WEEK 3 UNIT 5
FRAGMENTS AND CODE REUSE

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

Table of Contents

Preview

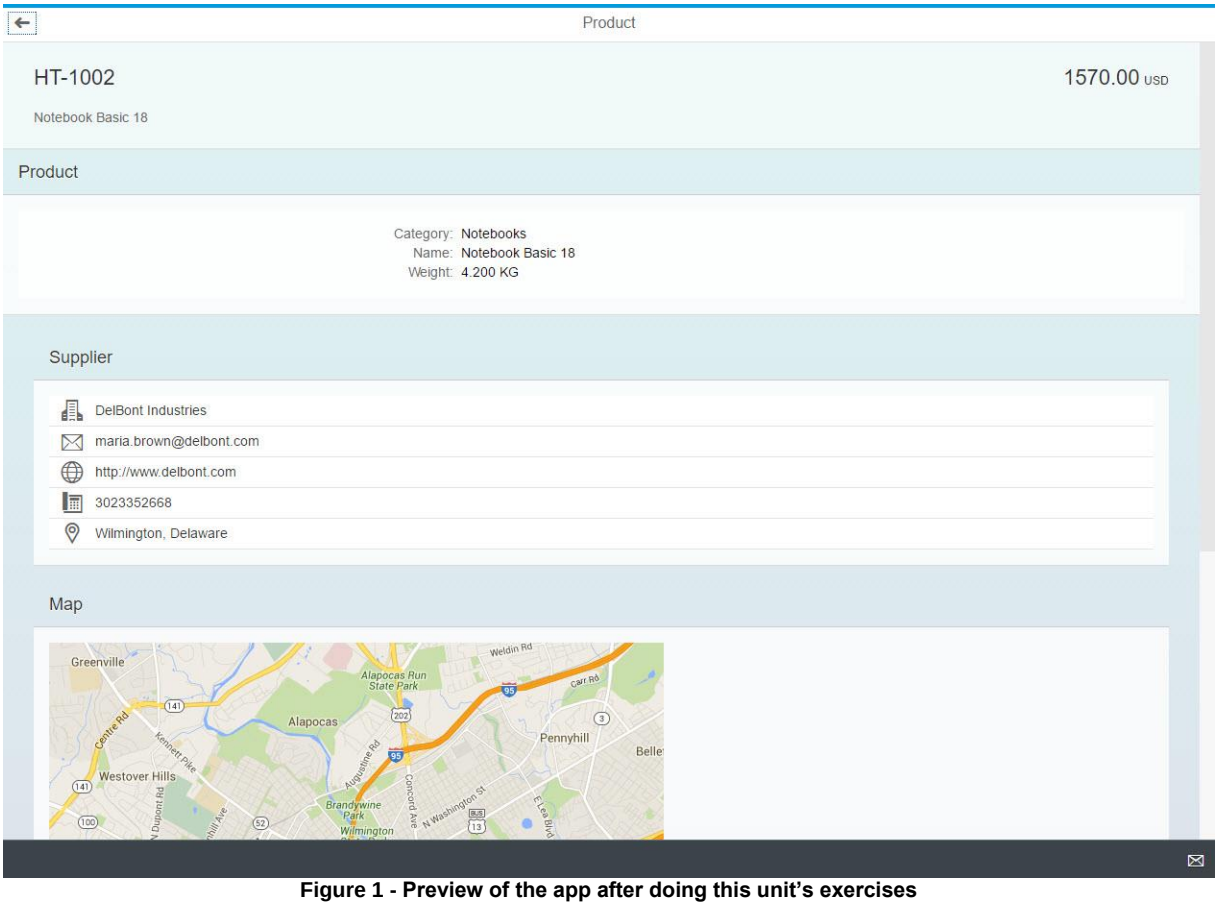

Figure 1 - Preview of the app after doing this unit's exercises

1 ADD A FRAGMENT

webapp/view/SupplierInfo.fragment.xml (NEW)

```
<core:FragmentDefinition
 xmlns="sap.m"
 xmlns:core="sap.ui.core"
 xmlns:form="sap.ui.layout.form">
 <Panel
    class="sapUiResponsiveMargin"
    width="auto"
    headerText="{i18n>supplierTitle}"
    expandable="{device>/system/phone}"
    expanded="false">
    <content>
       <List id="supplierList">
          <items>
             <StandardListItem icon="sap-icon://building"
title="{ToSupplier/CompanyName}"/>
             <StandardListItem icon="sap-icon://email"
title="{ToSupplier/EmailAddress}"/>
             <StandardListItem icon="sap-icon://world"
title="{ToSupplier/WebAddress}"/>
             <StandardListItem icon="sap-icon://phone"
title="{ToSupplier/PhoneNumber}"/>
             <StandardListItem icon="sap-icon://map"
title="{ToSupplier/Address/City}"/>
          </items>
       </List>
    </content>
 </Panel>

 <Panel
    class="sapUiResponsiveMargin sapUiHideOnPhone"
    width="auto"
    headerText="{i18n>mapTitle}">
    <Image src="{
       parts: [
          'ToSupplier/Address/Street',
          'ToSupplier/Address/PostalCode',
          'ToSupplier/Address/City',
          'ToSupplier/Address/Country'
       ],
       formatter: '.formatter.formatMapUrl'
    }" />
 </Panel>
</core:FragmentDefinition>
```

We are creating a fragment with the supplier info in the view directory of our app, simply **cutting out** two of our panels from the Object view we would like to reuse. We need to encapsulate these panels in a fragment definition, and also need to specify the corresponding namespaces, here. The Object view should not contain this part of the code, anymore.

We will next add the fragment to the Object view again, where we have taken the code from. Like this, the supplier information could easily be reused in different places/views of your application.

webapp/view/Object.view.xml

```
<mvc:View
 controllerName="opensap.manageproducts.controller.Object"
 xmlns="sap.m"
 xmlns:core="sap.ui.core"
 xmlns:mvc="sap.ui.core.mvc"
 xmlns:semantic="sap.m.semantic">
[...]
    <semantic:content>
      <ObjectHeader
        id="objectHeader"
        title="{ProductID}"
        responsive="true"
        number="{
          path: 'Price',
          formatter: '.formatter.numberUnit'
        }"
        numberUnit="{CurrencyCode}">
        <attributes>
          <ObjectAttribute text="{Name}"/>
        </attributes>
      </ObjectHeader>
      <Panel
        class="sapUiResponsiveMargin"
        width="auto"
        headerText="{i18n>productTitle}"
        expandable="{device>/system/phone}"
        expanded="true">
        <content>
          <form:SimpleForm id="objectForm">
            <form:content>
              <Label text="{i18n>productCategoryLabel}"/>
              <Text text="{Category}"/>
              <Label text="{i18n>productNameLabel}"/>
              <Text text="{Name}"/>
              <Label text="{i18n>productWeightLabel}"/>
              <Text text="{= ${WeightMeasure} + ' ' + ${WeightUnit}}"/>
            </form:content>
          </form:SimpleForm>
        </content>
      </Panel>
      <Panel
        class="sapUiResponsiveMargin"
        width="auto"
        headerText="{i18n>supplierTitle}"
        expandable="{device>/system/phone}"
        expanded="false">
        <content>
          <List id="supplierList">
            <items>
              <StandardListItem icon="sap-icon://building"
title="{ToSupplier/CompanyName}"/>
              <StandardListItem icon="sap-icon://email"
title="{ToSupplier/EmailAddress}"/>
              <StandardListItem class="sapUiVisibleOnlyOnDesktop"
icon="sap-icon://world" title="{ToSupplier/WebAddress}"/>
              <StandardListItem icon="sap-icon://phone"
title="{ToSupplier/PhoneNumber}"/>
              <StandardListItem icon="sap-icon://map"
title="{ToSupplier/Address/City}"/>
```

```
            </items>
        </List>
    </content>
</Panel>

<Panel
    class="sapUiResponsiveMargin sapUiHideOnPhone"
    width="auto"
    headerText="{i18n>mapTitle}">
    <Image src="{
        parts: [
            'ToSupplier/Address/Street',
            'ToSupplier/Address/PostalCode',
            'ToSupplier/Address/City',
            'ToSupplier/Address/Country'
        ],
        formatter: '.formatter.formatMapUrl'
    }" />
</Panel>
    <core:Fragment
fragmentName="opensap.manageproducts.view.SupplierInfo" type="XML"/>
</semantic:content>
```

When you re-run your application now, it should look just the same. The supplier information should be displayed in the same place in your Object view. Even the DOM of this view remains the same, which you can check in the developer tools. The fragment itself does not leave a trace, there.

2 CREATE A NESTED VIEW AND ITS CONTROLLER

webapp/view/ProductDetails.view.xml (NEW)

```
<mvc:View
 controllerName="opensap.manageproducts.controller.ProductDetails"
 xmlns="sap.m"
 xmlns:mvc="sap.ui.core.mvc"
 xmlns:form="sap.ui.layout.form">
 <Panel
    class="sapUiResponsiveMargin"
    width="auto"
    headerText="{i18n>productTitle}"
    expandable="{device>/system/phone}"
    expanded="true">
    <content>
      <form:SimpleForm id="objectForm">
        <form:content>
          <Label id="categoryLabel" text="{i18n>productCategoryLabel}"/>
          <Text id="category" text="{Category}"/>
          <Label text="{i18n>productNameLabel}"/>
          <Text text="{Name}"/>
          <Label text="{i18n>productWeightLabel}"/>
          <Text text="{= ${WeightMeasure} + ' ' + ${WeightUnit}}"/>
        </form:content>
      </form:SimpleForm>
    </content>
 </Panel>
</mvc:View>
```

We will now extract some more code out of our Object view and create a new view from it. Copy over the complete code from the panel containing the product data, and paste it into a new file in your view folder, which we will call ProductDetails.view.xml. Make sure you add the ids from the code above to the first Label and Text, which we will need for some code in our controller.
When you have pasted the code, don't forget to add the view definition which needs to wrap this code now. Make sure you also specify the correct controller name for the view, as will also create a new controller for it in a minute.

In the object view, you should now also include the new view, so the code will look like this:

webapp/view/Object.view.xml

```
…
    <semantic:content>
      <ObjectHeader
        id="objectHeader"
        title="{ProductID}"
        responsive="true"
        number="{
          path: 'Price',
          formatter: '.formatter.numberUnit'
        }"
        numberUnit="{CurrencyCode}">
        <attributes>
          <ObjectAttribute text="{Name}"/>
        </attributes>
      </ObjectHeader>
      <Panel
        class="sapUiResponsiveMargin"
        width="auto"
```

```
            headerText="{i18n>productTitle}"
            expandable="{device>/system/phone}"
            expanded="true">
            <content>
              <form:SimpleForm id="objectForm">
                <form:content>
                  <Label text="{i18n>productCategoryLabel}"/>
                  <Text text="{Category}"/>
                  <Label text="{i18n>productNameLabel}"/>
                  <Text text="{Name}"/>
                  <Label text="{i18n>productWeightLabel}"/>
                  <Text text="{= ${WeightMeasure} + ' ' + ${WeightUnit}}"/>
                </form:content>
              </form:SimpleForm>
            </content>
          </Panel>
          <mvc:XMLView viewName="opensap.manageproducts.view.ProductDetails"/>
          <core:Fragment
fragmentName="opensap.manageproducts.view.SupplierInfo" type="XML"/>
        </semantic:content>
…
```

What is still missing is the code for our controller.
Within the controller folder, we define the new controller, which will be called ProductDetails.controller.js

webapp/controller/ProductDetails.controller.js (NEW)

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

    return
Controller.extend("opensap.manageproducts.controller.ProductDetails", {

      formatter: formatter,

      onInit: function() {
        this.byId("categoryLabel").setVisible(false);
        this.byId("category").setVisible(false);
      }
    });

});
```

Within this controller, we just make an exemplary change to see that it is really used by our view.
The two controls from the nested view for which we have defined new ids should now be retrieved by these ids, and set to visible=false.

Running the app again will show nearly the same picture when you go to the Object view, with one difference: the label and text control for the product category should now be hidden.

Related Information

Reusing UI Parts: Fragments
Dialogs and other Popups as Fragments
API Reference: sap.ui.core.Fragment