openSAP Evolved Web Apps with SAPUI5

Week 3 Unit 4: Spicing up Your Scenario with Feature-Rich Controls

Exercises

PUBLIC

TABLE OF CONTENTS

SPICING UP YOUR SCENARIO WITH FEATURE-RICH CONTROLS

Summary

In this unit, you will enhance the existing controls by adding

- a sticky header to the table
- a floating footer to the semantic page
- a dynamic header to the object page layout

You will also add some new eye-catching controls

- `InfoLabel`, showing the delivery status for each order
- `StatusIndicator`, showing the delivery date of each order

Preview

Enrich the app by enabling some new fancy features that the framework offers. Examine some cool features that you get for free in UI5, which don't exist in other UI frameworks. The huge control set of UI5 is one of the key benefits. UI5 offers plenty of feature-rich UI controls ready to be used.

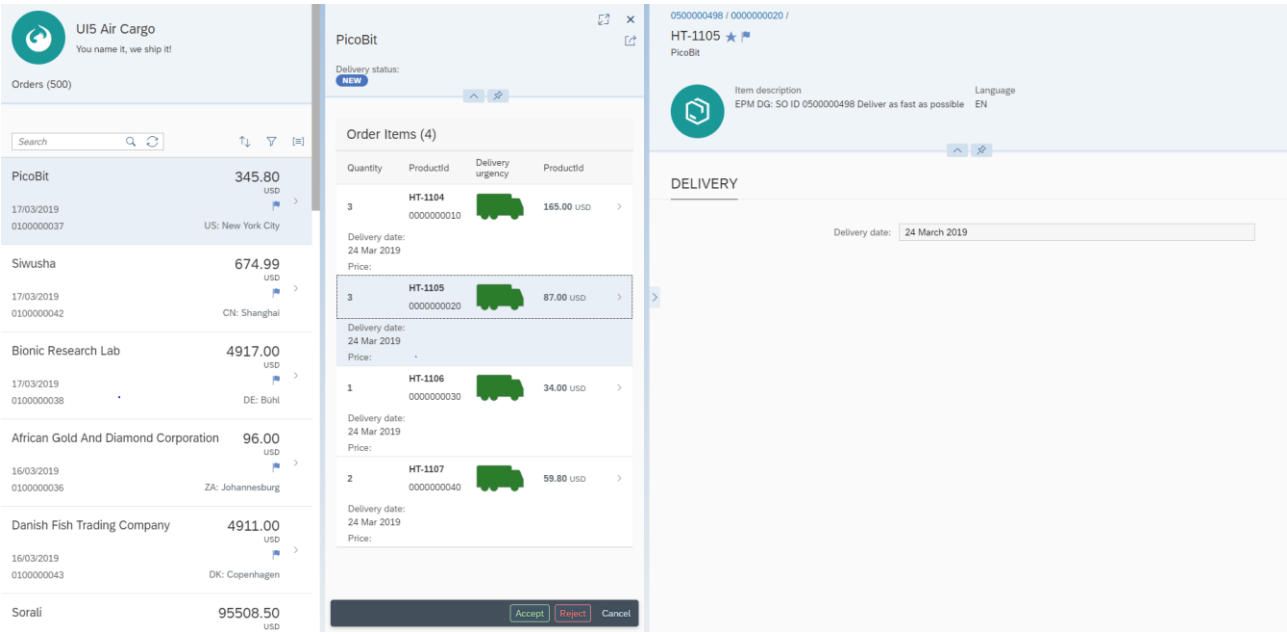

Figure 1 – The App includes two search fields now

ENABLING NEW FEATURES FOR EXISTING CONTROLS

Sticky table header

The first cool feature is the stickiness of the table header, which you can apply in the detail part of the `FlexibleColumnLayout`. It is useful when you want to optimize the screen space and show more information.

webapp/view/Detail.view.xml

```
…
   <Table
      id="lineItemsList"
      width="auto"
      items="{ToLineItems}"
      mode="SingleSelectMaster"
      sticky="ColumnHeaders,HeaderToolbar"
      updateFinished=".onListUpdateFinished"
      noDataText="{i18n>detailLineItemTableNoDataText}"
      busyIndicatorDelay="{detailView>/lineItemTableDelay}"
      selectionChange="action"
   xmlns:action="http://schemas.sap.com/sapui5/extension/sap.ui.core.CustomData/1"
   action:wiring="\{'selectionChange':\{'navigation':\{'routeName':'Info'\}\}\}">
      <headerToolbar>
         <Toolbar>
            <Title
               id="lineItemsTitle"
               text="{detailView>/lineItemListTitle}"
               titleStyle="H3"/>
         </Toolbar>
      </headerToolbar>
      …

   <Table>
</semantic:content>
 …
```

Note: Although the code is working, there might be some validation warnings in SAP Web IDE in some of the code lines. If this is the case, you can ignore them.

Set one of the properties of the `sap.m.Table` called `sticky`. It defines the section of the control that remains fixed at the top of the page during vertical scrolling as long as the control is in the viewport. To get the column headers and the top toolbar stacked, the value of this property is `ColumnHeaders,HeaderToolbar`.

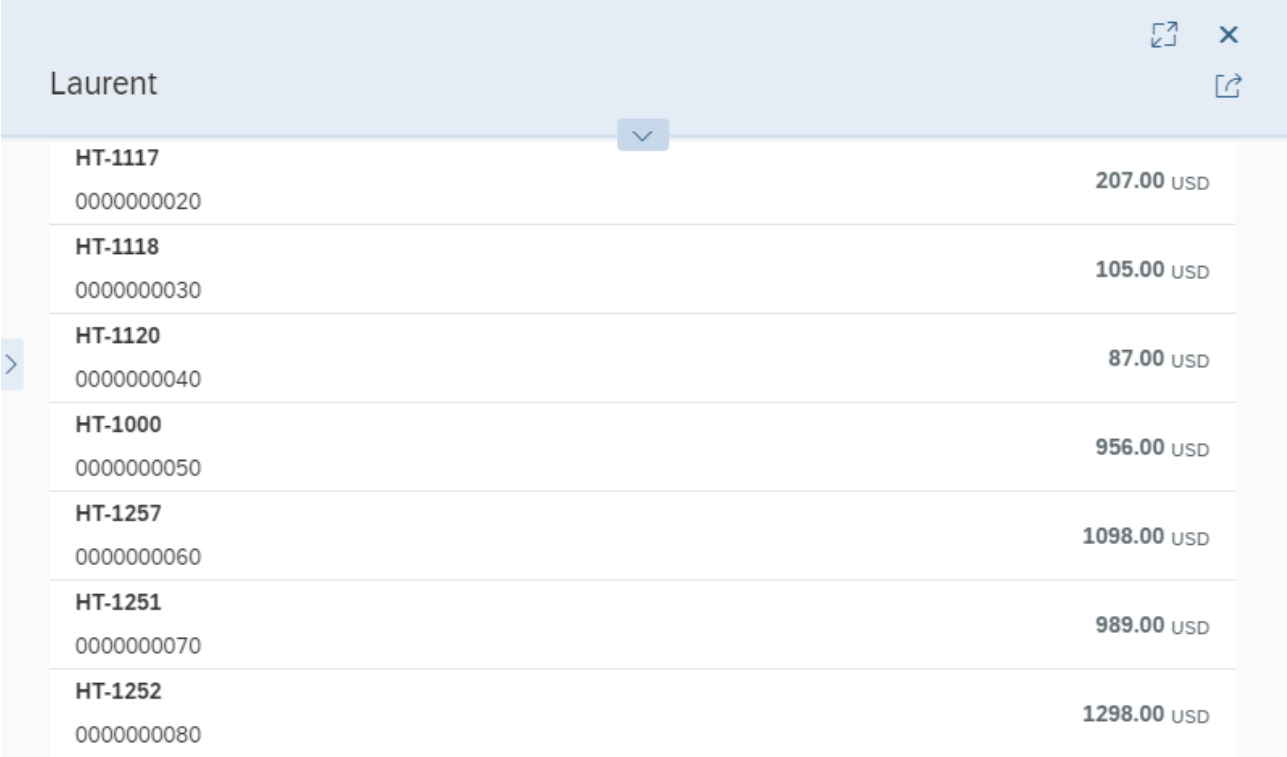

Figure 2 – Before adding the `sticky` property and after scrolling, the top of the table gets hidden

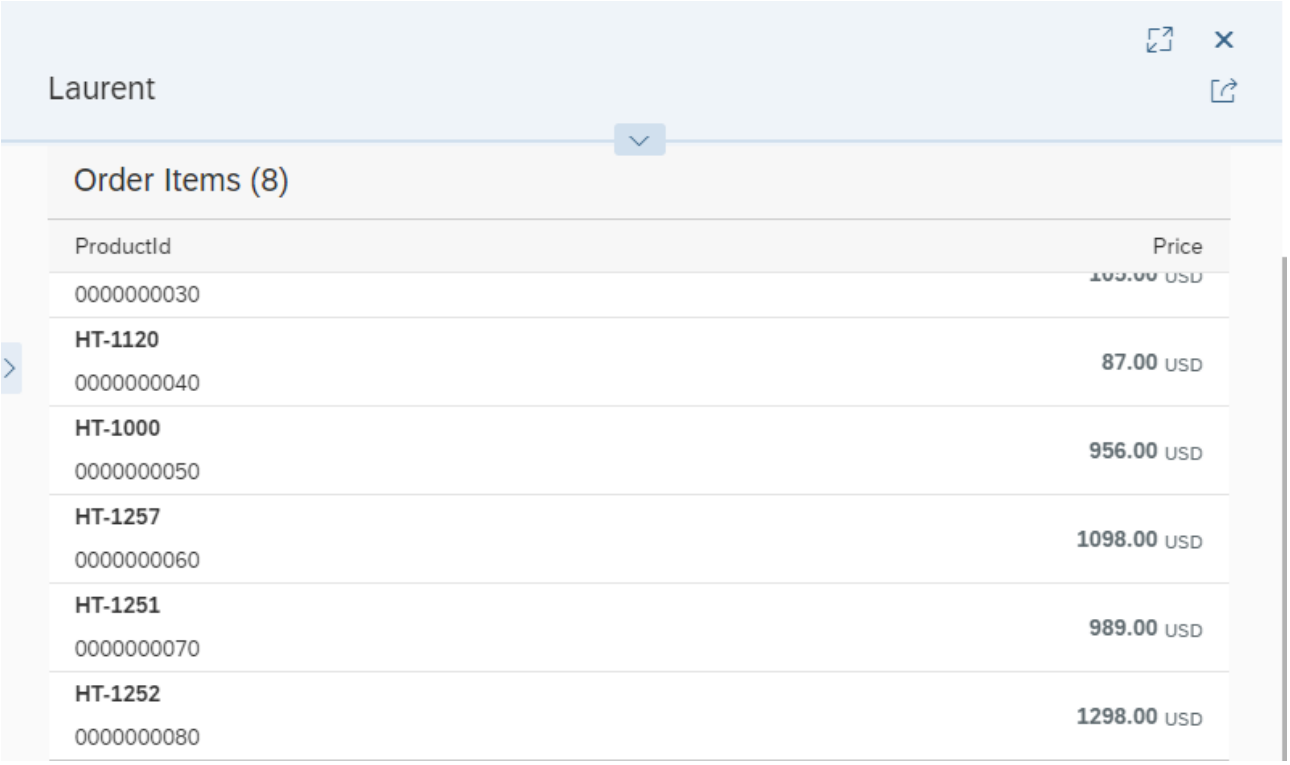

Figure 3 – Now, after scrolling the top toolbar and the column headers of the table get stacked and they appear always in the view

Floating footer in semantic page

As a `SemanticPage` is used in the detail view, let's add a footer part to it. In it, let's place some buttons just for demo purpose.

webapp/view/Detail.view.xml

```
…
<semantic:SemanticPage id="detailPage"
      busy="{detailView&gt;/busy}"
      busyIndicatorDelay="{detailView&gt;/delay}"
      showFooter="true">
   <semantic:titleHeading>
      <Title text="{CustomerName}"/>
   </semantic:titleHeading>

      …

   <semantic:positiveAction>
      <semantic:PositiveAction/>
   </semantic:positiveAction>

   <semantic:negativeAction>
      <semantic:NegativeAction/>
   </semantic:negativeAction>

   <semantic:footerCustomActions>
      <Button text="{i18n>DetailFooterCancel}"/>
   </semantic:footerCustomActions>
</semantic:SemanticPage>
 …
```

First, set the `showFooter` property of the semantic page to `true`. Then place the actions at the bottom. Let's include some semantic-specific buttons placed in the `FooterRight` area of the `SemanticPage` footer with default text value set to "Accept" and "Reject" via the corresponding `positiveAction` and `negativeAction` aggregations.

Right after that, add `footerCustomActions` – a button for the cancel action.

webapp/i18n/i18n.properties

```
…
#XGRP: Title for the NetAmount column in the ToLineItems table
detailLineItemTableUnitNumberColumn=Price

#XBUT: Text for the Cancel button in the footer
DetailFooterCancel=Cancel

#XTIT: Send E-Mail subject
…
```

A text for internationalization is also needed for the cancel button at the bottom.

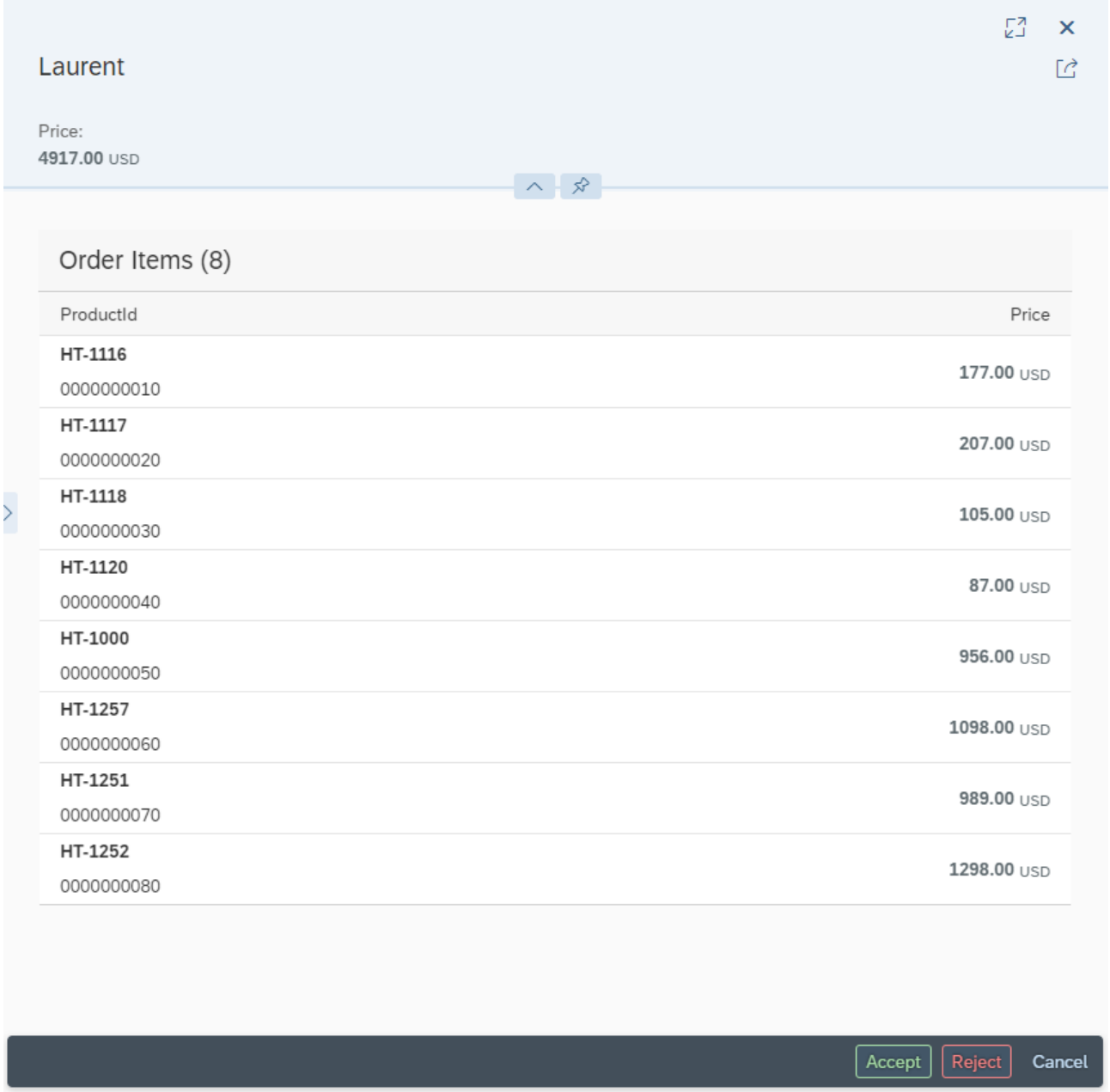

Figure 4 – The footer appears at the bottom of the page

Changing the info view

Let's show some more details for each item in the order in the info view by adding an `ObjectPageLayout` at the place of `List`. This control is used as a layout that allows apps to easily display information related to a business object, which makes sense in your app.

webapp/view/Info.view.xml

```
<mvc:View
   xmlns:core="sap.ui.core"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns="sap.m"
   xmlns:html="http://www.w3.org/1999/xhtml"
   controllerName="opensap.orders.controller.Info"
   xmlns:cd=http://schemas.sap.com/sapui5/extension/sap.ui.core.CustomData/1>
   <App>
      <pages>
         <Page title="{i18n>infoTitle}">
            <content>
                <List noDataText="Drop list items here"
                    id="list0" items="{path:'/SalesOrderLineItemSet',
                                       parameters:{expand:'ToProduct',
                    select:'ProductID,ToProduct/Name'}}">
                    <items>
                        <StandardListItem type="Navigation"
             title="{ToProduct/Name}"
                    description="{ProductID}"
                    icon="sap-icon://picture"
                    id="item0"/>
                    </items>
                </List>
            </content>
         </Page>
      </pages>
   </App>

</mvc:View>
```

As a next step, remove the existing `App` and the corresponding included libraries from the view.

webapp/view/Info.view.xml

```
<mvc:View
   controllerName="opensap.orders.controller.Info"
   height="100%"
   xmlns="sap.m"
   xmlns:mvc="sap.ui.core.mvc"
   xmlns:uxap="sap.uxap"
   xmlns:f="sap.f"
   xmlns:l="sap.ui.layout"
   xmlns:form="sap.ui.layout.form">

   <uxap:ObjectPageLayout
      id="objectPageLayout"
      showTitleInHeaderContent="true"
      alwaysShowContentHeader="false"
      preserveHeaderStateOnScroll="false"
      headerContentPinnable="true"
      isChildPage="true"
```

```
      enableLazyLoading="false">

      <uxap:headerTitle>
         <uxap:ObjectPageHeader>
            <uxap:breadcrumbs>
               <Breadcrumbs>
                  <Link text='{SalesOrderID}'/>
                  <Link text='{ItemPosition}'/>
               </Breadcrumbs>
            </uxap:breadcrumbs>
         </uxap:ObjectPageHeader>
      </uxap:headerTitle>

      <uxap:headerContent>
         <FlexBox wrap="Wrap" fitContainer="true" alignItems="Stretch">
            <f:Avatar class="sapUiSmallMarginEnd sapUiSmallMarginTop"
               src="sap-icon://product"
               displaySize="L"/>

            <l:VerticalLayout
               class="sapUiSmallMarginEnd sapUiSmallMarginTop">
               <Label text="{i18n>infoItemDescription}"/>
               <Text text="{ToHeader/Note}"/>
            </l:VerticalLayout>
            <l:VerticalLayout
               class="sapUiSmallMarginEnd sapUiSmallMarginTop">
               <Label text="{i18n>infoItemDescriptionLanguage}"/>
               <Text text="{ToHeader/NoteLanguage}"/>
            </l:VerticalLayout>
         </FlexBox>
      </uxap:headerContent>

      <uxap:sections>
         <uxap:ObjectPageSection title="{i18n>infoItemDeliveryTitle}">
            <uxap:subSections>
               <uxap:ObjectPageSubSection>
                  <uxap:blocks>
                     <form:SimpleForm editable="true">
                        <Label text="{i18n>infoItemDeliveryDateDescr}"/>
                        <DatePicker
                           dateValue="{DeliveryDate}"
                           displayFormat="long"
                           editable="false"/>
                     </form:SimpleForm>
                  </uxap:blocks>
               </uxap:ObjectPageSubSection>
            </uxap:subSections>
         </uxap:ObjectPageSection>
      </uxap:sections>
   </uxap:ObjectPageLayout>
</mvc:View>
```

After that, include the `ObjectPageLayout` and set its `headerTitle` part, containing `breadcrumbs`, as well as the `headerContent` part. This contains an `Avatar` with a placeholder picture as well as the description text from the model. In addition, add a section in the `ObjectPageLayout` to display the delivery date also from the model.

webapp/i18n/i18n.properties

```
…

#XBUT: Text for the send e-mail button
sendEmail=Send E-Mail

#~~~ Info View ~~~~~~~~~~~~~~~~~~~~~~~~~~

#XFLD: Text for item description of each item in the order
infoItemDescription=Item description

#XFLD: Text for item description language of each item in the order
infoItemDescriptionLanguage=Language

#XGRP: Title of the details group of information about each item
infoItemDetailsTitle=Details

#XGRP: Title of the delivery group of information about each item
infoItemDeliveryTitle=Delivery

#XFLD: Text for the delivery date of each item in the order
infoItemDeliveryDateDescr=Delivery date

#~~~ Not Found View ~~~~~~~~~~~~~~~~~~~~~~

#XTIT: Not found view title
notFoundTitle=Not Found

…
```

Include the new texts to be translated for the view in the internationalization file.

webapp/view/Detail.view.xml

```
…

   <Table id="lineItemsList"
      width="auto"
      items="{ToLineItems}"
      mode="SingleSelectMaster"
      …
      >
      <headerToolbar>
         …

      </headerToolbar>
      <columns>
         …
      </columns>
      <items>
         <ColumnListItem selected="{= ${ItemPosition} ===
${appView>/selectedItemId}}">
               …

      </items>
   </Table>

…
```

In the detail part, ensure that the selected `ColumnListItem` will stay selected, for example even after a refresh is done. Mark the item which has its ID the same as the one selected.

webapp/controller/Detail.controller.js

```
sap.ui.define([
   "./BaseController",
   "sap/ui/model/json/JSONModel",
   "../model/formatter",
   "sap/m/library",
   "sap/ui/Device"
], function (BaseController, JSONModel, formatter, mobileLibrary, Device) {
   "use strict";

   return BaseController.extend("opensap.orders.controller.Info", {

      …

      toggleFullScreen: function () {

         …

      },

      /**
       *@memberOf opensap.orders.controller.Detail
       * @param {sap.ui.base.Event} oEvent pattern match event in route 'object'
       */
      action: function (oEvent) {
         var that = this;
         var actionParameters =
JSON.parse(oEvent.getSource().data("wiring").replace(/'/g, "\""));
         var eventType = oEvent.getId();
         var aTargets = actionParameters[eventType].targets || [];
         aTargets.forEach(function (oTarget) {
            var oControl = that.byId(oTarget.id);
            if (oControl) {
               var oParams = {};
               for (var prop in oTarget.parameters) {
                  oParams[prop] = oEvent.getParameter(oTarget.parameters[prop]);
               }
               oControl[oTarget.action](oParams);
            }
         });
         var oNavigation = actionParameters[eventType].navigation;
         if (oNavigation) {
            var oParams = {};
            (oNavigation.keys || []).forEach(function (prop) {
               oParams[prop.name] = encodeURIComponent(JSON.stringify({
                  value:
oEvent.getSource().getBindingContext(oNavigation.model).getProperty(prop.name),
                  type: prop.type
               }));
            });
            if (Object.getOwnPropertyNames(oParams).length !== 0) {
      this.getOwnerComponent().getRouter().navTo(oNavigation.routeName, oParams);
            } else {
      this.getOwnerComponent().getRouter().navTo(oNavigation.routeName);
            }
         }
         var bReplace = !Device.system.phone;
         this.getRouter().navTo("Info", {
```

```
            objectId : (oEvent.getParameter("listItem") ||
oEvent.getSource()).getBindingContext().getProperty("SalesOrderID"),
            itemPosition : (oEvent.getParameter("listItem") ||
oEvent.getSource()).getBindingContext().getProperty("ItemPosition")
         }, bReplace);
      }
   });
});
```

In the `Detail.controller.js`, remove the code needed for managing the previous view of the end column of the flexible column layout.

When the user chooses an item from the order, navigate to the end column and load the corresponding data. Also check if the device the app is run on is a phone. If so, show only the third column. If not, open the third column or change its binding.

webapp/controller/Info.controller.js

```
sap.ui.define([
   "./BaseController"
], function (BaseController) {
   "use strict";

   return BaseController.extend("opensap.orders.controller.Info", {

      /**
       * Called when a controller is instantiated and its View controls (if
       * available) are already created.
       * Can be used to modify the View before it is displayed, to bind event
       * handlers and do other one-time initialization.
       * @memberOf opensap.orders.view.Info
       */
      onInit: function () {
   this.getRouter().getRoute("Info").attachPatternMatched(this._onInfoMatched, this);
      },

      /**
       * Binds the view to the object path and expands the aggregated line items.
       * @function
       * @param {sap.ui.base.Event} oEvent pattern match event in route 'object'
       * @private
       */
      _onInfoMatched: function (oEvent) {
         this.getModel("appView").setProperty("/layout", "ThreeColumnsMidExpanded");
         var sObjectId = oEvent.getParameter("arguments").objectId,
            sItemPosition = oEvent.getParameter("arguments").itemPosition;

         this.getModel("appView").setProperty("/layout", "ThreeColumnsEndExpanded");
         this.getModel().metadataLoaded().then(function () {
            var sObjectPath = this.getModel().createKey("SalesOrderLineItemSet", {
               SalesOrderID: sObjectId,
               ItemPosition: sItemPosition
            });
            this.getView().bindElement({
               path: "/" + sObjectPath,
               parameters: {
                  'expand': 'ToHeader'
               }
            });
         }.bind(this));
      }
```

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

In the `Info.controller.js` load the binding data when the route gets hit and adjust the layout mode of the FCL.

webapp/manifest.json

```
{
   "_version": "1.12.0",
   "sap.app": {

…

   },

…

   "sap.ui5": {
      "rootView": {
         "viewName": "opensap.orders.view.App",
         "type": "XML",
         "async": true,
         "id": "app"
      },
      "dependencies": {
         "minUI5Version": "1.60.0",
         "libs": {
            "sap.ui.core": {},
            "sap.m": {},
            "sap.f": {},
            "sap.ui.layout": {},
            "sap.uxap": {},
            "sap.tnt": {},
            "sap.suite.ui.commons": {}
         }
      },

…

      "routing": {
         "config": {
            "routerClass": "sap.f.routing.Router",
            "viewType": "XML",
            "viewPath": "opensap.orders.view",
            "controlId": "layout",
            "controlAggregation": "beginColumnPages",
            "bypassed": {
               "target": "notFound"
            },
            "async": true
         },
         "routes": [
            {
               "pattern": "",
               "name": "master",
               "target": "master"
            },
            {
               "pattern": "SalesOrderSet/{objectId}",
               "name": "object",
```

```
                "target": [
                   "master",
                   "object"
                ]
             },
             {
                "pattern": "Info",
                "pattern": "SalesOrderSet/{objectId}/Item/{itemPosition}",
             "name": "Info",
                "target": [
                   "master",
                   "object",
                   "Info"
                ]
             }
          ],
          "targets": {

…

          }
       }
    }
}
```

As the app has to hit a route to display the information for the clicked item, modify the routes in the `manifest.json`.

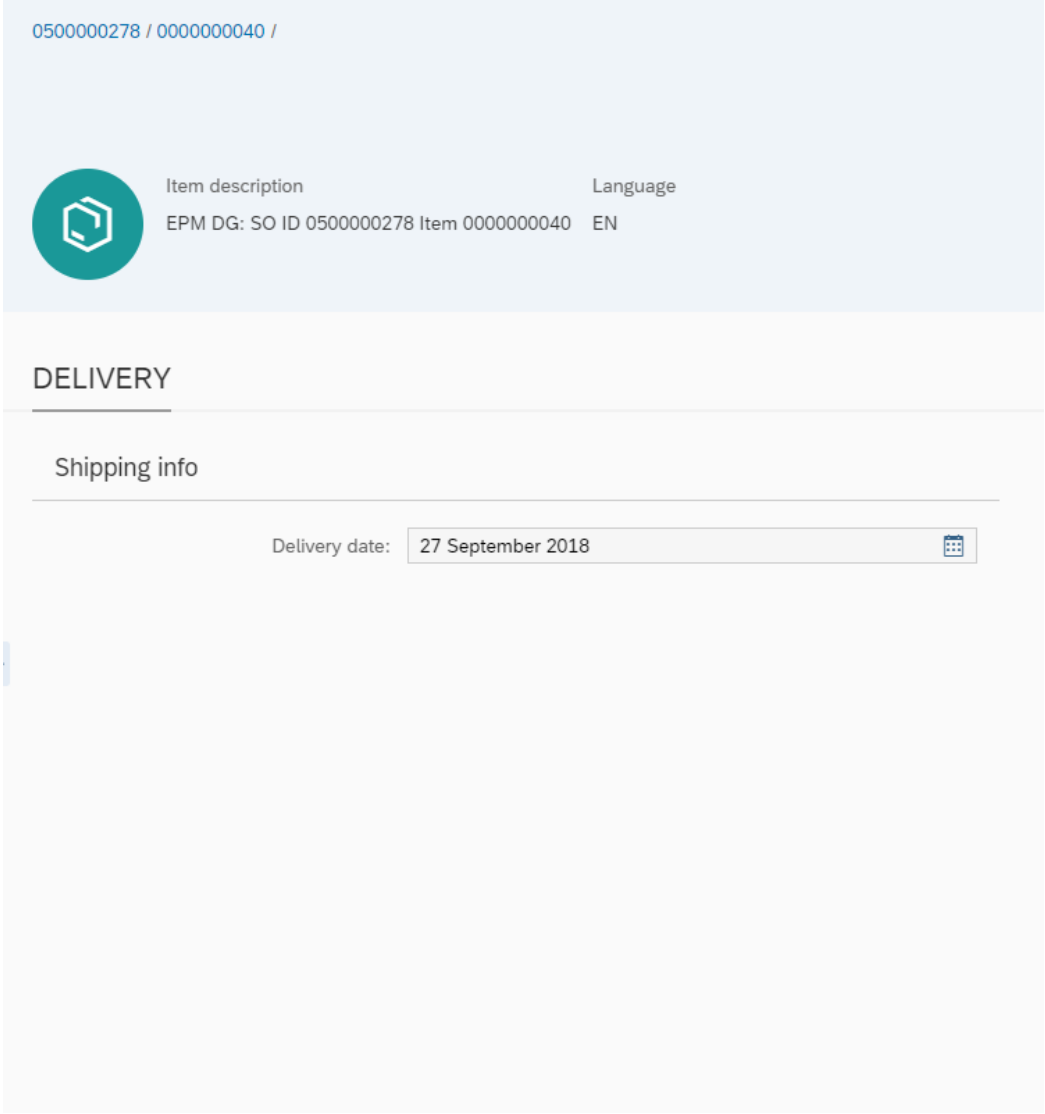

Figure 5 – The footer appears at the bottom of the page

As now you have already contextually connected the end column to an item from the middle one, a good idea is to visualize this connection for the user.

webapp/view/Detail.view.xml

```
…

   <Table id="lineItemsList"
      width="auto"
      items="{ToLineItems}"
      mode="SingleSelectMaster"
      …
      >
      <headerToolbar>
         …

      </headerToolbar>
      <columns>
         …
```

```
      </columns>
      <items>
         <ColumnListItem
            selected="{= ${ItemPosition} === ${appView>/selectedItemId} }"
            type="Navigation">           …

      </items>
   </Table>

…
```

Add a type `Navigation` to the `ColumnListItems` of the `Table`, in order to give a clue to the user that the items are interactive and more information about each clicked item will be shown.

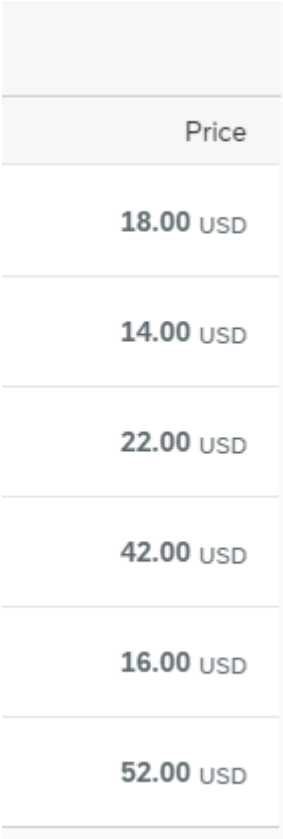

Figure 6 – Before adding the property

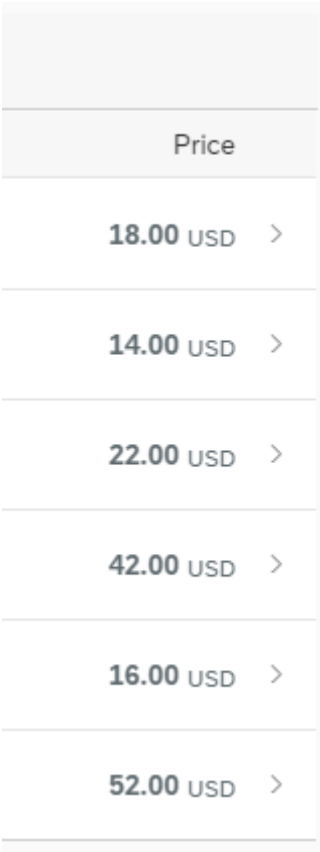

Figure 7 – After adding the property

Header area in object page layout

Now, let's enhance the info view of the app and the header part of the `ObjectPageLayout`. Now, it contains a `headerTitle` and `headerContent` part. Let's enhance the `headerTitle` by making it a bit more dynamic.

webapp/view/Info.view.xml

```
…

<uxap:ObjectPageLayout
   id="objectPageLayout"
   showTitleInHeaderContent="true"
   alwaysShowContentHeader="false"
   preserveHeaderStateOnScroll="false"
   headerContentPinnable="true"
   isChildPage="true"
   enableLazyLoading="false">

   <uxap:headerTitle>
      <uxap:ObjectPageDynamicHeaderTitle>
         <uxap:breadcrumbs>
            <Breadcrumbs>
               <Link text='{SalesOrderID}'/>
               <Link text='{ItemPosition}'/>
            </Breadcrumbs>
         </uxap:breadcrumbs>

         <uxap:expandedHeading>
            <FlexBox wrap="Wrap"
               fitContainer="true"
               alignItems="Center">
               <Title text="{ProductID}"
                  wrapping="true"
               class="sapUiTinyMarginEnd sapUiTinyMarginTop"/>
               <FlexBox wrap="NoWrap"
                  fitContainer="true"
                  alignItems="Center"
                  class="sapUiTinyMarginEnd">
                  <ObjectMarker type="Favorite"
                     class="sapUiTinyMarginEnd"/>
                  <ObjectMarker type="Flagged"/>
               </FlexBox>
            </FlexBox>
         </uxap:expandedHeading>
         <uxap:expandedContent>
            <Text text="{ToHeader/CustomerName}"/>
         </uxap:expandedContent>
      </uxap:ObjectPageDynamicHeaderTitle>
   </uxap:headerTitle>

      …

</uxap:ObjectPageLayout>
…
```

The dynamic part of the header title has four important aggregations which control the content – `expandedHeading`, `snappedHeading`, `expandedContent` and `snappedContent`.

For now, add the content to be shown when the header is expanded. In the `expandedHeading` include a `Title`, some `ObjectMarker` controls and a text in the `expandedContent`.

The header title area can now be clicked/tapped to expand/collapse the dynamic header. An arrow button is positioned either below the header content (when header is expanded) or below the header title (when the header is collapsed). The expand/collapsed state of the header can be toggled by either clicking on the header title area, or the arrow button.

webapp/view/Info.view.xml

```
…

<uxap:ObjectPageLayout
   id="objectPageLayout"
   showTitleInHeaderContent="true"
   alwaysShowContentHeader="false"
   preserveHeaderStateOnScroll="false"
   headerContentPinnable="true"
   isChildPage="true"
   enableLazyLoading="false">

   <uxap:headerTitle>
      <uxap:ObjectPageDynamicHeaderTitle>
         <uxap:breadcrumbs>
            <Breadcrumbs>
               <Link text='{SalesOrderID}'/>
               <Link text='{ItemPosition}'/>
            </Breadcrumbs>
         </uxap:breadcrumbs>

         <uxap:expandedHeading>

            …

         </uxap:expandedHeading>
         <uxap:snappedHeading>
            <FlexBox wrap="Wrap"
               fitContainer="true"
               alignItems="Center">
               <FlexBox wrap="NoWrap"
                  fitContainer="true"
                  alignItems="Center"
                  class="sapUiTinyMarginEnd">
                  <f:Avatar
                     src="sap-icon://product"
                     displaySize="S"
                     class="sapUiTinyMarginEnd"/>
                  <Title text="{ProductID}"
                     wrapping="true"
                     class="sapUiTinyMarginEnd"/>
               </FlexBox>
               <FlexBox wrap="NoWrap"
                  fitContainer="true"
                  alignItems="Center"
                  class="sapUiTinyMarginEnd">
                  <ObjectMarker type="Favorite"
                        class="sapUiTinyMarginEnd"/>
                  <ObjectMarker type="Flagged"/>
               </FlexBox>
            </FlexBox>
         </uxap:snappedHeading>
         <uxap:expandedContent>
```

```
            <Text text="{ToHeader/CustomerName}"/>
         </uxap:expandedContent>
         <uxap:snappedContent>
            <Text text="{ToHeader/CustomerName}"/>
         </uxap:snappedContent>
      </uxap:ObjectPageDynamicHeaderTitle>
   </uxap:headerTitle>

      …

</uxap:ObjectPageLayout>
…
```

When the dynamic header is snapped, the `headerContent` of the object page gets hidden. That's why you need to move the most important information from the `headerContent` to the `snappedHeading` and `snappedContent`.

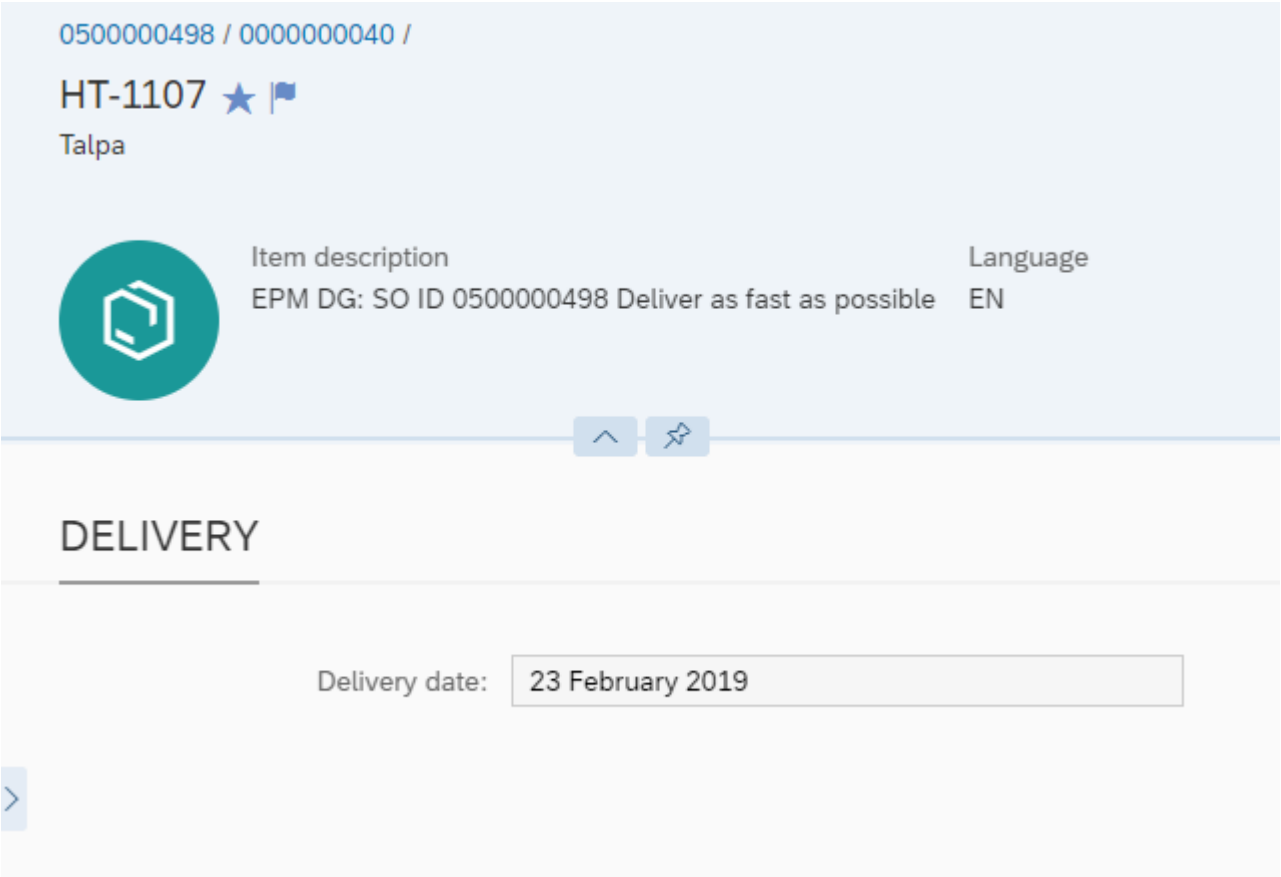

Figure 8 – The Header Title area is expanded

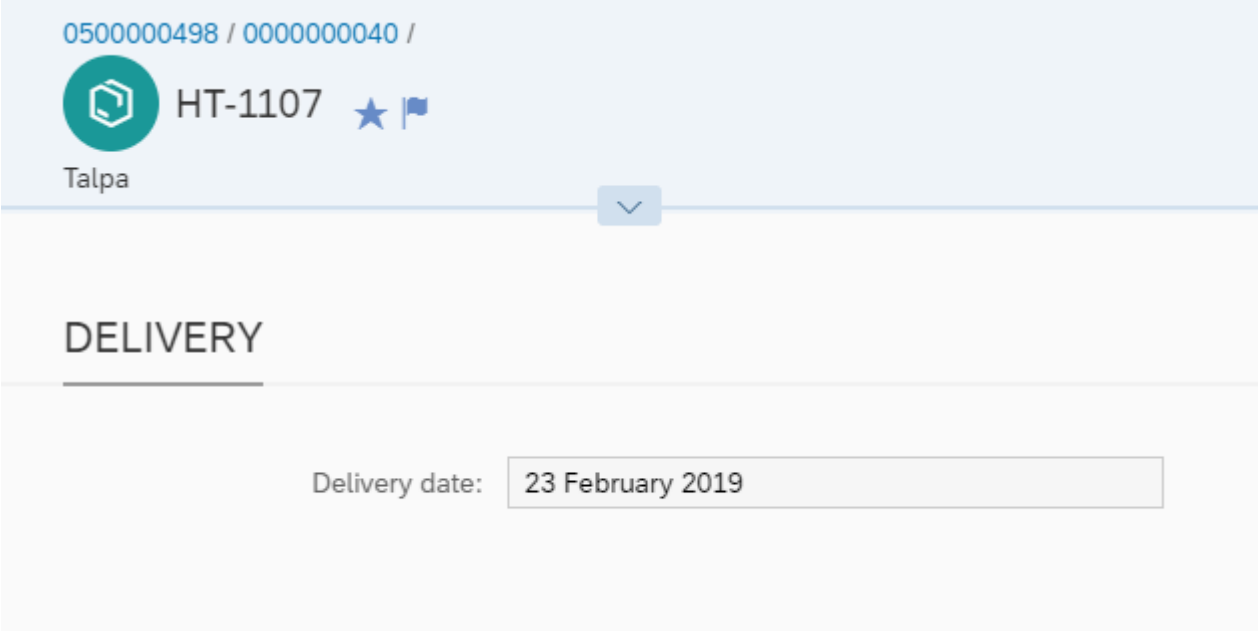

Figure 9 – The Header Title area is snapped

NEW FEATURE RICH CONTROLS

InfoLabel showing delivery status

An important part of each delivery is the delivery status. It would be great if the app shows if each order is accepted, canceled, new, etc. at a glance. Looks like the `sap.tnt.InfoLabel` control would help out. The `InfoLabel` is a small non-interactive control which contains text information and non-semantic color chosen from a list of predefined color schemes.

webapp/view/Detail.view.xml

```
<mvc:View
   controllerName="opensap.orders.controller.Detail"
   xmlns="sap.m"
   xmlns:semantic="sap.f.semantic"
   xmlns:tnt="sap.tnt"
   xmlns:mvc="sap.ui.core.mvc">

   <semantic:SemanticPage
      id="detailPage"
      busy="{detailView>/busy}"
      busyIndicatorDelay="{detailView>/delay}"
      showFooter="true">

      <semantic:titleHeading>
         <Title text="{CustomerName}"/>
      </semantic:titleHeading>

      <semantic:headerContent>
         <ObjectAttribute title="{i18n>priceTitle}"/>
         <ObjectNumber
            id="objectHeaderNumber"
            number="{
               path: 'NetAmount',
               formatter: '.formatter.currencyValue'
            }"
            unit="{CurrencyCode}"
         />
         <ObjectAttribute title="{i18n>StatusDesc}"/>
         <tnt:InfoLabel
            text="{LifecycleStatusDescription}"/>
      </semantic:headerContent>

      …

   </semantic:SemanticPage>
</mvc:View>
```

In the `headerContent` aggregation of the `SemanticPage` in the detail part of the `FlexibleColumnLayout`, use the `InfoLabel` to replace the `ObjectNumber` generated from the template which shows the `NetAmount`.

Include the `tnt` library as well, since the `InfoLabel` control is contained there. Then define it in the `headerContent` aggregation with its `text` property coming from the model.

webapp/i18n/i18n.properties

```
…

#XBUT: Text for the send e-mail button
sendEmail=Send E-Mail

#XTIT: Title text for the price
priceTitle=Price

#XTIT: Title text for the delivery status
StatusDesc=Delivery status

#~~~ Info View ~~~~~~~~~~~~~~~~~~~~~~~~~~

#XFLD: Text for item description of each item in the order
infoItemDescription=Item description

…
```

Include the `i18n` text instead of the `Price` text.

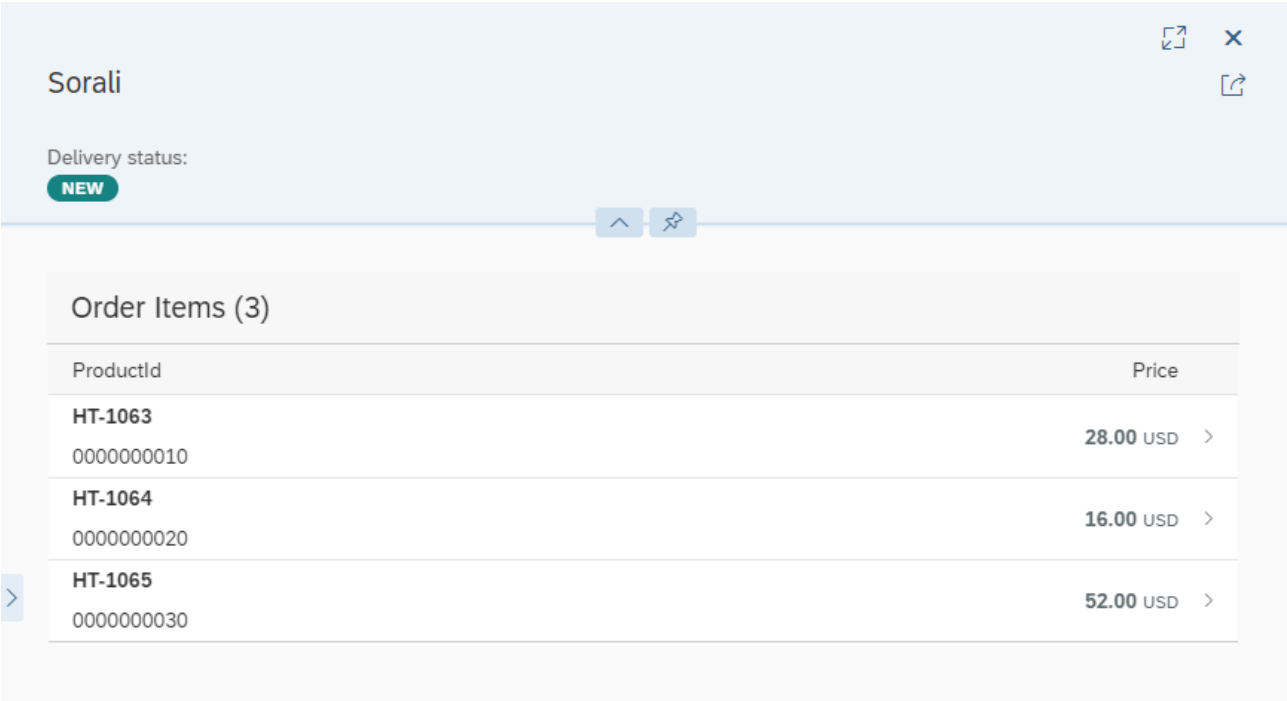

Figure 10 – The app now shows the delivery status of the order

The `InfoLabel` also has a property called `colorScheme`. It specifies the color filling of the control and accepts a digit as a value. The default `colorScheme` is “7” as displayed in the view with the greenish color now.

Let’s use this property and change the color of the control via a formatter function.

webapp/view/Detail.view.xml

```
<mvc:View xmlns="sap.m"
   xmlns:mvc="sap.ui.core.mvc"
```

```
   xmlns:semantic="sap.f.semantic"
   xmlns:tnt="sap.tnt"
   controllerName="my.opensap.w2u3.controller.Detail">
      <semantic:SemanticPage id="detailPage"
              busy="{detailView&gt;/busy}"
              busyIndicatorDelay="{detailView&gt;/delay}"
              showFooter="true">
         <semantic:titleHeading>
            <Title text="{CustomerName}"/>
         </semantic:titleHeading>
         <semantic:headerContent>
            <ObjectAttribute title="{i18n>StatusDesc}"/>
            <tnt:InfoLabel
               text="{LifecycleStatusDescription}"
               colorScheme="{
                  path: 'LifecycleStatusDescription',
                  formatter: '.formatter.deliveryStatus'
               }"/>
         </semantic:headerContent>

      …

   </semantic:SemanticPage>
</mvc:View>
```

In the XML view, define the path of the value from the model, which will be used for conversion, and the path to the formatter function. In it, the value to be converted is received as a first parameter.

webapp/model/formatter.js

```
sap.ui.define([
   ], function () {
      "use strict";

      return {
         /**
          * Rounds the currency value to 2 digits
          *
          * @public
          * @param {string} sValue value to be formatted
          * @returns {string} formatted currency value with 2 digits
          */
         currencyValue : function (sValue) {
            if (!sValue) {
               return "";
            }

            return parseFloat(sValue).toFixed(2);
         },

         /**
          * Converts the delivery status value from the data into a
          * number used for the colorScheme propery of the InfoLabel
          * control.
          *
          * @public
          * @param {string} sValue The value to be formatted
          * @returns {int} The formatted delivery status value for the
          * InfoLabel colorScheme
          */
         deliveryStatus: function(sValue) {
```

```
            switch(sValue) {
               case "New": return 5;
               case "In Progress": return 1;
               case "Canceled": return 3;
               case "Closed": return 7;
               default: return 7;
            }
         }
      };

   }
);
```

In the formatter, state the custom conversion logic and return the value to be passed to the `colorScheme` property.

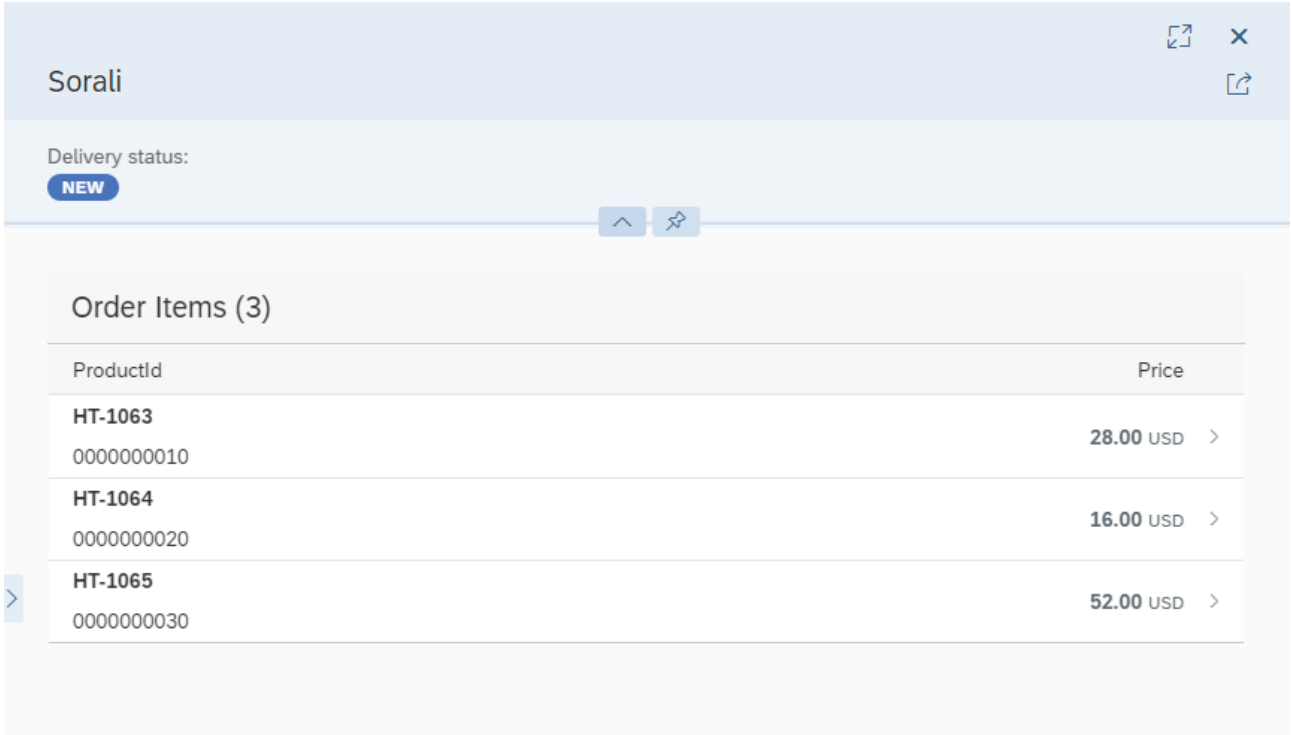

Figure 11 – The delivery status is colored differently in each status of the order

`StatusIndicator` showing the days to delivery for each item

The delivery date and the quantity of the items in the order are useful information as well. Let's add them to the table in the detail part. It would be even better if there was a better visual clue for noticing how urgent each delivery is.

The `sap.suite.ui.commons.statusindicator.StatusIndicator` control displays a value between 0 and 100 via a SVG. The image is filled according to a given value between 0 and 100. There are a couple of predefined library shapes in UI5. Let's use one of them in the demo.

webapp/view/Detail.view.xml

```
<mvc:View
   controllerName="opensap.orders.controller.Detail"
   xmlns="sap.m"
   xmlns:semantic="sap.f.semantic"
   xmlns:tnt="sap.tnt"
   xmlns:si="sap.suite.ui.commons.statusindicator"
   xmlns:mvc="sap.ui.core.mvc">
…

<Table id="lineItemsList"
   width="auto"
   items="{ToLineItems}"
   mode="SingleSelectMaster"

      …

   <columns>
      <Column>
         <Text text="{i18n>detailLineItemTableIDQuantity}"/>
      </Column>
      <Column>
         <Text text="{i18n>detailLineItemTableIDColumn}"/>
      </Column>
      <Column
         minScreenWidth="Tablet"
         demandPopin="true">
         <Text text="{i18n>detailLineItemTableDeliveryDate}"/>
      </Column>
      <Column>
         <Text text="{i18n>detailLineItemTableDeliveryUrgency}"/>
      </Column>
      <Column minScreenWidth="Tablet"
         demandPopin="true"
         hAlign="End">
         <Text text="{i18n>detailLineItemTableUnitNumberColumn}"/>
      </Column>
   </columns>
   <items>
      <ColumnListItem
         selected="{= ${ItemPosition} === ${appView>/selectedItemId} }"
         type="Navigation">
         <cells>
            <ObjectNumber number="{Quantity}"/>
            <ObjectIdentifier title="{ProductID}"
text="{ItemPosition}"/>
            <Text text="{
               path: 'DeliveryDate',
               type: 'sap.ui.model.type.Date',
               formatOptions: {
                  style: 'medium'
```

```
                }
            }"/>
            <si:StatusIndicator
                id="statusIndicator"
                width="4.5rem"
                height="3.5rem"
                value="{= (Date.parse(${DeliveryDate}) - 1546300801000) / 52560000
}">
                <si:ShapeGroup>
                    <si:LibraryShape
                        id="customShape0"
                        shapeId="vehicle_truck_1"/>
                </si:ShapeGroup>
                <si:propertyThresholds>
                    <si:PropertyThreshold
                        fillColor="Error"
                        toValue="50"/>
                    <si:PropertyThreshold
                        fillColor="Critical"
                        toValue="65"/>
                    <si:PropertyThreshold
                        fillColor="Neutral"
                        toValue="80"/>
                    <si:PropertyThreshold
                        fillColor="Good"
                        toValue="100"/>
                </si:propertyThresholds>
            </si:StatusIndicator>
            <ObjectNumber number="{
                path: 'NetAmount',
                formatter: '.formatter.currencyValue'}"
   unit="{CurrencyCode}"/>
        </cells>
      </ColumnListItem>
   </items>
</Table>

…
```

Now there are three columns for the table in the detail view – one showing the item quantity, one showing the actual delivery date and one showing visually how many days are left until delivery.

Add three new columns to the `columns` aggregation of the `sap.m.Table` and three `ColumnListItem` cells containing a `sap.m.ObjectNumber`, a `sap.m.Text` and a `sap.suite.ui.commons.statusindicator.StatusIndicator` accordingly.

In the `ShapeGroup` aggregation of the latter, pass the SVG shape to be displayed. In the current case this is the predefined "`vehicle_truck_1`" shape. For more information you can see the StatusIndicator documentation page.

webapp/i18n/i18n.properties

```
…


#XTIT: Title of the ToLineItems table
detailLineItemTableHeadingCount=Procuts ({0})

#XGRP: Title for the Quantity column in the ToLineItems table
```

```
detailLineItemTableIDQuantity=Quantity

#XGRP: Title for the ProductID column in the ToLineItems table
detailLineItemTableIDColumn=Name

#XGRP: Title for the NetAmount column in the ToLineItems table
detailLineItemTableUnitNumberColumn=Price

#XGRP: Title for the DeliveryDate column in the ToLineItems table
detailLineItemTableDeliveryDate=Delivery date

#XGRP: Title for the days until DeliveryDate column in the ToLineItems table
detailLineItemTableDeliveryUrgency=Delivery urgency

…
```

Some texts for internationalization are needed as well.

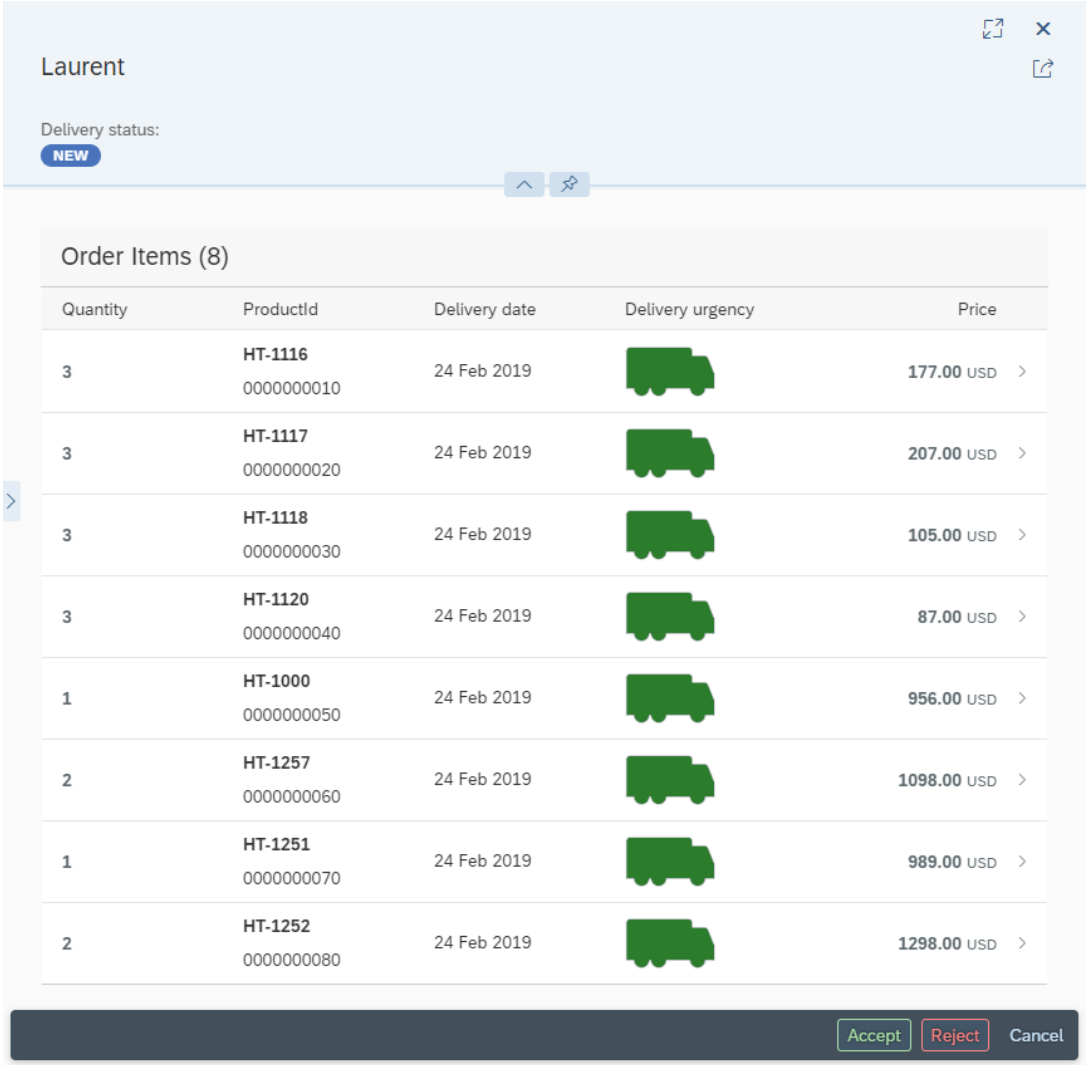

Figure 12 – StatusIndicator in the table in the Detail page

★ CHALLENGE YOURSELF: ADD PRODUCT DETAILS TO THE INFO PAGE

This task does not come with a predefined solution and can be solved creatively – dive a bit deeper into the topics and exchange with other learners to make the most out of your learning experience. Good luck!

Summary

One of the clients of the app – UI52 Air Cargo – was amazed by the progress on the order browser app. While navigating through the app and trying out the new features, the idea to display more information on the products of an order came up.

With all essential information about products in one place, the delivery department can process orders more efficiently and faster. Can you add a section to the `ObjectPageLayout` with the following product details?

Details

- **Gross Price:** 103.53 USD
- **Net Price:** 87.00 USD
- **Tax:** 16.53 USD
- **Quantity:** 10 EA

Preview

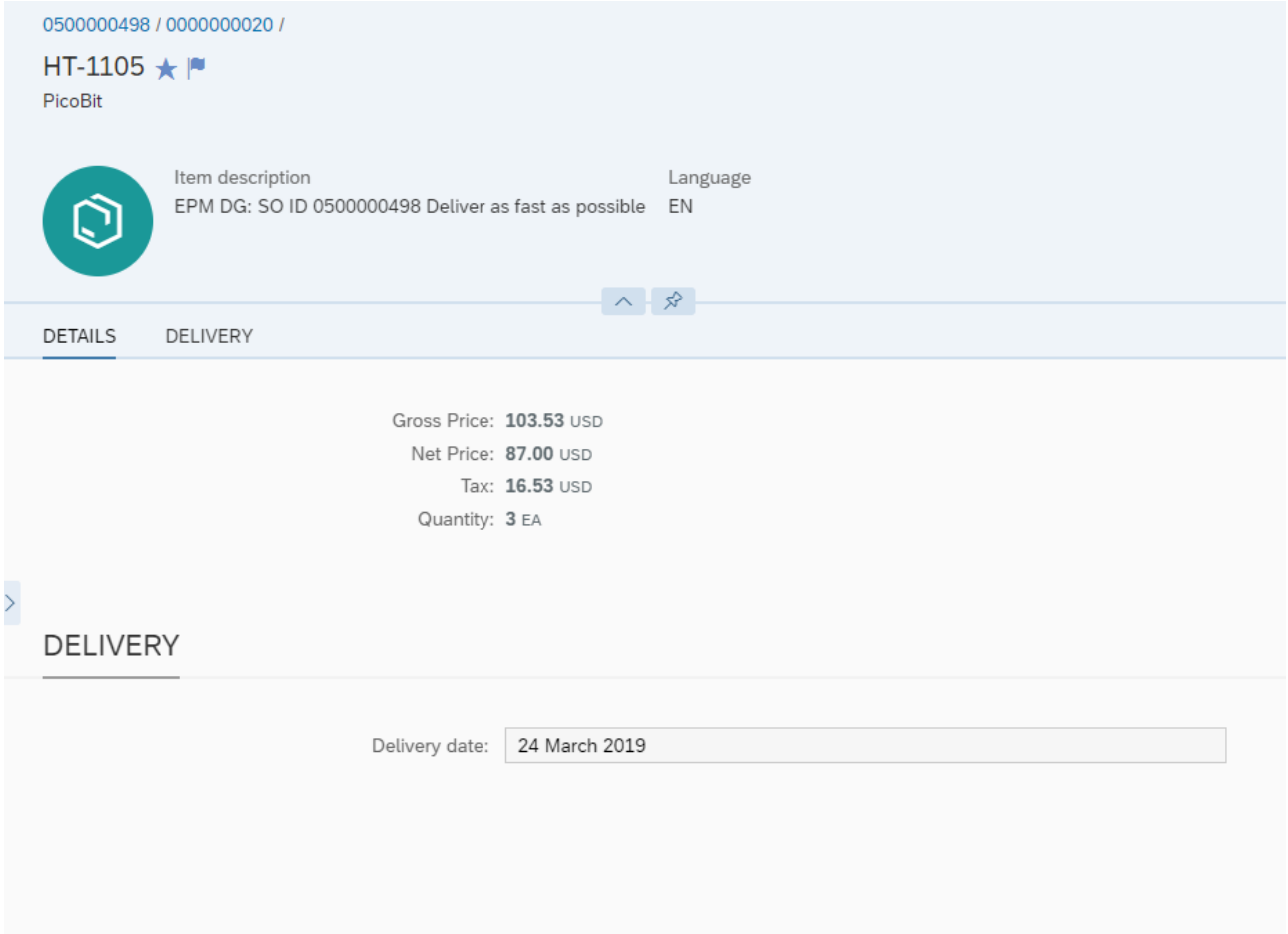

Figure 13 – The requested product details on the info page

Hints:

- The third column of the `FlexibleColumnLayout` shows details about a product
- In the service metadata you can see some product-related fields which are not used yet
- Add some controls to a new section of the `ObjectPageLayout`
- Especially, the `sap.m.ObjectNumber` control may be handy

RELATED MATERIAL

- Demo Kit: Object Page Layout
- Example: InfoLabel Control
- Demo Kit: StatusIndicator

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 production 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 cause by use of the Code, except where such damages were caused by SAP with intent or with gross negligence.

www.sap.com/contactsap

THE BEST RUN SAP®