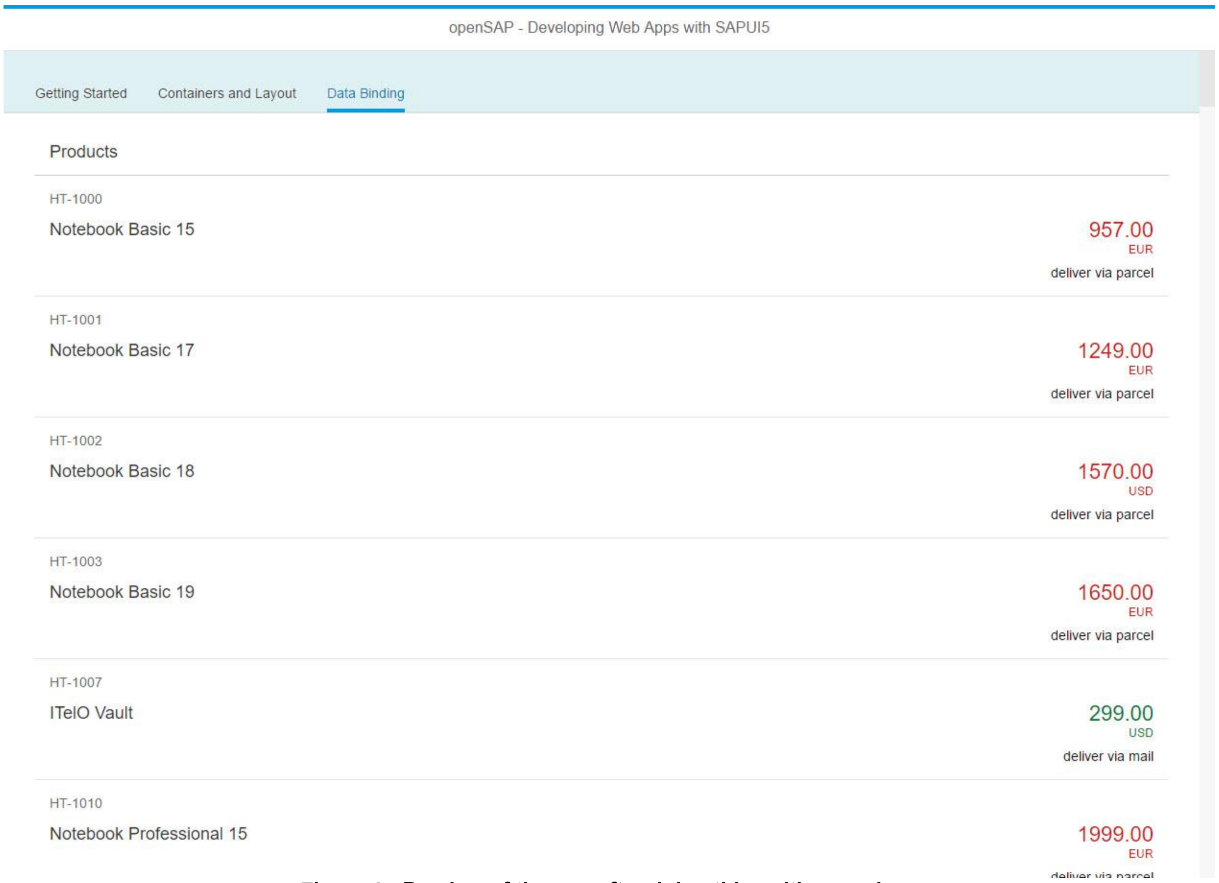
WEEK 2 UNIT 2
WORKING WITH EXPRESSIONS AND FORMATTERS

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

Table of Contents

Preview



openSAP - Developing Web Apps with SAPUI5

Products

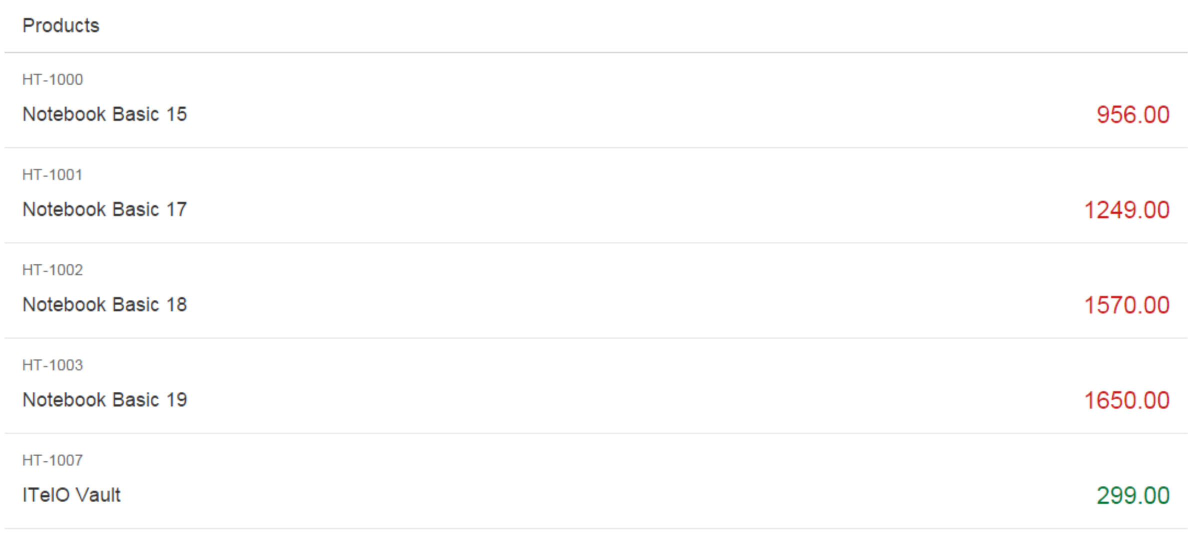
HT-1000 Notebook Basic 15	957.00 EUR deliver via parcel
HT-1001 Notebook Basic 17	1249.00 EUR deliver via parcel
HT-1002 Notebook Basic 18	1570.00 USD deliver via parcel
HT-1003 Notebook Basic 19	1650.00 EUR deliver via parcel
HT-1007 ITelO Vault	299.00 USD deliver via mail
HT-1010 Notebook Professional 15	1999.00 EUR

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

1 EXPRESSION BINDING

In this step, we use expression binding to add a status to the list items based on the data from the backend service.

Preview



Products

HT-1000
Notebook Basic 15 956.00

HT-1001
Notebook Basic 17 1249.00

HT-1002
Notebook Basic 18 1570.00

HT-1003
Notebook Basic 19 1650.00

HT-1007
ITelO Vault 299.00

Figure 2: Number state set via expression

webapp/view/App.view.xml

```
<mvc:View
…
  <IconTabFilter
  text="{i18n>dataBindingFilter}" key="db">
  <content>
     <List
        headerText="{i18n>productListTitle}"
        items="{/ProductSet}">
        <items>
           <ObjectListItem
              title="{Name}"
              number="{Price}"
              numberState="{= ${Price} > 500 ? 'Error' : 'Success'}"
              intro="{ProductID}">
           </ObjectListItem>
        </items>
     </List>
  </content>
</IconTabFilter>
…
</mvc:View>
```

We add the property `numberState` in our declarative view and introduce a new binding syntax that starts with = inside the brackets. This symbol is used to initiate a new binding syntax, it's called an expression and can do simple calculation logic like the ternary operator shown here.

The condition of the operator is a value from our data model. A model binding inside an expression binding has to be escaped with the `$` sign as you can see in the code. We set the state to `'Error'` (the number will appear in red) if the price is higher than 500 and to `'Success'` (the number will appear in green) otherwise.

Expressions are limited to a particular set of operations that help formatting the data such as `Math` expression, comparisons, and such. You can look up the possible operations in the documentation.

Conventions

- Only use expression binding for trivial calculations.

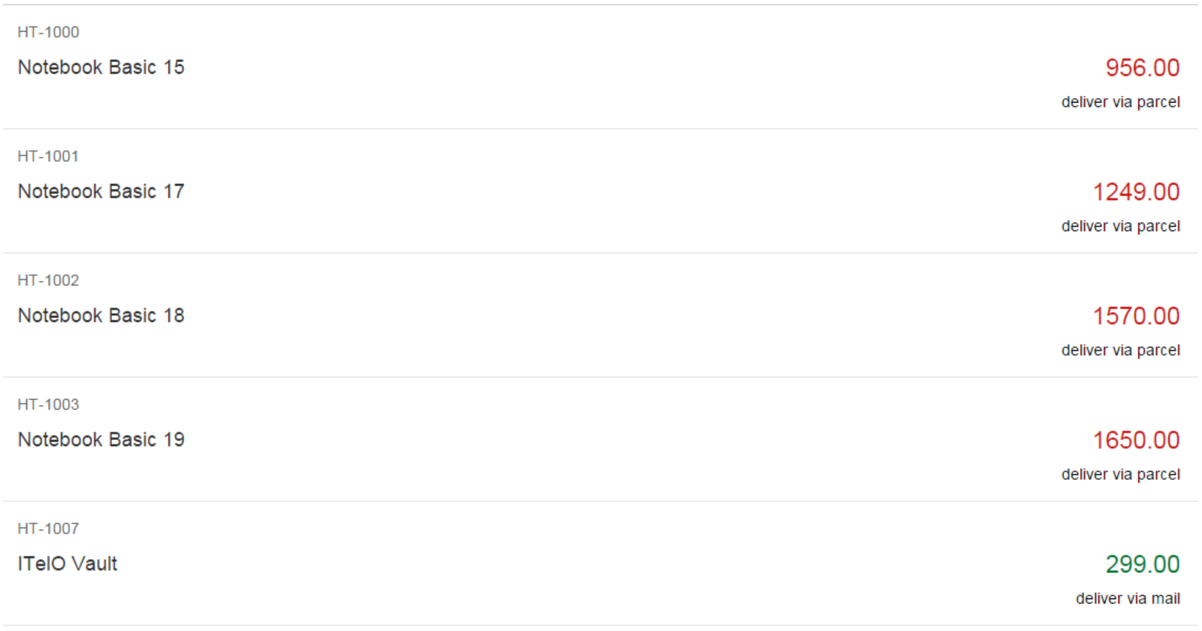
Related Information

Expression Binding

2 CUSTOM FORMATTER

In this step, we add a custom formatter to our list item that can perform more complex conversion and formatting of data.

Preview



HT-1000
Notebook Basic 15 956.00
deliver via parcel

HT-1001
Notebook Basic 17 1249.00
deliver via parcel

HT-1002
Notebook Basic 18 1570.00
deliver via parcel

HT-1003
Notebook Basic 19 1650.00
deliver via parcel

HT-1007
ITelO Vault 299.00
deliver via mail

Figure 3: Status added using a custom formatter

webapp/model/formatter.js (NEW)

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

 return {
   delivery: function(sMeasure, iWeight) {
      var oResourceBundle =
this.getView().getModel("i18n").getResourceBundle(),
         sResult = "";

      if(sMeasure === "G") {
         iWeight = iWeight / 1000;
      }
      if (iWeight < 0.5) {
         sResult = oResourceBundle.getText("formatterMailDelivery");
      } else if (iWeight < 5) {
         sResult = oResourceBundle.getText("formatterParcelDelivery");
      } else {
         sResult = oResourceBundle.getText("formatterCarrierDelivery");
      }

      return sResult;
   }
 };
});
```

We create a new file called `formatter.js` into the `model` folder and use the already introduced `sap.ui.define` syntax. This file returns an object with the `delivery` formatter function that can take several input parameters. We can pass them over from our view later. The formatter function we define takes two parameters and returns a text value from the i18n bundle based on the input parameters.

webapp/controller/App.controller.js

```
sap.ui.define([
  "sap/ui/core/mvc/Controller",
  "sap/m/MessageToast",
  "opensap/myapp/model/formatter"
], function (Controller, MessageToast, formatter) {
  "use strict";

  return Controller.extend("opensap.myapp.controller.App", {

     formatter : formatter,

     onShowHello : function () {
        …
     }
  });
});
```

To load our formatter functions, we have to add it to the app controller. This controller simply stores the loaded formatter functions in a local property `formatter` so that we can access them in the view. The `formatter` reference is loaded as an additional dependency in the `sap.ui.define` statement.

webapp/view/App.view.xml

```
<mvc:View …>
  <App>
     <pages>
        <Page …>
           …
                       <items>
                          <ObjectListItem …>
                             <firstStatus>
                                <ObjectStatus text="{
                                   parts: [
                                      {path: 'WeightUnit'},
                                      {path: 'WeightMeasure'}
                                   ],
                                   formatter : '.formatter.delivery'
                                }"/>
                             </firstStatus>
                          </ObjectListItem>
                       </items>
           …
        </Page>
     </pages>
  </App>
</mvc:View>
```

We add the `firstStatus` aggregation to our `ListItem` and use the `ObjectStatus` control to display the information we provide with our formatter function. For this we use the extended binding syntax with multiple parts. In parts we can pass in the paths to multiple entries in our model to a formatter function. In our case we use the `WeightUnit` and the `WeightMeasure` fields from the model as expected by the formatter function. A `"."`in front of the formatter name means that the function is looked up in the controller of the current view. There we defined a property `formatter` that holds our formatter functions, so we can access it by `.formatter.delivery`.

webapp/i18n/i18n.properties

```
# App Descriptor
…

# Data Binding Content
productListTitle=Products
formatterMailDelivery=deliver via mail
formatterParcelDelivery=deliver via parcel
formatterCarrierDelivery=deliver via carrier
```

In a last step we then have to simply add the translatable texts we use in the formatter function to the internationalization file of the application.

Related Information

Defining a Formatter