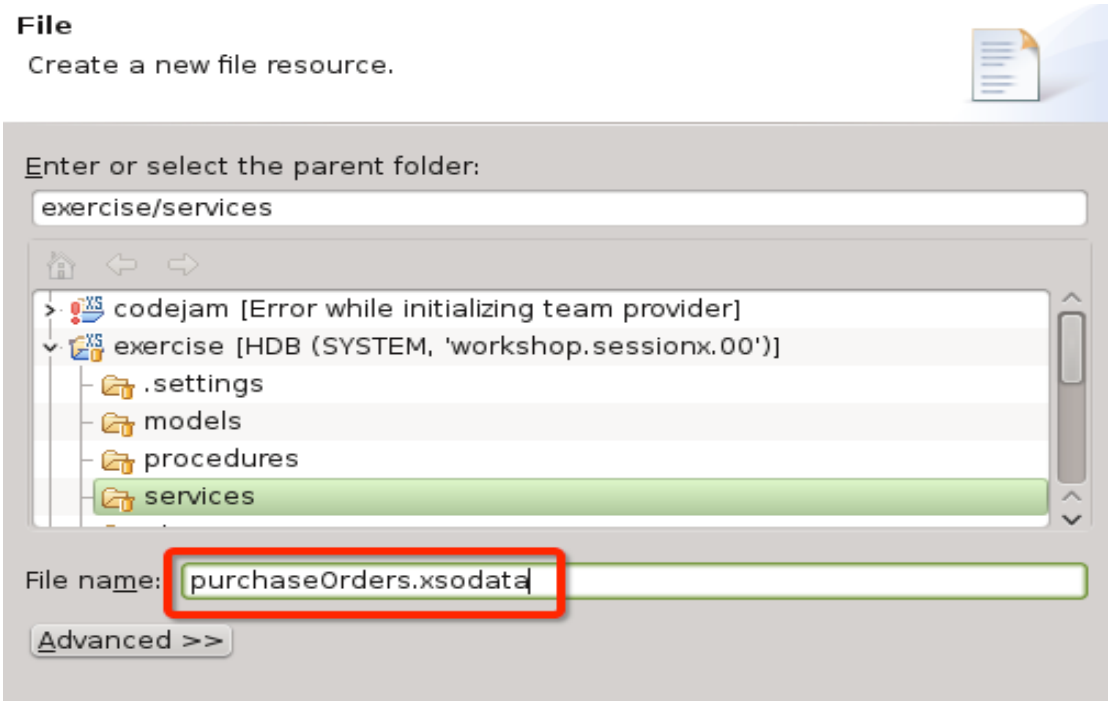
Exercise 4: OData Service

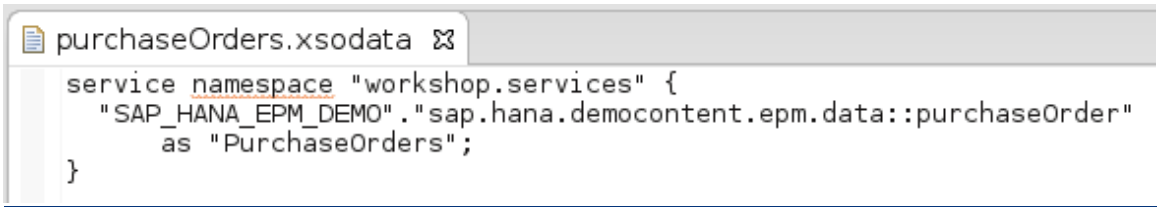
The XSEngine contains a special tool for the creation of OData Services without needing to perform server side coding to expose the data in your HANA views or tables and can be simply consumed by your applications.

Creating a simple OData Service

1. To create an OData service from an existing HANA table or view, you need only define an XSODATA service definition file. By applying what you've learned in the previous exercises; create a new file called **purchaseOrders.xsodata** in the services package of your project.



2. We want to define an OData service to expose the purchase order table. The syntax of the XSODATA service is relative easy for this use case. We need only define a namespace (**workshop.services**), the table schema (**SAP_HANA_EPM_DEMO**), the name of the HANA Table we will base the service from (**sap.hana.democontent.epm.data::purchaseOrder**) and the name of the OData entity (**PurchaseOrders**). Therefore the content of the XSODATA file would be.



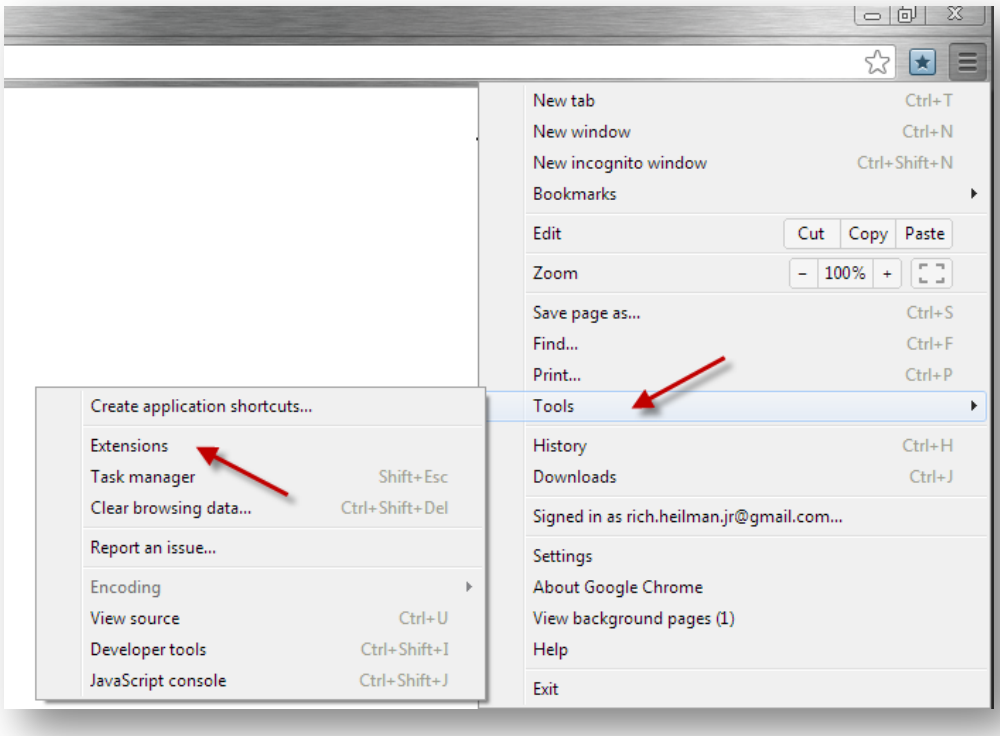
Source Code:

```
service namespace "workshop.services" {
        "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::purchaseOrder"
        as "PurchaseOrders";
}
```

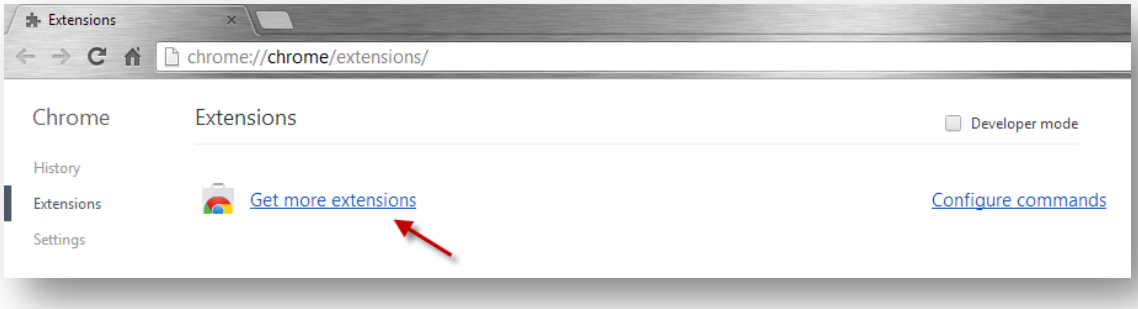
3. Commit and activate your service.

4. Before you run the service, we will ask you to install some extensions in your Chrome browser for better display of JSON or XML files. Open the Chrome browser. Click on the "Customize and control Google

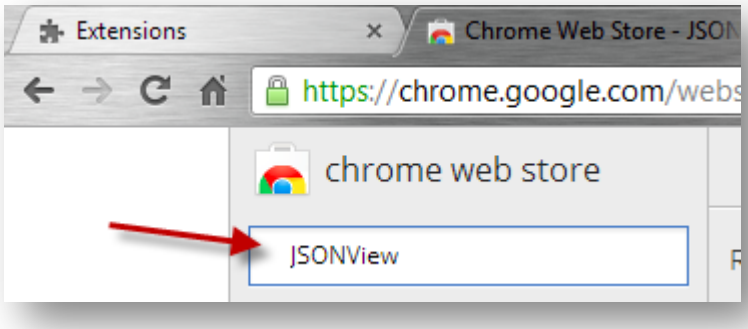
Chrome" button. Click "Tools", then "Extensions".



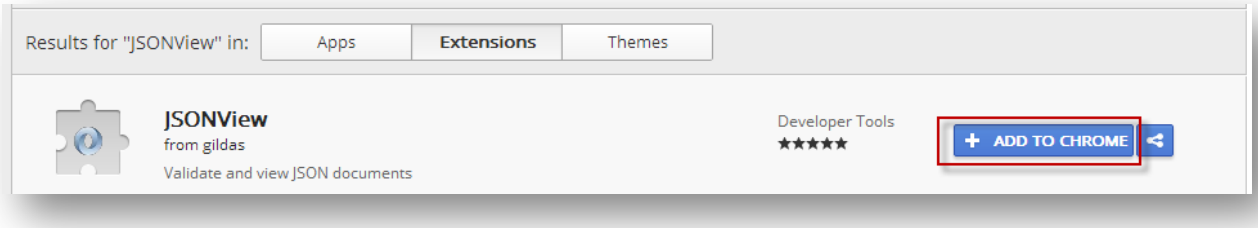
5. Click on "Get more extensions".



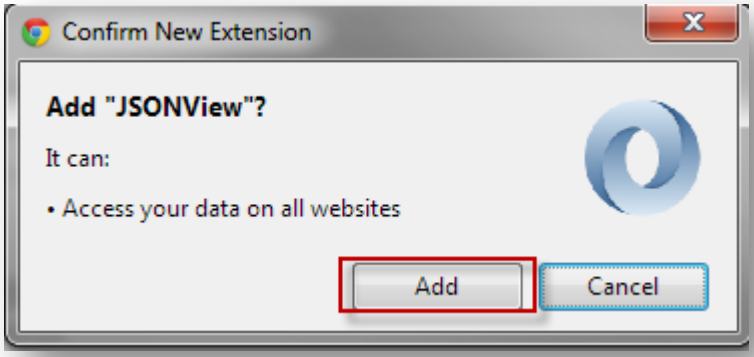
6. Enter JSONView and hit enter.



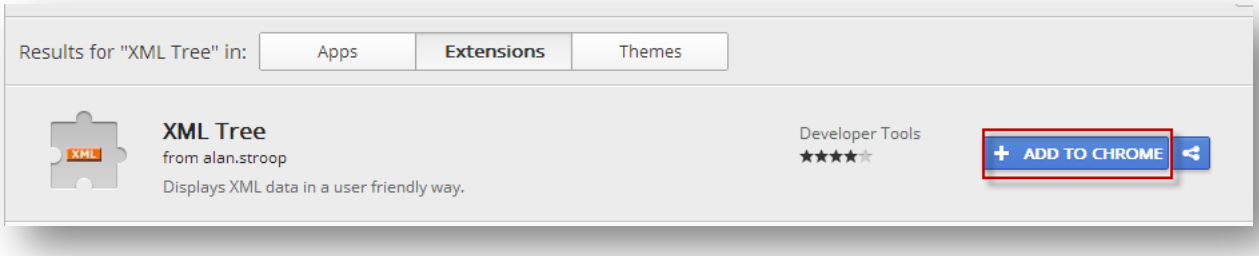
7. Next to JSONView, click “Add to Chrome”.



8. Click “Add”.



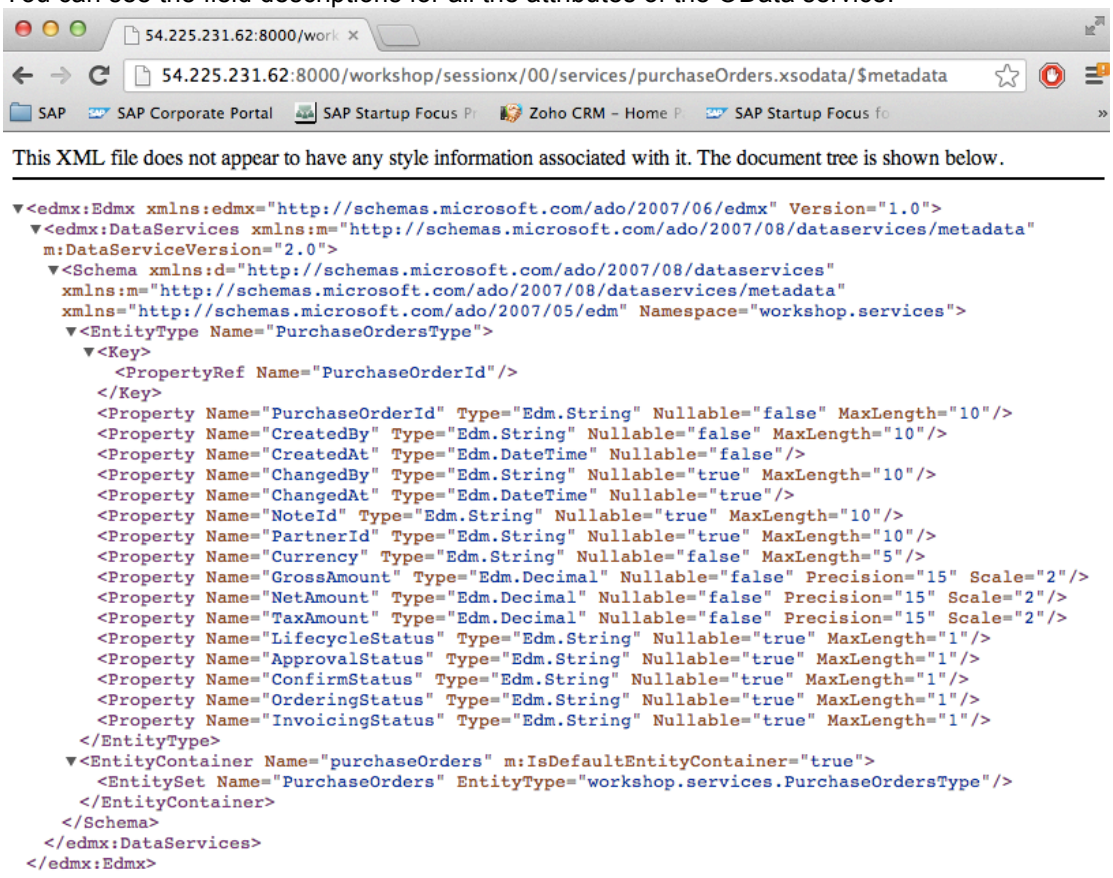
9. Next, add another Chrome component called XML Tree in the same way.



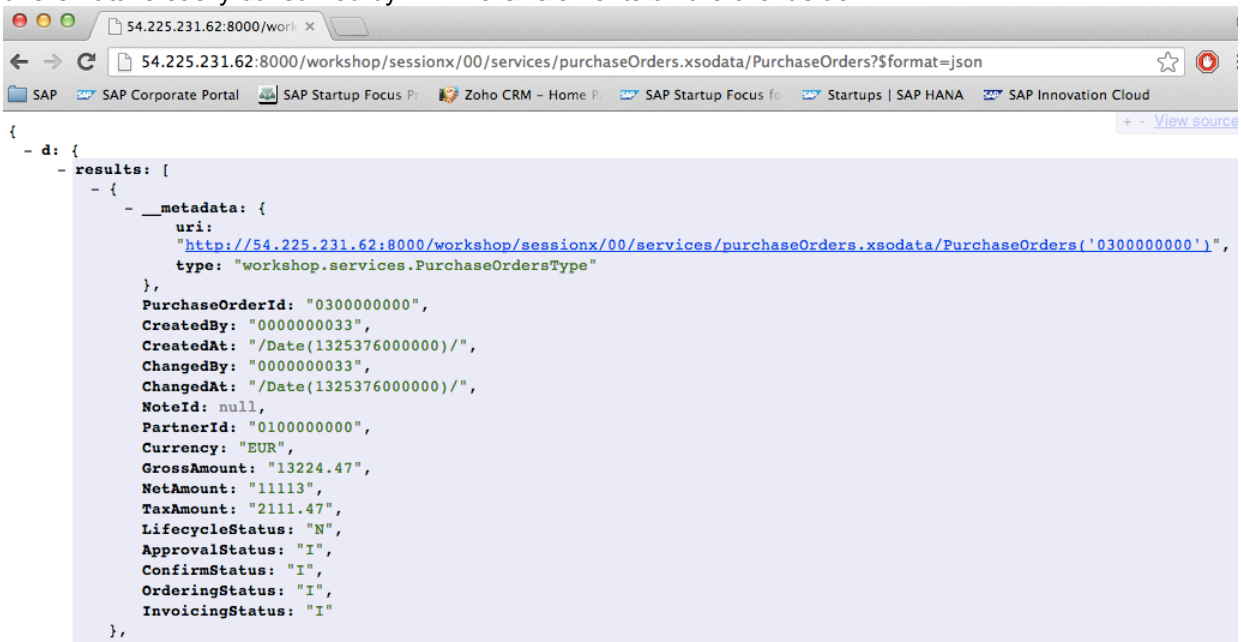
10. Now run the service from Chrome. The URL to run your service would be http://<host>:<port>/workshop/session#/XX/services/purchaseOrders.xsodata/$metadata

    where # is your session letter and XX is your group number. For example if your session letter was X and your group number was 00 then the URL would be:

http://<host>:<port>/workshop/sessionx/00/services/purchaseOrders.xsodata/$metadata
You can see the field descriptions for all the attributes of the OData service.



11. In order to view the data of the entity, you would append **PurchaseOrders** to the end of the URL and we can also try to use the json format instead of the xml format.
http://<host>:<port>/workshop/sessionx/00/services/purchaseOrders.xsodata/PurchaseOrders?$format=json
You are now able to see the data from the purchaseOrder table. In later steps of this exercise you will see how this OData is easily consumed by HTML5 UI elements on the client side.



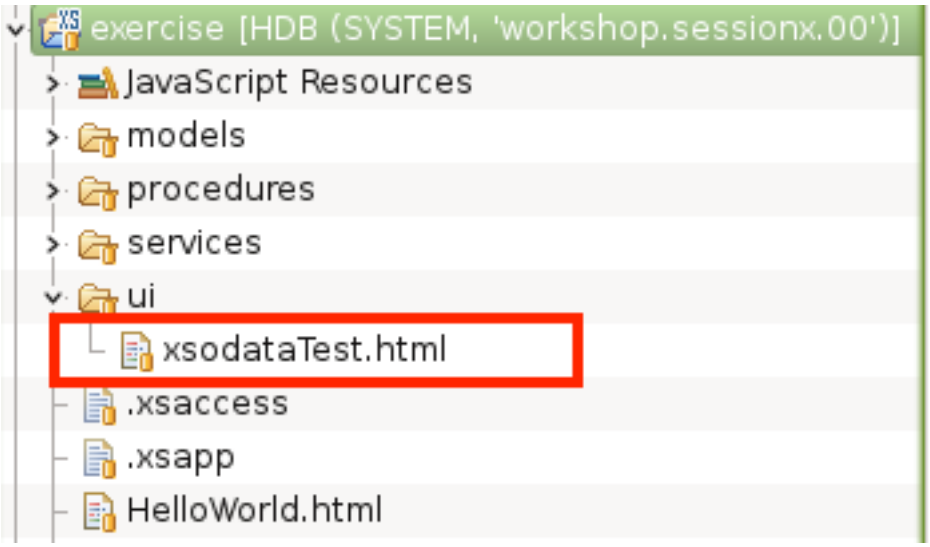
You can also experiment with standard OData URL parameters like $top, $skip, or $filter. These options are interpreted and handled by the OData service of the XSEngine for you. You get complex service handling

without any coding. For example the following URL would return only three business partner records and would skip the first five records. Such parameters are helpful when implementing server side scrolling, filtering, or sorting in table UI elements.
http://<host>:<port>/workshop/sessionx/00/services/purchaseOrders.xsodata/PurchaseOrders?$format=json&$top=3&$skip=5

Calling the OData Service From the User Interface

1. As we've seen, the OData Service isn't necessarily designed to be easily human readable. However they are very well suited for enabling data exchange to mobile devices and browser based applications. In this exercise we will build a user interface table which is bound to our XSODATA service we've designed up to this point.

2. In the project, choose to create a new html file **xsodataTest.html** in the **ui** folder.



3. We will use the SAPUI5 framework to build a table to show the business partners. SAPUI5 is the html5 framework to build web applications as well as mobile applications, following the MVC design pattern and it is already installed in the HANA system. You can download the UI5 eclipse based IDE from the SCN: http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/webcontent/uuid/c08465d5-b833-2f10-e59d-f67a5cb54d2f

   In this example, we will not use the UI5 IDE but just continue with our HANA studio. The differences are we cannot build a UI5 project in MVC patterns, with particular UI5 views and controllers. However, we can still reuse the UI5 library in a single html file.

   Here is the example to load the UI5 library. We will use the "sap.ui.commons" and "sap.ui.table" in this example.

```
<!-- load UI5 library, which is already installed in your HANA system -->
<script src="/sap/ui5/1/resources/sap-ui-core.js" id="sap-ui-bootstrap"
    data-sap-ui-libs="sap.ui.commons,sap.ui.table"
    data-sap-ui-theme="sap_goldreflection">
</script>
```

   And here is the example how to call the oData services and bind the results to UI5 model.

```
  // Create UI5 Model with data source from the odata service we just defined
  var oModel = new sap.ui.model.odata.ODataModel(
          "../services/purchaseOrders.xsodata/", false);
```

   If you do not want to type code, copy the full code to your html file:

```
<!DOCTYPE HTML>
<html>
```

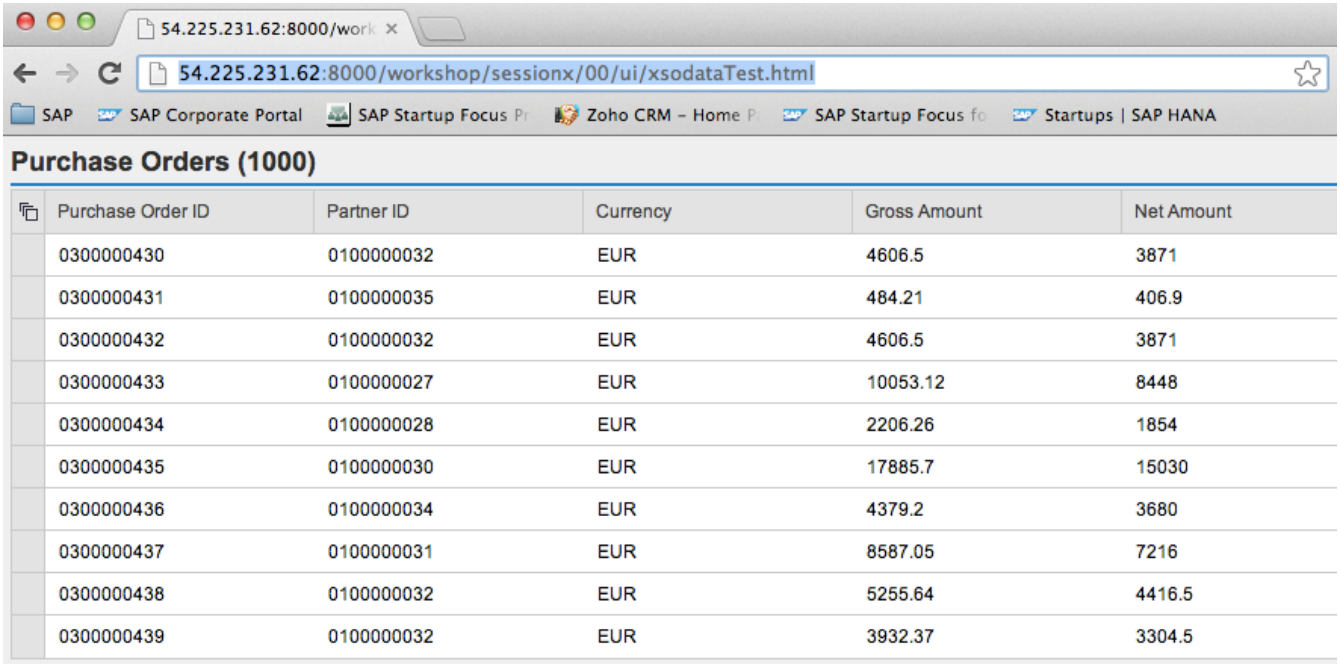
```
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">

<!-- load UI5 library, which is already installed in your HANA system -->
<script src="/sap/ui5/1/resources/sap-ui-core.js" id="sap-ui-bootstrap"
        data-sap-ui-libs="sap.ui.commons,sap.ui.table"
        data-sap-ui-theme="sap_goldreflection">
</script>


<script>
        // Create UI5 Model with data source from the odata service we just defined
        var oModel = new sap.ui.model.odata.ODataModel(
                        "../services/purchaseOrders.xsodata/", false);

        var oControl;
        oTable = new sap.ui.table.Table("test", {
                tableId : "tableID",
                visibleRowCount : 10
        });
        oTable.setTitle("Purchase Orders");

        //Table Column Definitions
        oControl = new sap.ui.commons.TextField()
                        .bindProperty("value", "PurchaseOrderId");
        oTable.addColumn(new sap.ui.table.Column({
                label : new sap.ui.commons.Label({
                        text : "Purchase Order ID"
                }),
                template : oControl,
                sortProperty : "PurchaseOrderId",
                filterProperty : "PurchaseOrderId",
                width : "125px"
        }));

        oControl = new sap.ui.commons.TextField().bindProperty("value",
                        "PartnerId");
        oTable.addColumn(new sap.ui.table.Column({
                label : new sap.ui.commons.Label({
                        text : "Partner ID"
                }),
                template : oControl,
                sortProperty : "PartnerId",
                filterProperty : "PartnerId",
                width : "125px"
        }));

        oControl = new sap.ui.commons.TextField().bindProperty("value",
                        "Currency");
        oTable.addColumn(new sap.ui.table.Column({
                label : new sap.ui.commons.Label({
                        text : "Currency"
                }),
                template : oControl,
                width : "125px"
        }));

        oControl = new sap.ui.commons.TextField()
                        .bindProperty("value", "GrossAmount");
```

```
        oTable.addColumn(new sap.ui.table.Column({
                label : new sap.ui.commons.Label({
                        text : "Gross Amount"
                }),
                template : oControl,
                width : "125px"
        }));

        oControl = new sap.ui.commons.TextField()
                        .bindProperty("value", "NetAmount");
        oTable.addColumn(new sap.ui.table.Column({
                label : new sap.ui.commons.Label({
                        text : "Net Amount"
                }),
                template : oControl,
                width : "125px"
        }));

        oTable.setModel(oModel);

        //Create Sorter and Bind to the Business Partner Entity
        var sort1 = new sap.ui.model.Sorter("PurchaseOrderId");
        oTable.bindRows("/PurchaseOrders", sort1);

        var iNumberOfRows = oTable.getBinding("rows").iLength;
        oTable.setTitle("Purchase Orders" + " (" + iNumberOfRows + ")");

        oTable.placeAt("content");
</script>

</head>
<body class="sapUiBody" role="application">
        <div id="content"></div>
</body>
</html>
```

4. Save, commit and activate your UI content.

5. Test your **xsodataTest.html** application. The URL to run your test application would be http://<host>:<port>/workshop/session#/XX/ui/xsodataTest.html where # is your session letter and XX is your group number. For example if your session letter was X and your group number was 00 then the URL would be:

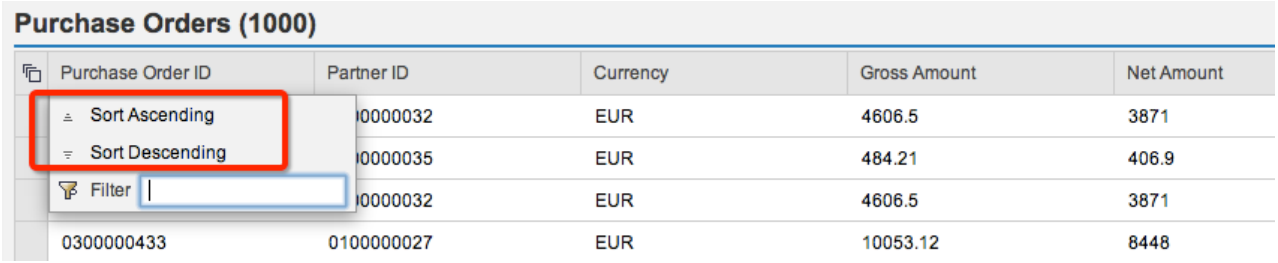
   http://<host>:<port>/workshop/sessionx/00/ui/xsodataTest.html



Purchase Orders (1000)

Purchase Order ID	Partner ID	Currency	Gross Amount	Net Amount
0300000430	0100000032	EUR	4606.5	3871
0300000431	0100000035	EUR	484.21	406.9
0300000432	0100000032	EUR	4606.5	3871
0300000433	0100000027	EUR	10053.12	8448
0300000434	0100000028	EUR	2206.26	1854
0300000435	0100000030	EUR	17885.7	15030
0300000436	0100000034	EUR	4379.2	3680
0300000437	0100000031	EUR	8587.05	7216
0300000438	0100000032	EUR	5255.64	4416.5
0300000439	0100000032	EUR	3932.37	3304.5

6. Try the sort or filter ability on one of the columns to test out the built-in features of the OData Service.



The Best-Run Businesses Run SAP™