

WEEK 3 UNIT 3

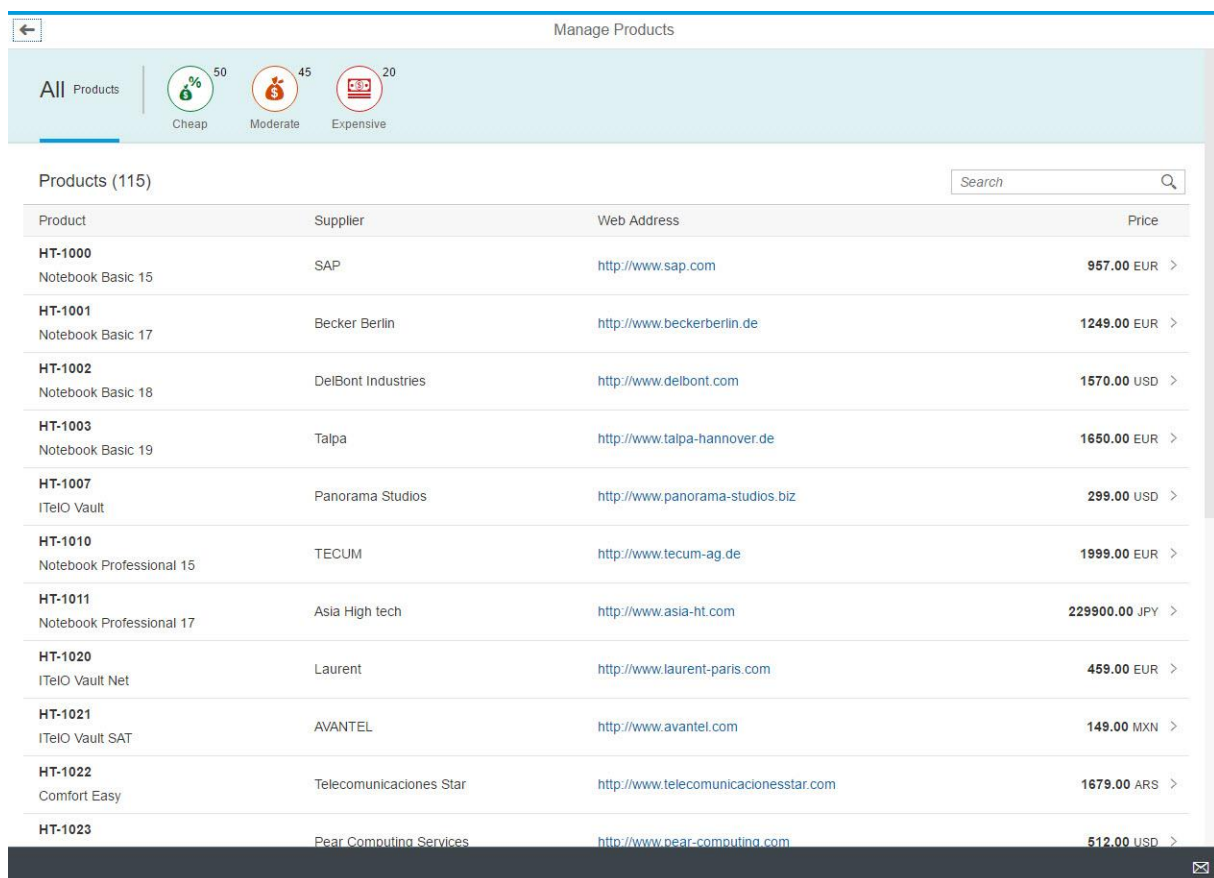
ADDING QUICK FILTERS

Please perform the exercises below in your app project as shown in the video.

Table of Contents

1	Update the view to show a filter bar	2
2	Add the filters to the controller	5
3	(Optional) Add Counts for the Filters	7

Preview



The screenshot shows the 'Manage Products' app interface. At the top, there's a navigation bar with a back arrow and the title 'Manage Products'. Below the navigation bar, there's a filter bar with three filters: 'All Products' (selected), 'Cheap' (50 items), 'Moderate' (45 items), and 'Expensive' (20 items). Below the filter bar, there's a search bar and a table of products. The table has four columns: Product, Supplier, Web Address, and Price. The table lists 11 products, each with a product ID, name, supplier, web address, and price in various currencies. A dark blue bar at the bottom contains an email icon.

Product	Supplier	Web Address	Price
HT-1000 Notebook Basic 15	SAP	http://www.sap.com	957.00 EUR >
HT-1001 Notebook Basic 17	Becker Berlin	http://www.beckerberlin.de	1249.00 EUR >
HT-1002 Notebook Basic 18	DelBont Industries	http://www.delbont.com	1570.00 USD >
HT-1003 Notebook Basic 19	Talpa	http://www.talpa-hannover.de	1650.00 EUR >
HT-1007 ITeIO Vault	Panorama Studios	http://www.panorama-studios.biz	299.00 USD >
HT-1010 Notebook Professional 15	TECUM	http://www.tecum-ag.de	1999.00 EUR >
HT-1011 Notebook Professional 17	Asia High tech	http://www.asia-ht.com	229900.00 JPY >
HT-1020 ITeIO Vault Net	Laurent	http://www.laurent-paris.com	459.00 EUR >
HT-1021 ITeIO Vault SAT	AVANTEL	http://www.avantel.com	149.00 MXN >
HT-1022 Comfort Easy	Telecomunicaciones Star	http://www.telecomunicacionesstar.com	1679.00 ARS >
HT-1023	Pear Computing Services	http://www.pear-computing.com	512.00 USD >

Figure 1 - Preview of the app after doing this unit's exercises

1 UPDATE THE VIEW TO SHOW A FILTER BAR

For easily finding products in a certain price range, we will add a quick filter for the worklist table. Users can press the filter tabs to display the products according to whether they are cheap, moderate, or expensive. The table will update accordingly and show only the products matching the criteria.

webapp/view/Worklist.view.xml

```

...
<semantic:FullscreenPage
  id="page"
  navButtonPress="onNavBack"
  showNavButton="true"
  title="{i18n>worklistViewTitle}">
  <semantic:content>
    <IconTabBar
      id="iconTabBar"
      select="onQuickFilter"
      expandable="false"
      applyContentPadding="false">
      <items>
        <IconTabFilter
          key="all"
          showAll="true"
          count="{i18n>worklistFilterProductsAllCount}"
          text="{i18n>worklistFilterProductsAll}"/>
        <IconTabSeparator/>
        <IconTabFilter
          key="cheap"
          icon="sap-icon://waiver"
          iconColor="Positive"
          text="{i18n>worklistFilterCheap}"/>
        <IconTabFilter
          key="moderate"
          icon="sap-icon://loan"
          iconColor="Critical"
          text="{i18n>worklistFilterModerate}"/>
        <IconTabFilter
          key="expensive"
          icon="sap-icon://money-bills"
          iconColor="Negative"
          text="{i18n>worklistFilterExpensive}"/>
      </items>
      <content>
        <!-- put the existing table here -->
      </content>
    </IconTabBar>
  </semantic:content>

```

We first update the view and add the new UI for the quick filter to the content aggregation of the `sap.m.SemanticPage` control just before the `table`. In the content aggregation of the `IconTabBar` we put the existing table. The `IconTabBar` contains a `sap.m.IconTabFilter` for each of the following filter options:

- **All products**
This tab will simply show the overall number of products that has been returned by the data service. This tab will show a larger number only (optional) and no icon by using the `showAll` property.
- **Cheap**
This tab will show all the products that are below a certain price range. We choose a matching icon from the icon font and set the icon color to the semantic `Positive` state so that it will appear in green.
- **Moderate**
This tab will show products that have a moderate price with the warning state which will make the icon appear in orange.
- **Expensive**
This tab will show products that have a high price. The semantic `Negative` state will let the icon appear in red.

As usual the UI texts for the tabs are linked to the resource bundle file and will be added later.

webapp/view/Worklist.view.xml

```
...  
<Table  
  id="table"  
  class="sapUiResponsiveMargin"  
  ...>
```

Do not forget to remove the CSS class `sapUiResponsiveMargin` from the table to remove the spacing between the `IconTabBar` and the table and make the UI look nicer.

Note:

Each `IconTabFilter` element has a `key` property that is used to identify the tab that was pressed in the event handler `onQuickFilter` that is registered on the `IconTabBar` control directly. The event handler implementation does the actual filtering on the table and is defined in the controller.

webapp/i18n/i18n.properties

```
...  
  
#XTIT: The title of the products quick filter  
worklistFilterProductsAll=Products  
  
#XTIT: The count for the all products quick filter  
worklistFilterProductsAllCount=All  
  
#XTIT: The title of the cheap products filter  
worklistFilterCheap=Cheap  
  
#XTIT: The title of the moderate products filter  
worklistFilterModerate=Moderate  
  
#XTIT: The title of the expensive products filter  
worklistFilterExpensive=Expensive  
  
#~~~ Object View ~~~~~  
...
```

Add the new translation texts to your resource bundle file.

You can now run the app to check the UI, but the filters will have no effect yet

2 ADD THE FILTERS TO THE CONTROLLER

webapp/controller/Worklist.controller.js

```

return
BaseController.extend("opensap.manageproducts.controller.Worklist", {

    formatter: formatter,

    _mFilters: {
        cheap: [new sap.ui.model.Filter("Price", "LT", 100)],
        moderate: [new sap.ui.model.Filter("Price", "BT", 100, 1000)],
        expensive: [new sap.ui.model.Filter("Price", "GT", 1000)]
    },

    /* ===== */
    /* lifecycle methods */
    /* ===== */

    /**
     * Called when the worklist controller is instantiated.
     * @public
     */
    onInit : function () {

```

We create an object `_mFilters` that contains a filter for each tab. We will use the filters for filtering the table. The properties in `_mFilters` correlate to the keys of the `IconTabFilter` controls we defined above in the `Worklist.view.xml` file. This way we can easily access a filter for a given tab based on the key of the corresponding tab.

Creating a simple filter requires a binding path as first parameter of the filter constructor (e.g. "Price"), a filter operator (e.g. "GT") as second argument, and a value to compare (e.g. 1000) as the third argument. We create such filters for all three tabs with different filter operators as described in the view part above.

Note:

You may notice that we have different currencies in the product table. If we create a filter solely on the price field then it might not reflect the actual value of the product in a common currency like EUR. However, we want to keep this example free of any currency conversions and therefore live with this side-effect.

webapp/controller/Worklist.controller.js

```

/* ===== */
/* event handlers */
/* ===== */

/**
 * Event handler when a filter tab gets pressed
 * @param {sap.ui.base.Event} oEvent the filter tab event
 * @public
 */
onQuickFilter: function(oEvent) {
    var sKey = oEvent.getParameter("key"),
        oFilter = this._mFilters[sKey],
        oTable = this.byId("table"),
        oBinding = oTable.getBinding("items");
    if (oFilter) {
        oBinding.filter(oFilter);
    } else {
        oBinding.filter([]);
    }
},

```

Next, we implement the handler for the `select` event of the `IconTabBar`. In this event handler we get a reference to the binding for the `items` aggregation of our `table` and store it in the variable `oBinding`. Then we read the parameter `key` from the `event` object to find out which tab has been selected.

This `key` is used to get the correct filter for the selected tab. Next, we simply call the `filter` method on `oBinding` and pass the correct filter of the selected tab. If no filter matches the `key` the user has selected, the first tab should be displayed and all filters should be cleared. To achieve this we pass an empty array into the filtering.

The filters are always applied as an array on the binding level, so you don't need to take care of managing the data, the data binding features of SAPUI5 will automatically do it.

Now run the app again and click on the filter icons on top of the table. The products should be filtered according to the selection in the filter bar and the count should match the number of items displayed.

Related Information

[API Reference: sap.ui.model.ListBinding.filter](#)

3 (OPTIONAL) ADD COUNTS FOR THE FILTERS

If you like you can enrich this example by adding the count for each filter. We can read the amount of products matching the criteria by querying the service with the filter:

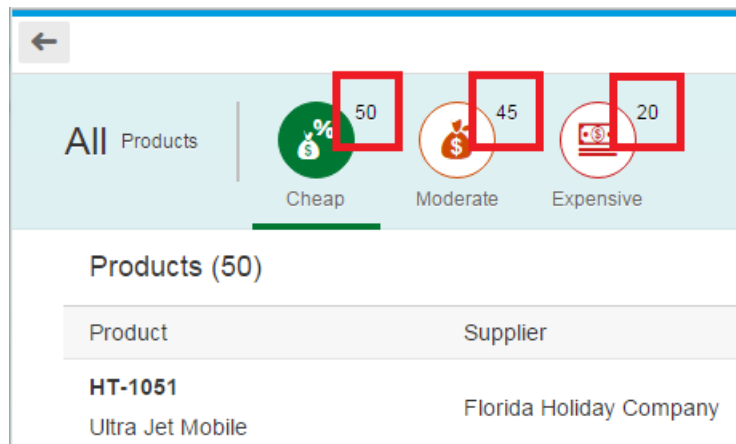


Figure 2 – xxx

webapp/view/Worklist.view.xml

```
...
<IconTabBar
  id="iconTabBar"
  select="onQuickFilter"
  expandable="false"
  applyContentPadding="false">
  <items>
    <IconTabFilter
      key="all"
      showAll="true"
      count="{i18n>WorklistFilterProductsAllCount}"
      text="{i18n>WorklistFilterProductsAll}"/>
    <IconTabSeparator/>
    <IconTabFilter
      key="cheap"
      icon="sap-icon://waiver"
      iconColor="Positive"
      count="{worklistView>/cheap}"
      text="{i18n>WorklistFilterCheap}"/>
    <IconTabFilter
      key="moderate"
      icon="sap-icon://loan"
      iconColor="Critical"
      count="{worklistView>/moderate}"
      text="{i18n>Moderate}"/>
    <IconTabFilter
      key="expensive"
      icon="sap-icon://money-bills"
      iconColor="Negative"
      count="{worklistView>/expensive}"
      text="{i18n>WorklistFilterExpensive}"/>
  </items>
</IconTabBar>
```

The `count` property of each tab is now bound to a local view model and the number will be updated in the controller later in this step.

webapp/controller/Worklist.controller.js

```
...
/**
 * Called when the worklist controller is instantiated.
 * @public
 */
onInit : function () {

    ...

    // Model used to manipulate control states
    oViewModel = new JSONModel({
        ...
        tableBusyDelay : 0,
        cheap: 0,
        moderate: 0,
        expensive: 0
    });
    this.setModel(oViewModel, "worklistView");
}
...
```

We add properties for the counters into the local view model of the worklist controller. We initialize the three values with 0 each. The counters in the view are bound to these properties.

webapp/controller/Worklist.controller.js

```
onUpdateFinished : function (oEvent) {
    // update the worklist's object counter after the table update
    var sTitle,
        oTable = oEvent.getSource(),
        oModel = this.getModel(),
        oViewModel = this.getModel("worklistView"),
        iTotItems = oEvent.getParameter("total");
    // only update the counter if the length is final and
    // the table is not empty
    if (iTotItems && oTable.getBinding("items").isLengthFinal()) {
        sTitle = this.getResourceBundle().getText("...", [iTotItems]);
        // iterate the filters and request the count from the server
        jQuery.each(this._mFilters, function (sFilterKey, oFilter) {
            oModel.read("/ProductSet/$count", {
                filters: oFilter,
                success: function (oData) {
                    var sPath = "/" + sFilterKey;
                    oViewModel.setProperty(sPath, oData);
                }
            });
        });
    }
};
```



```
    } else {  
        sTitle = this.getResourceBundle().getText("...");  
    }  
    this.getModel("worklistView").setProperty("/...", sTitle);  
},  
...  

```

In the `onUpdateFinished` function which is called when data is fetched from the model, we get the count of all products by triggering a read operation on the model with the appropriate filter. To loop over the filter objects we use [jQuery's each function](#). For each property in `mFilters` a callback function is called that gets the key and the filter object as parameter. The filter is a helper object of SAPUI5 that defines the condition for each tab on the data binding level. We already created the filters in the `onInit` function.

Note:

The `v2.ODataModel` will automatically bundle these `read` requests to one batch request to the server (if batch mode is enabled).

In the `success` handler of each `read` operation we update the corresponding property in the view model with the real count of the matching items that were returned by the service.

Now run the app again and check that the numbers next to the filters are shown.

Related Information

[API Reference: sap.ui.model.ListBinding.filter](#)

Coding Samples

Any software coding or code lines/strings ("Code") provided in this documentation are only examples and are not intended for use in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules for certain SAP coding. SAP does not warrant the correctness or completeness of the Code provided herein and SAP shall not be liable for errors or damages caused by use of the Code, except where such damages were caused by SAP with intent or with gross negligence.