HD3C08 – Tables

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  The Table control is versatile control that is fundamental to business applications. This case describes how to create responsive tables in SAPUI5.  **PREREQUISITES**  HD1sCO3 – The Base Application |
| **Target Audience**  Undergraduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

# The Table Control

Along with Lists, Tables are an essential element of business transactions.

## Create the Table view

Add a new object for the Tables view to the views.json file. You can find and icon [here](https://openui5.hana.ondemand.com/iconExplorer.html). Also, add a route to the Component.js file. Next, create two new files called **Tables.view.xml** and **Tables.controller.js.**

### Tables.view.xml

Copy the code below into the **Tables.view.xml** file.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.Tables"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m">  <Page  showNavButton = "true"  navButtonPress = "handleNavButtonPress"  title="Start Coding Tables!" >  <Table id="idProductsTable"  inset="false"  items="{gbi>/Products}">  <headerToolbar>  <Toolbar>  <Label text="Products"></Label>  </Toolbar>  </headerToolbar>  <columns>  <Column  width="12em">  <Text text="Product" />  </Column>  <Column>  <Text text="Product No." />  </Column>  <Column>  <Text text="Division" />  </Column>  </columns>  <items>  <ColumnListItem>  <cells>  <Text  text="{gbi>ProductName}" />  </cells>  <cells>  <Text  text="{gbi>ID}" />  </cells>  <cells>  <Text  text="{gbi>Division}" />  </cells>  </ColumnListItem>  </items>  </Table>    </Page>  </mvc:View> |

Listing 1

The Table control has three sections in this code: headerToolbar, columns, and Items. The headerToolbar is self-explanatory. You can add Labels as is done in this code but you can also add other controls such as Input and Button controls.

The columns section is used to define the columns of the table. This includes styling information such as the width as well as column headings.

The items section is used to define the rows of the table. There are different controls you can use to define the rows with different formatting options. In this case, a ColumnListItem control is used. The table columns are defined by cells in the ColumnListItem control. In this code, the properties are shown using Text controls but you can use other types of controls as well. For example, you can include form controls like Input and Checkbox controls.

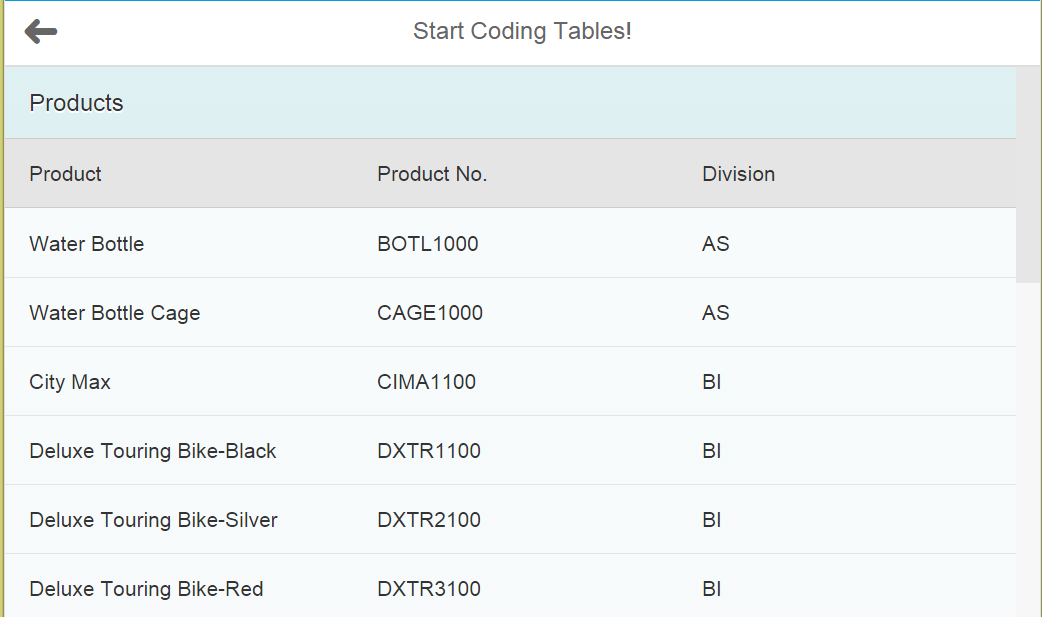
### Tables.controller.js

Copy the code shown below into the **Tables.controller.js** file:

|  |
| --- |
| onInit : function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  },  handleNavButtonPress: function(){  this.router.navTo("Master", {  from: "Tables"  });  } |

Listing 2

Select **index.html,** click on the run button  to run the application. You should have a Tables tile on the first page. When you click it, you will see a page with a table like the one shown below.



## Some Formatting Options

There are a variety of options you can use to format the table:

|  |  |
| --- | --- |
| Option |  |
| minScreenWidth | This sets the minimum screen width that a column will be visible. You can use CSS sizes or you can use the sap.m.ScreenSize enumeration(e.g: "Phone", "Tablet", "Desktop", "Small", "Medium", "Large", ....). If you set this to Tablet then the column would be visible on a tablet but not a phone. |
| demandPopin | Works in conjunction with minScreenWidth. Setting this property to true, shows this column as pop-in instead of hiding it. A pop-in is moved below the values in the first column on the same row. |
| hAlign | This determines how the values are aligned horizontally in the column. The possible values are Left, Center and Right. |
| width | Sets the width of the column in CSS units. |

The minScreenWidth and demandPopin are used to create responsive tables that adjust to the size of the screen. Let’s say we want to ensure that the Product and Inventory Qty columns don’t change regardless of the size of the screen. However, we want the Division column to be hidden on screens smaller than a tablet and for the Product Category and Price columns to appear below Product on small screens.

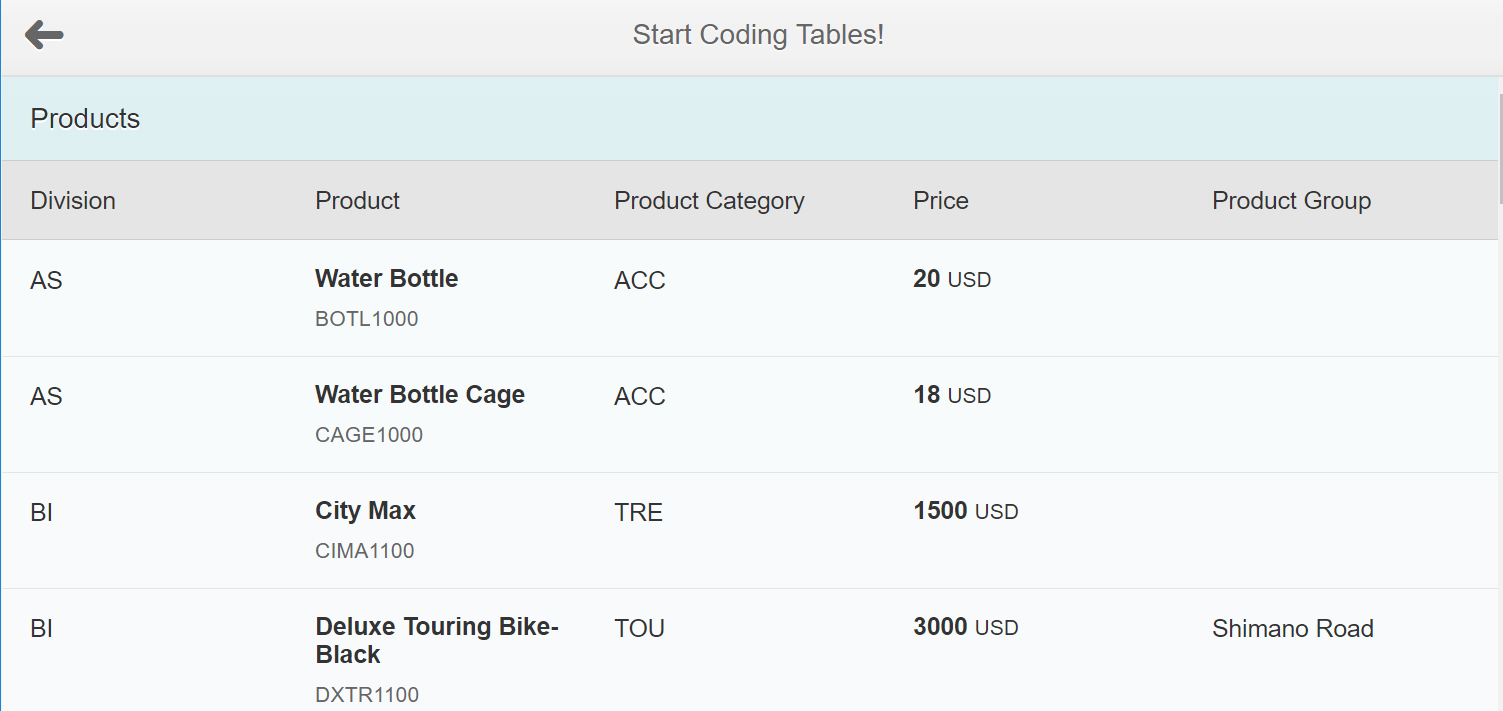
### Tables.view.xml

Adjust the columns and items definitions as shown below in **Tables.view.xml**:

|  |
| --- |
| <columns>  <Column>  <header>  <Text text="Product" />  </header>  </Column>  <Column  minScreenWidth="Tablet">  <header>  <Text text="Division" />  </header>  </Column>  <Column  minScreenWidth="Tablet"  demandPopin='true'>  <header>  <Text text="Product Category" />  </header>  </Column>  <Column  minScreenWidth="Tablet"  demandPopin='true'>  <header>  <Text text="Price" />  </header>  </Column>  <Column>  <header>  <Text text="ProductGroup" />  </header>  </Column>  </columns>  <items>  <ColumnListItem>  <cells>  <ObjectIdentifier  title="{gbi>ProductName}"  text="{gbi>ID}" />  <Text  text="{gbi>Division}" />  <Text  text="{gbi>ProductCategory}" />  <ObjectNumber  number="{gbi>Price}"  unit="USD" />  <Text  text="{gbi>ProductGroup}" />  </cells>  </ColumnListItem>  </items> |

Listing 3

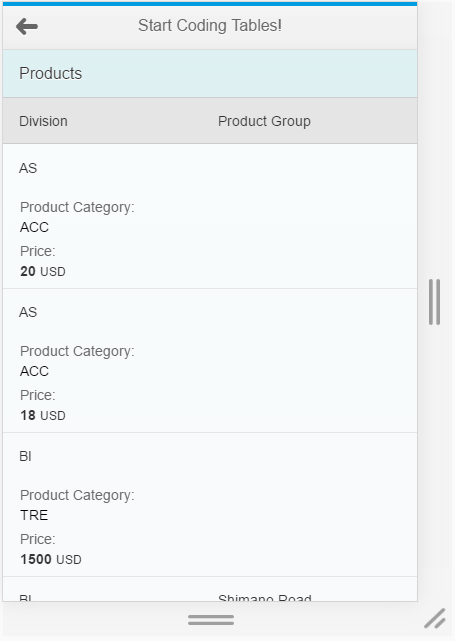
This code adds a number of columns and uses the [ObjectIdentifier](https://sapui5.netweaver.ondemand.com/sdk/explored.html#/entity/sap.m.ObjectIdentifier/samples) and [ObjectNumber](https://sapui5.netweaver.ondemand.com/sdk/explored.html#/entity/sap.m.ObjectNumber/samples) objects to format the data. The table appears like the image below on a large screen:



When you reduce the screen size, the table adjusts. You can adjust the screen size by using Run in Application Preview or, if you are using Chrome, you can open the Developer Tools (right-click the browser window and select Inspect Element) and then click the mobile phone emulator that appears to the left on the toolbar:

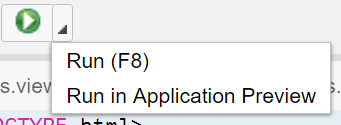


When you choose a phone emulator the Division column is hidden and the Product Category and Price appear below the Product information. Note that when you change the type of emulator you may have to refresh the browser.

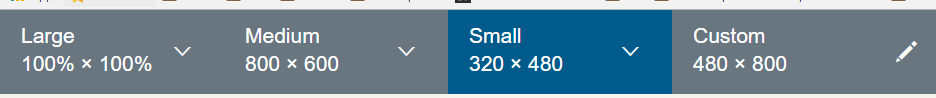


You can use the dropdown list at the top of the screen to try different devices:

You can also use the Application Preview to try different display options.



You can adjust the screen size:



You can also change the orientation or use the QR code to run the application on a physical device.



## Grouping Records Using Merged Cells

It’s relatively easy to group rows on a column.

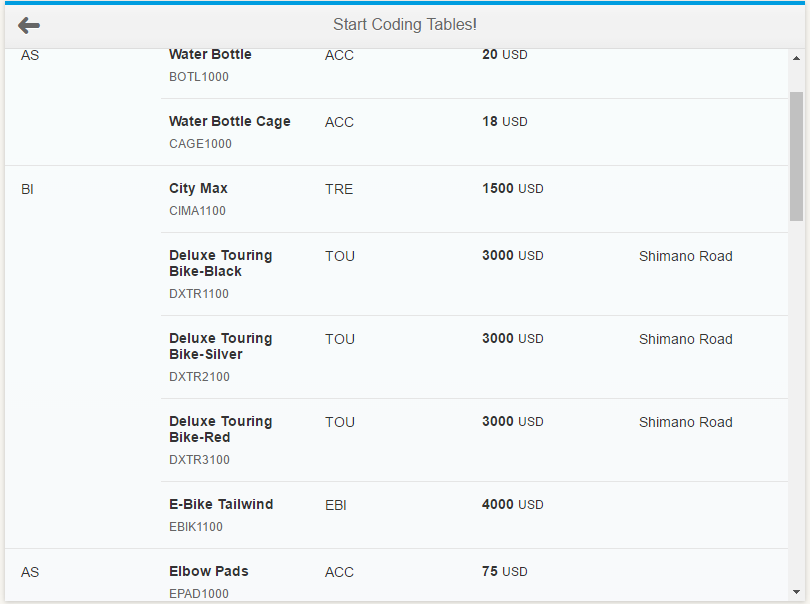
### Tables.view.xml

To create a grouping column on Product Category modify the columns definition in **Tables.view.xml** as shown below. The Product Category is moved to the first column and the mergeDuplicates=”true” attribute is added to the Column element.

|  |
| --- |
| <columns>  <Column>  <header>  <Text text="Division" />  </header>  </Column>  <Column  minScreenWidth="Tablet">  <header>  <Text text="Product" />  </header>  </Column>  <Column  minScreenWidth="Tablet"  demandPopin='true'>  <header>  <Text text="Product Category" />  </header>  </Column>  <Column  minScreenWidth="Tablet"  demandPopin='true'>  <header>  <Text text="Price" />  </header>  </Column>  <Column>  <header>  <Text text="ProductGroup" />  </header>  </Column>  </columns>  <items>  <ColumnListItem>  <cells>  <Text  text="{gbi>Division}" />  <ObjectIdentifier  title="{gbi>ProductName}"  text="{gbi>ID}" />  <Text  text="{gbi>ProductCategory}" />  <ObjectNumber  number="{gbi>Price}"  unit="USD" />  <Text  text="{gbi>ProductGroup}" />  </cells>  </ColumnListItem>  </items> |

Listing 4

The table is now grouped on the product category column:



## View Settings

In this section we will add a dialog box which will provide the user with some options for manipulating the table. Make the following changes.

### Tables.view.xml

Replace the headerToolbar in the Tables.view.xml file with the code shown below:

|  |
| --- |
| <headerToolbar>  <Toolbar>  <Label text=*"Products"*></Label>  <ToolbarSpacer></ToolbarSpacer>  <Button icon=*"sap-icon://drop-down-list"* press=*"handleViewSettingsDialogButtonPressed"* />  </Toolbar>  </headerToolbar>  <infoToolbar>  <Toolbar  id=*"vsdFilterBar"*  active=*"true"*  visible=*"false"* press=*"[handleViewSettingsDialogFilterBarPressed, views.control.tableViewSettingsDialog]"*>  <Label id=*"vsdFilterLabel"* text=*"?"* />  </Toolbar>  </infoToolbar> |

Listing 5

The file should like this:



### Tables.controller.js

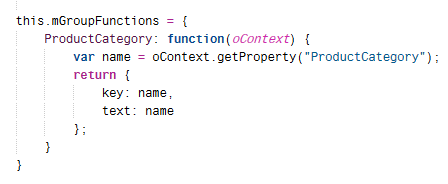
Replace all the code with the code shown below:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("ui5.controller.Tables", {  **onInit: function() {**  **this.router = sap.ui.core.UIComponent.getRouterFor(this);**  **this.mGroupFunctions = {**  **ProductCategory: function(oContext) {**  **var name = oContext.getProperty("ProductCategory");**  **return {**  **key: name,**  **text: name**  **};**  **}**  **};**  **},**    **handleNavButtonPress: function() {**  **this.router.navTo("Master", {**  **from: "Tables"**  **});**  **},**  **onExit: function() {**  **//Destroy the dialog when the user leaves the view**  **if (this.\_oDialog) {**  **this.\_oDialog.destroy();**  **}**  **},**  **handleViewSettingsDialogButtonPressed: function() {**  **//If the dialog hasn't been created, create it**  **if (!this.\_oDialog) {**  **this.\_oDialog = sap.ui.xmlfragment("ui5.view.Dialog", this);**  **}**  **// Synchronize the compact style between the view and the dialog**  **jQuery.sap.syncStyleClass("sapUiSizeCompact", this.getView(), this.\_oDialog);**  **this.\_oDialog.open();**  **},**  **handleConfirm: function(oEvent) {**  **//When the user clicks OK on the dialog, get references to aggregation binding of the table**  **var oBinding = this.getView().byId("idProductsTable").getBinding("items");**  **//The mParams property contains the options the user selected**  **var mParams = oEvent.getParameters();**  **//The aSorters array will contain all the properties the use selected to sort the table**  **var aSorters = [];**  **if (mParams.groupItem) {**  **var gPath = mParams.groupItem.getKey();**  **var gDescending = mParams.groupDescending;**  **var vGroup = this.mGroupFunctions[gPath];**  **aSorters.push(new sap.ui.model.Sorter(gPath, gDescending, vGroup));**  **}**  **var sPath = mParams.sortItem.getKey();**  **var bDescending = mParams.sortDescending;**  **aSorters.push(new sap.ui.model.Sorter(sPath, bDescending));**  **oBinding.sort(aSorters);**  **}**  });  }); |

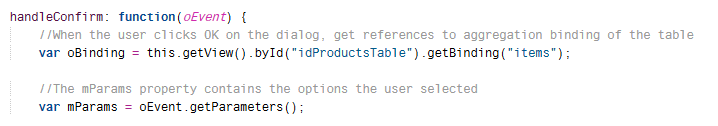
Listing 6

The handleViewSettingsDialogButtonPressed function handles opening the dialog. It checks to see if the Dialog has been instantiated before (previously opened) and then instantiated it if necessary and then opens the dialog.

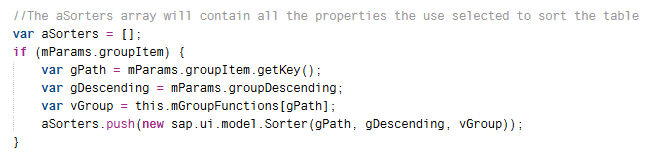
In the onInit function a function is defined that will be used to help group the data.



The handleConform function processes the results of the dialog. The code retrieves reference to the table’s aggregation binding. The sorting and grouping are handled by adding the sorting and grouping objects to the binding. The user’s choices are included in the oEvent object passed into the handler.

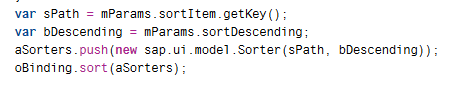


The next part processes the grouping options.



The groupItem property contains the field the user selected to group the data on. If mParams.groupItem exists then the user selected to group on a field. The code inside the if statement retrieves the path to the property in the model of the field to group on (gPath), creates a variable that defines the way data will be sorted when grouped (bDescending), creates an object to handle the grouping (vGroup) then pushes a Sorter object with the grouping parameters into the aSorters array. Grouping is a subset of sorting because the records must be sorted to be grouped.

The final part of the function handles the sorting options selected by the user.



First, the path to the sorting property is found (sPath), the direction of sorting is found (bDescending) and then a object that defines the sorting is pushed into the aSorters array. The last statement applies the Sorting objects in the aSorters array to the binding.

### Dialog.fragment.xml

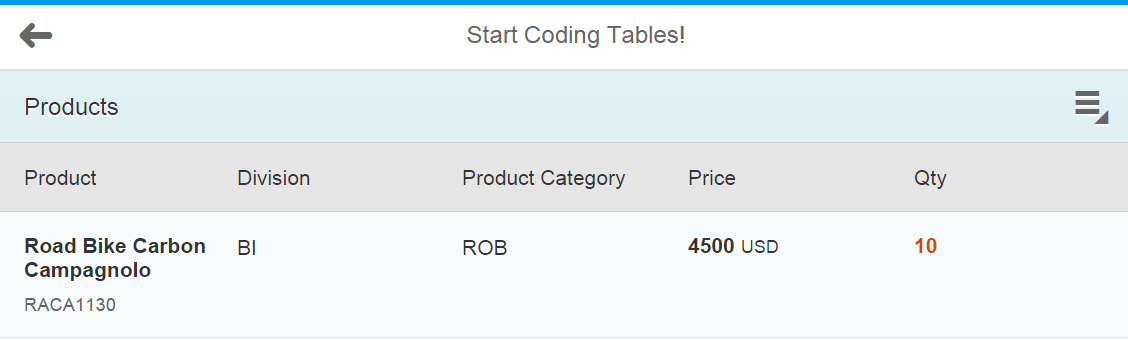
Create a new file in the **view** folder called **Dialog.fragment.xml** and insert the code shown below:

|  |
| --- |
| <core:FragmentDefinition  xmlns="sap.m"  xmlns:core="sap.ui.core">  <ViewSettingsDialog confirm="handleConfirm">  <sortItems>  <ViewSettingsItem text="Product" key="ProductName" selected="true" />  <ViewSettingsItem text="Product Category" key="ProductCategory" />  <ViewSettingsItem text="Material Number" key="ID" />  <ViewSettingsItem text="Price" key="Price" />  </sortItems>  <groupItems>  <ViewSettingsItem text="Product Category" key="ProductCategory" />  </groupItems>  </ViewSettingsDialog>  </core:FragmentDefinition> |

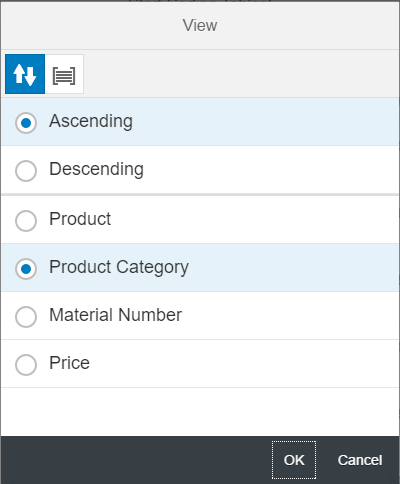
Listing 7

This code creates a dialog box and formats it to provide options for sorting and grouping the data in the table.

When you run the application you will see an icon in the top right part of the header.

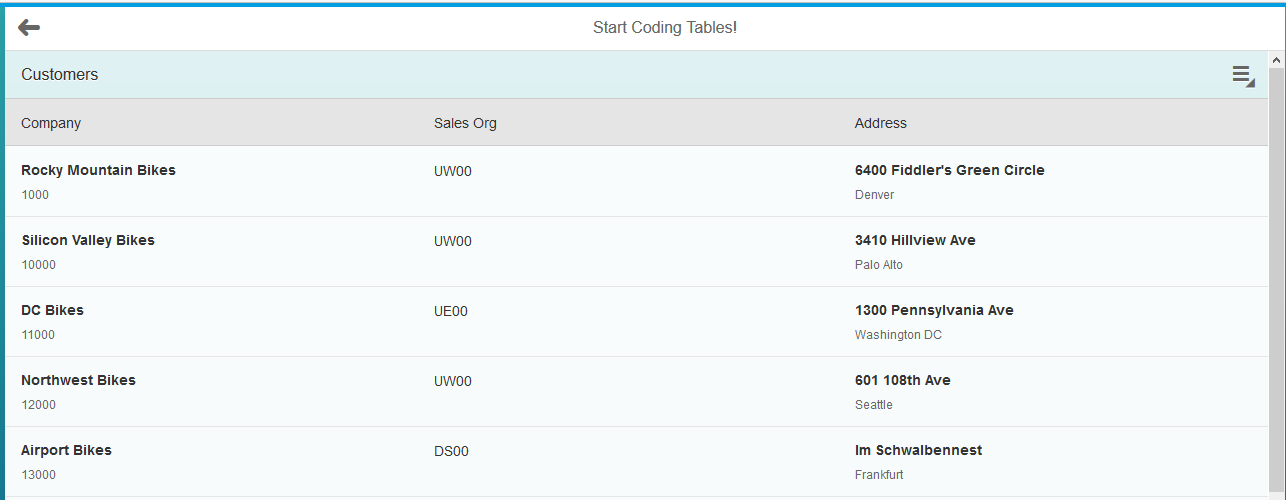


Clicking it will open a dialog. Try a few of the options.

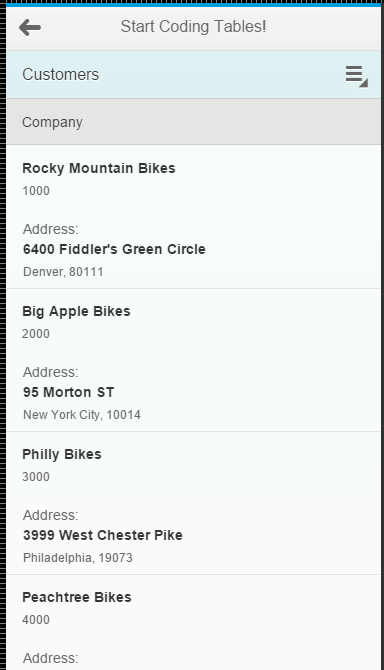


# Exercise

Create a view that shows a table of customers. The table should look similar to the image below when the screen size is tablet size or larger:



For small screens the Sales Org column should be hidden and the Address column should appear below the Company information.



You should also be able to sort and group:

|  |  |
| --- | --- |
|  |  |