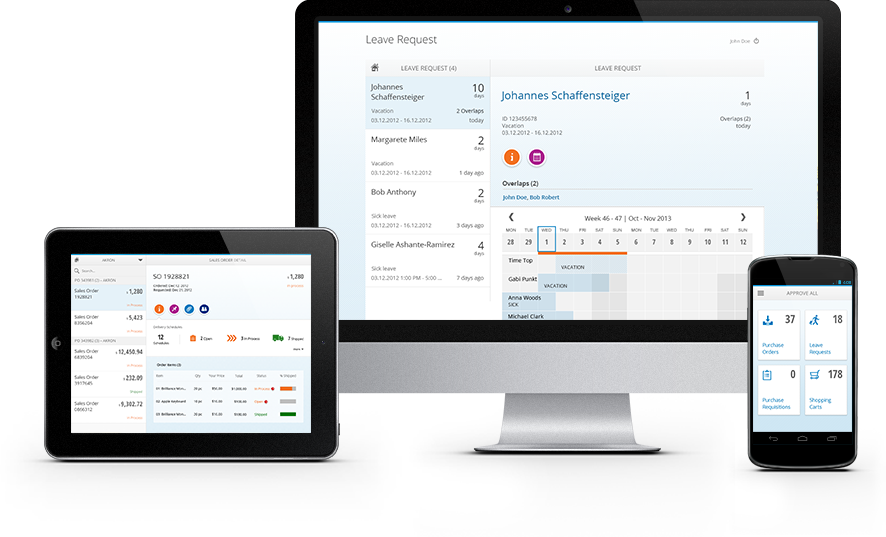
HD3C06 – The Base Application

|  |  |
| --- | --- |
| **Product**  HANA Platform  **Level**  Undergraduate/Graduate  Beginner  **Focus**  Application Development  SAPUI5  **Authors** Ross Hightower  Leigh Jin | MOTIVATION  This case describes creating the base application which is used in all of the SAPUI5 techniques cases using the Web Development Workbench.  **PREREQUISITES**  HD3C01 –Hello World  HD3C03 – Data Binding  HD3C04 - Routing |

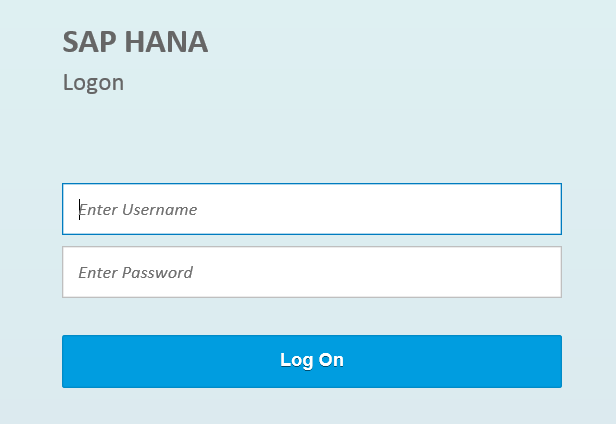


# The Base Application

This case describes a base application that serves as a Launchpad for most of the cases in the SAPUI5 skills series. The application has a main page with tiles which can be pressed to navigate to the views created in the cases.

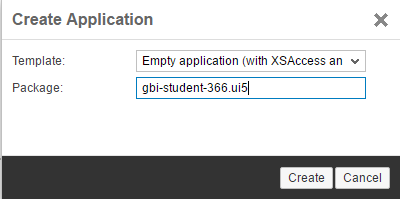
# Create the Application

Navigate to the Web-Based Development Workbench using the URL provided by your UCC and logon using the credentials provided by your UCC.

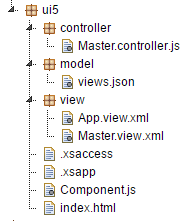


Open the **Editor.**

Right-click your package and select **Create Application,** choose **Empty application (with XSAccess and XSApp)**, and add **.UI5** to the Package then click **Create.**



Create three more packages under **UI5** package named **model, controller** and **view**. Create five new files named **Master.controller.js**, **Master.view.xml**, **App.view.xml, views.json** and **Component.js**. The project structure should looks similar to the following:



### Index.html

Open **index.html** file and modify its code as shown below. This is the standard SAPUI5 application:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge" />  <meta charset="UTF-8">    <title>UA SAPUI5 Skills App</title>  <script id="sap-ui-bootstrap"  src="/sap/ui5/1/resources/sap-ui-core.js"  data-sap-ui-theme="sap\_bluecrystal"  data-sap-ui-libs="sap.m"  data-sap-ui-xx-bindingSyntax="complex"  data-sap-ui-resourceroots='{  "ui5": "./"  }'>  </script>  <script>  // now create a new, reusable component called ui5 (like our namespace)  sap.ui.getCore().attachInit(function() {  new sap.m.Shell("shell",{  app : new sap.ui.core.ComponentContainer({  height : "100%",  name : "ui5"  })  }).placeAt("content"); });  </script>  </head>  <body class="sapUiBody" id="content">  </body>  </html> |

Listing

An explanation of this code can be found in case HD1C01 – The Hello World MVC Application.

### Component.js

Enter the code in the **Component.js** as shown below:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent",  "sap/ui/model/odata/ODataModel"  ], function(UIComponent, ODataModel) {  "use strict";  return UIComponent.extend("routing.Component", {  metadata: {  rootView: "routing.view.App",  routing: {  config: {  viewType: "XML",  viewPath: "routing.view",  transition: "slide",  clearTarget: true,  targetControl: "idAppControl"  },  routes: [  {  pattern: "",  name: "Master",  view: "Master",  targetAggregation: "pages"  }  ]  }  },  init: function() {  UIComponent.prototype.init.apply(this, arguments);  this.getRouter().initialize();  var oModel = new ODataModel(**MODEL URL**);  this.setModel(oModel,'gbi');  }  });  }); |

Listing

Replace MODEL URL with either:

1. If you completed the persistence model exercise (HD3C05) use the URL of your OData services.
2. If your instructor is supplying the URL

### Add a Route to the Master View

The default view is called Master.view.xml. Add a route to the Component.js file to load this view when the application loads. Refer to HD2C04 - Routing in SAPUI5 Applications to see how to do this.

### Add a Model

Now add a statement to add a model that references the OData services you created in HD2dC02w – Creating the Persistence Model. Refer to HD2C03 – Data Binding for how to create the statement and where to put it. The code to add an OData model is:

new sap.ui.model.odata.ODataModel(“”)

You identify the model using the service document URL that you can get when your run the gbi.xsodata file. Use the name gbi for the model.

### The App View

Right-click the **view** package and create file called **App.view.xml.**

#### App.view.xml

Insert the following code in **App.view.xml:**

|  |
| --- |
| <mvc:View  xmlns:mvc="sap.ui.core.mvc"  displayBlock="true"  xmlns="sap.m" >  <App id="idAppControl" />  </mvc:View> |

Listing

The only thing this view does is provide the App control into which views will be loaded by the router.

### The Master View

Create a file in the view package called **Master.view.xml.** The project structure now looks like:

#### Master.view.xml

Insert the following code in **Master.view.xml:**

|  |
| --- |
| <mvc:View height="100%" controllerName="ui5.controller.Master" xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc" xmlns:commons="sap.suite.ui.commons">  <Page enableScrolling="false" title="Start Coding!">    <customHeader>  <Toolbar>  <Image src="http://go.sap.com/dam/application/shared/logos/sap-logo.png.adapt.72\_36.false.png" />  <ToolbarSpacer />  <Label text="Let's Start Coding!" />  <ToolbarSpacer />  </Toolbar>  </customHeader>    <TileContainer id="caseTiles" height="50%" tiles="{ }">  <StandardTile press="handleTilePress" icon="sap-icon://{views>icon}"  type="{views>type}" number="{views>number}" numberUnit="{views>numberUnit}" title="{views>title}"  info="{views>info}" infoState="{views>infoState}" />  </TileContainer>    <TileContainer id="exerciseTiles" height="50%" tiles="{ }">  <StandardTile press="handleTilePress" icon="sap-icon://{views>icon}"  type="{views>type}" number="{views>number}" numberUnit="{views>numberUnit}" title="{views>title}"  info="{views>info}" infoState="{views>infoState}" />  </TileContainer>  </Page>  </mvc:View> |

Listing

This is an XML view that creates two TileContainers and define StandardTile controls as templates for the tiles that will created. It will be the Launchpad for the views you create in this series of cases.

The tiles property of the caseTiles TileContainer will be bound to the ViewCollection in the view model which we will define in the view’s controller (you will have add the data bindings later in the case). This is an example of an aggregation binding, a binding to a collection of data elements. SAPUI5 will create a tile for each item in the collection using the StandardTile control as the template. When you complete cases in this series, you will add them to the launchpad by adding an object to the views.json file.

#### Master.controller.js

Create a file in the controller package called **Master.controller.js** Insert the following code into Master.controller.js:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("ui5.controller.Master", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  },  handleTilePress: function(evt) {  var oView = evt.getSource().getInfo();  this.router.navTo(oView, {  from: "Master"  });  }  });  }); |

Listing

The onInit function is called when the view is first instantiated. It retrieves a reference to the application’s router which we can use to navigate to other views.

This code adds a click handler, handleTilePress, which is called when a tile is pressed. The press property of the StandardTile template points to this method. The first line retrieves the info property of the tile which we will set to the name of the view to which we want to navigate. It then invokes the router, passing the destination view name.

#### views.json

Insert the following code in **views.json:**

|  |
| --- |
| {  "ViewsCollection" : [  {  "icon" : "play",  "number" : "1",  "title" : "Resources",  "info" : "Resources"  }  ]  } |

Listing

This code defines a collection called ViewsCollection in JSON format. The file will be used to add tiles when completing later cases in this series.

### Master.controller.js

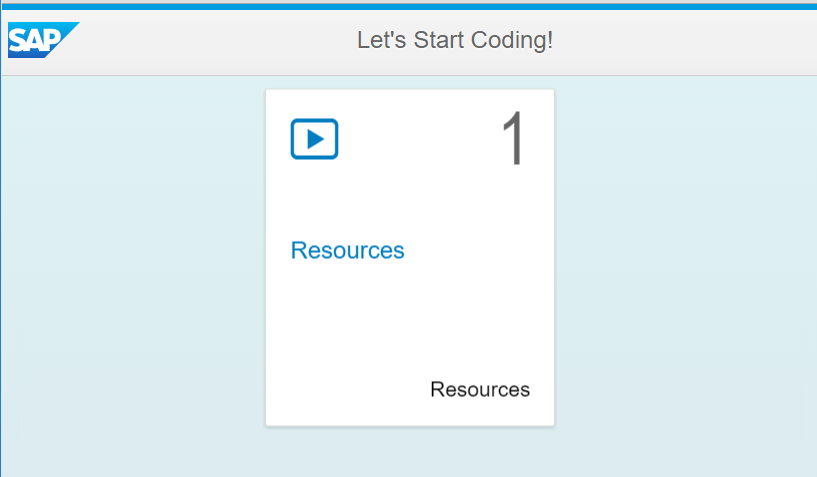
Now add code to the onInit function in the Master controller that creates a model that refers to the views.json file. By creating the model in the view’s controller, the model is only available to the view. The only difference in the syntax when assigning a model in a view is that you add the function getView() to the assignment code:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller**",**  **"sap/ui/model/json/JSONModel"**  ], function(Controller**,JSONModel**) {  "use strict";  return Controller.extend("ui5.controller.Master", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);    **var oModel = new JSONModel("model/views.json");**  **this.getView().setModel(oModel,"views");**  },  handleTilePress: function(evt) {  var oView = evt.getSource().getInfo();  this.router.navTo(oView, {  from: "Master"  });  }  });  }); |

Listing

### Bind the TileContainer Controls to the views Model

One last thing to do is to bind the first TileContainer control in the Master view to the ViewCollection collection in the views model. If you run the application now it should look like this:



If you click the tile, nothing happens so let’s add the Resources view next.

### Add the Resources View

#### Resources.view.xml

Create a file in the view package called **Resources.view.xml.** Paste the code shown below into it.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.Resources"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m">  <Page  title="Resources on the Web"  showNavButton = "true"  navButtonPress="handleNavButtonPress">    <HBox fitContainer="true" justifyContent="Center" alignItems="Center" width="60%" class="gridMarginTop">  <List headerText="SAPUI5 Resources" items="{resources>/ResourceCollection}">  <ObjectListItem  type="Active"  press="handleListPress"  title = "{resources>title}">  <attributes>  <ObjectAttribute text="{resources>description}" />  </attributes>  </ObjectListItem>  </List>  </HBox>    </Page>  </mvc:View> |

Listing

This code creates a List control bound to the ResourceCollection that we will create below. It will create a list item defined by the ObjectListItem template for each object in ResourceCollection.

#### Resources.controller.js

Create a file in the controller package called **Resources.controller.js.** Paste the code shown below into it.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/ui/model/json/JSONModel"  ], function(Controller, JSONModel) {  "use strict";  return Controller.extend("ui5.controller.Resources", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  var iModel = new JSONModel('model/resources.json');  this.getView().setModel(iModel, 'resources');  },  handleNavButtonPress: function() {  this.router.navTo("Master", {  from: "Resources"  });  },  handleListPress: function(evt) {  var url = evt.getSource().getBindingContext('resources').getProperty('url');  window.open(url, '\_blank');  }  });  }); |

Listing

A model is created for the view in the onInit function that references the resources.json file we will add to the model package.

The handleNavButtonPress is invoked when the user clicks the back navigation button. The function uses the router to navigate back to the Master view.

The handlListPress function is invoked when the user taps an item on the list. The function evt.getSource() retrieves a reference to the item that was tapped. The function getBindingContext() retrieves an object that contains the data bound to the tapped item and the getProperty() function retrieves the value of the URL property. The last line opens a new browser tab using the URL.

#### resources.json

Create a file called **resources.json** in the models package and insert the code shown below.

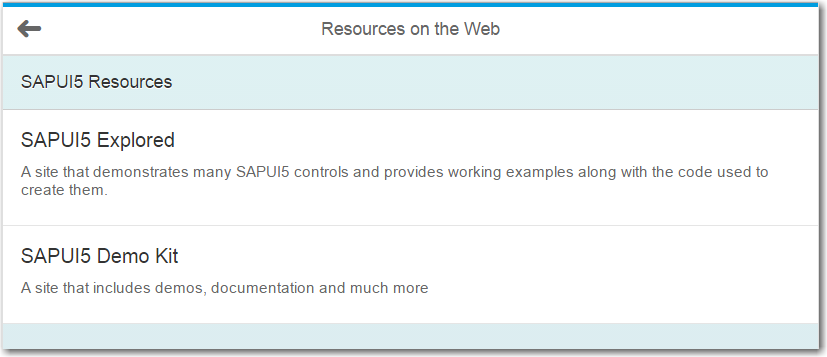
|  |
| --- |
| {  "ResourceCollection" : [  {  "title": "SAPUI5 Explored",  "description": "A site that demonstrates many SAPUI5 controls and provides working examples along with the code used to create them.",  "url": "https://sapui5.hana.ondemand.com/sdk/explored.html"    },  {  "title": "SAPUI5 Demo Kit",  "description": "A site that includes demos, documentation and much more",  "url": "https://sapui5.hana.ondemand.com/sdk/#content/Overview.html"  }    ]  } |

Listing

#### Add a Route to the Resources in Component.js

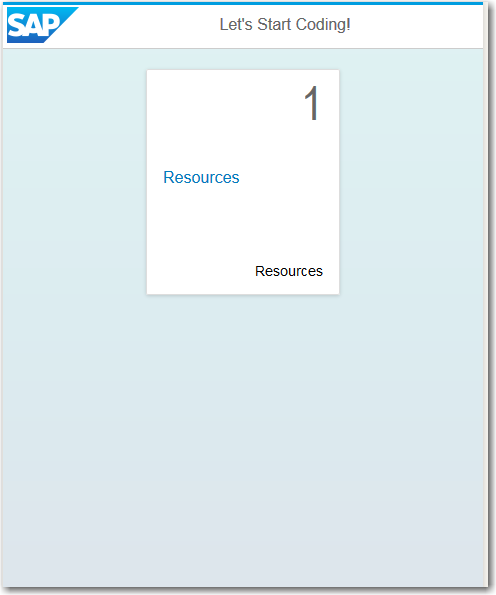
Now add a route to Component.js that will enable navigation to the Resources view.

Now when you run the app, you can click on the tile to navigate to the Resources view.



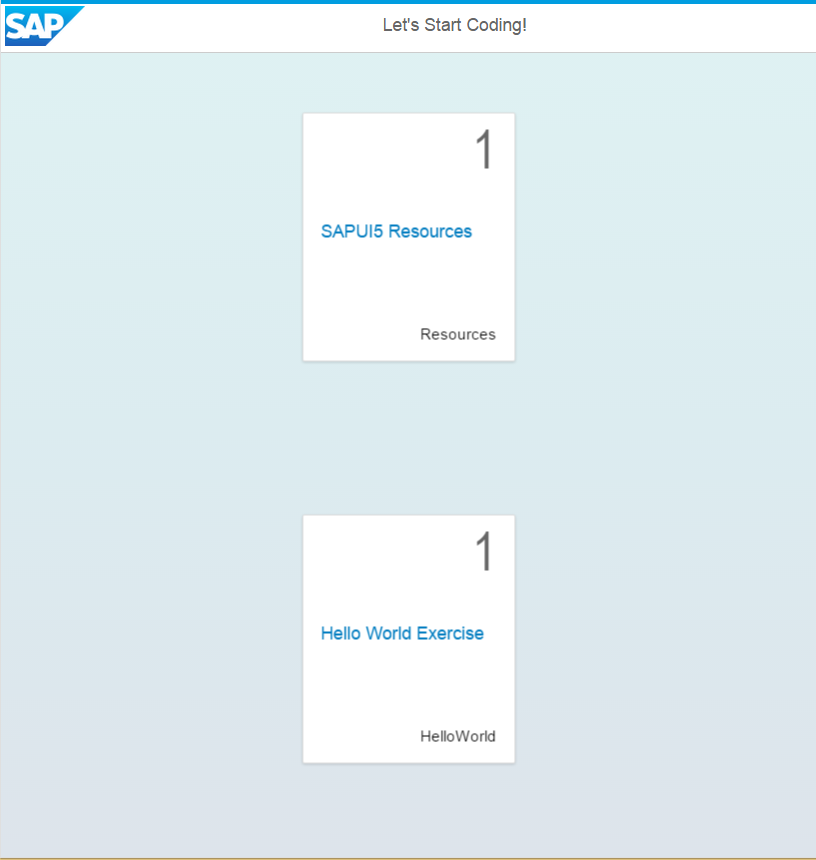
If you click one of the list items, a new browser tab will open with the page indicated by the corresponding URL in the resources.json file.

Return to the editor and then use the drop-down list on the Execute button and select **Run in Application Preview.**



The application preview allows you to see how the application looks in different screen sizes and orientations. There is also a QR Code in the upper right corner that will allow you to load the application on a physical device with typing the URL manually.

## Exercise

1. Add at least three more resources to the resources.json file.
2. Add a new collection to the views.json file called ExerciseCollection. This will be used to add tiles for the exercises in later cases. Bind the second TIleContainer control in the Master view to this collection. Remember that the info property must be the name of the view to which you are going to navigate when the tile is clicked.
3. Create a new view called HelloWorld.view.xml with the following code (You will have to update the reference to the controller at the top of the code)

|  |
| --- |
| <mvc:View height="100%"  controllerName="ui5.controller.HelloWorld"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m"  xmlns:c="sap.suite.ui.commons">  <Page enableScrolling="false" title="Hello World!"  showNavButton = "true"  navButtonPress="handleNavButtonPress">    <c:GenericTile  frameType="TwoByOne"  header="Hello World!"  backgroundImage="http://mindmajix.com/wp-content/uploads/2014/10/sap-ui5-online-training.png" />  </Page>  </mvc:View> |

Listing

1. Create the HelloWorld.controller.js file which has two functions: init and handleNavButtonPress. These would be similar to the same functions in Resources.controller.js except that no model is created.
2. Add the route to the Component.js file.

When you click the HelloWorld tile, the HelloWorld view will open.

