SAP Mobility 101

Tutorial 10 – Split View

# Objective of Exercise

## Build an example application

The objective of this exercise is to build an HTML page that uses JavaScript and output values in a table.

## Note

* We recommend that you use a chrome browser for testing
* Eclipse Juno would be needed for this Tutorial.

# Task 1: Design your layout

In this step you should decide what screens should be the master screen and what screen should be the detail screen.

Detail screen

Master screen

The master screen will slide over the Detail screen.

Let for this tutorial, let a welcome message be displayed on the detail screen, then on the master screen a category can be selected. The category that will be selected, can change the contents of the detail screen.

# Task 2: Change from app to Split App:

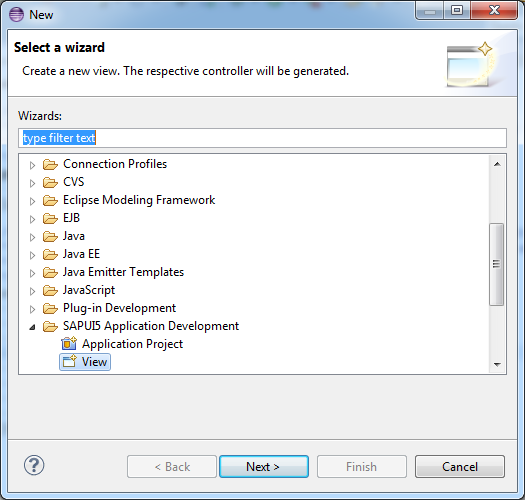
In your index.html file, change the app that you created from a App to a SplitApp:

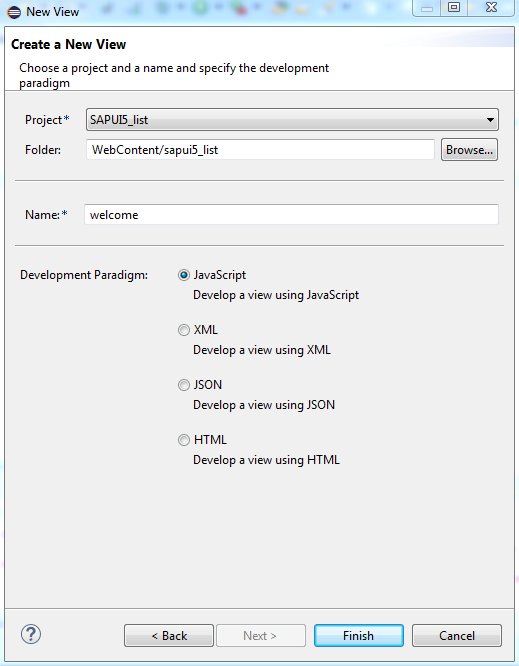
**var** app = **new** sap.m.SplitApp(“appId”,{});

# Task 3: Create a welcoming page

Create a new view in the webcontent folder and call it Detailmain:

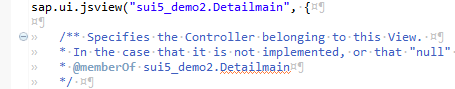
rightClick on WebContent> new>Other>SAPUI5 Application Project:



Next> enter a name for the new screen(welcome)> change the folder location to the sui5\_list folder within WEBContent:>finish

Two new files will be created, the Welcome.view.js file, and the Welcome.controller.js file.

In your index.html, add another page, the view name to be used can be found at the top of your Welcome.view.js file:



Add the new page to the app using the addPage function.

Give the new page an id.

The code for a new page in the index.html file should look like this:

**var** Welcome\_page = sap.ui.view({

id:"Welcome",

viewName:"sapui5\_list.welcome",

type:sap.ui.core.mvc.ViewType.JS

# })

# Task 4: Add the pages to the app

Add Category page as a Master page, and welcome page as a detail page:

The addMasterPage and addDetailPage functions take the page variable name as parameters:

app.addMasterPage(Categ\_page);

app.addDetailPage(Welcome\_page);

The initial page of a normal app were set in the definition of the app, for this exercise, let’s set the initial pages with the set method. The set methods take the page id as parameters:

app.setInitialMaster("Welcome");

app.setInitialDetail("Categories");

We want the master page to slide over the Detail Page, thus set the transition detail to “slide”, since we are using a mobile device, it would be appropriate to hide the master page until it is called, this can be set by the setMode function:

app.setDefaultTransitionNameDetail("slide");

app.setMode("HideMode");

The total code should look like this:

app.addMasterPage(Categ\_page);

app.addDetailPage(Welcome\_page);

app.setInitialMaster("Welcome");

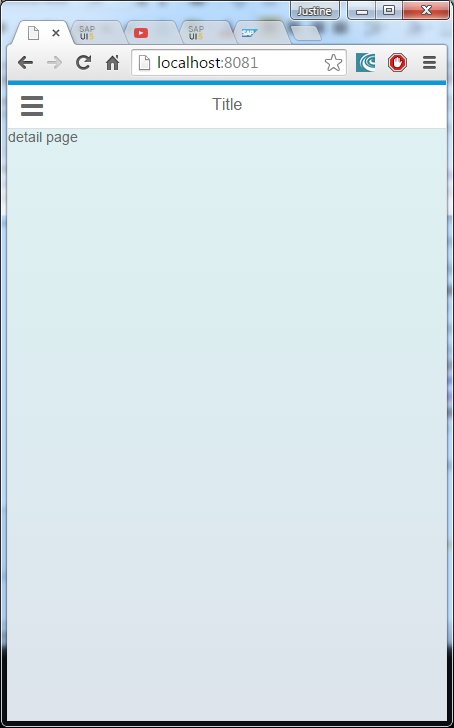
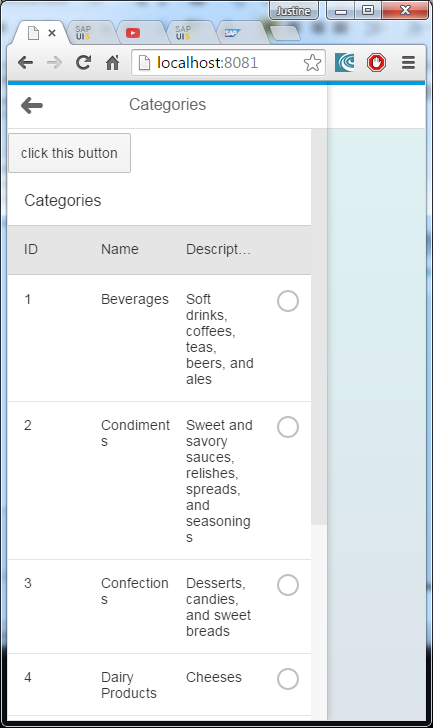
app.setInitialDetail("Categories");

app.setDefaultTransitionNameDetail("slide");

app.setMode("HideMode");

app.placeAt("content");

Save the project and run in your browser:

# Task 5: Create a list

In the product list Controller:

onInit: **function**() {

**var** oModel = **new** sap.ui.model.odata.v2.ODataModel(

"proxy/http/services.odata.org/V2/Northwind/Northwind.svc/",

{

json: **true**

}

);

sap.ui.getCore().setModel(oModel, 'data2');

},

Create a list with id: “products” and in singleselect mode in the ProductList.View:

**var** list = **new** sap.m.List({

id: "Products",

mode: sap.m.ListMode.SingleSelect,

//select: oController.

});

To connect to the database, we want to create a template of a standard list item:

**var** otemplate = **new** sap.m.StandardListItem({

id: "Plist",

title: "{data2>ProductName}",

description:"{data2>CategoryID}"

});

The ID of the list will be Plist, the title of each list item will be the Productname of the content of ‘data’ which is the odata model we’ve set earlier, the description of each item will be the CategoryID.

Bind the list with aggregation: the ‘items’ will be bound to the data in the Products table in the ‘data’ model in the template form.

list.bindAggregation("items", {

path: "data2>/Products",

template: otemplate});

the code should now look like this if the list is added to the page:

createContent : **function**(oController) {

**var** list = **new** sap.m.List({

id: "Products",

mode: sap.m.ListMode.SingleSelect,

//select: oController.

});

**var** otemplate = **new** sap.m.StandardListItem({

id: "Plist",

title: "{data2>ProductName}",

description:"{data2>CategoryID}"

});

list.bindAggregation("items", {

path: "data2>/Products",

template: otemplate});

**var** page1 = **new** sap.m.Page({

title: "Products",

enableScrolling: **true**,

showNavButton: **true**,

navButtonPress: **function**(){

app.back();

},

content: [

list

]

});

**return** page1;

}

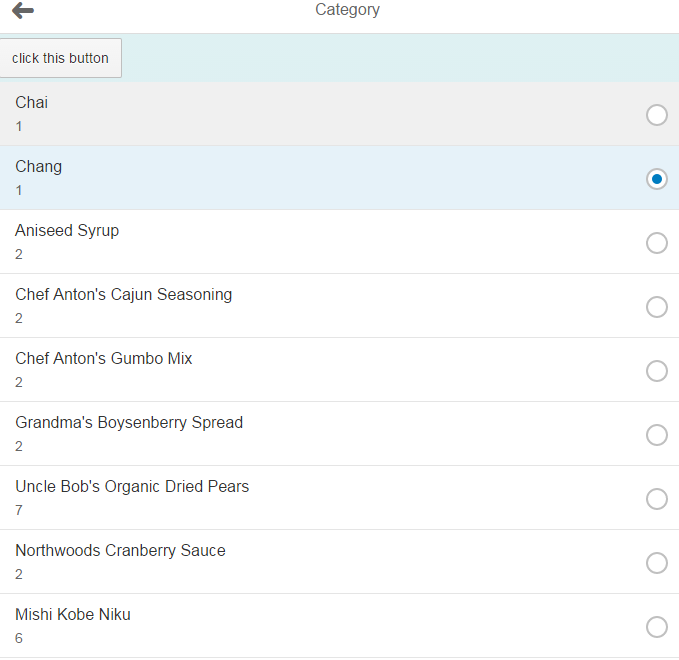
# Task 6: Add the second detail page

In your index.html file, add the page2 to the detail page:

app.addMasterPage(Categ\_page);

app.addDetailPage(Welcome\_page).addDetailPage(Prod\_page);

save and run in your browser:



# Task 7: Respond to selected item

Previously we have set that the handleRowPress function would handle the event of a row being pressed and get the categoryID of the row being pressed:

**var** name = e.getParameter("listItem");

**var** path1 = name.oBindingContexts.data1.sPath;

**var** item = sap.ui.getCore().getModel('data1').getProperty(path1);

console.log(path1);

Create your own JSON model of the object ‘name, since the data is in JSON format:

**var** myModel = **new** sap.ui.model.json.JSONModel(item);

sap.ui.getCore().setModel(myModel, 'selected');

With the toDetailView function, navigate to page2:

app.toDetail('Products','slide');

The entire function for handleRowPress should look like this:

handleRowPress : **function**(e){

**var** name = e.getParameter("listItem");

**var** path1 = name.oBindingContexts.data1.sPath;

**var** item = sap.ui.getCore().getModel('data1').getProperty(path1);

console.log(item);

**var** myModel = **new** sap.ui.model.json.JSONModel(item);

sap.ui.getCore().setModel(myModel, 'selected');

app.toDetail('Products', 'slide');

},

save and run to check if your app navigates to the products list when a row on the table are selected.

# Task 8: Filter the products by Category:

Add the filter created in the previous tutorial to page2, add it as a property in the bindAggregation method:

**var** oFilters = [

**new** sap.ui.model.Filter(

"CategoryID", "EQ", "2"

)];

list.bindAggregation("items", {

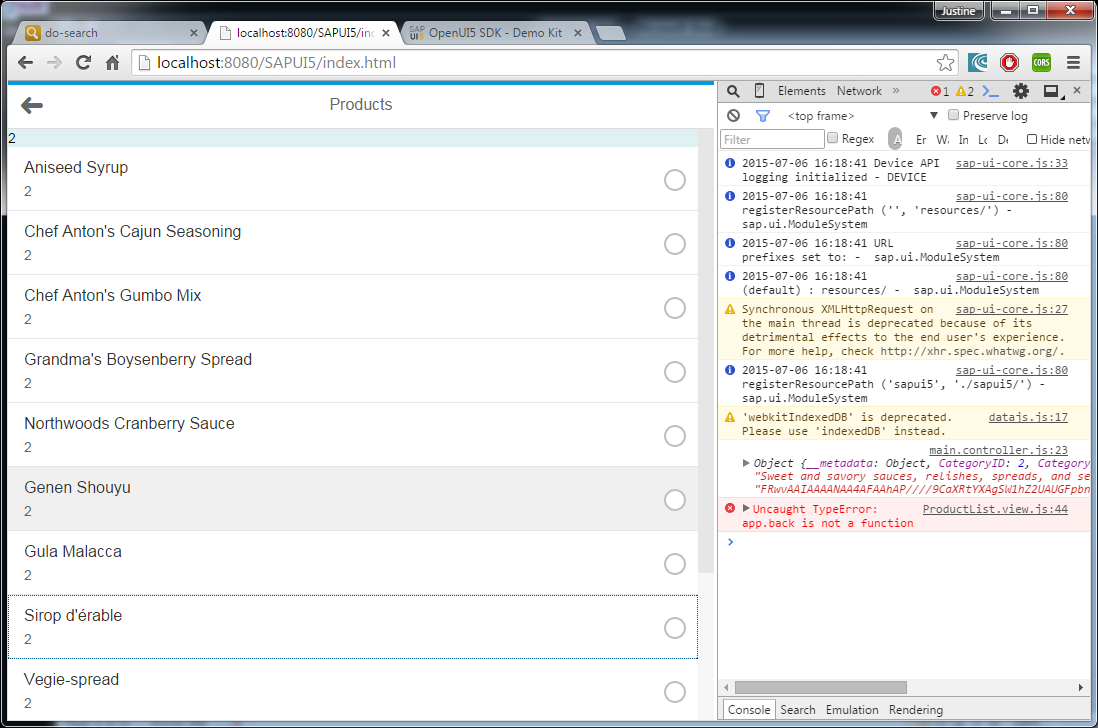
path: "data>/Products",

template: otemplate,

filters: oFilters});

save and run:

The list should be filtered so only the category 2 products are shown.



# Task 9: Filter the products by selected category

Since we are creating the list before we actualy chose the filter item to be used, the program will give an error, thus remove the part where you add the product page:

app.addMasterPage(Categ\_page);

app.addDetailPage(Welcome\_page);

Add the product page when the Category is selected:##ERROR #I’m working on this

handleRowPress : **function**(e){

**var** name = e.getParameter("listItem");

**var** path1 = name.oBindingContexts.data1.sPath;

**var** item = sap.ui.getCore().getModel('data1').getProperty(path1);

console.log(item);

**var** myModel = **new** sap.ui.model.json.JSONModel(item);

sap.ui.getCore().setModel(myModel, 'selected');

app.addDetailPage(Prod\_page);

app.toDetail('Products', 'slide');

},

Change the filter value of you filter in ProductList.main, to the selected value: