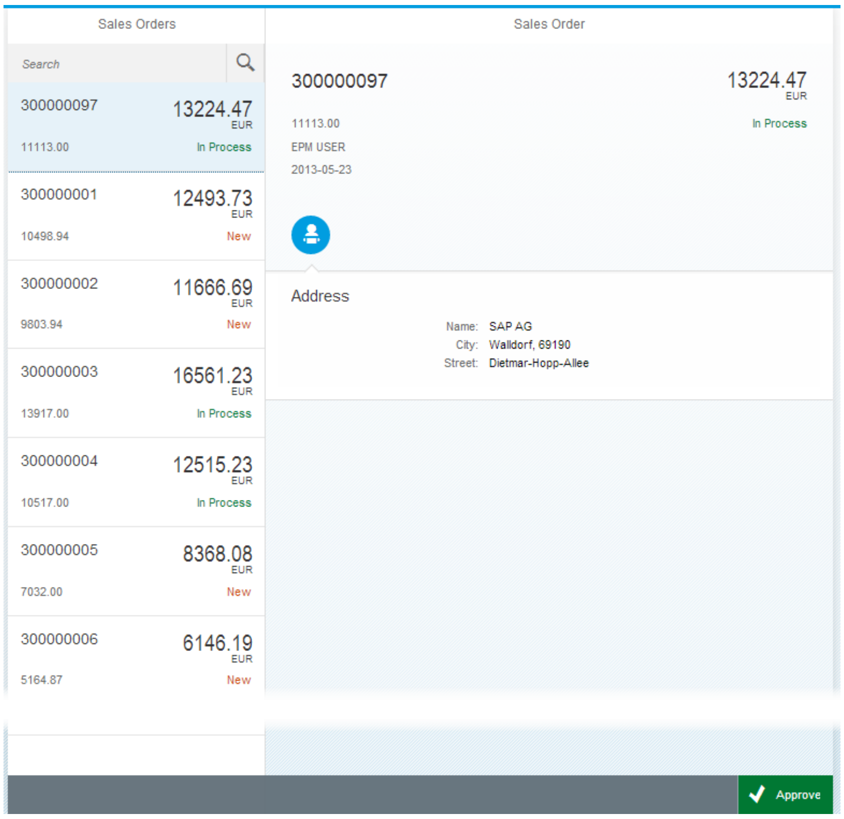
EXERCISE 9 – LINE ITEM

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:



Sales Orders

Search

300000097	13224.47 EUR	
11113.00	In Process	
300000001	12493.73 EUR	
10498.94	New	
300000002	11666.69 EUR	
9803.94	New	
300000003	16561.23 EUR	
13917.00	In Process	
300000004	12515.23 EUR	
10517.00	In Process	
300000005	8368.08 EUR	
7032.00	New	
300000006	6146.19 EUR	
5164.87	New	

Sales Order

300000097 13224.47 EUR

11113.00 In Process

EPM USER

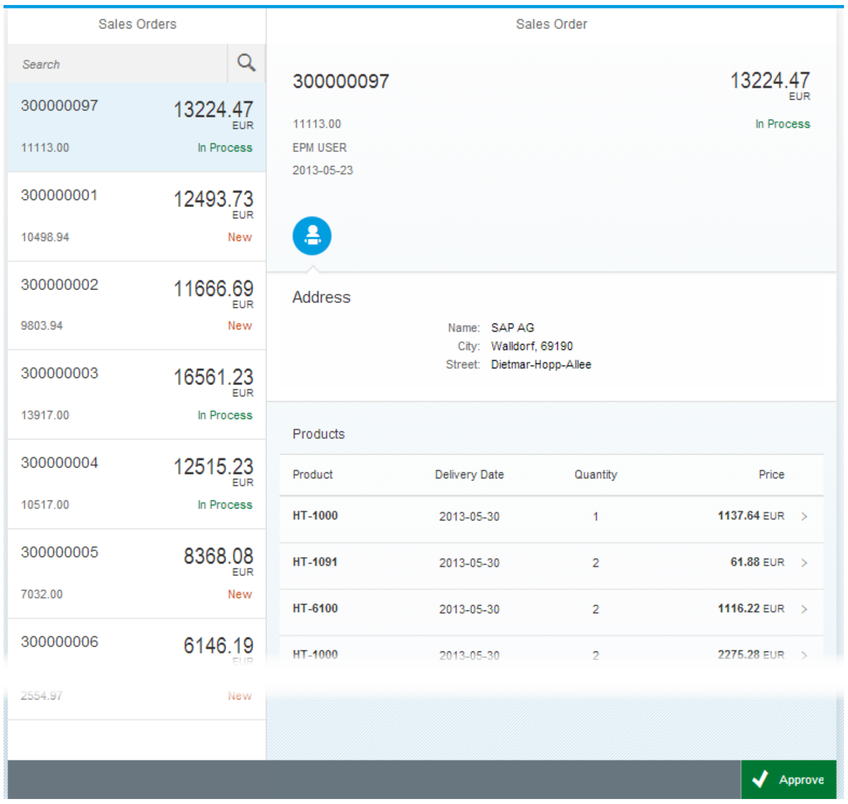
2013-05-23

Address

Name: SAP AG
City: Walldorf, 69190
Street: Dietmar-Hopp-Allee

Approve

After:



Sales Orders

Search

300000097	13224.47 EUR	
11113.00	In Process	
300000001	12493.73 EUR	
10498.94	New	
300000002	11666.69 EUR	
9803.94	New	
300000003	16561.23 EUR	
13917.00	In Process	
300000004	12515.23 EUR	
10517.00	In Process	
300000005	8368.08 EUR	
7032.00	New	
300000006	6146.19 EUR	
2554.97	New	

Sales Order

300000097 13224.47 EUR

11113.00 In Process

EPM USER

2013-05-23

Address

Name: SAP AG
City: Walldorf, 69190
Street: Dietmar-Hopp-Allee

Products

Product	Delivery Date	Quantity	Price
HT-1000	2013-05-30	1	1137.64 EUR
HT-1091	2013-05-30	2	61.88 EUR
HT-6100	2013-05-30	2	1116.22 EUR
HT-1000	2013-05-30	2	2275.28 EUR

Approve

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 *ColumnListItem* 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

- Add more message texts

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
LineItemTableHeader=Products
LineItemTitle=Product
```

util/Formatter.js

- 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";
              }
       }
};
```

view/Detail.view.xml

- We set a CSS class on the page control that will set proper margins on the table control in this page.
- 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>
```

view/Detail.controller.js

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

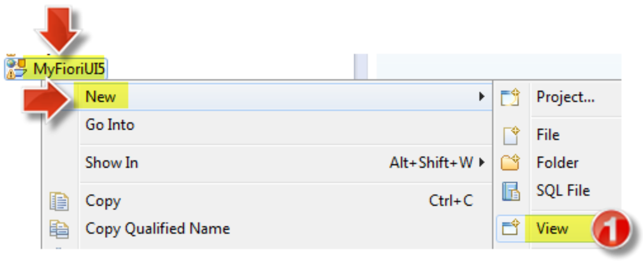
```
…

        handleApprove : function (evt) {
…
        },

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

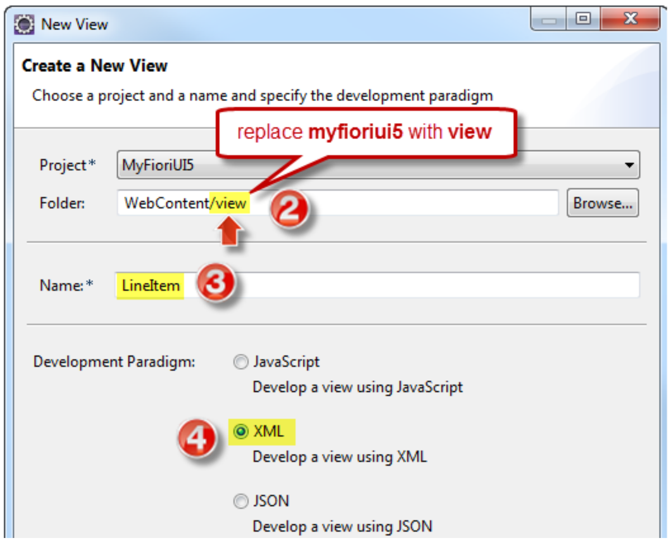
view/LineItem.view.xml (ADD NEW XML view LineItem, REPLACE all initial code)

- 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.




New View

Create a New View

Choose a project and a name and specify the development paradigm

replace **myfioriui5** with **view**

Project*: MyFioriUI5

Folder: WebContent/view

Name:* LineItem

Development Paradigm:
- JavaScript — Develop a view using JavaScript
- [✓ XML] — Develop a view using XML
- JSON — Develop a view using JSON

```
<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>
```

view/LineItem.controller.js (gets NEWLY added with view/LineItem.view.xml, REPLACE all initial code)

- We only need to handle the back navigation to the **Detail** page

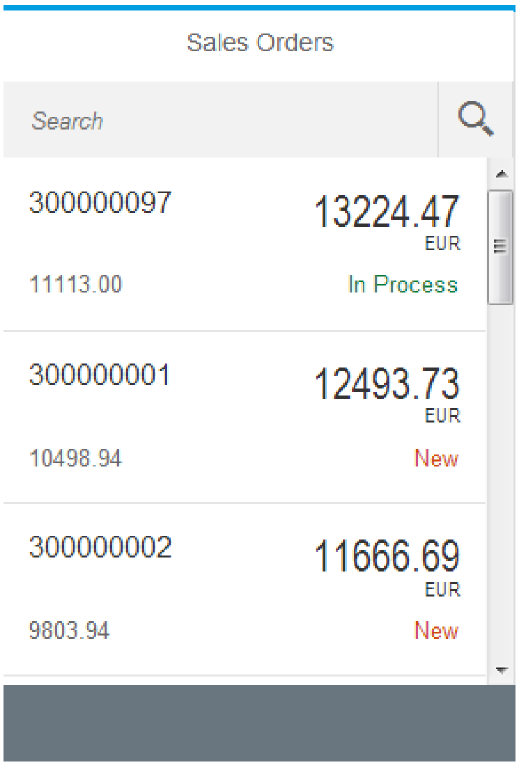
```
sap.ui.controller("sap.ui.demo.myFiori.view.LineItem", {

        handleNavBack : function (evt) {
                this.nav.back("Detail");
        }
});
```

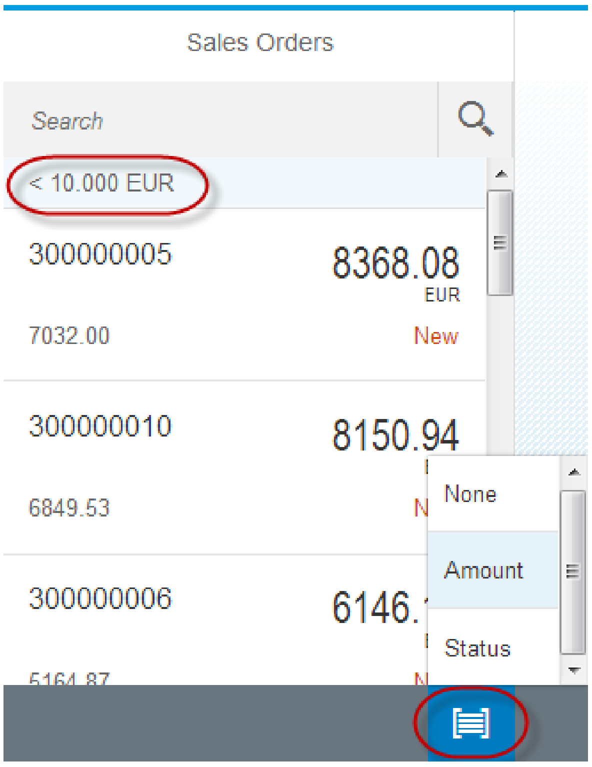
EXERCISE 10 – GROUPING

Objective

Add a "**Select**" to the master list that lets the user select a grouping. Handle the user selection and apply the grouping to the data binding.

Preview

Before: 

After: 

Description

We're almost there. In this last exercise we're going to add grouping features that can be applied when aggregation bindings are sorted. In this case the binding is the one between the sales orders in the data model and the items aggregation in the *List* control in the *Master* view.

We'll create a new file in the '*util*' folder, containing two custom grouping functions. We'll add a *Select* control to the *Bar* in the Page footer in the *Master* view, and in the corresponding controller, we will handle the button press with a function '*handleGroup*' that updates the data binding of the list.

Changes

i18n/messageBundle.properties

- Add more message texts

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
LineItemTableHeader=Products
LineItemTitle=Product
MasterGroupNone=None
MasterGroupStatus=Status
MasterGroupAmount=Amount
```

util/Grouper.js (ADD NEW JavaScript file)

- This new file contains two functions that implement the logic to group sales orders by
    - **Status** (simple string comparison)
    - **Amount** (a little bit more sophisticated price checks)

```
jQuery.sap.declare("sap.ui.demo.myFiori.util.Grouper");

sap.ui.demo.myFiori.util.Grouper = {

	bundle : null, // somebody has to set this

	LifecycleStatus : function (oContext) {
		var status = oContext.getProperty("LifecycleStatus");
		var text = sap.ui.demo.myFiori.util.Grouper.bundle.getText("StatusText" + status,
"?");
		return {
			key: status,
			text: text
		};
	},

	GrossAmount : function (oContext) {
		var price = oContext.getProperty("GrossAmount");
		var currency = oContext.getProperty("CurrencyCode");
		var key = null,
			text = null;
		if (price <= 5000) {
			key = "LE10";
			text = "< 5000 " + currency;
		} else if (price > 5000 && price <= 10000) {
			key = "LE100";
			text = "< 10.000  " + currency;
		} else if (price > 10000) {
			key = "GT100";
			text = "> 10.000 " + currency;
		}
		return {
			key: key,
			text: text
		};
	}
};
```

view/Master.view.xml

- Add a select control to the footer of the master page to choose a criteria for grouping

```
<core:View
	controllerName="sap.ui.demo.myFiori.view.Master"
	xmlns="sap.m"
	xmlns:core="sap.ui.core" >
	…
		<footer>
			<Bar>
				<contentRight>
					<Select
						id="groupSelect"
						change="handleGroup"
						icon="sap-icon://group-2"
						type="IconOnly"
						selectedKey="None"
						autoAdjustWidth="true" >
						<core:Item
							key="None"
							text="{i18n>MasterGroupNone}"/>
						<core:Item
							key="GrossAmount"
```

```
                                                                        text="{i18n>MasterGroupAmount}"/>
                                                              <core:Item
                                                                        key="LifecycleStatus"
                                                                        text="{i18n>MasterGroupStatus}"/>
                                                    </Select>
                                          </contentRight>
                                </Bar>
                      </footer>
            </Page>
</core:View>
```

view/Master.controller.js

- Require the new “**Grouper.js**” file
- Implement the “**handleGroup**” function
    - Compute the sorter object that will perform the grouping
    - Apply the grouping to the data binding

```
jQuery.sap.require("sap.ui.demo.myFiori.util.Formatter");
jQuery.sap.require("sap.ui.demo.myFiori.util.Grouper");

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);
        },

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

        handleGroup : function (evt) {

                // compute sorters
                var sorters = [];
                var item = evt.getParameter("selectedItem");
                var key = (item) ? item.getKey() : null;
                if ("GrossAmount" === key || "LifecycleStatus" === key) {
                        sap.ui.demo.myFiori.util.Grouper.bundle =
this.getView().getModel("i18n").getResourceBundle();
                        var grouper = sap.ui.demo.myFiori.util.Grouper[key];
                        sorters.push(new sap.ui.model.Sorter(key, true, grouper));
                }

                // update binding
                var list = this.getView().byId("list");
                var oBinding = list.getBinding("items");
                oBinding.sort(sorters);
        }
});
```

Further Reading:

- Select API: https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.Select.html www.sap.com