VP Softwaretechnologien WS2016/2017

SAP HANA

***H**igh Performance **An**alytic **A**ppliance*

Dr. Schwaiger Roland

Vorstellung

roland.schwaiger@facet.at

Dr. Roland Schwaiger

Located

Bad Dürrnberg, Hallein, AT

Background

Mathematics (University Salzburg)
Computer Sciences (University Salzburg, Bowling Green State University)
Project & Process Management (SMBS – University of Salzburg Business School)

Profession

SAP Technical Consultant (Cert. SAP Development Consultant)
SAP Trainer
Project Coach (Cert. Scrum Master)
Software Architect
Software Developer (SAP AG, Walldorf, DE and Customer Development Projects)
Author (check out Amazon and/or www.citeseer.com)
Lecturer (University Salzburg, FH Salzburg)

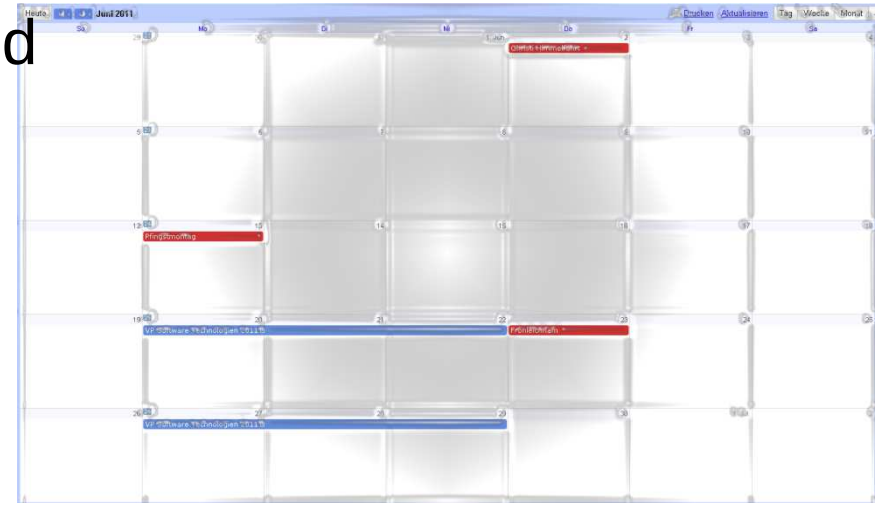
Inhalte/Organisation

1. Motivation
2. LV Überblick
3. Abschluss LV

Blockveranstaltung (www.facet.at)

Datum: 01.02 – 03.02

Zeit: 09:00-12:00 und
13:00-17:00



Motivation

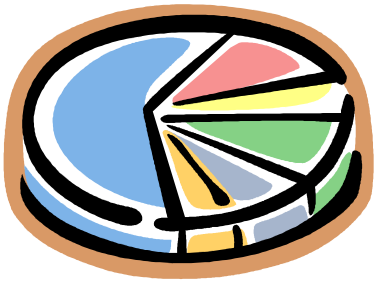
Die drei Schwerpunkte, die wir behandeln werden sind:

1. **SAP HANA Entwicklung**
2. **oData** (REST) Service Erstellung im SAP Backend
3. **Web Anwendungen** mit Hilfe von SAP UI5

und das Schritt für Schritt in hoffentlich leicht verdaulicher Art und Weise.

VP Overview

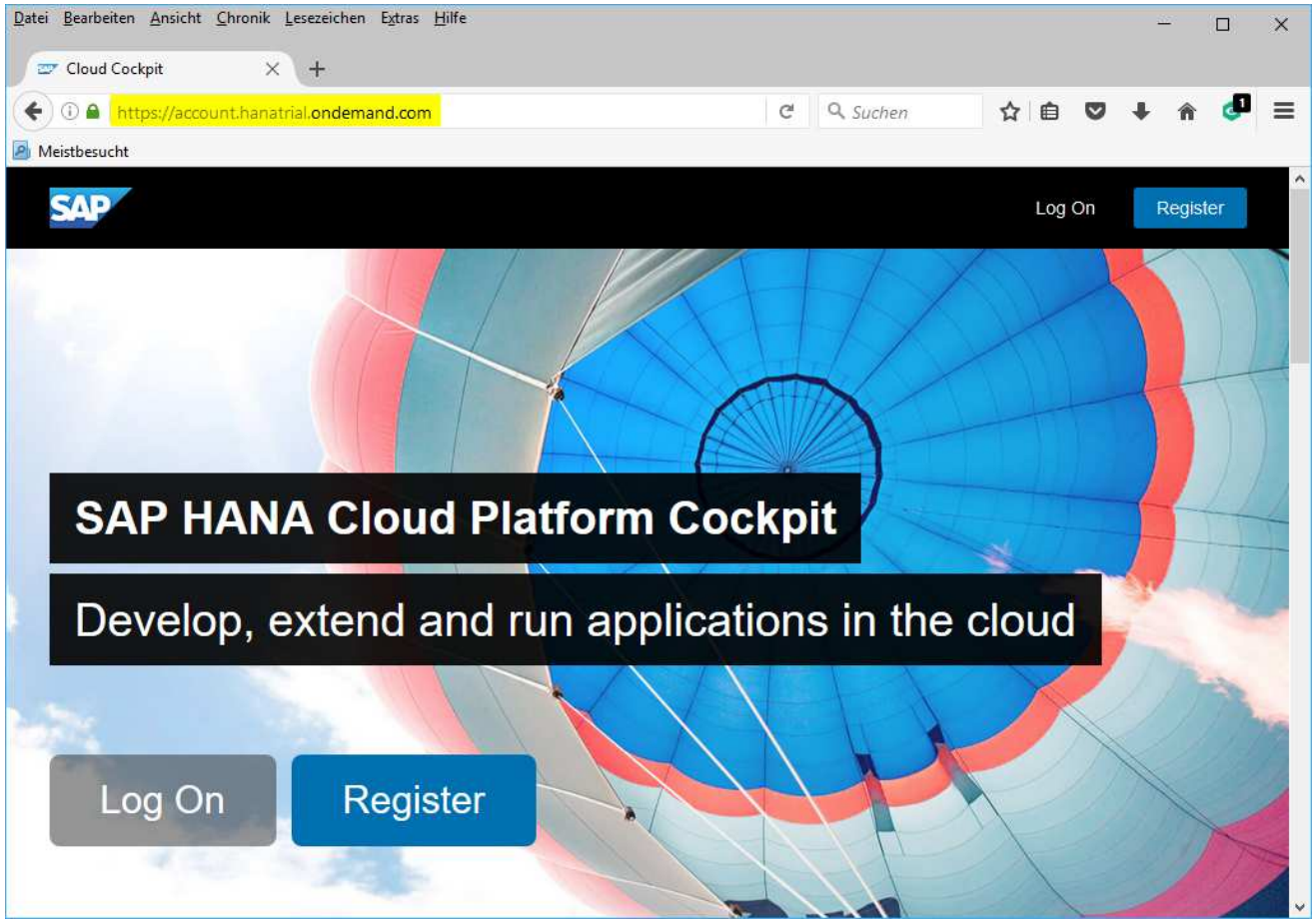
First things first Logon, Get to know each other, DB Setup, User Setup, Package creation, Hello World, …
Introduction to HANA Infrastructure, SAP HANA Architecture, Web IDE, Web Based Dev Tool, …
Development Tools Eclipse, Web IDE, HANA …
UIs with SAP UI5 UI5 Model, View, Controller, I18N, …
SAP HANA Extended Application Services (XS) Packages, Schema, Roles,
The Persistence Model CDS,
oData Services
SQLScript
Server Side Java Script
Connectivity 4 HANA XS
? IoT
? Honeycomb



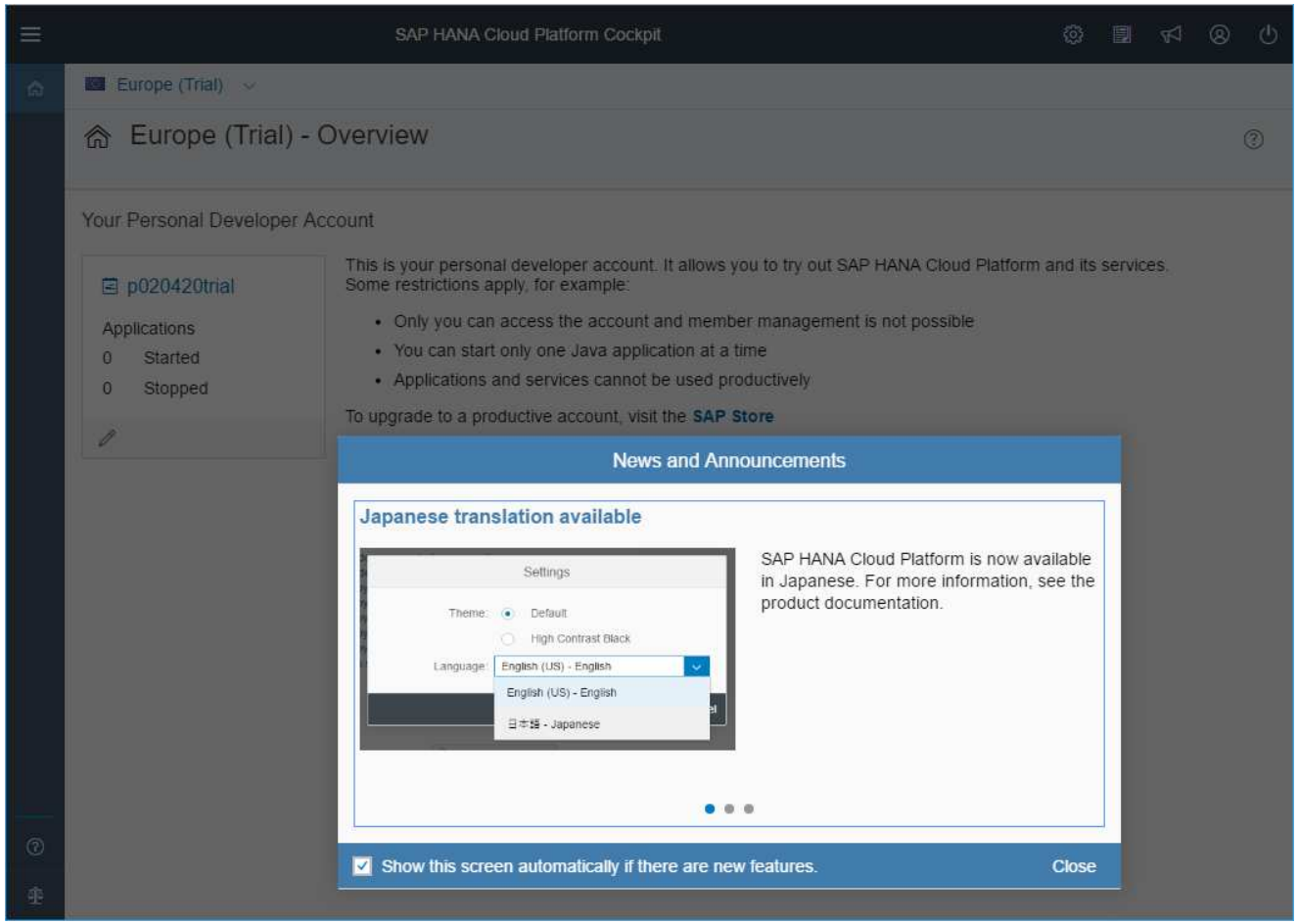
FIRST THINGS FIRST!

Get prepared for development

HANA Login

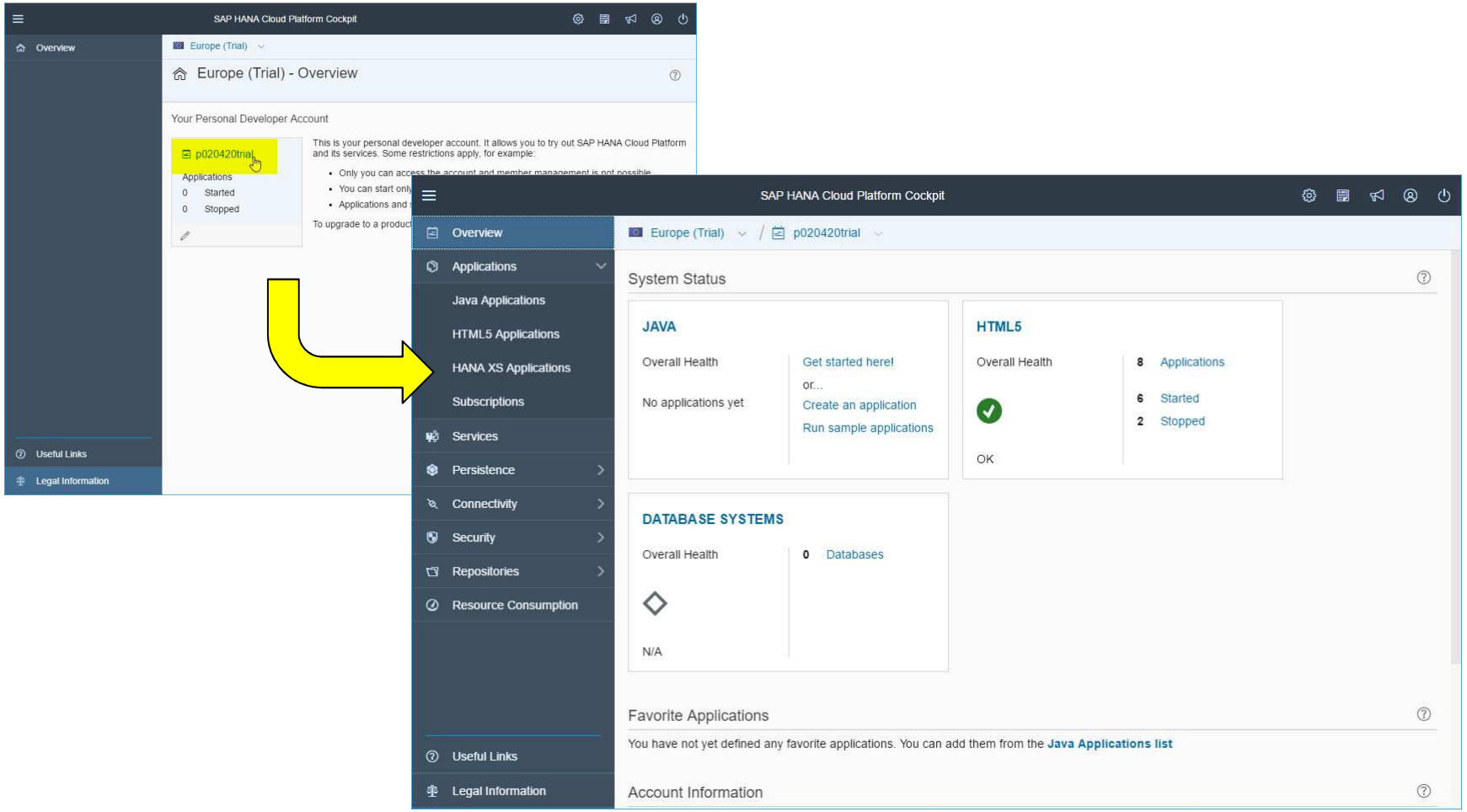

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
Cloud Cockpit
https://account.hanatrial.ondemand.com
Suchen
Meistbesucht
SAP
Log On
Register
SAP HANA Cloud Platform Cockpit
Develop, extend and run applications in the cloud
Log On
Register

Entry Screen



Navigate to

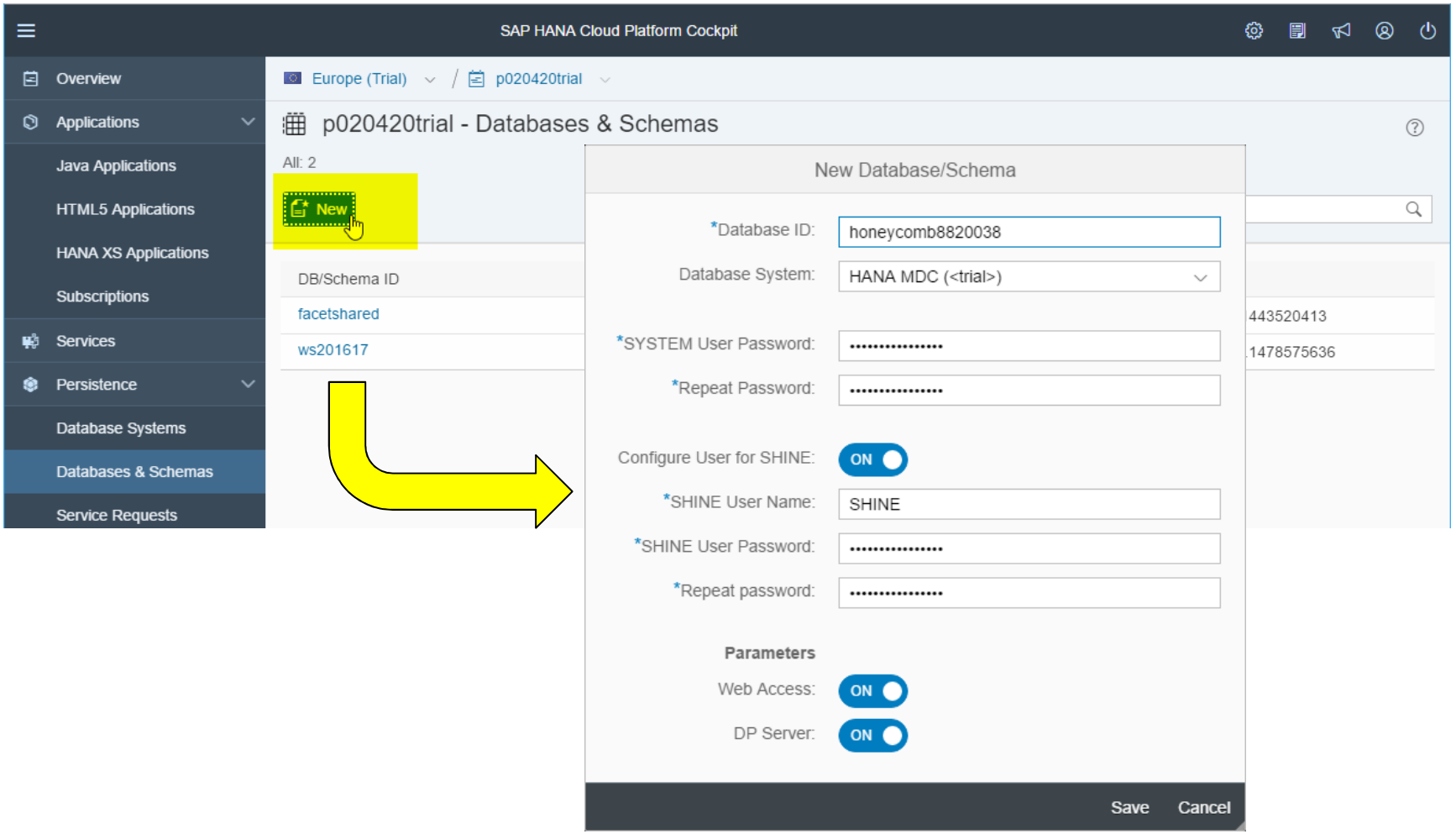
SAP HANA Cloud Platform Cockpit



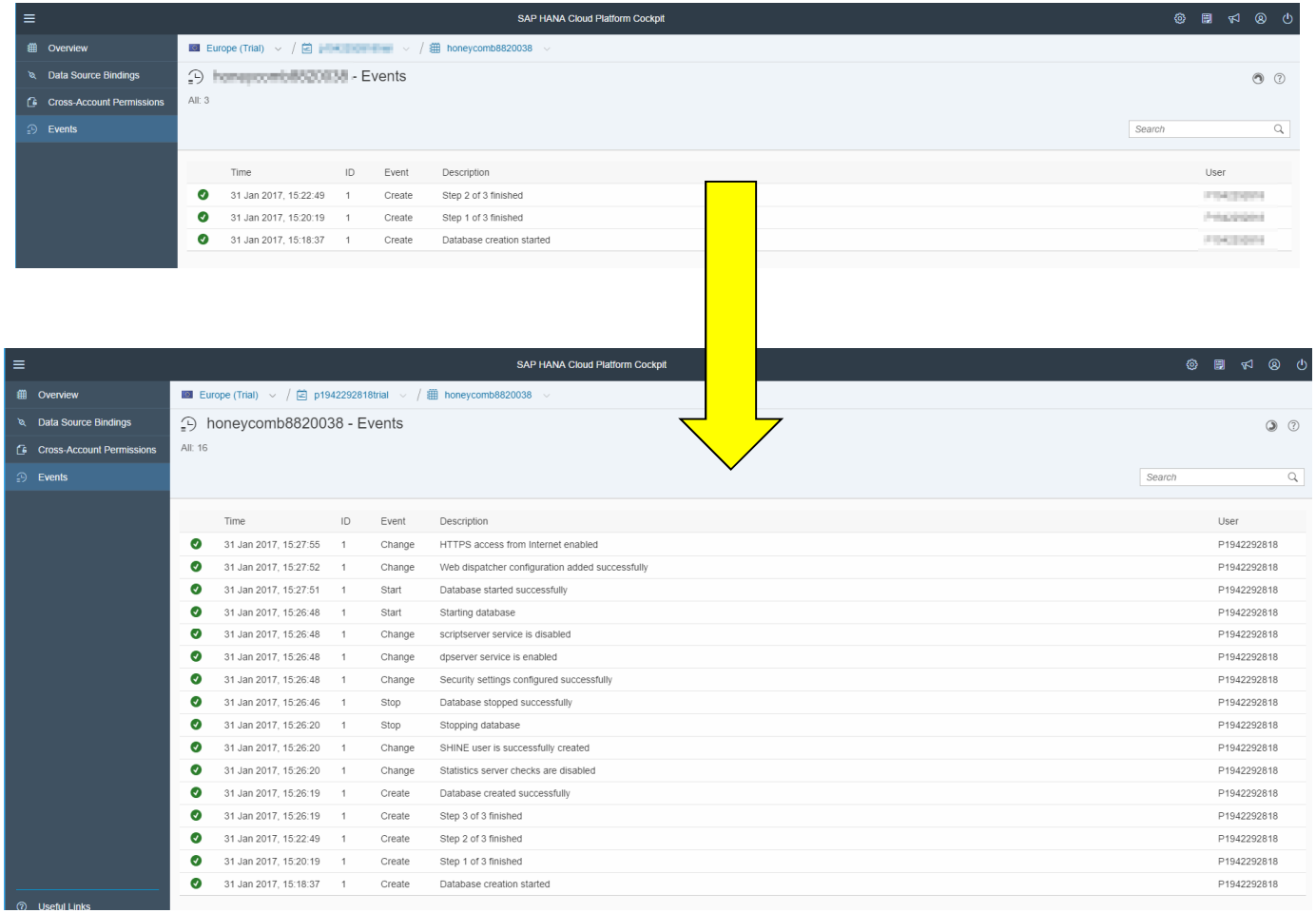
SETUP MULTITENANT DATABASE CONTAINER (MDC)

- Create an MDC instance
- Create the SYSTEM user

Setup MDC
Create DB

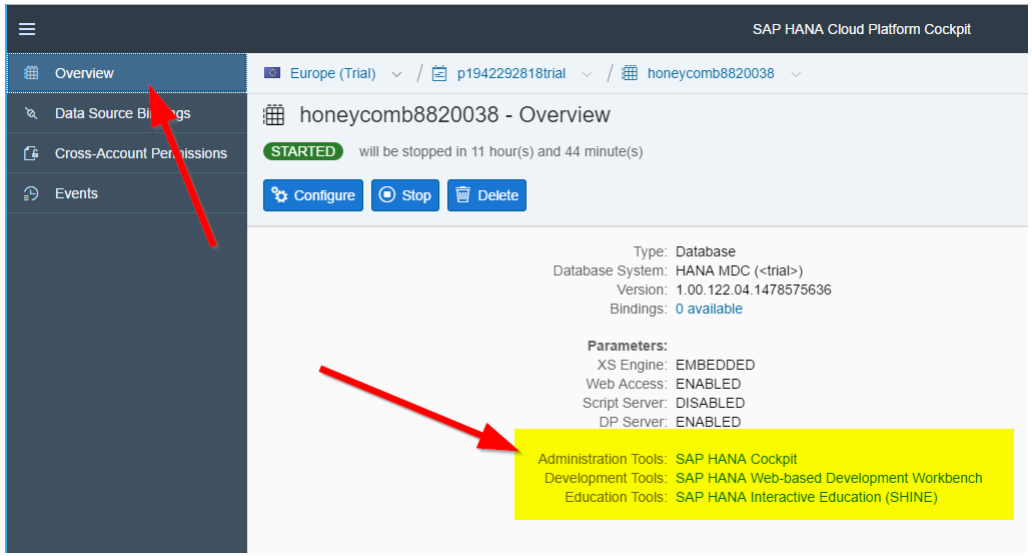


Setup MDC
Create DB - Steps



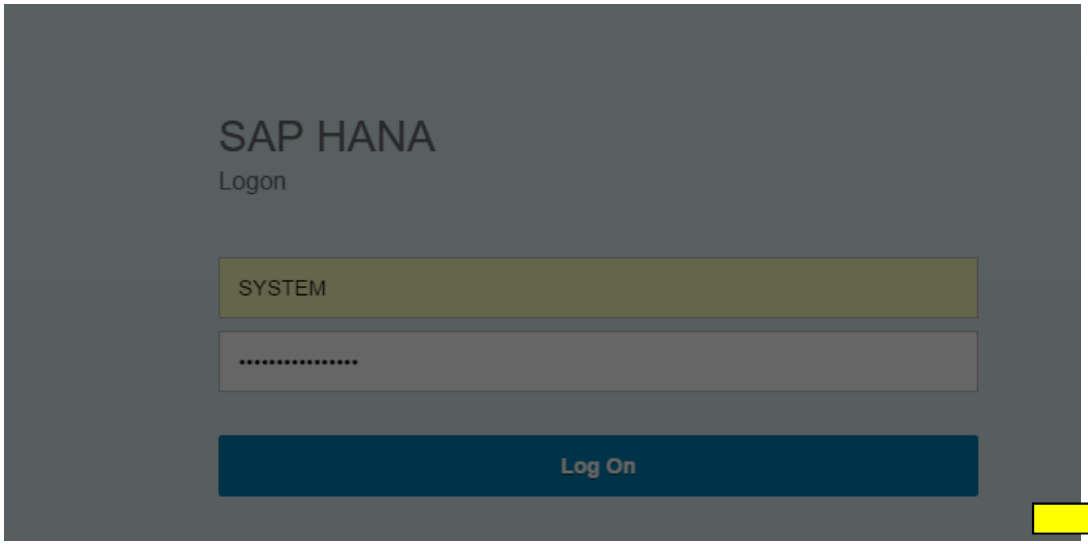
Setup MDC Select DB

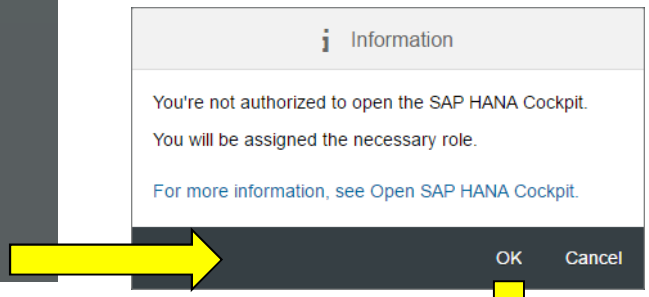
- Switch to Overview
- Or
- Select the newly created SAP HANA MDC database in the list of databases



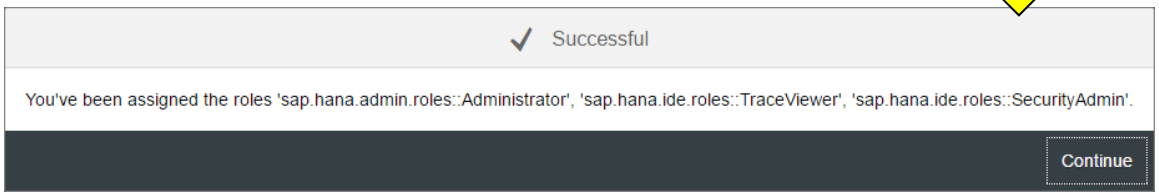
Setup MDC
Open SAP HANA Cockpit

- In the overview that is shown in the lower part of the screen, open the **SAP HANA Cockpit** link under **Administration Tools**.



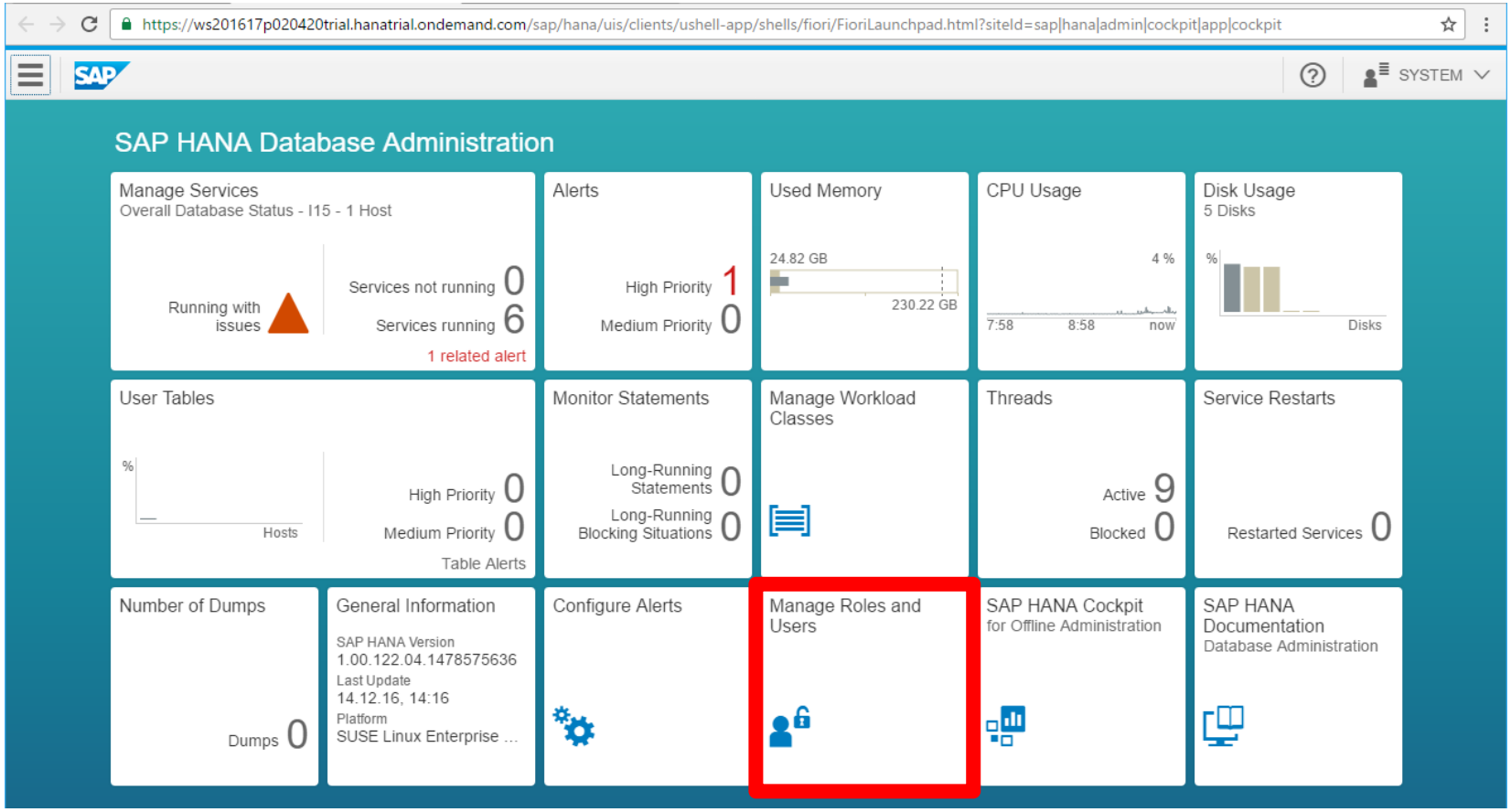


- In the Enter Username field, enter **SYSTEM**, then enter the **password** you determined for the SYSTEM user in the Enter Password field.



SAP HANA Cockpit

SAP HANA Database Administration

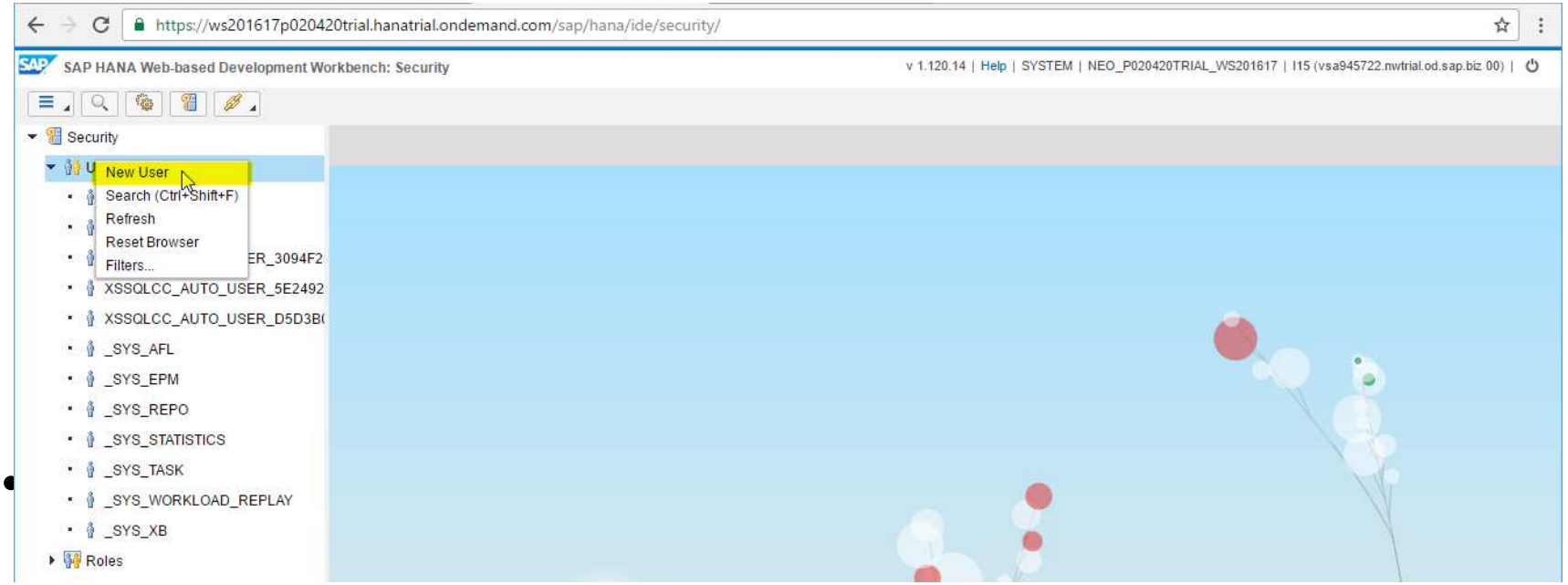


SYSTEM user

- Add roles:
  - sap.hana.xs.ide.roles::EditorDeveloper
  - sap.hana.xs.ide.roles::Developer
  - sap.hana.xs.lm.roles::Administrator
  - *sap.hana.xs.debugger::Debugger*
  - *sap.hana.xs.admin.roles::HTTPDestViewer*
  - *sap.hana.xs.admin.roles::HTTPDestAdministrator*
  - *sap.hana.xs.admin.roles::TrustStoreViewer*
  - *sap.hana.xs.admin.roles::TrustStoreAdministrator*

Create User

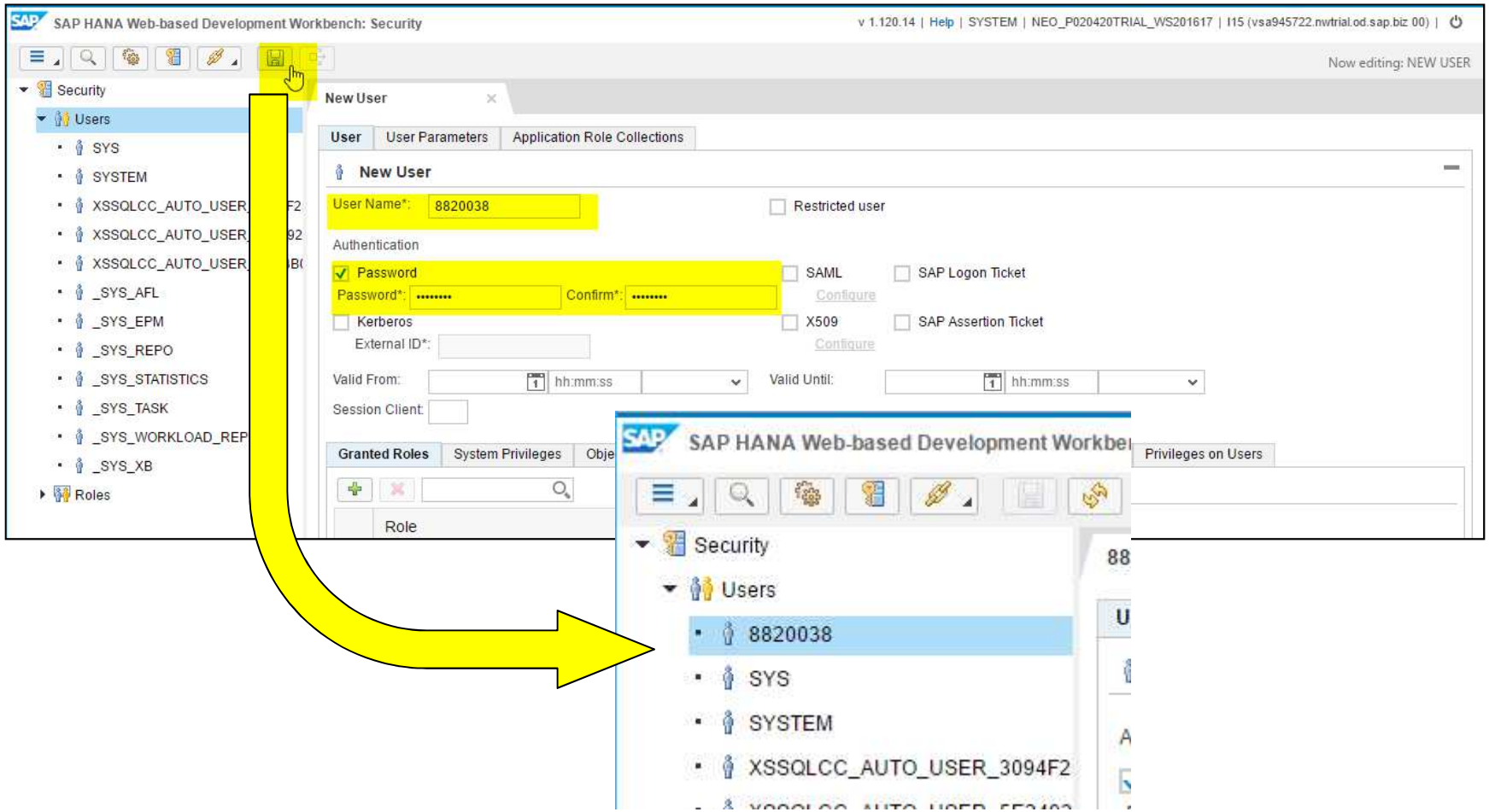
- Choose **Manage Roles and Users**.
- To create database users and assign them the required roles, expand the **Security** node.
- Open the context menu for the **Users** node and choose **New User**.



Create User

- On the **User** tab, provide a name for the new user.The user name always appears in upper case letters.
- In the **Authentication** section, make sure the **Password** checkbox is selected and enter a password.Note: The password must start with a letter and only contain uppercase and lowercase letters ('a' - 'z', 'A' - 'Z'), and numbers ('0' - '9').
- To create the user, choose **Save**.The new database user is displayed as a new node under the **Users** node.

Create User


SAP HANA Web-based Development Workbench: Security
Now editing: NEW USER
Security
Users
SYS
SYSTEM
_SYS_AFL
_SYS_EPM
_SYS_REPO
_SYS_STATISTICS
_SYS_TASK
_SYS_XB
Roles
New User
User
User Parameters
Application Role Collections
New User
User Name*:
8820038
Restricted user
Authentication
Password
Password*:
Confirm*:
SAML
SAP Logon Ticket
Configure
Kerberos
External ID*:
X509
SAP Assertion Ticket
Configure
Valid From:
hh:mm:ss
Valid Until:
hh:mm:ss
Session Client:
Granted Roles
System Privileges
Privileges on Users
Role
SAP HANA Web-based Development Workbe
Security
Users
8820038
SYS
SYSTEM
XSSQLCC_AUTO_USER_3094F2

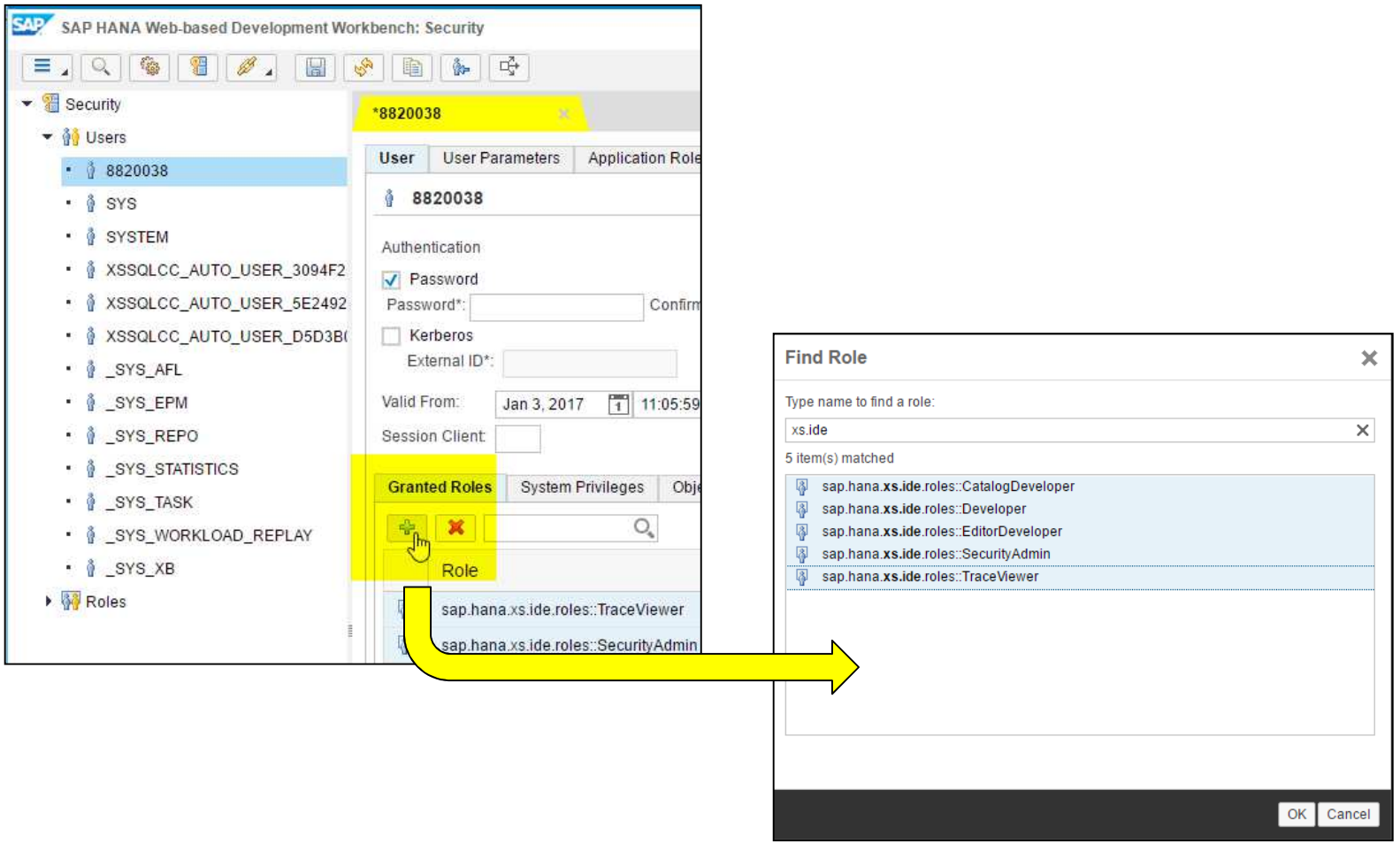
Grant Roles

- To assign your user the roles with the required permissions for working with SAP HANA Web-based Development Workbench, go to the Granted Roles section and choose the + (Add Role) button.
- Type **xs.ide** in the search field and select all roles in the result list.
- Choose Ok.

- The roles are added on the Granted Roles tab.
- Repeat the last two steps to assign the **CONTENT_ADMIN** role to the user.
- Save your changes.

- Caution: At this point, you are still logged on with the SYSTEM user. You can only use your new database user to work with **SAP HANA Web-based Development Workbench** by logging out from **SAP HANA Cockpit** first. Otherwise, you would automatically log in to the **SAP HANA Web-based Development Workbench** with the SYSTEM user instead of your new database user. Therefore, choose the **Logout** button before you continue to work with the **SAP HANA Web-based Development Workbench**, where you need to log on again with the new database user.

Grant Roles

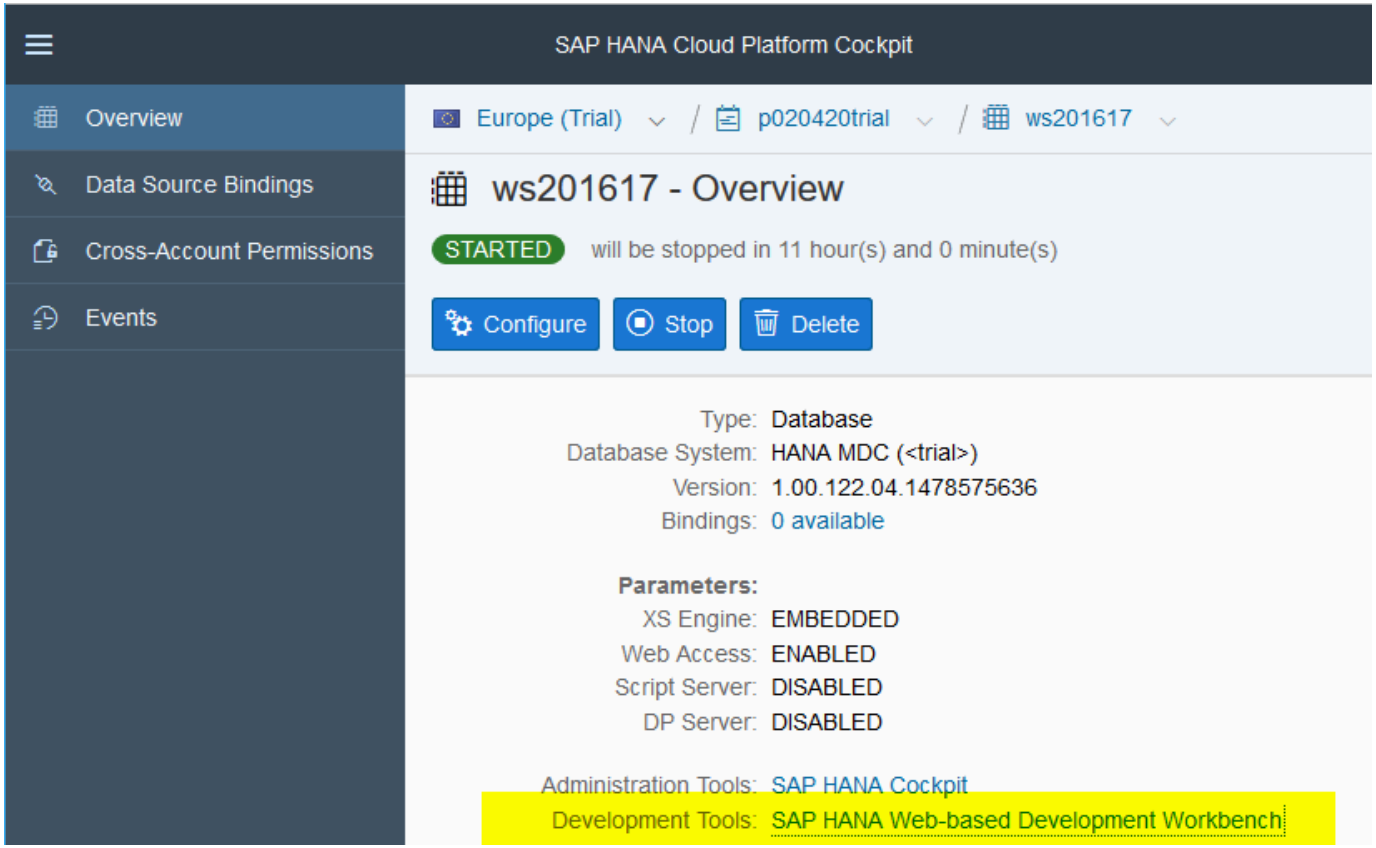

SAP HANA Web-based Development Workbench: Security
Security
Users
8820038
SYS
SYSTEM
XSSQLCC_AUTO_USER_3094F2
XSSQLCC_AUTO_USER_5E2492
XSSQLCC_AUTO_USER_D5D3B(
_SYS_AFL
_SYS_EPM
_SYS_REPO
_SYS_STATISTICS
_SYS_TASK
_SYS_WORKLOAD_REPLAY
_SYS_XB
Roles
*8820038
User
User Parameters
Application Role
8820038
Authentication
Password
Password*:
Confirm
Kerberos
External ID*:
Valid From:
Jan 3, 2017
11:05:59
Session Client:
Granted Roles
System Privileges
Obje
Role
sap.hana.xs.ide.roles::TraceViewer
sap.hana.xs.ide.roles::SecurityAdmin
Find Role
Type name to find a role:
xs.ide
5 item(s) matched
sap.hana.xs.ide.roles::CatalogDeveloper
sap.hana.xs.ide.roles::Developer
sap.hana.xs.ide.roles::EditorDeveloper
sap.hana.xs.ide.roles::SecurityAdmin
sap.hana.xs.ide.roles::TraceViewer
OK
Cancel

HANA Dev Workbench

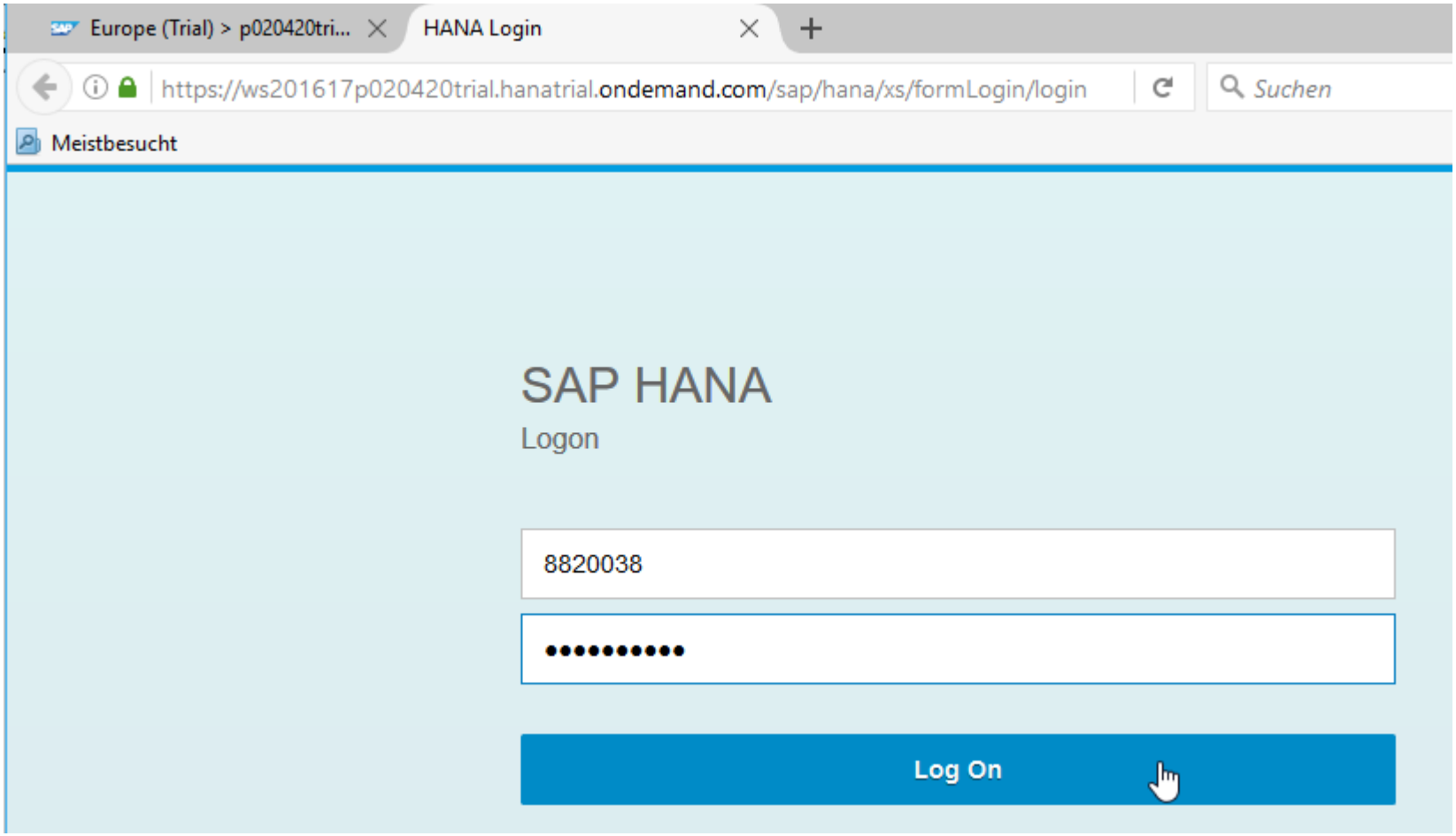
- Open the SAP HANA Cloud Platform cockpit.
- Select the relevant database from the list and choose **SAP HANA Web-based Development Workbench** under Development Tools.
- Log on with your newly created user.

- **Note**: If you log on to the SAP HANA Web-based Development Workbench for the first time, you are prompted to change your initial password.

HANA Dev Workbench Call



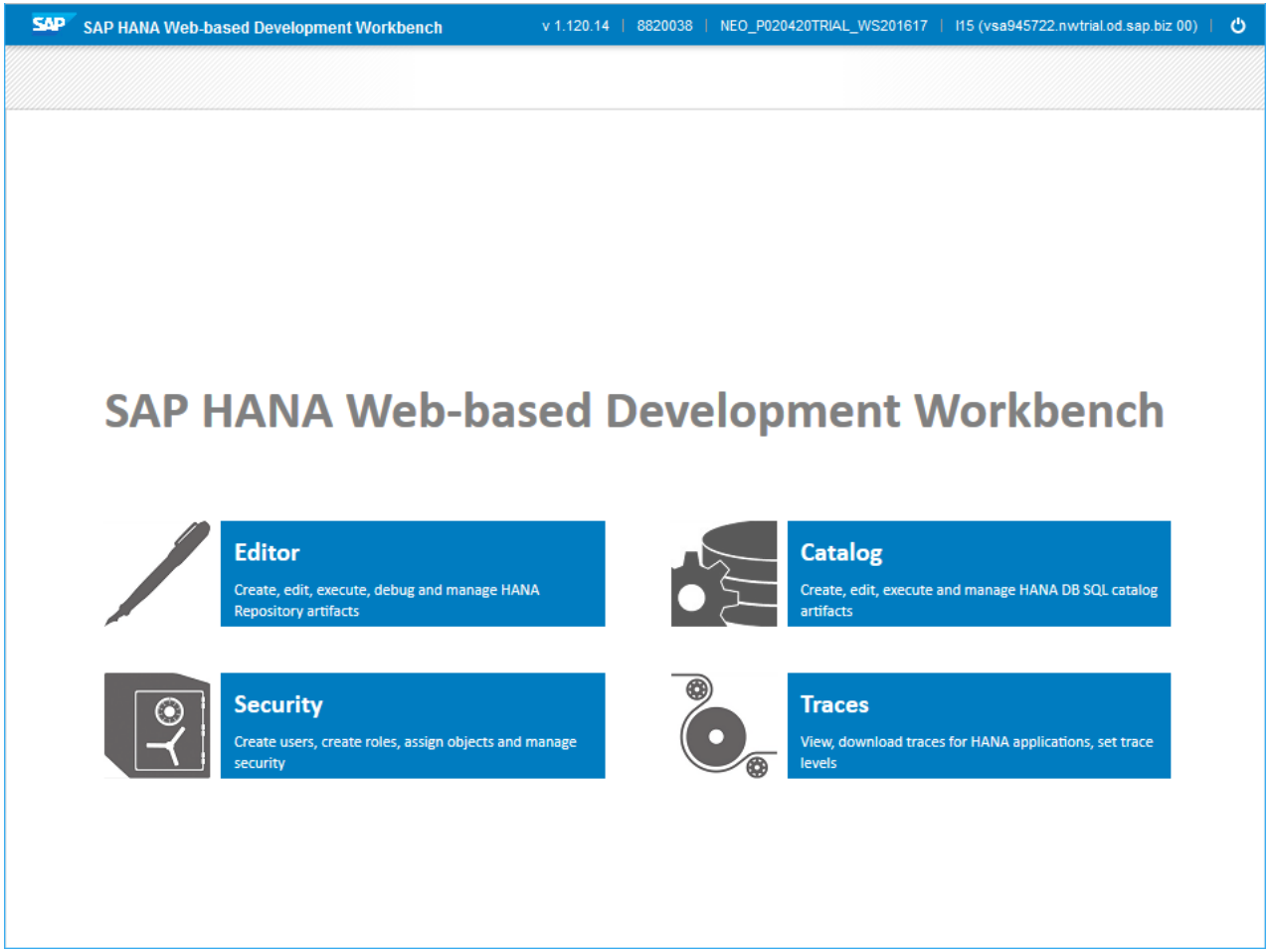
HANA Dev Workbench
Login with DB User



HANA Dev Workbench Start

- Select the Editor.
- The editor is displayed. The header shows the details for your user and database. Hover over the entry for the SID to view the details.

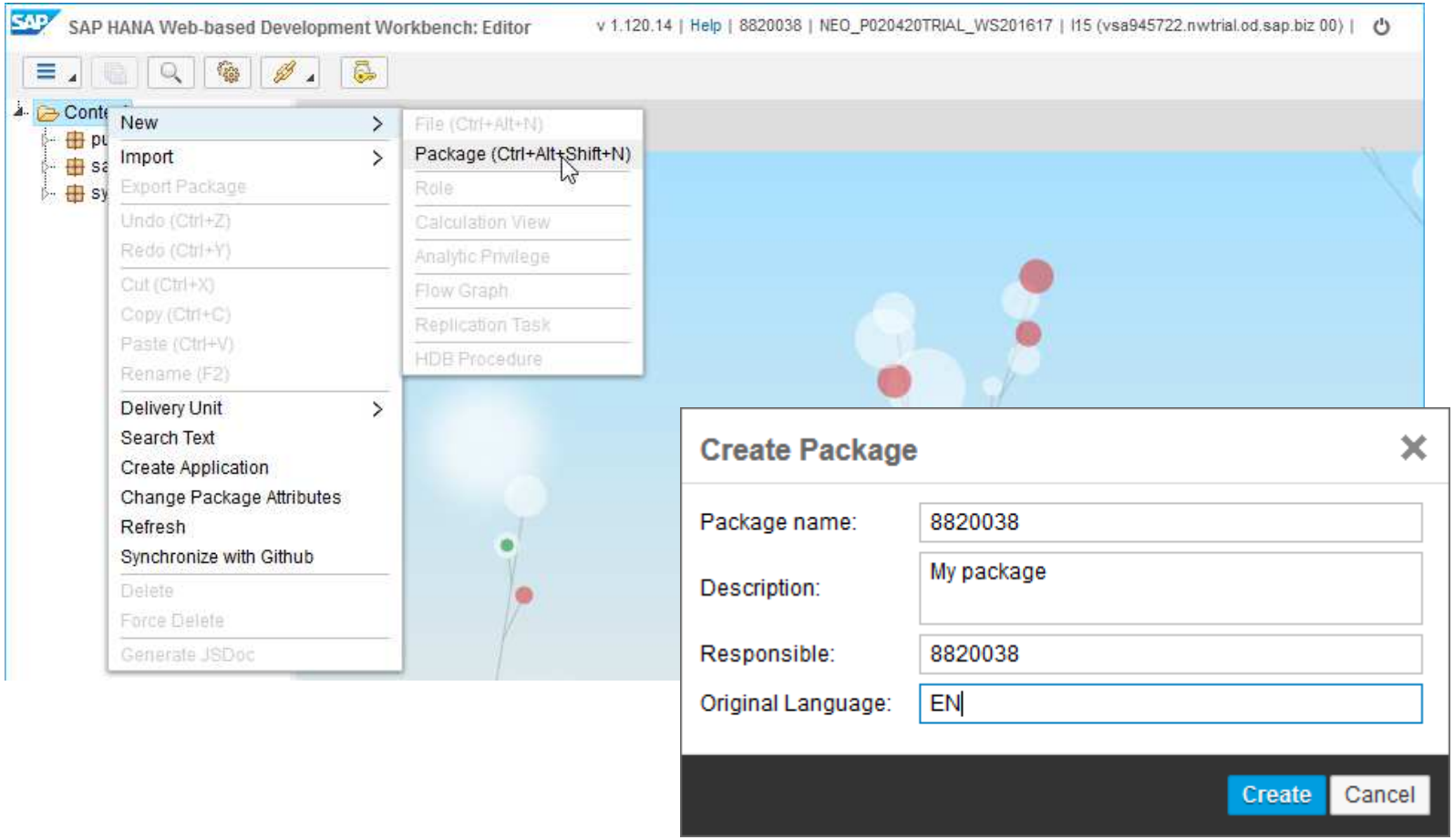
HANA Dev Workbench Start



HANA Dev Workbench
Create New Package

- Create a new package by choosing **New Package** from the context menu for the **Content** folder.
- Enter a package name (! No Umlaute!)
- The package appears under the Content folder node.

HANA Dev Workbench
Create New Package

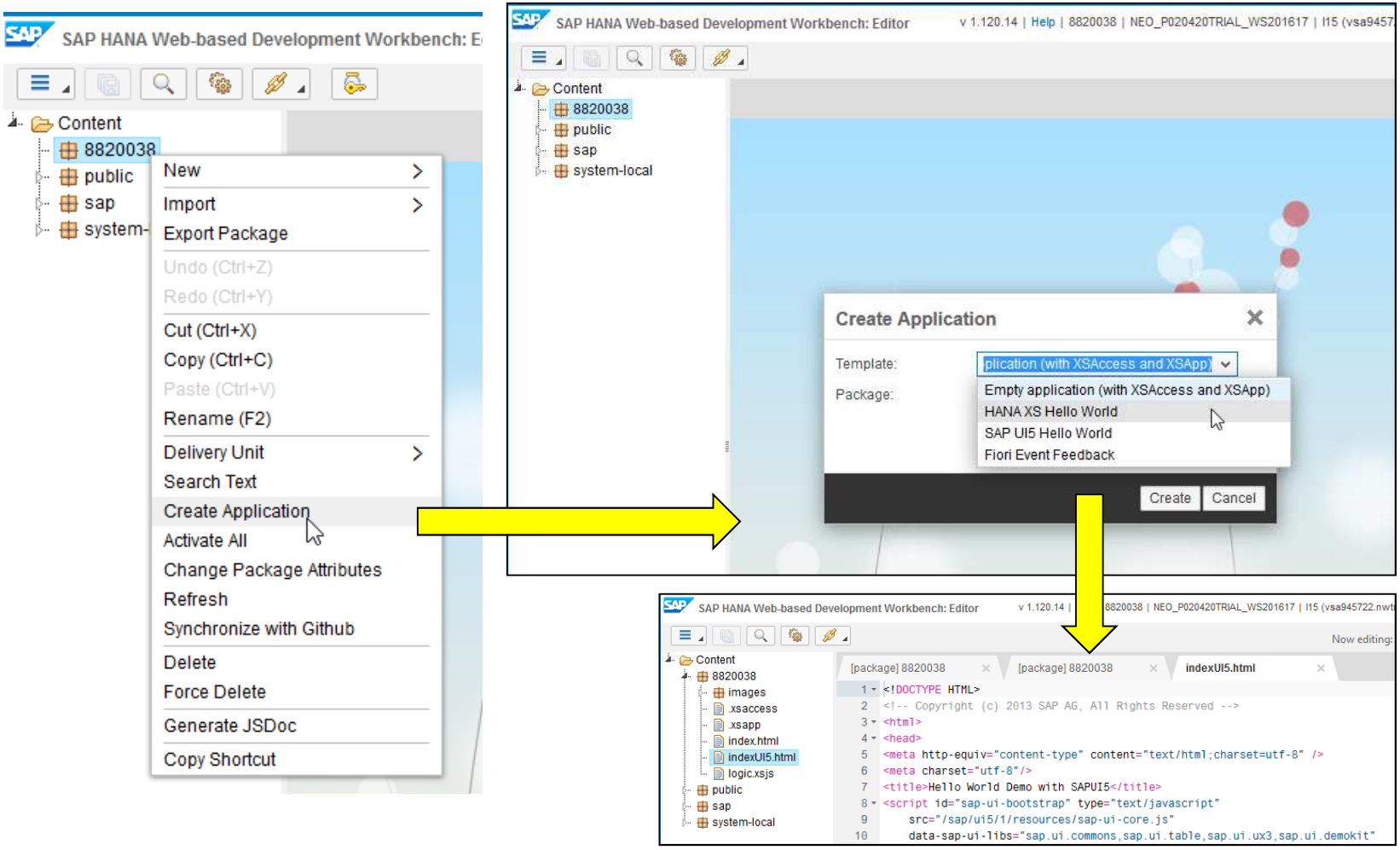


HANA Dev Workbench
Create New Application

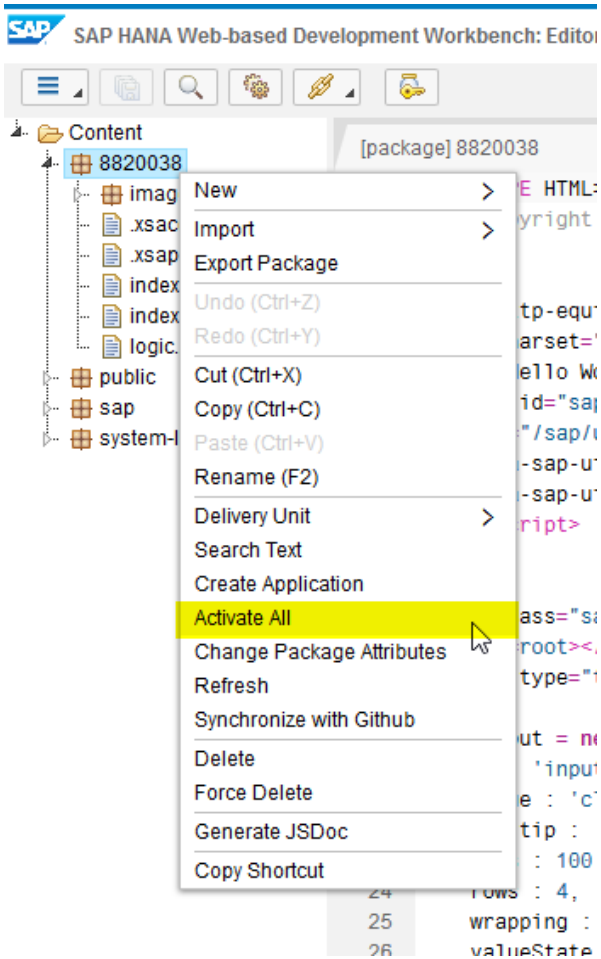
- From the context menu for the new package node, choose **Create Application**.
- Select **HANA XS Hello World** as template and choose Create.
- Open the files under the new package hierarchy to view them in the editor.

- Only if you are using an SAP HANA MDC database: From the context menu for the new package node, choose **Activate All**.

HANA Dev Workbench
Create New Application



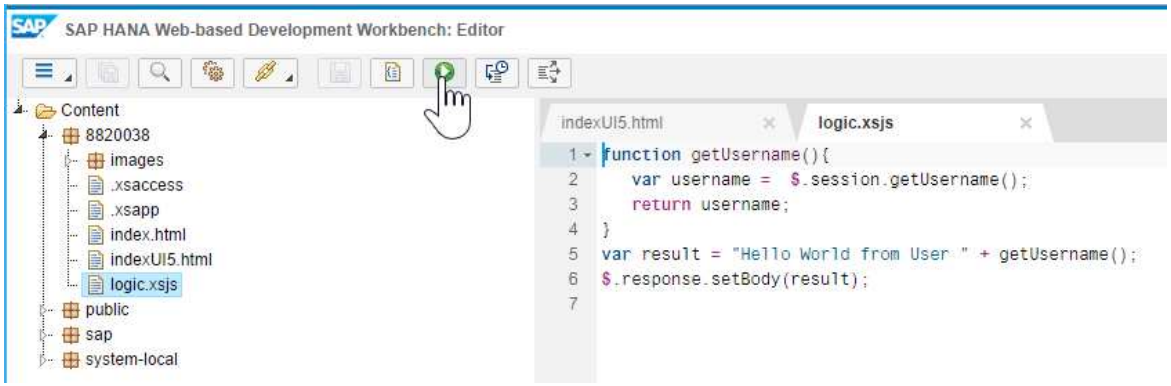
HANA Dev Workbench
Activate All



HANA Dev Workbench Test

- In the Editor of the SAP HANA Web-based Development Workbench, select the **logic.xsjs** file from the newly created package and choose **Run**.
- The program is deployed and displayed in the browser:

Hello World from User <Your User>.



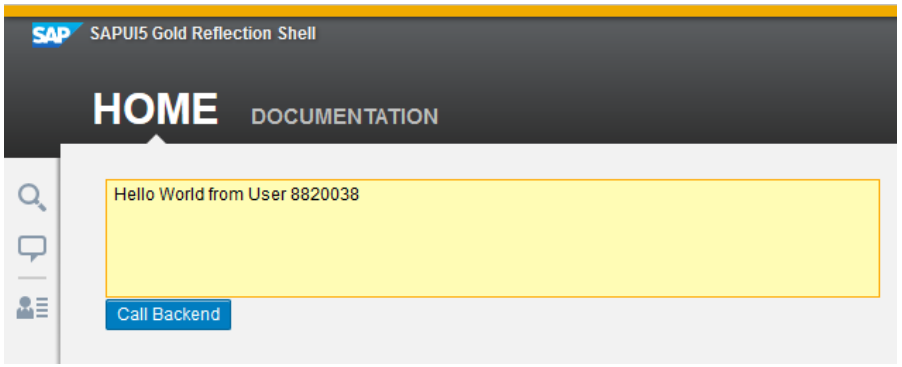
- Note: If you have used an SAP HANA XS database for creating your SAP HANA XS application, you can also launch your application from the SAP HANA Cloud Platform cockpit.

HANA Dev Workbench
Test Output

- logic.xsjs

Hello World from User 8820038

- indexUI5.html



- index.html

INTRODUCTION TO HANA

- Introduce SAP HANA
- Explain the system architecture and the impacts on development model
- Comparing the **XS Classic** and XS Advanced ecosystem

Introduction to HANA
(New) HANA Paradigm

„Do as much as you can in the database to get the best performance“

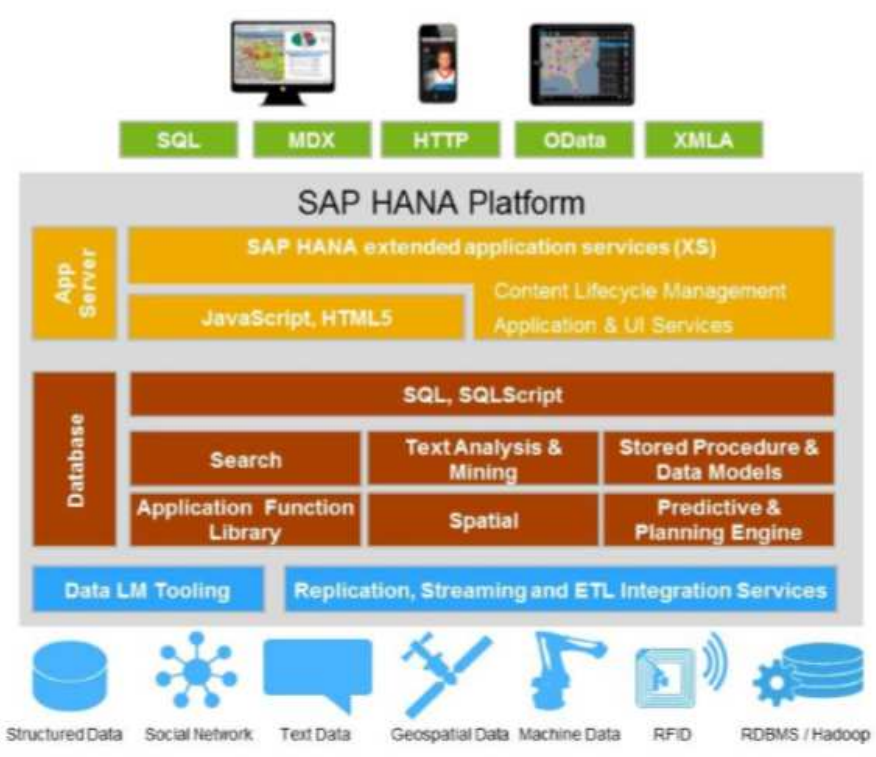
Introduction to HANA
HANA Platform Overview

Open, standard interfaces
- Supporting all types of devices

Integrated Application Server Components
- Native application infrastructure

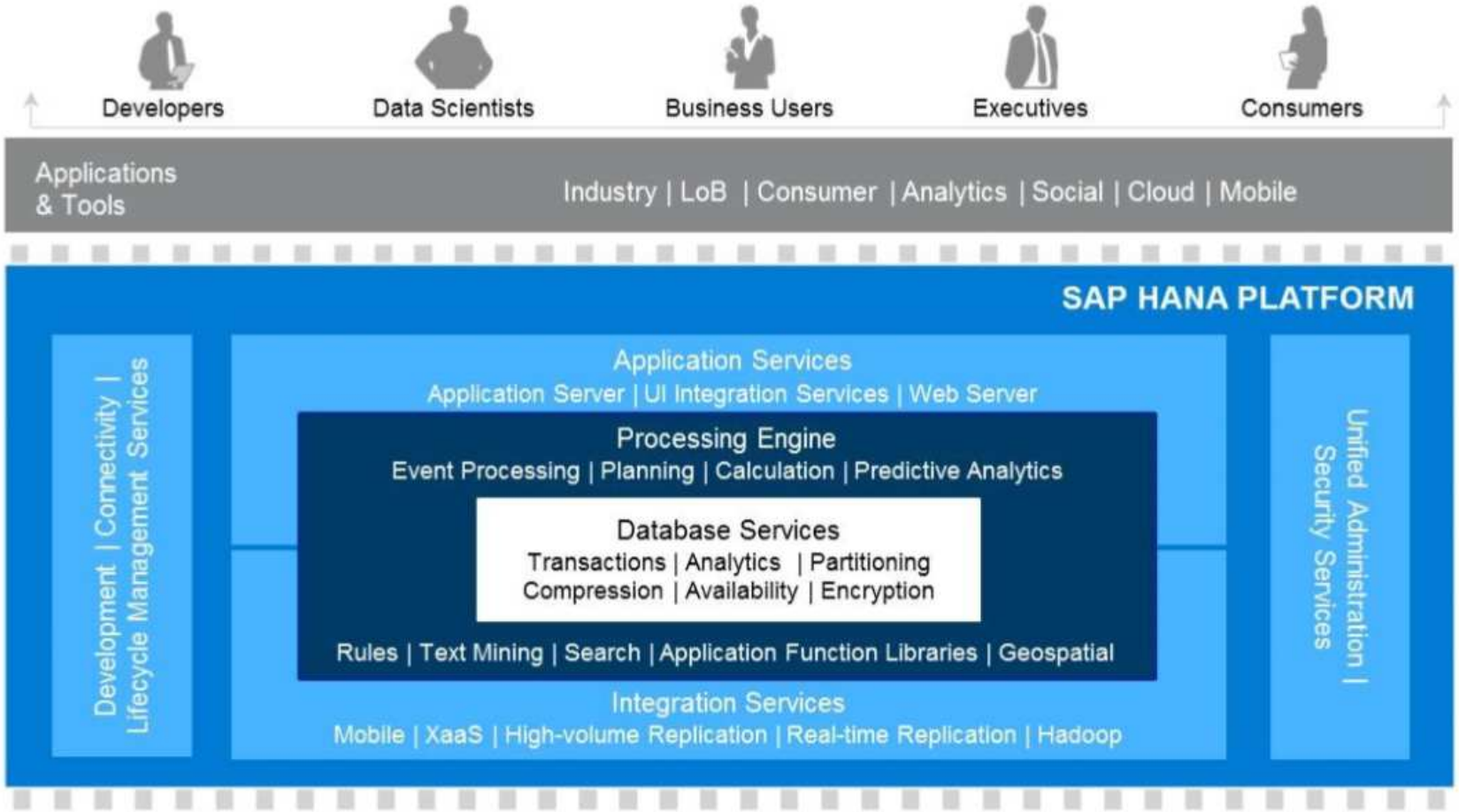
One DB for OLAP and OLTP Workloads
- With built-in functions for data-intensive processing

Data persistence and integration
- Integrate any data from any source
- Ready for Big Data Scenarios



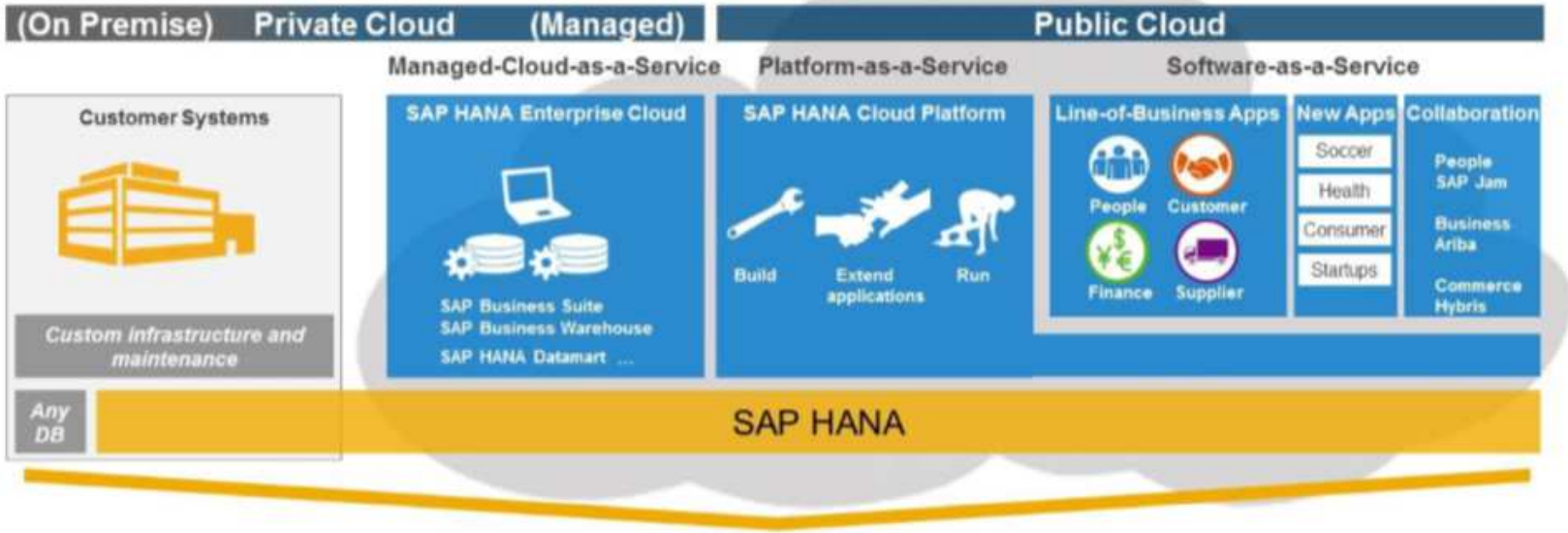
© SAP

Introduction to HANA
HANA Platform



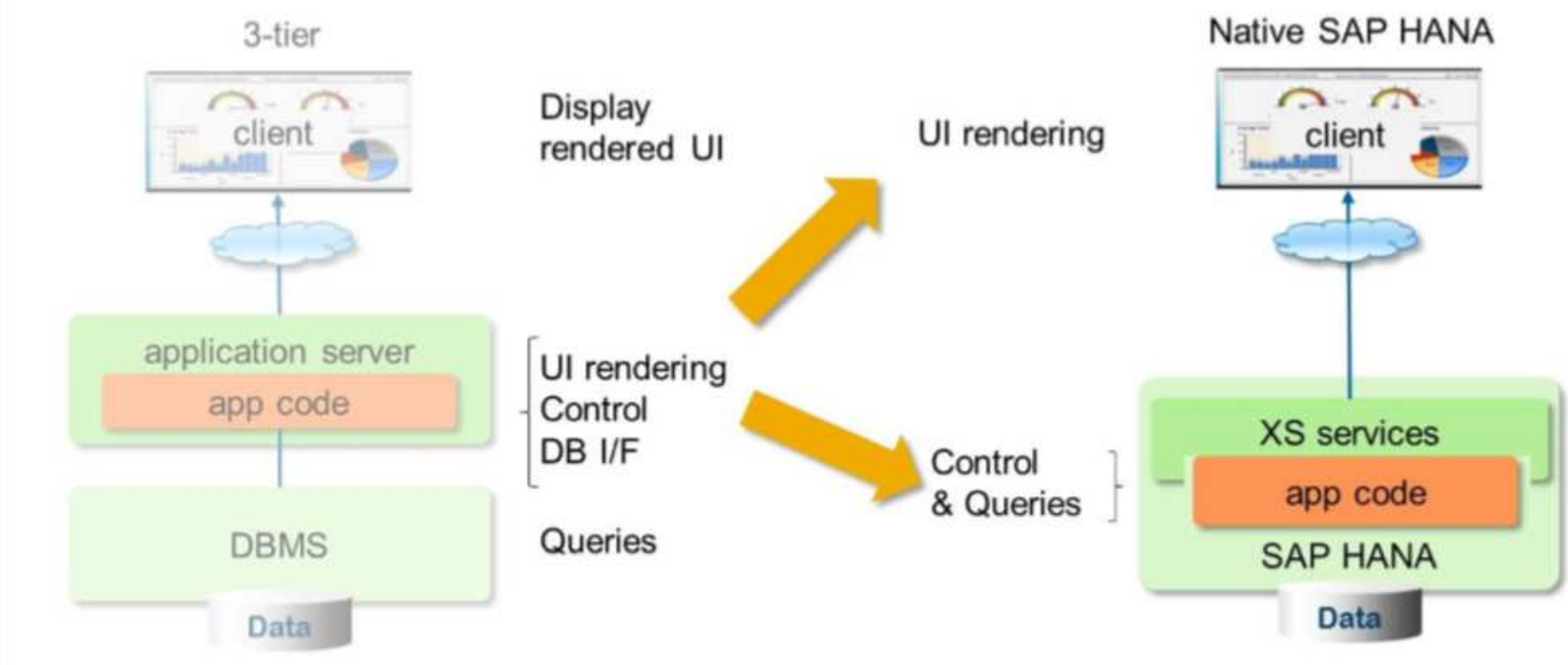
© SAP

Introduction to HANA
SAP Cloud



© SAP

Introduction to HANA
3-tier vs HANA Apps

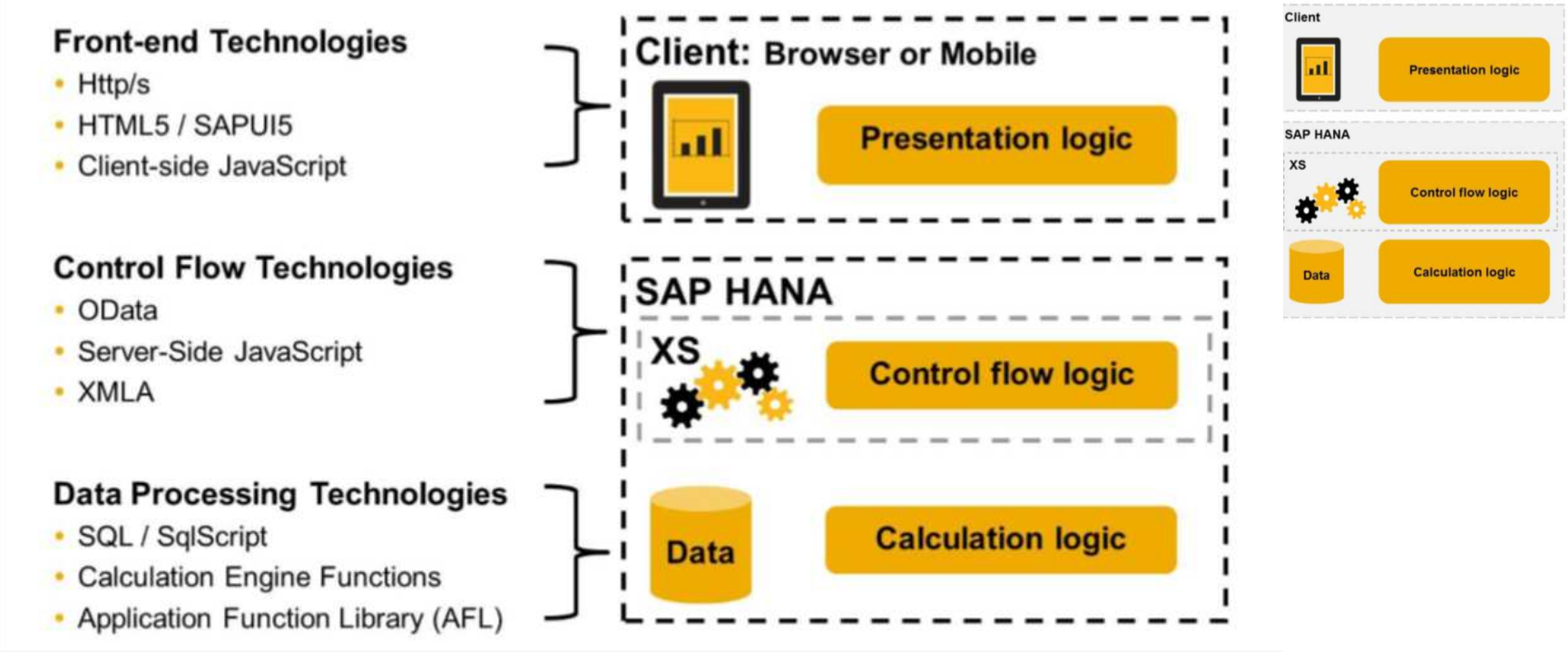


© SAP

Introduction to HANA

HANA Dev Model

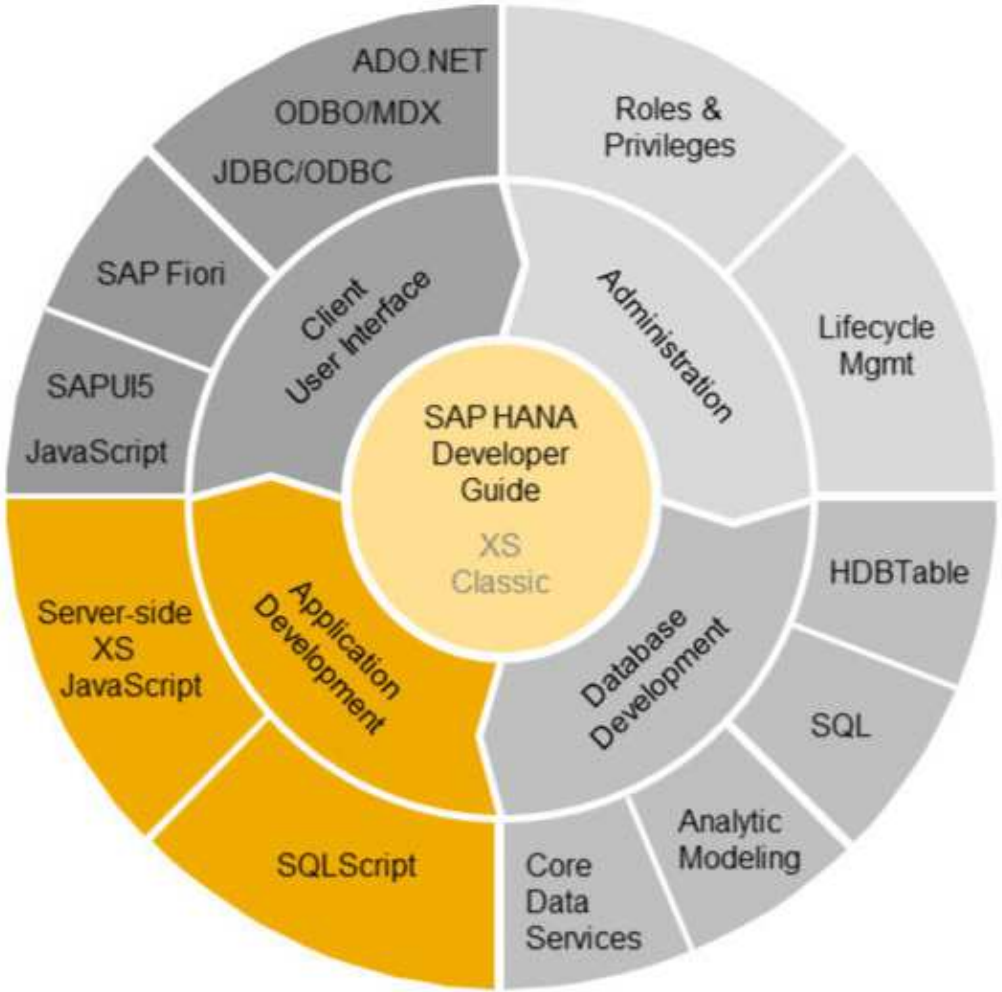
SAP HANA Extended Application Services



SAP HANA XS advanced model from SPS 11 on.

© SAP

Introduction to HANA
Dev in HANA XS Classical Model



DEVELOPMENT TOOLS

- Explain basic concepts of SAP HANA Studio
- Explain SAP HANA Web-based Development Workbench
- Explain XS Advanced Application Development Tools

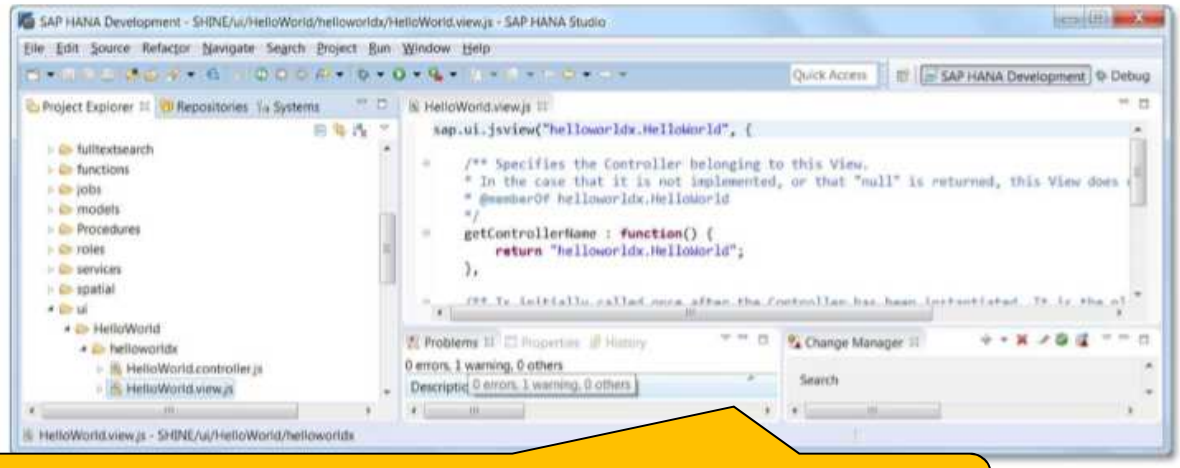
Development Tools
SAP HANA Studio

$$SAP\ HANA\ Studio$$
$$= EclipseIDE\ for\ Java\ EE\ Developers + SAP\ Plugins$$

Application Development in SAP HANA requires the use of an **Eclipse IDE**.

On top of Eclipse, the following SAP Plug-ins have to be installed:

- **SAP HANA Studio**: Used for the **sever-side** development.
- **SAPUI5 Development Tools**: Used for the **client-side** development.

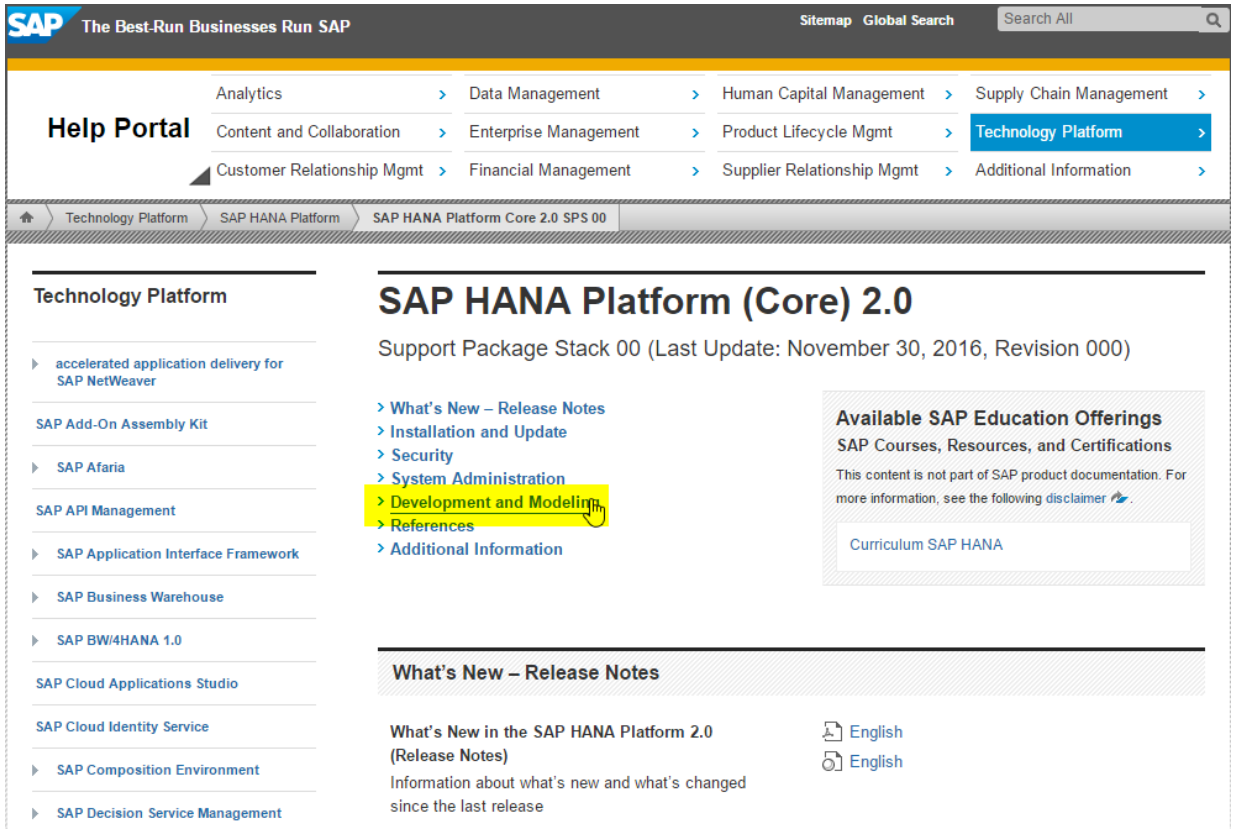


Update Site:
https://tools.hana.ondemand.com/[neon|mars]

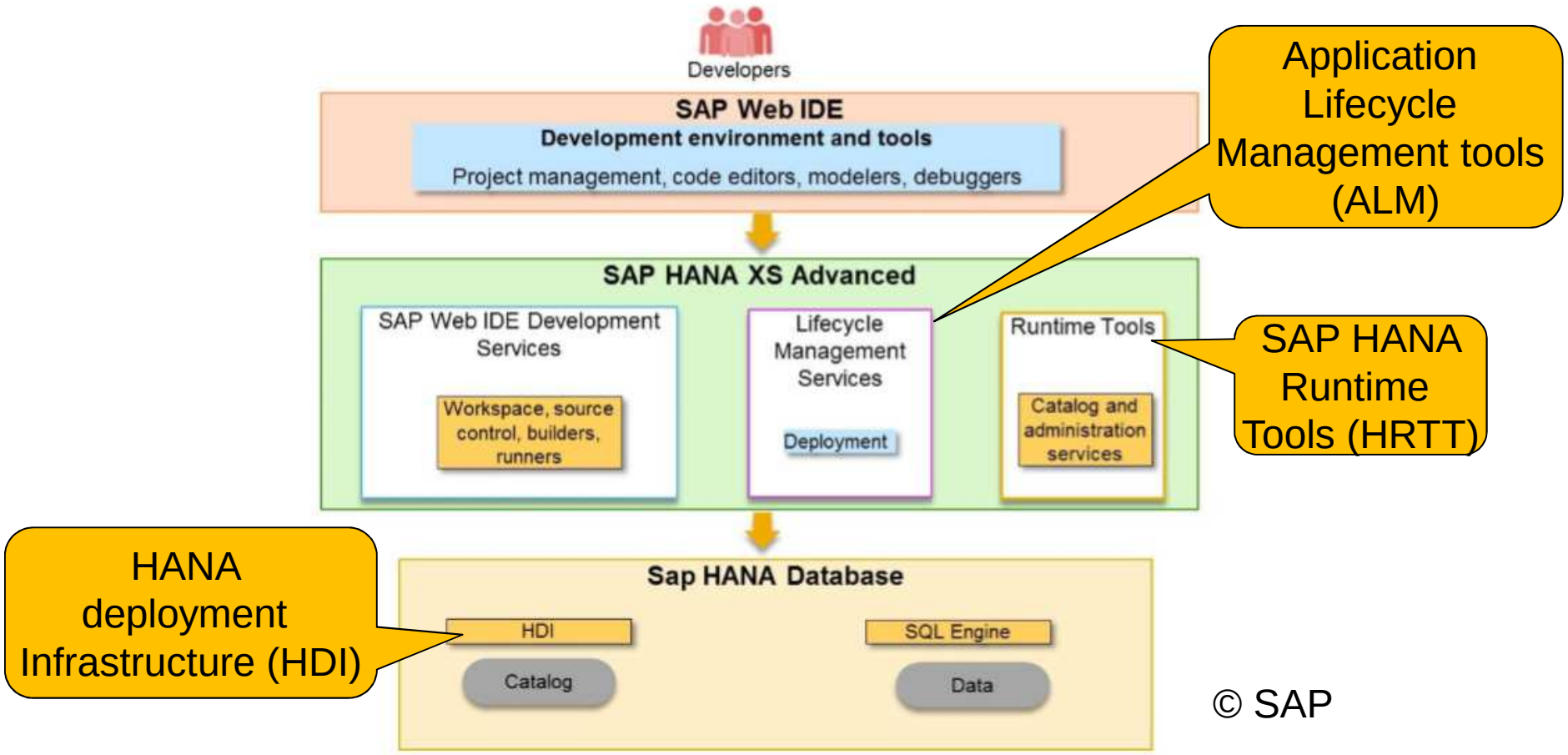
© SAP

Development Tools
SAP HANA Dev Guide

http://help.sap.com/hana_platform



XSA Application Development Tools



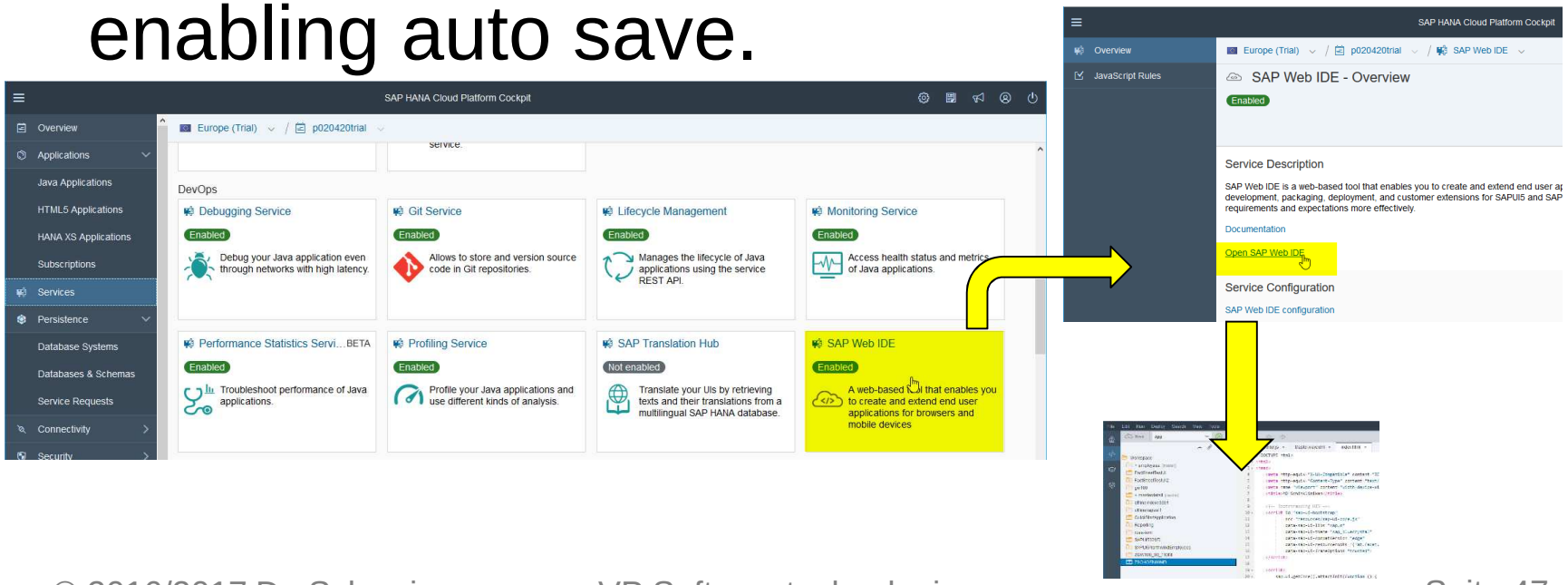
© SAP

Exercise
Web IDE

- Start SAP Web IDE for SAP HANA and Customize your code editor (That is how the Web IDE is going to look like in HANA Trial, once SP 11 is activated)
- Navigate to the SAP HANA Web-based Development Workbench

Solution WebIDE

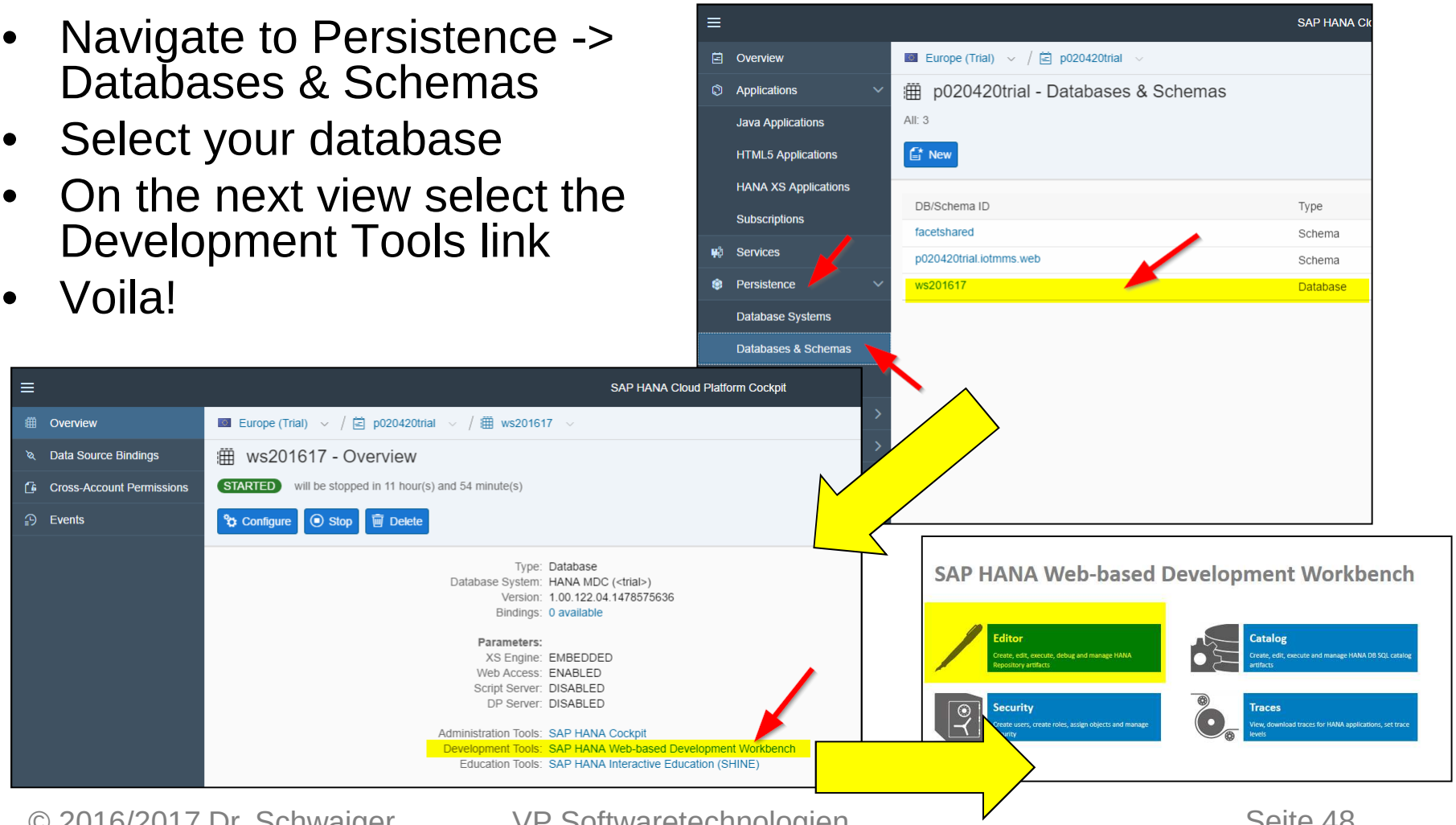
- Navigate to the Web IDE
- Customize your code editor by choosing a theme you like, changing the font size and enabling auto save.



Solution

Web-based Development Workbench

- Navigate to Persistence -> Databases & Schemas
- Select your database
- On the next view select the Development Tools link
- Voila!



BUILDING UIS WITH SAPUI5

JavaScript

- JS is the basis e.g. for Node.js or SAP HANA XSJS
- It is a scripting language
- It is loosely typed, i.e. determined during runtime

Exercise: Plain JS

1. Using Notepad, create a file named <matnr>-1.1.htm
2. In the file, copy the following content:
<script> var a = 2; var b = 3: var c = a+b; alert("c = "+c); </script>
3. Save the file.
4. Open the file using Chrome Developer Tools.
5. Set a breakpoint in the first line after the <script> tag. Execute the script and observe the execution stopping at the breakpoint.
6. Set 3 watch expressions to display the content of variables a, b, and c.
7. Execute the script step-by-step until it ends.

Function

Create an HTML file named <matnr>-1.2.htm containing a JavaScript program based on the following description:

1. Create a JavaScript function named Multiply with 2 input parameter. The function returns the sum of the two parameters.
2. Create 2 numeric variables, a and b, with values 4 and 5.
3. Display the result in an alert window.
4. Execute the multiplication of the variables using the Multiply function.

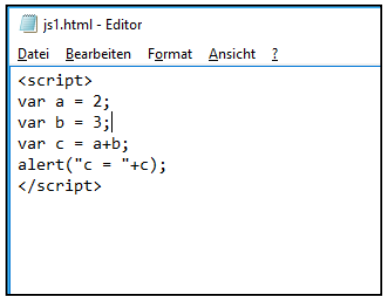
Object

Create an HTML file named <matnr>-1.3.htm containing a JavaScript program based on the following description:

1. Create an object with the following 3 items: ● firstName with value “John” ● lastName with value “Smith” ● a function named FullName, returning name and surname.
2. Execute the function and display the full name in an alert window.

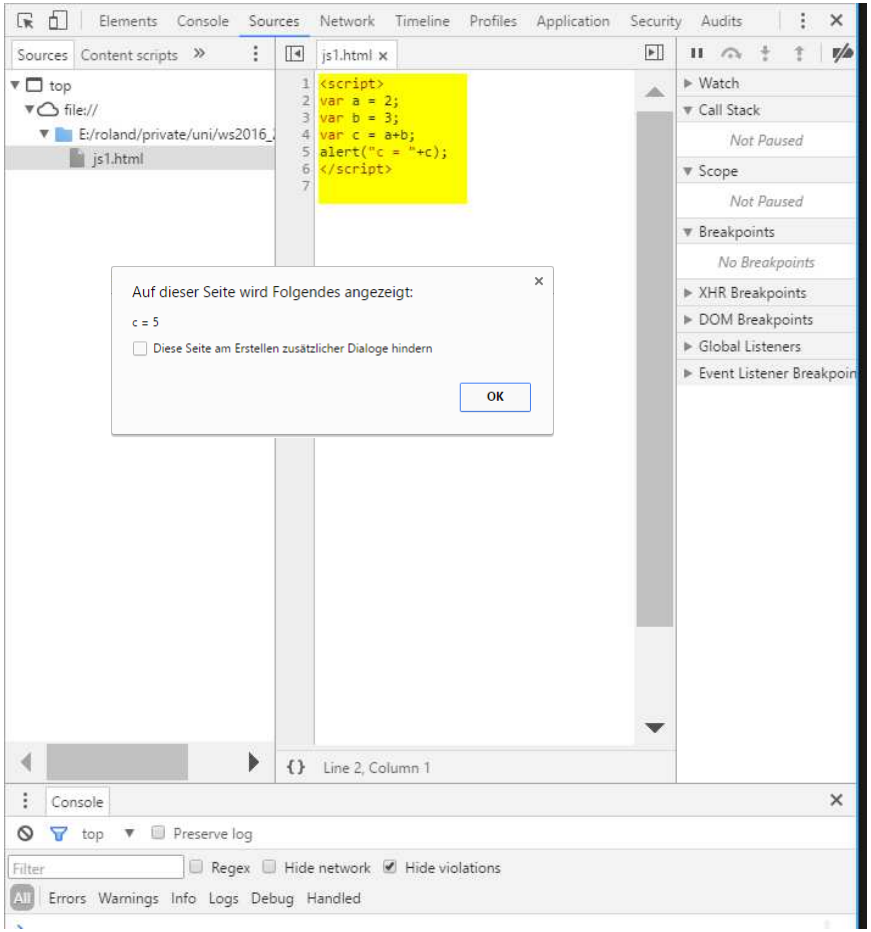
Solution
Plain JS

```
<script>
    var a = 2;
    var b = 3:
    var c = a+b;
    alert("c = "+c);
</script>
```




```
<script>
    function Multiply(x,y) { return x*y; }

    var a = 4, b = 5;
    alert("Result = " + Multiply(a, b));

</script>
```

```
<script>

    obj = {
        firstName: "John",
        lastName: "Smith",
        FullName: function(){
            return this.firstName + " " + this.lastName
        }
    };
    alert( obj.FullName() );

</script>
```



Create a SAPUI5 Project

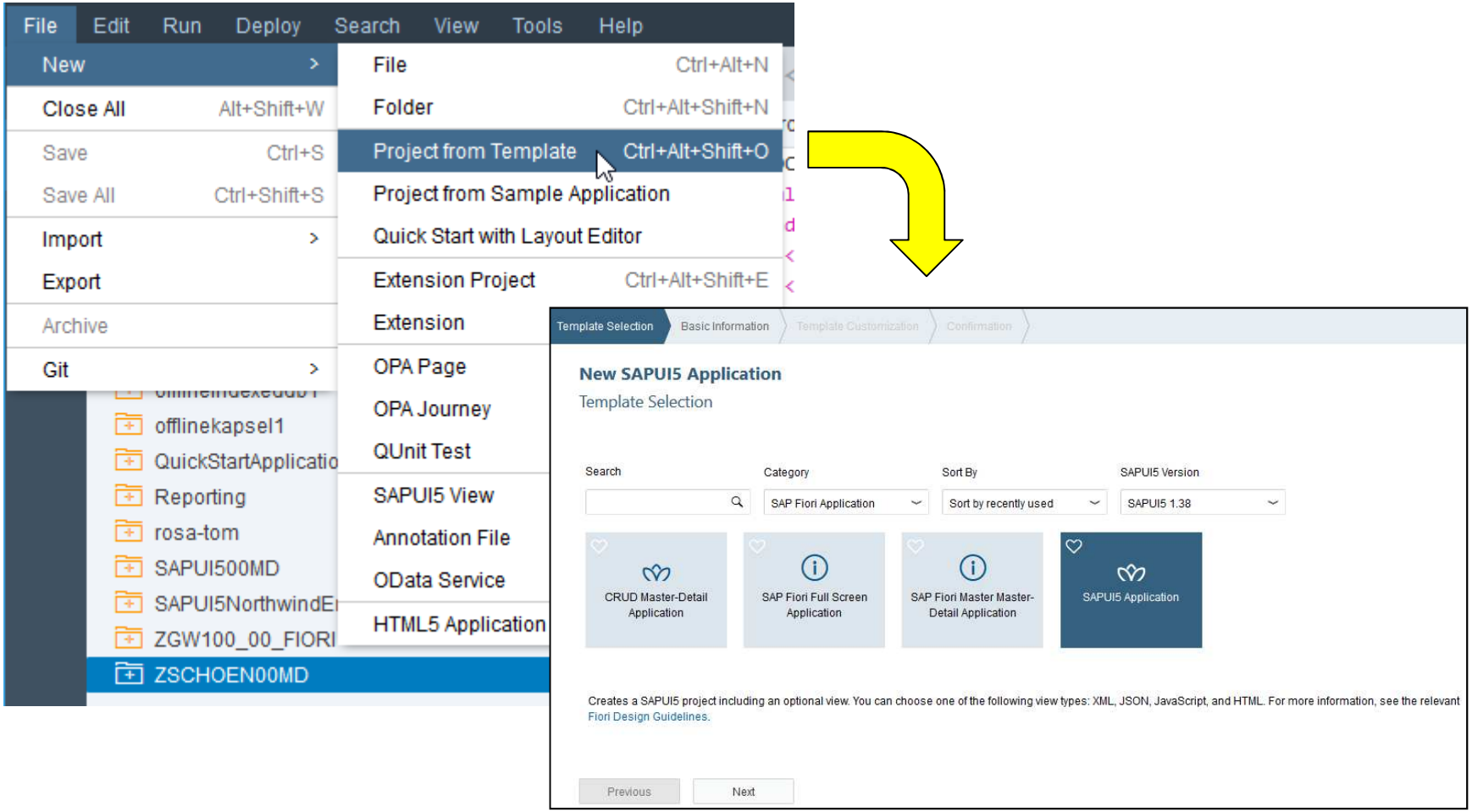
Attention!

In the next steps we are using the **WEB IDE not** in the context of the **XS Applications** but in the context of cloud development of SAP UI5 development.

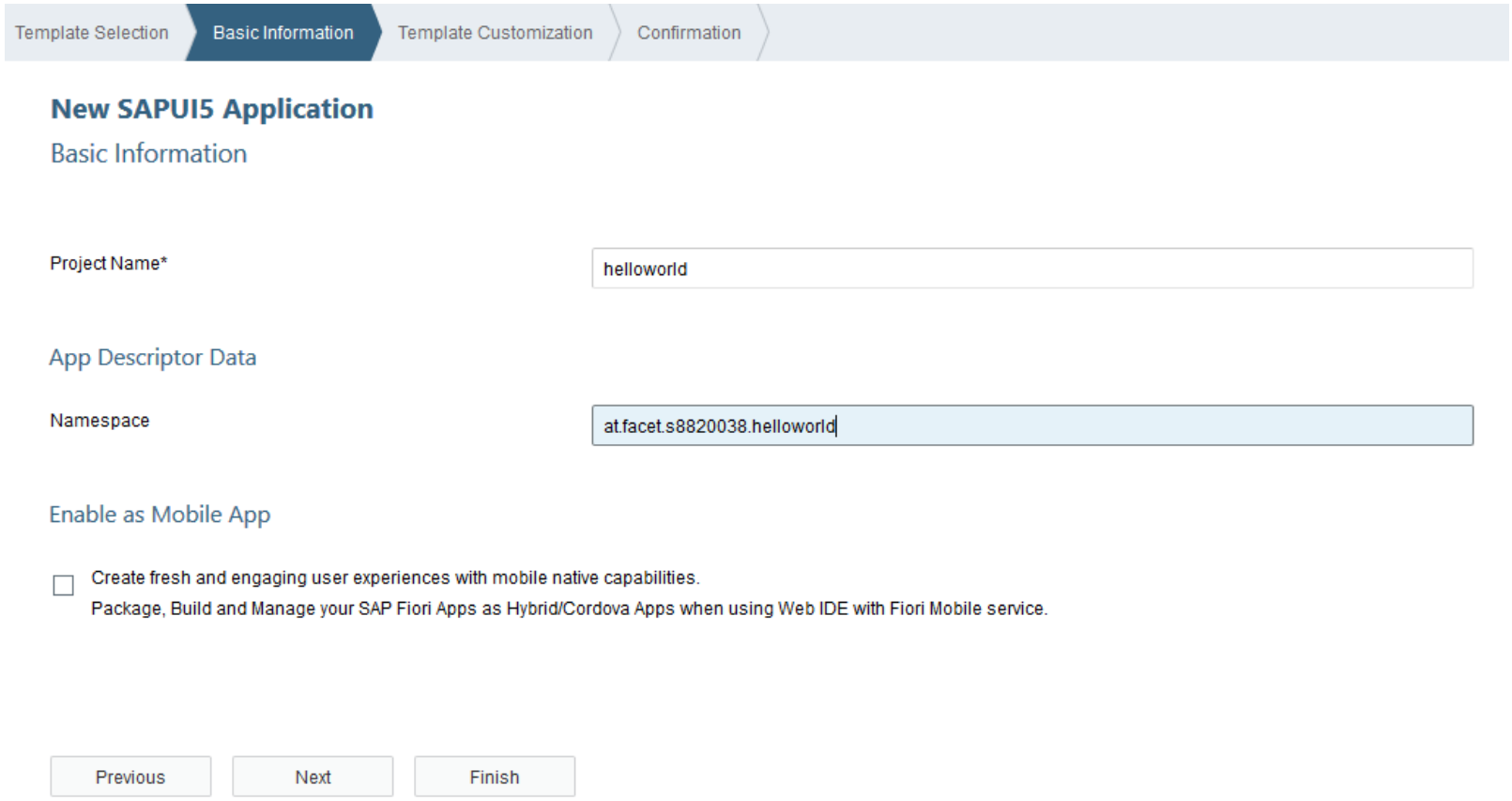
Lateron we are switching to XS applications development and we are going to use the **Web based development tools** for XSC Applications.

!!!2 different things!!!

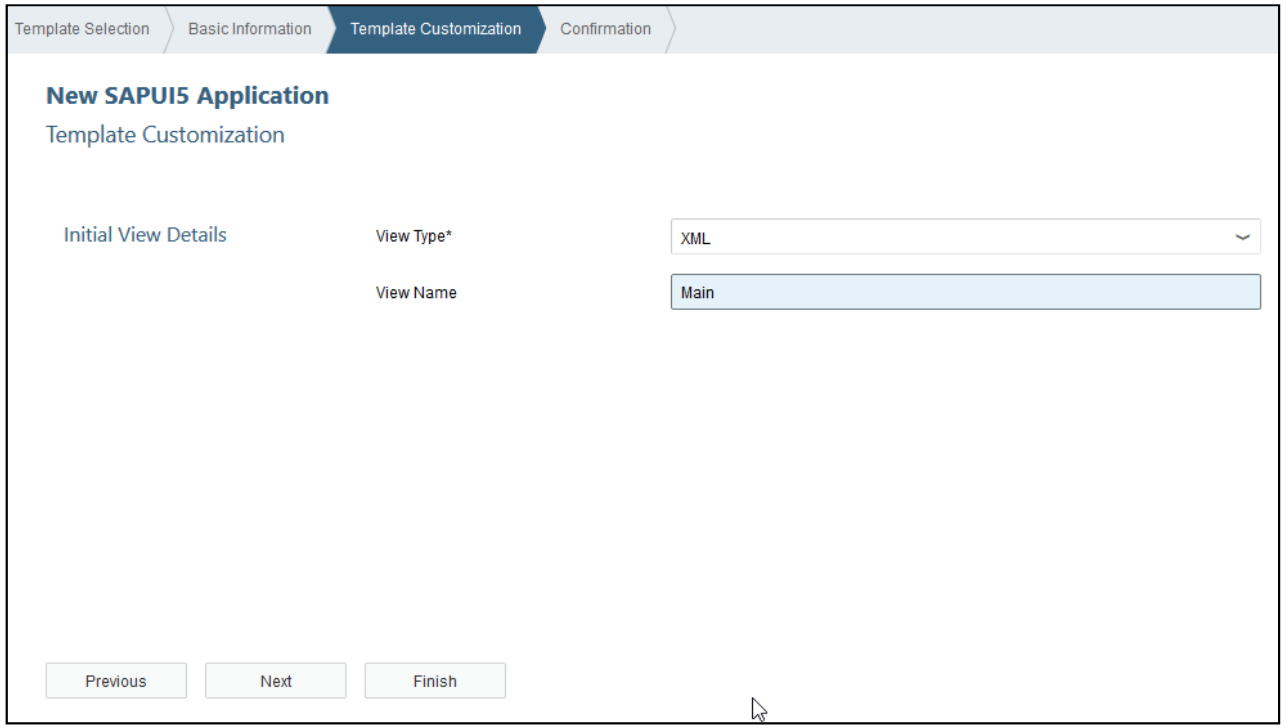
UI5
WebIDE
Create a SAPUI5 Project



UI5
WebIDE
Create a SAPUI5 Project

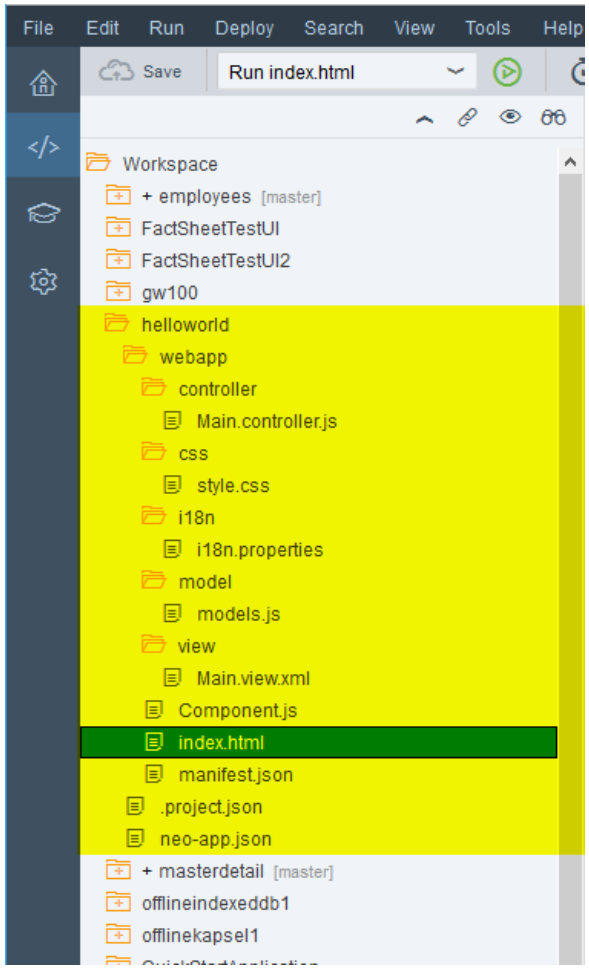


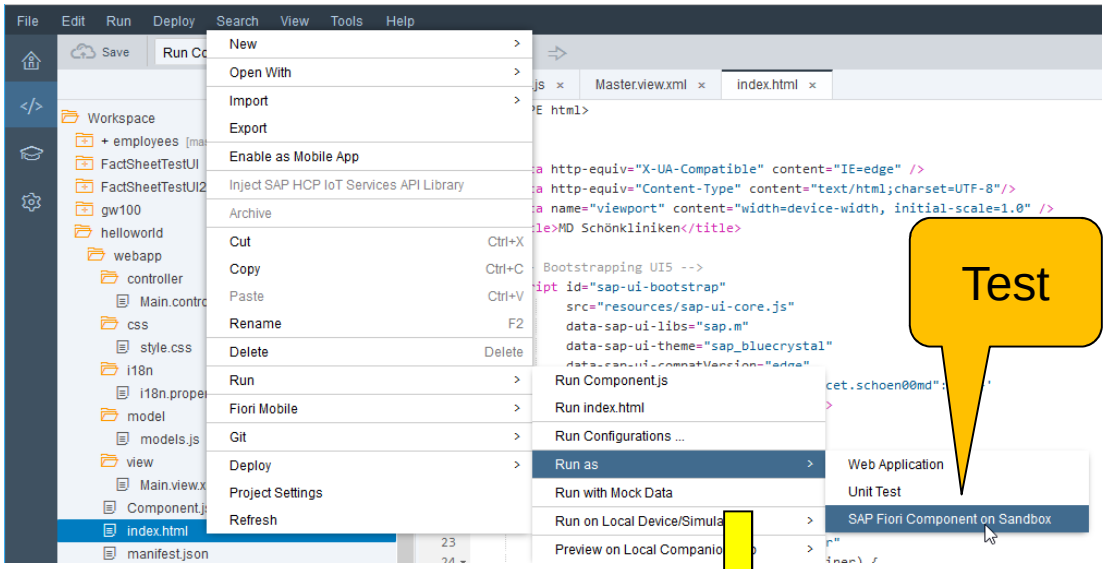
UI5
WebIDE
Create a SAPUI5 Project

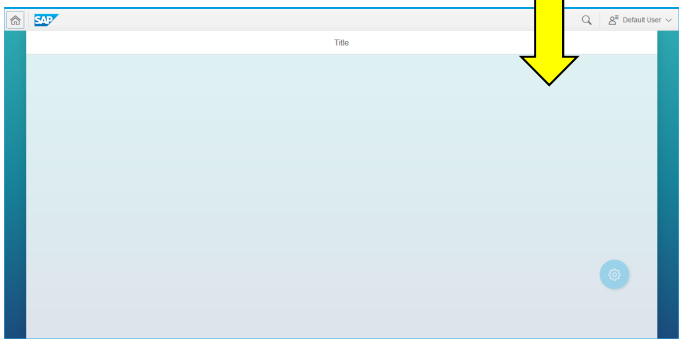


- Click Next and then Finish in the next screen

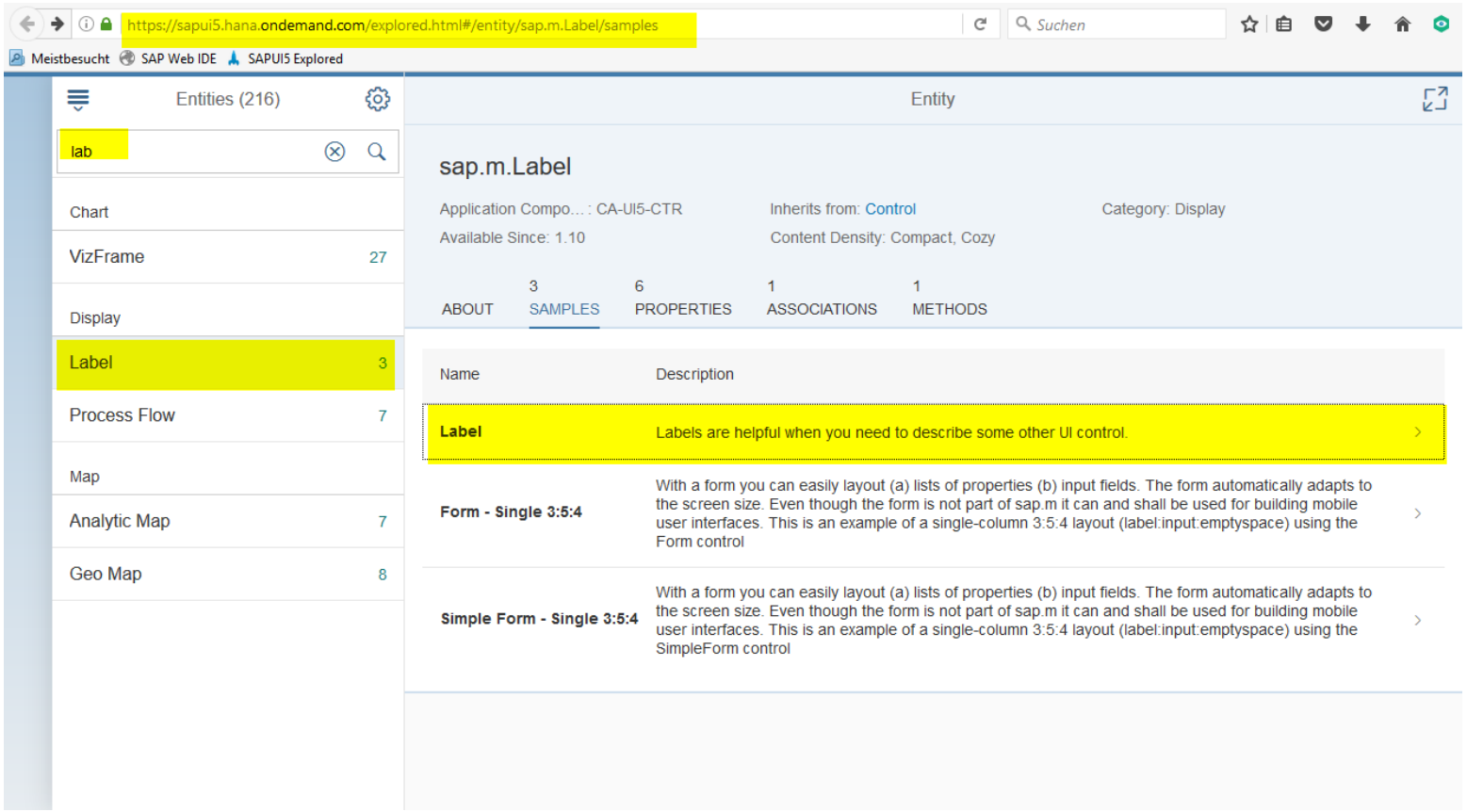
UI5
WebIDE
Create/Test a SAPUI5 Project



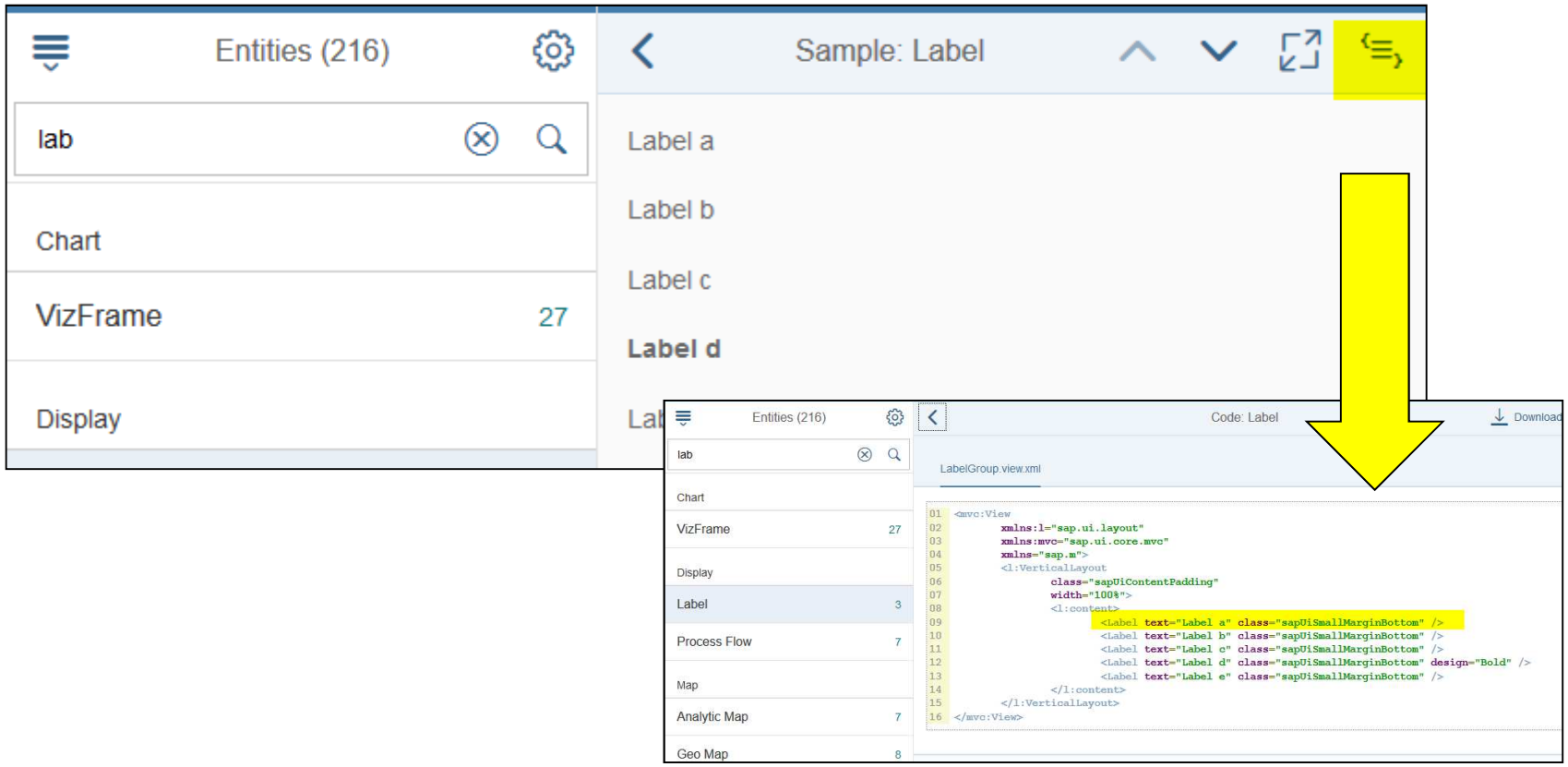




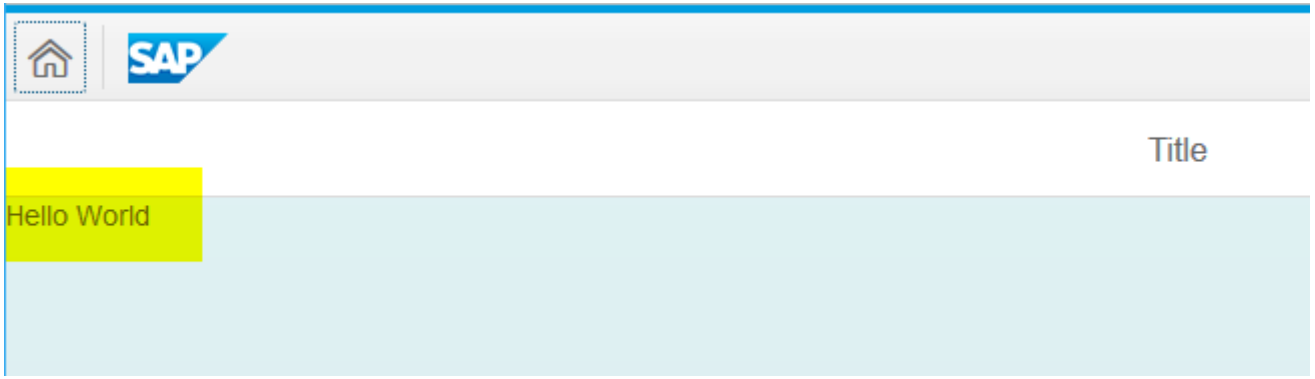
UI5
WebIDE
Create Label and Textfield



UI5
WebIDE
Create Label and Textfield



UI5
WebIDE
Create Label and Textfield

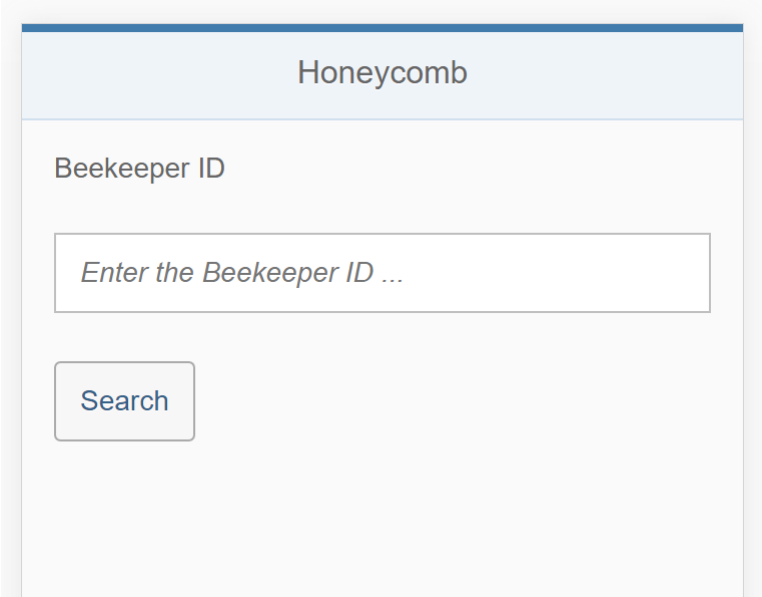


Exercise
WebIDE
SAPUI5

- Create a HelloWorld SAPUI5 Application with the WebIDE
- Insert a Label
- Insert an Inputfield
- Insert a Button

Solution WebIDE SAPUI5

- Just follow the previous steps



Solution WebIDE SAUI5 View

```
<mvc:View controllerName="at.facet.hwHellloWorld.controller.Main"
   xmlns:html="http://www.w3.org/1999/xhtml" xmlns:mvc="sap.ui.core.mvc"
                  xmlns:l="sap.ui.layout" displayBlock="true" xmlns="sap.m">
                  <App>
                                 <pages>
                                                <Page title="{i18n>title}">
                                                               <content>
                                                                              <l:VerticalLayout class="sapUiContentPadding"
width="100%">
                                                                                             <Label text="Beekeeper ID"
class="sapUiSmallMarginBottom"/>
                                                                                             <Input type="Text"
                                                                                                    class="sapUiSmallMarginBottom"

placeholder="{i18n>placeholderBeekeeperID}"/>
                                                                                             <Button text="Search"
press="onPress"></Button>
                                                                              </l:VerticalLayout>
                                                               </content>
                                                </Page>
                                 </pages>
                  </App>
</mvc:View>
```

Solution
WebIDE
SAUI5 Controller

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

		return Controller.extend("at.facet.hwHellloWorld.controller.Main", {

			onPress: function(oEvent){
				debugger;
				var value =
this.getView().byId("beekeeperID").getValue();
				alert("Hello I am here" + value);
			}

		});
});
```

SAPUI5 Structure

`index.html` is the default starting point of the application. It contains 3 main areas:

- Bootstrap - Contains configuration including the libraries to be loaded, resource root location, and theme.
- Application – Initial construction of application including `placeAt` method to attach application into primary UI Area
- UI-Area – Contains primary UI anchor.

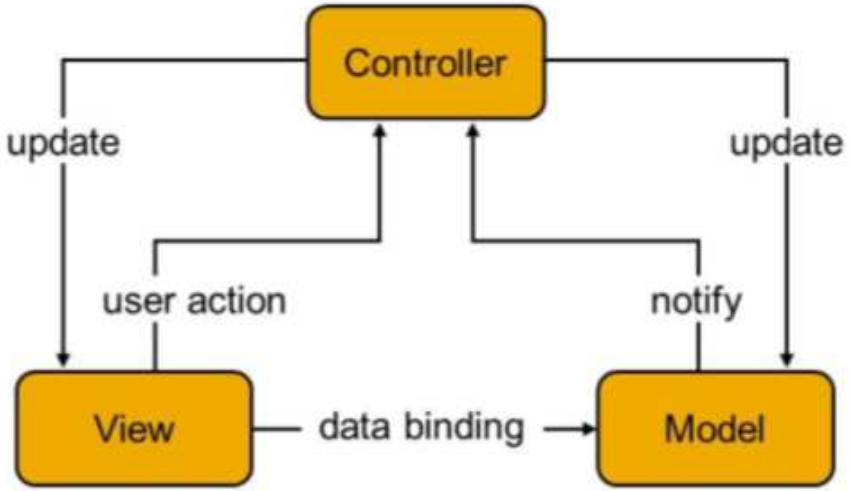
Best practice keeps minimal coding in `index.html`

© SAP

SAPUI5 MVC

The Model View Controller (MVC) concept is used in SAPUI5 to separate the representation of information from the user interaction.

- Model – Manages application data
- View – Defines and renders the UI
- Controller – reacts to view events and user interaction by modifying the view and model



The separation has the following advantages:

- It provides better readability, maintainability, and extensibility.
- It allows you to change the view without touching the underlying business logic and to define several views of the same data.

Views and controllers usually form a 1:1 relationship (but standalone controllers and views are possible).

© SAP

SAPUI5 Model

A **model** holds the data, provides methods to fetch the data, and to set and update data. The model types are as follows:

- **JSON** – Client-side model. Intended for small datasets and can be used to bind controls to JavaScript object data.
- **XML** – Client-side model. Intended for small datasets. Does not support server based paging or loading of deltas.
- **Resource** – Handles resource bundles. Mainly for text translation.
- **oData** – Server-side model. Binds controls to oData services. Client only knows currently fetched rows and fields. Sorting and filtering only on the server.

Multiple models and types are possible within a view, and the models may be set at different levels such as at the application level or for an individual control.

The data binding is not dependent on the type of model used.

© SAP

SAPUI5 View

The view is responsible for defining and rendering the UI. SAPUI5 supports the following predefined view types:

- **XML** – The user interface is defined in an XML file or string. (Supports a mix of XML and plain HTML.) Preferred approach for Fiori apps.
- **JSON** – The user interface is defined in a file or string in JSON format.
- **JS** – The user interface is constructed with JavaScript.
- **HTML** – The user interface is constructed with HTML.

Custom view types can be defined by extending base class sap.ui.core.mvc.View

```
<core:View xmlns:core="sap.ui.core"
    xmlns:mvc="sap.ui.core.mvc"
    xmlns="sap.m" controllerName="sap.sapx05.view.View1"
    xmlns:html="http://www.w3.org/1999/xhtml">
    <Page title="Exercise 5 - XML View">
        <content>
          <Label id="lblFirstName" text="First Name:" />
          <Input id="inpFirstName" />
          <Label id="lblLastName" text="Last Name:" />
          <Input id="inpLastName" />
          <Button id="btnSubmit" text="Submit" press="onBtnClick" />
        </content>
    </Page>
</core:View>
```

```
{
    "Type": "sap.ui.core.mvc.JSONView",
    "controllerName": "sap.sapx05.view.View1JSON",
    "content": [{
        "Type": "sap.m.Page",
        "title": "Exercise 5 - JSON View",
        "content": [
            {"Type":"sap.m.Label",
            "id": "lblFirstName",
            "text": "First Name:"},
            {"Type":"sap.m.Input",
            "id": "inpFirstName",
            "text":""},
            {"Type":"sap.m.Label",
            "id": "lblLastName",
            "text": "Last Name:"},
            {"Type":"sap.m.Input",
            "id": "inpLastName",
            "text":""},
            {"Type":"sap.m.Button",
            "id":"btnSubmit",
            "text": "Submit"}
        ]
    }]
}
```

SAPUI5 View Conventions

- View names are capitalized
- All views are stored in the view folder
- Names of XML views always end with *.view.xml
- The default XML namespace is sap.m
- Other XML namespaces use the last part of the SAP namespace as alias (for example, mvc for sap.ui.core.mvc

SAPUI5 Controller

SAPUI5 uses the controller to separate the view logic from the model logic. It has the following lifecycle hooks:

- **onInit()** – Called when a view is instantiated and its controls (if available) have already been created; used to modify the view before it is displayed to bind event handlers and do other one-time initialization.
- **onExit()** – Called when the view is destroyed; used to free resources and finalize activities.
- **onAfterRendering()** – Called when the view has been rendered and, therefore, its HTML is part of the document; used to do post-rendering manipulations of the HTML. SAPUI5 controls get this hook after being rendered.
- **onBeforeRendering()** – Invoked before the controller view is re-rendered and **not** before the first rendering; use onInit() for invoking the hook before the first rendering.

A controller can also define additional methods that serve as event handlers or additional functionality offered by the controller.

SAPUI5 Controller Conventions

- Controller names are capitalized
- Controllers carry the same name as the related view (if there is a 1:1 relationship)
- Event handlers are prefixed with **on**
- Controller names always end with *.**controller.js**

SAPUI5
Common UI Controls

- Mobile:
  https://sapui5.hana.ondemand.com/explored.html
- General (old):
  https://sapui5.hana.ondemand.com/#content/Controls/index.html

SAPUI5
Control Samples

SAPUI5 provides different types of UI-controls:

- Simple controls
- Value holders
- Layouts
- Complex controls
- Dialogs
- UX3 controls

© SAP

SAPUI5
Control in XML View

```
<Control propertyX="value" …
         propertyY="{ subpropX:'value' subpropY:'value' …}"
         propertyZ="{ subpropX:{ sspX:'val' sspY:'val' …} …}"
>
<aggregation1></aggregation1>
<aggregation2></aggregation2>
…
</Control>
```

- *Control* – Name of the control.
- *propertyX Y Z* – Property for the specific control. See Documentation. Could also be an event.
- *subpropX Y sspX Y* – Sub-properties in JSON string format. Can be nested.
- *aggregation1 2* – Additional groups of data, such as like nested controls, layout data, and so on.

© SAP

SAPUI5
sap.m.Input

Selected Properties:

- **type** – HTML type for input tag. Supported by browsers natively (Text, Number, Email, Phone, …)
- **placeholder** – Text to display in field before data entry.
- **description** – Description of field data to be displayed after field.
- **fieldWidth** – Width when using description

Events:

- **liveChange** – triggered as data is entered. (Assumed in the controller.)


```
<Input type="Number"
       placeholder="Enter Number..."
       fieldWidth="25%"
       description="Units"
       liveChange="onChange">
</Input>
```


```
onChange        : function(oEvent) {
    var input = oEvent.getSource();
    if (input.getValue() < 10) {
        input.setValueState(sap.ui.core.ValueState.Error);
        input.setValueStateText("Value must be at least 10");
    } else {
        input.setValueState(sap.ui.core.ValueState.None);
    }
}
```

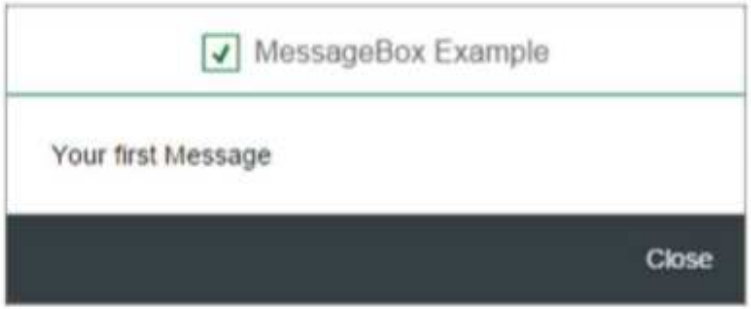
© SAP

SAPUI5
sap.m.MessageBox

Specialization of sap.m.Dialog with configured texts and buttons.

Additional Properties:

- `title` – Displayed on top of the pop-up.
- `icon` – Icon for message type. Supported values: Error, Information, Success, None, Question, Success, Warning.
- `actions` – Button to be added to message. Supported Values: Abort, Cancel, Close, Delete, Ignore, No, Ok, Retry, Yes.
- `onClose` – Method triggered when user presses a button.




```
var message = "Your first Message";
sap.m.MessageBox.show(message, {
    icon: sap.m.MessageBox.Icon.SUCCESS,
    title: "MessageBox Example",
    actions: sap.m.MessageBox.Action.CLOSE,
    verticalScrolling: false,
    horizontalScrolling: false
});
```

© SAP

SAPUI5
sap.ui.table.Table

The Table control provides a set of sophisticated and comfort functions for table design.

Additional Properties/Aggregations:

- `rows` – Includes binding information for the data.
- `columns` – Defines each column and can contain a data template based on the bound data.
- `sortProperty` – At the column level supports user selected sorting based on the configured data.
- Many other options. Also see **sap.m.Table** for selected functionality optimized for mobile devices.

ID	First Name	Last Name
1	Nancy	Davolio
2	Andrew	Fuller
3	Janet	Leverling
4	Margaret	Peacock

```
<t:Table id="oDataTable" rows="{path:'/Employees'}">
    <t:columns>
        <t:Column sortProperty="EmployeeID">
            <m:Label text="ID"/>
            <t:template>
                <m:Text text="{EmployeeID}"/>
            </t:template>
        </t:Column>
        <t:Column sortProperty="FirstName">
            <m:Label text="First Name"/>
            <t:template>
                <m:Text text="{FirstName}"/>
            </t:template>
        </t:Column>
        <t:Column sortProperty="LastName">
            <m:Label text="Last Name"/>
            <t:template>
                <m:Text text="{LastName}"/>
            </t:template>
        </t:Column>
    </t:columns>
</t:Table>
```

SAPUI5 Layout

- **sap.m.FlexBox** – Flexible box layout
- **sap.ui.layout.FitFlex** – A fixed part for multiple controls and an adaptable part for one control
- **sap.ui.layout.Grid** – 12 column flow layout
- **sap.ui.layout.HorizontalLayout**
- **sap.ui.layout.ResponsiveFlowLayout** – Controls blown up to fit one line
- **sap.ui.layout.VerticalLayout**

sap.ui.commons.layout – layouts focused on desktop devices:

- **AbsoluteLayout**
- **BorderLayout**
- **MatrixLayout**
- HorizontalLayout – deprecated, use **sap.ui.layout** version.
- ResponsiveFlowLayout – deprecated, use **sap.ui.layout** version
- VerticalLayout – deprecated, use **sap.ui.layout** version

© SAP

SAPUI5
Example sap.m.FlexBox

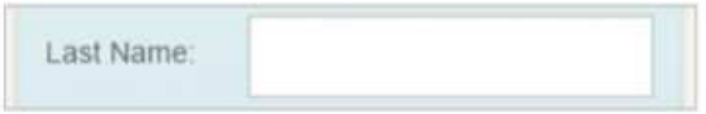
Selected Properties:

- **alignItems** – Options include: Baseline, Center, End, Inherit, Start, and Stretch.
- **justifyContent** – Options include: Center, End, Inherit, SpaceAround, SpaceBetween, and Start.
- **height / width** – CSS sizes, such as "1px" or "2em" or "50%".

Aggregations:

- **items** – Contains any controls.

```
<FlexBox alignItems="Center" justifyContent="SpaceAround">
    <items>
        <Label id="lblLastName" text="Last Name:"/>
        <Input id="inpLastName"/>
    </items>
</FlexBox>
```



In this example, the controls are centered on the line and the space is evenly distributed between the controls.

© SAP

SAPUI5
i18n

- Create an i18n.properties.[de|en|…] for translateable texts

```
// set i18n model on view
        var i18nModel = new ResourceModel({
           bundleName: 8820038.i18n.i18n
        });
        this.getView().setModel(i18nModel, i18n);
```

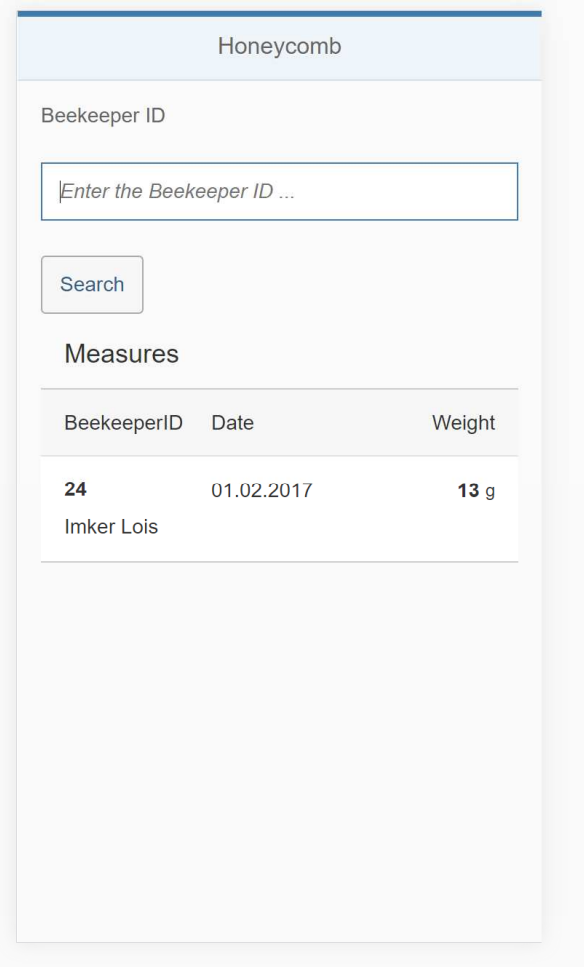
I18n Conventions

- The resource model for internationalization is called the i18n model.
- The default filename is i18n.properties.
- Resource bundle keys are written in (lower) camelCase.
- Resource bundle values can contain parameters like {0}, {1}, {2}, …
- Never concatenate strings that are translated, always use placeholders.
- Use Unicode escape sequences for special characters.

Exercise
WebIDE
SAPUI5

- Internationalize the App
- React to the Button press in the controller
- Alert the value of the Inputfield
- Send a MessageBox
- Insert a sap.m.Table
- Insert a Json model
- Set the Json model data in the table

Solution WebIDE SAPUI5



Solution
WebIDE
SAUI5 View - Table

```
<Table id="idMeasureTable" inset="false" items="{ path: '/MeasureCollection', sorter: { path: 'beekeeperID' } }">
<headerToolbar>
<Toolbar>
<Title text="Measures" level="H2"/>
</Toolbar>
</headerToolbar>
<columns>
<Column>
<Text text="BeekeeperID"/>
</Column>
<Column>
<Text text="Date"/>
</Column>
<Column
hAlign="Right">
<Text text="Weight"/>
</Column>
</columns>
<items>
<ColumnListItem>
<cells>
<ObjectIdentifier title="{beekeeper_id}" text="{beekeeper_name}"/>
<!-- <Text text="{measure_date}"/> -->
<!-- <Text text="{path: 'measure_date',
type : 'sap.ui.model.type.Date',
formatOptions: {pattern: 'dd/MM/yyyy'}
}"></Text> -->
<Text text="{path: 'measure_date', formatter: '.formatDate'}"></Text>
<ObjectNumber number="{measure_weight}" unit="{measure_unit}"/>
</cells>
</ColumnListItem>
</items>
</Table>
```

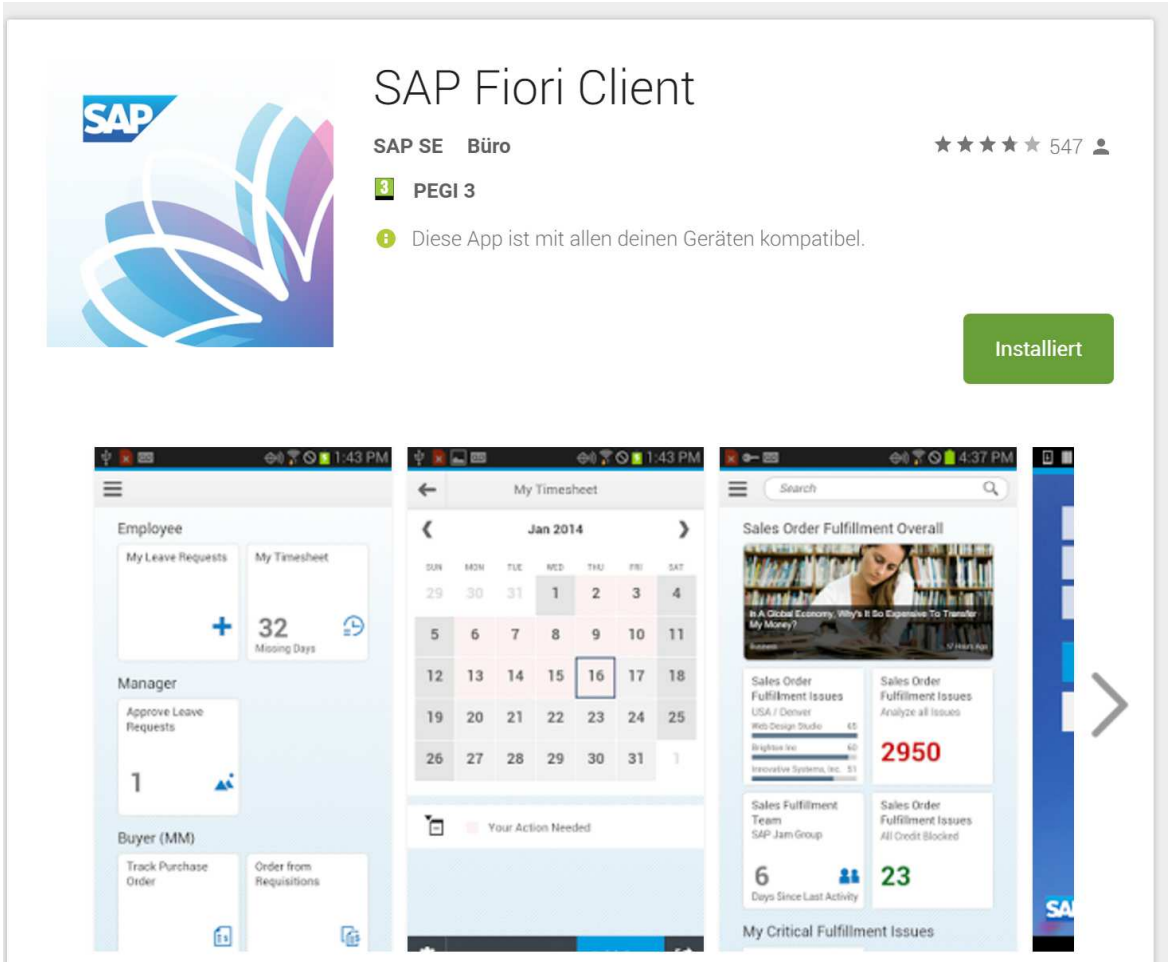
Solution
WebIDE
SAUI5 Controller

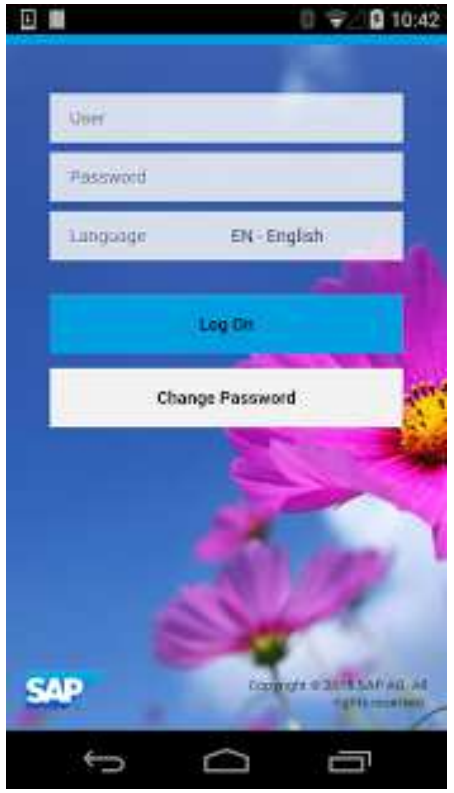
```
sap.ui.define([
		"sap/ui/core/mvc/Controller",
		"sap/m/MessageBox"
], function(Controller, MessageBox) {
		"use strict";
		return Controller.extend("at.facet.hwHellloWorld.controller.Main", {
				// Init the object
				onInit: function() {
						// set explored app's demo model on this sample
						var oModel = new sap.ui.model.json.JSONModel("model/measures.json");
						this.getView().setModel(oModel);
				},
				// React on the button
				onPress: function(oEvent) {
						debugger;
						var value = this.getView().byId("beekeeperID").getValue();
						//alert("Hello I am here " + value);
						//sap.m.MessageBox.show("Hello I am here " + value,
						// Falls sap.m.MessageBox in function Parameters definiert
						MessageBox.show("Hello I am here " + value, {
								icon: sap.m.MessageBox.Icon.SUCCESS,
								title: "Hello Beekeeper",
								actions: sap.m.MessageBox.Action.CLOSE,
								verticalScrolling: false,
								horizontalScrolling: false
						});
				},
				formatDate: function(date) {
						// Deklaration laden
						jQuery.sap.require("sap.ui.core.format.DateFormat");
						// Format definieren und Objekt instanziieren
						var oDateFormat = sap.ui.core.format.DateFormat.getDateTimeInstance({
								pattern: "dd.MM.YYYY"
						});
						// Formatierungsergebnis
						return oDateFormat.format(new Date(date));
				}
		});
});
```

Solution
WebIDE
SAUI5 model/measures.json

```
{
        "MeasureCollection": [{
                        "beekeeper_id": "24",
                        "beekeeper_name" : "Imker Lois",
                        "measure_date": "2017-02-01",
                        "measure_weight": "13",
                        "measure_unit" : "g"
                }

        ]
}
```

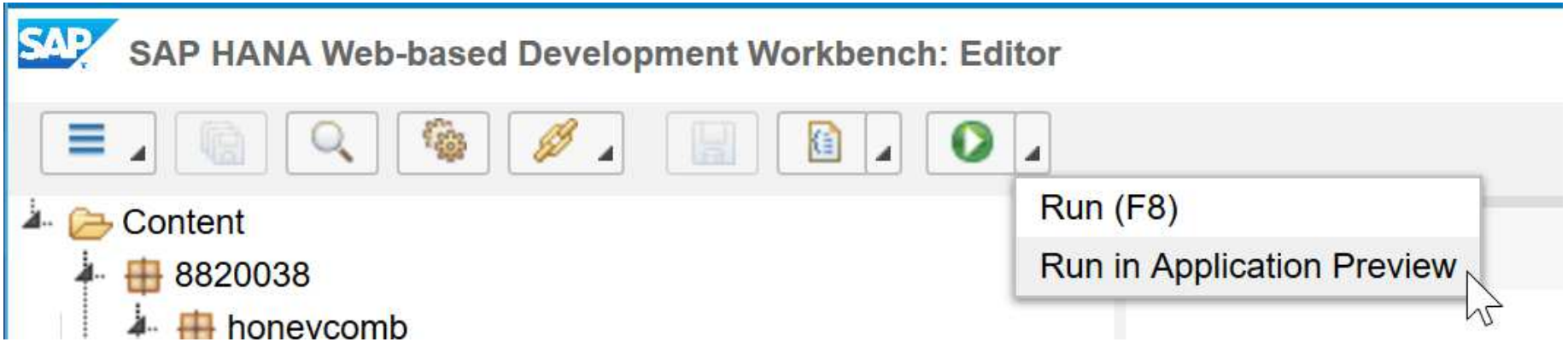
SAPUI5
Check the App in your Mobile
Install the SAP Fiori Client, e.g. on Android

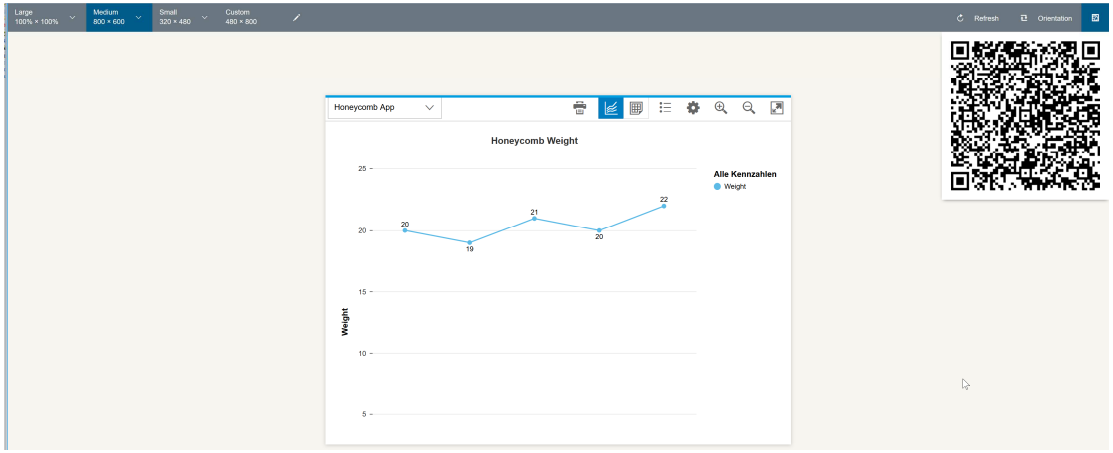




SAPUI5
Check the App in your Mobile
XSC Apps

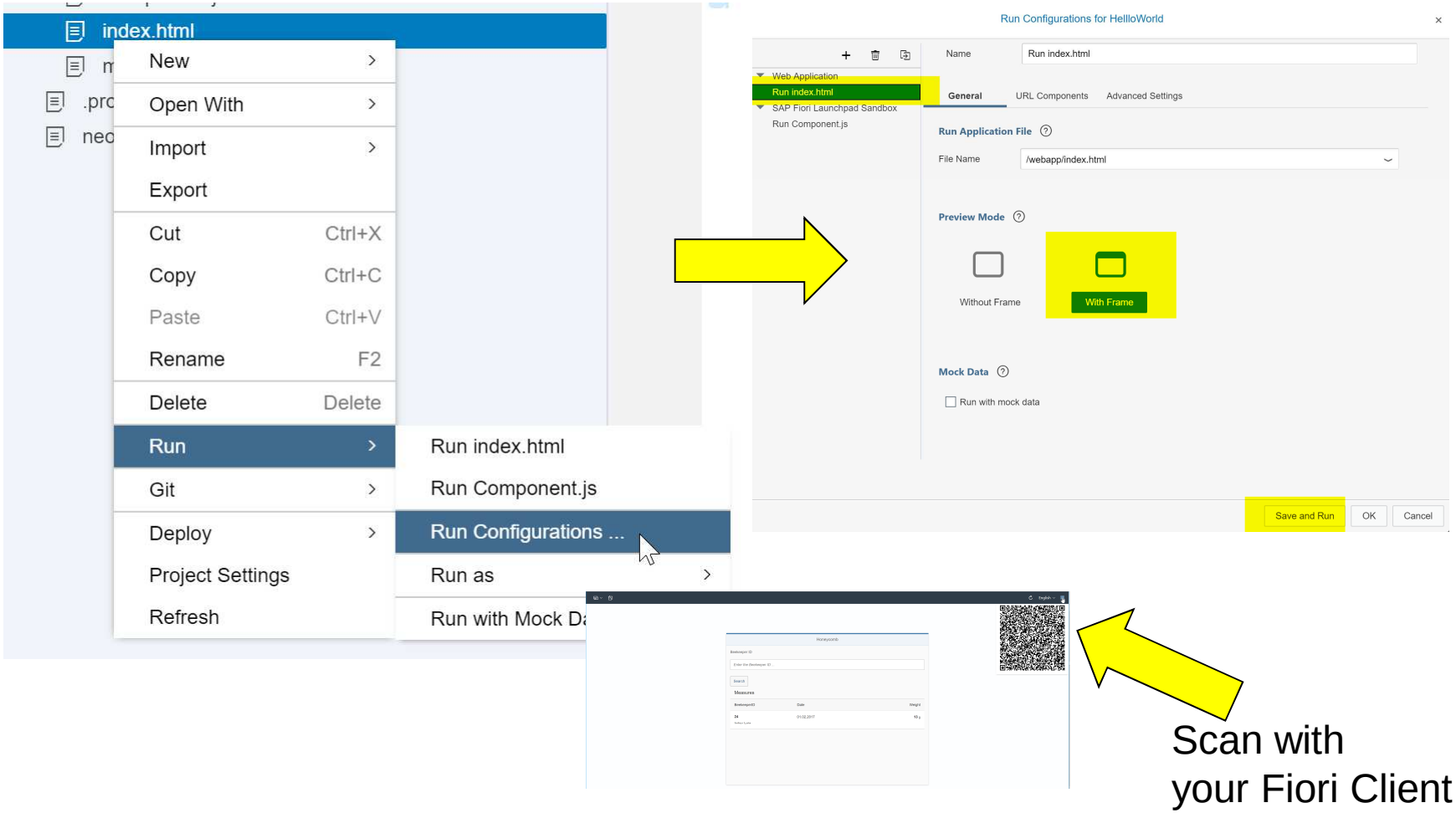
Precondition: Installed SAP Fiori Client on your smartphone





Scan with your Fiori Client

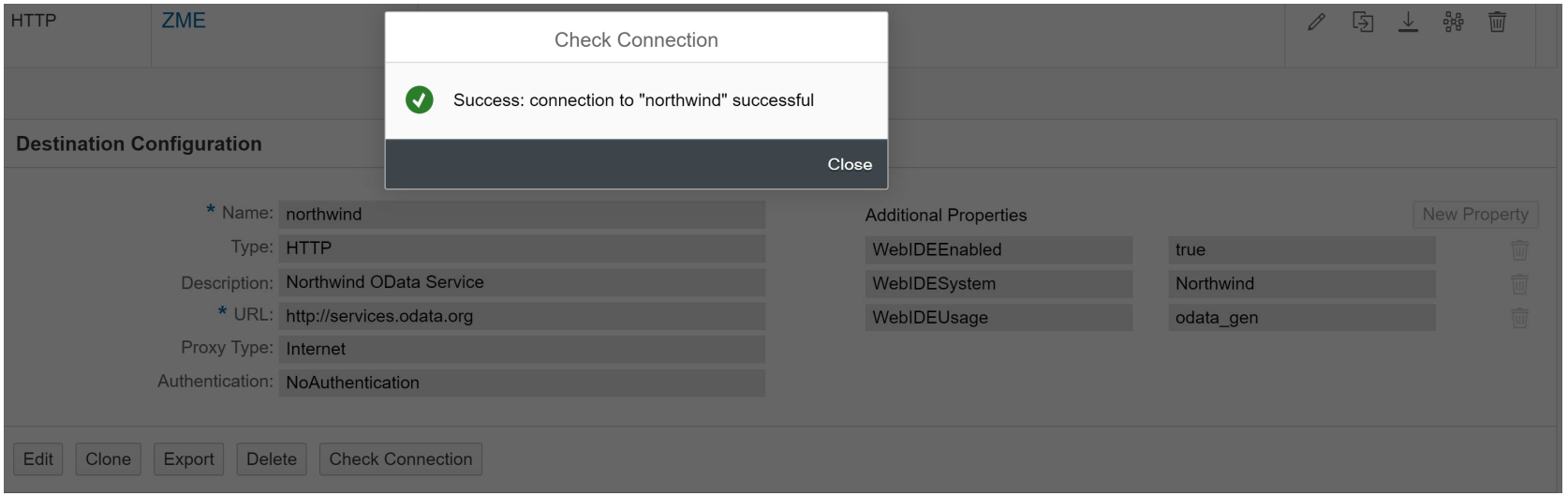
SAPUI5
Check the App in your Mobile WebIDE



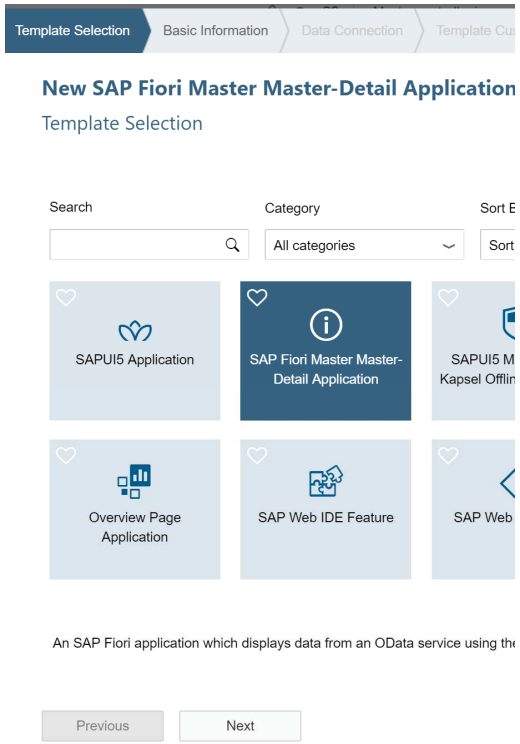
Scan with
your Fiori Client

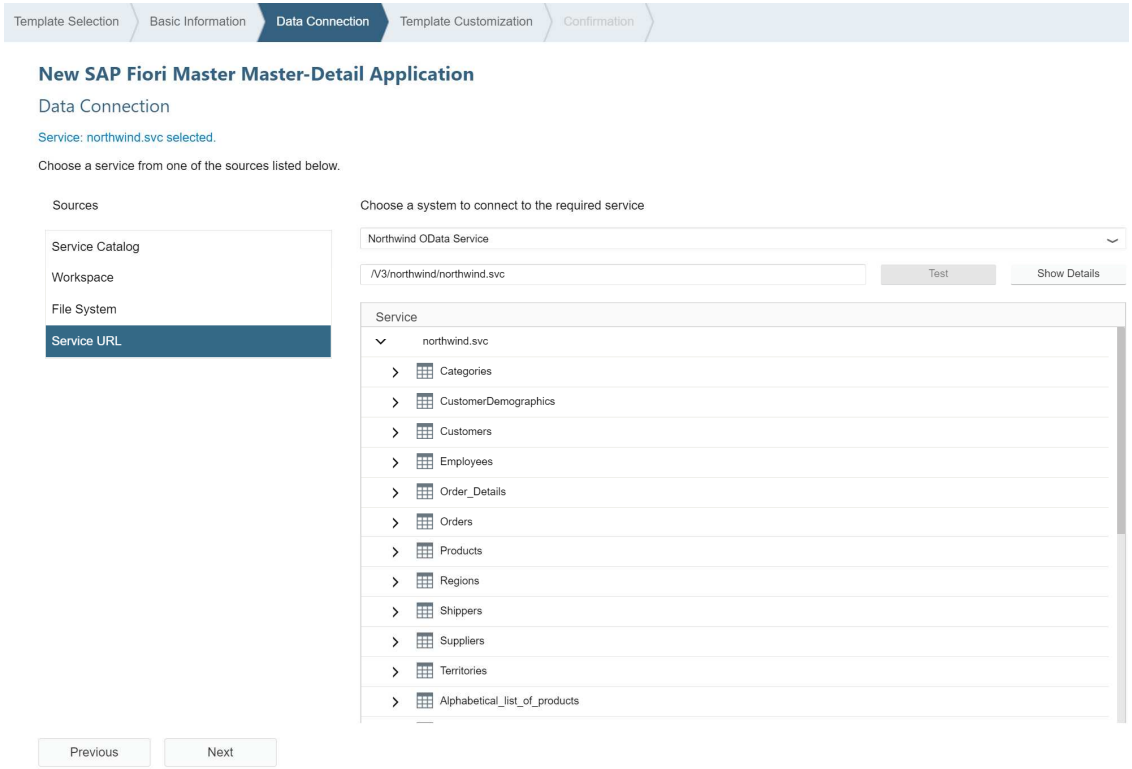
Automatic UI5 Generation
Destinations
Northwind

- Connectivity -> Destination
- New Destination

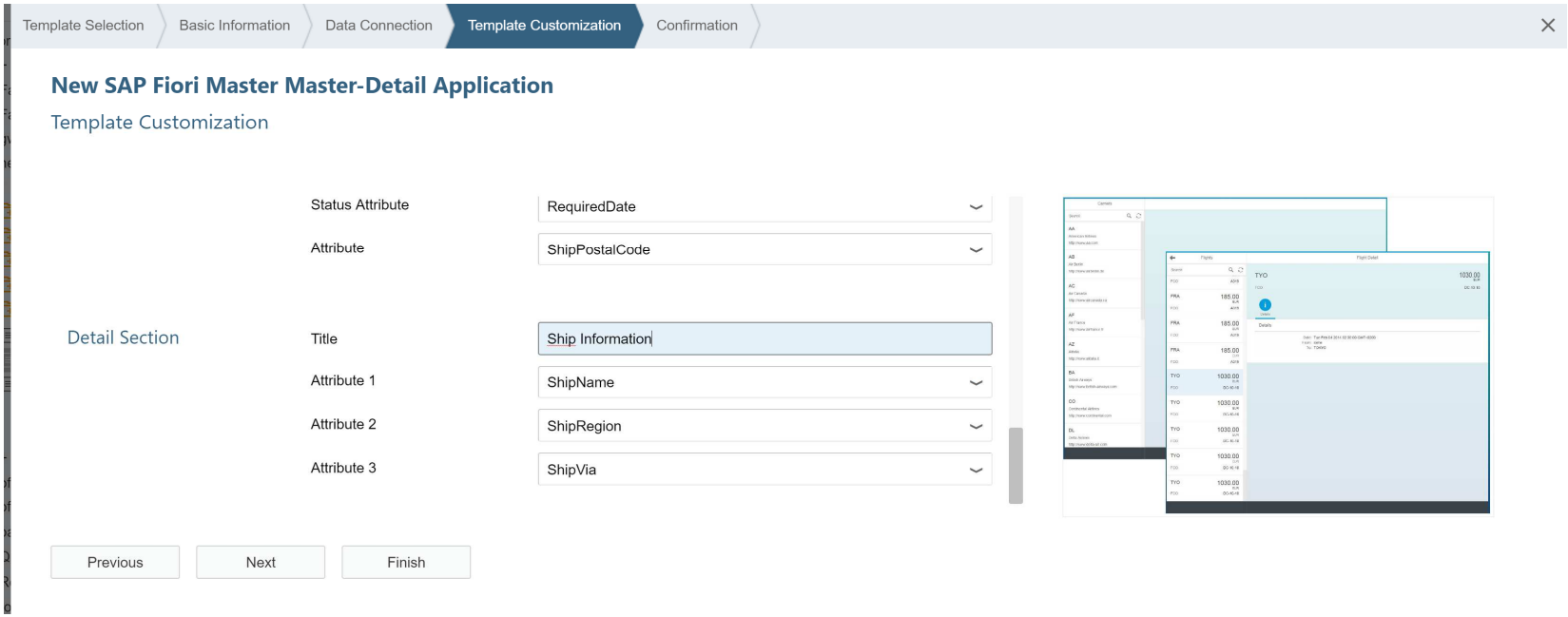


Automatic UI5 Generation Template

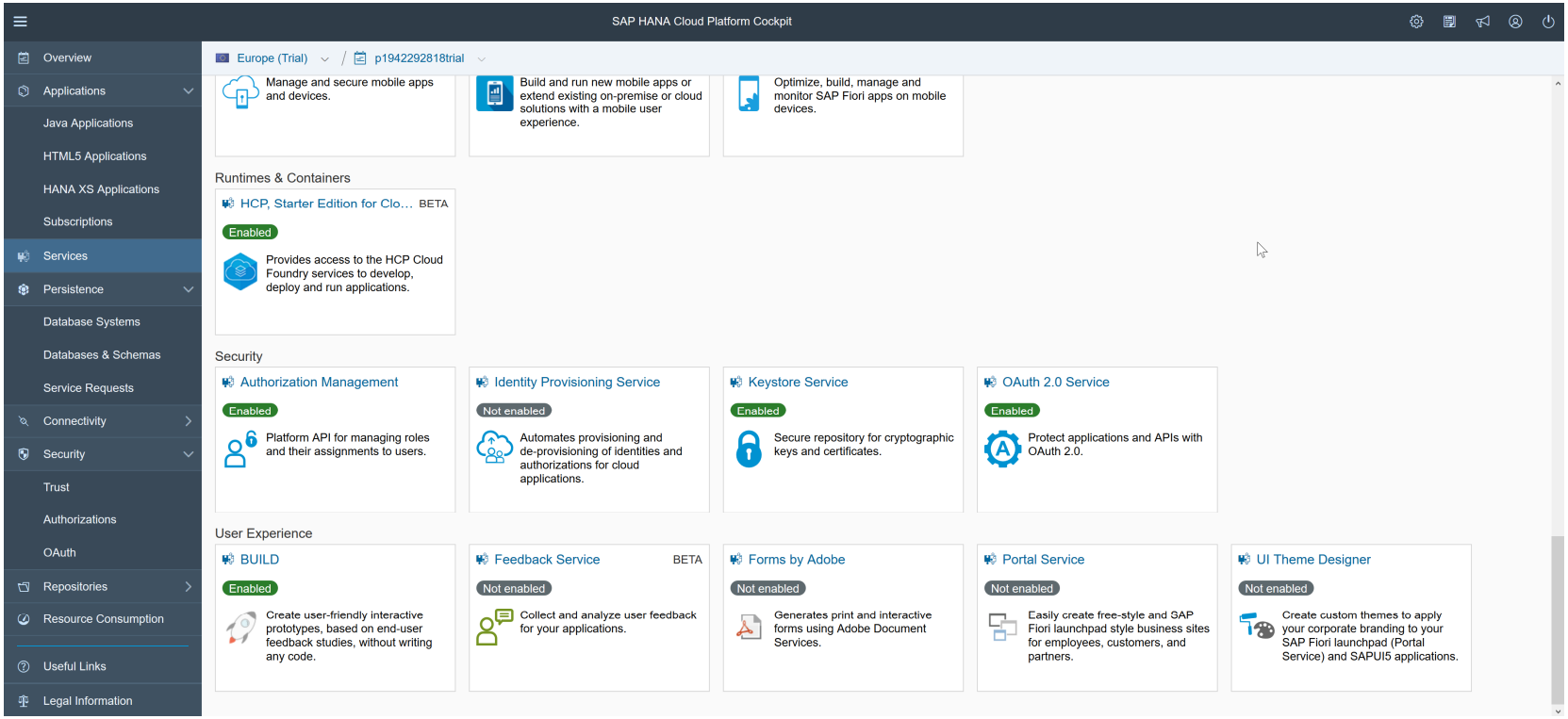




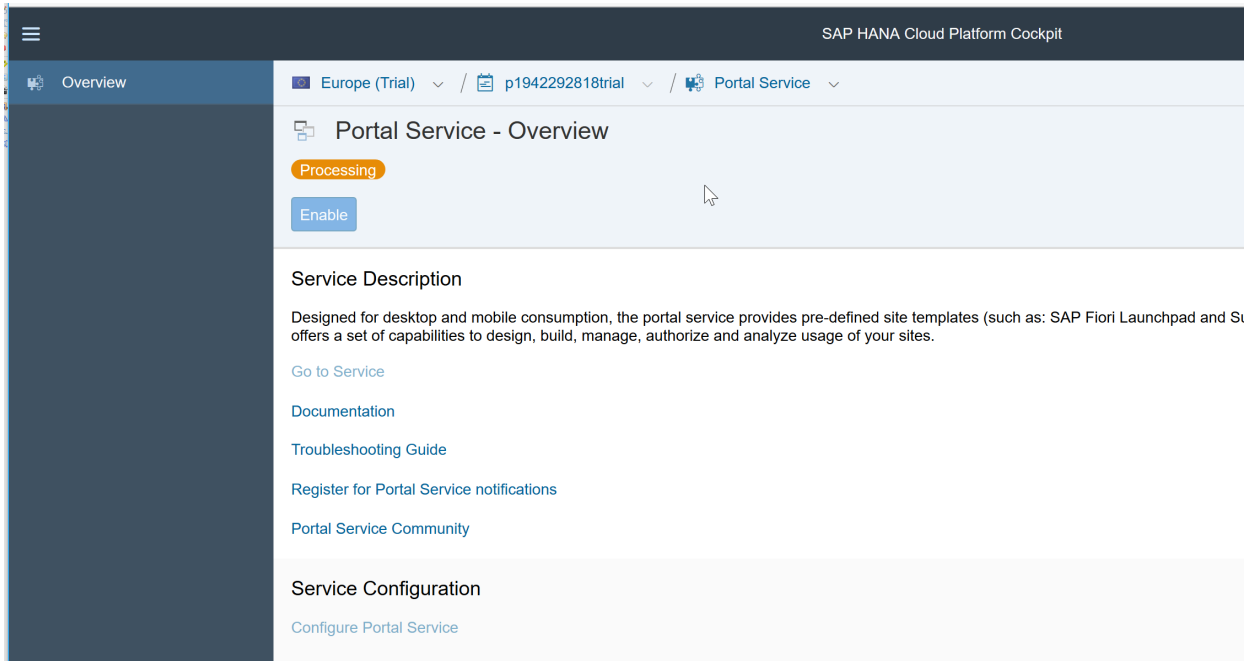
Automatic UI5 Generation Template



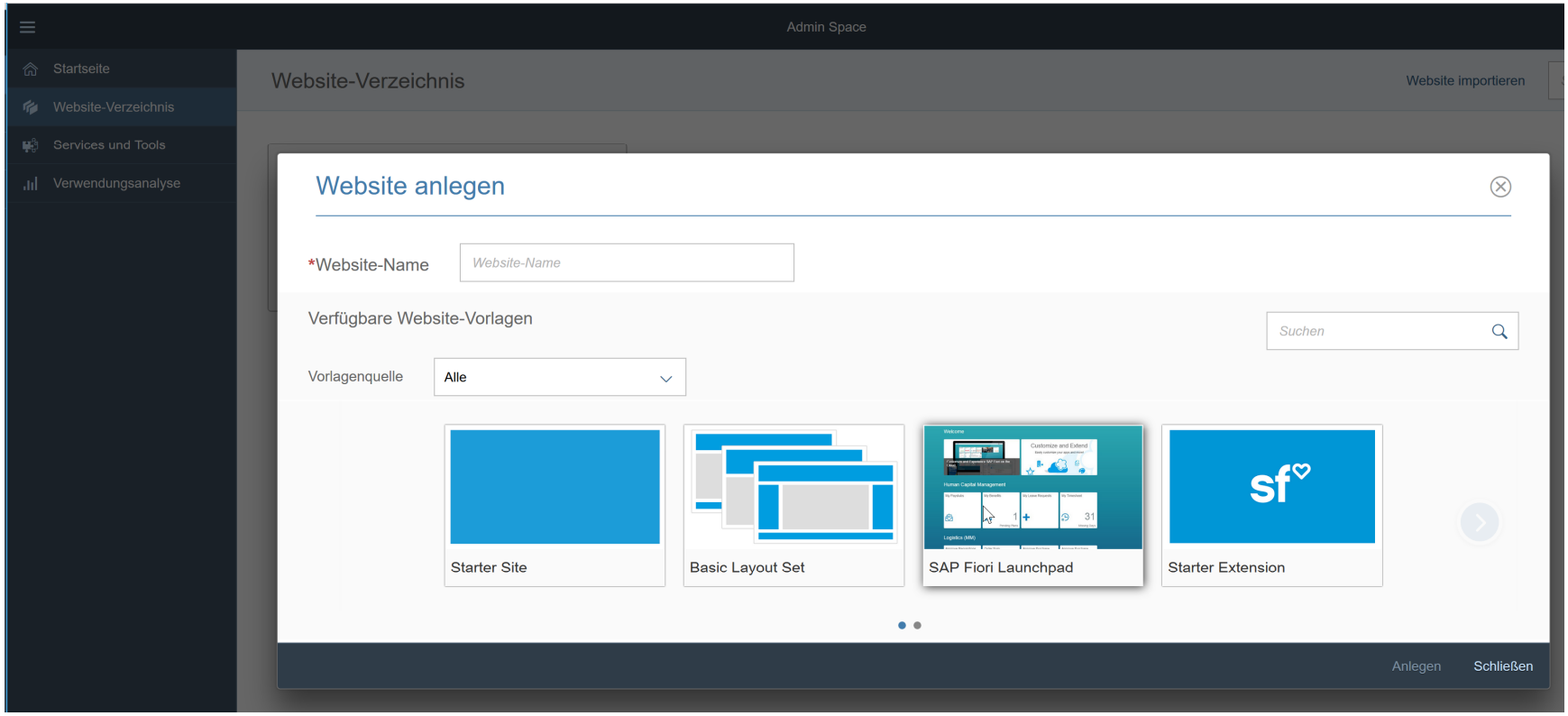
Launchpad HANA



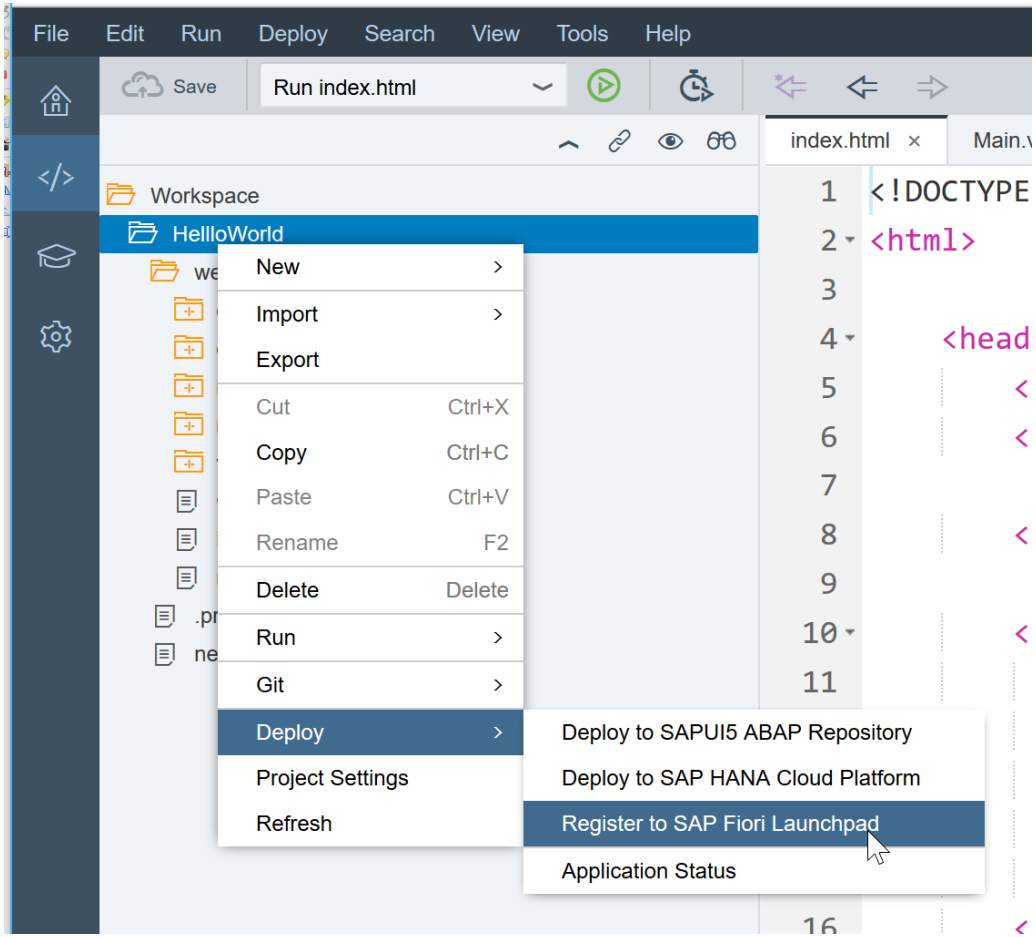
Launchpad HANA



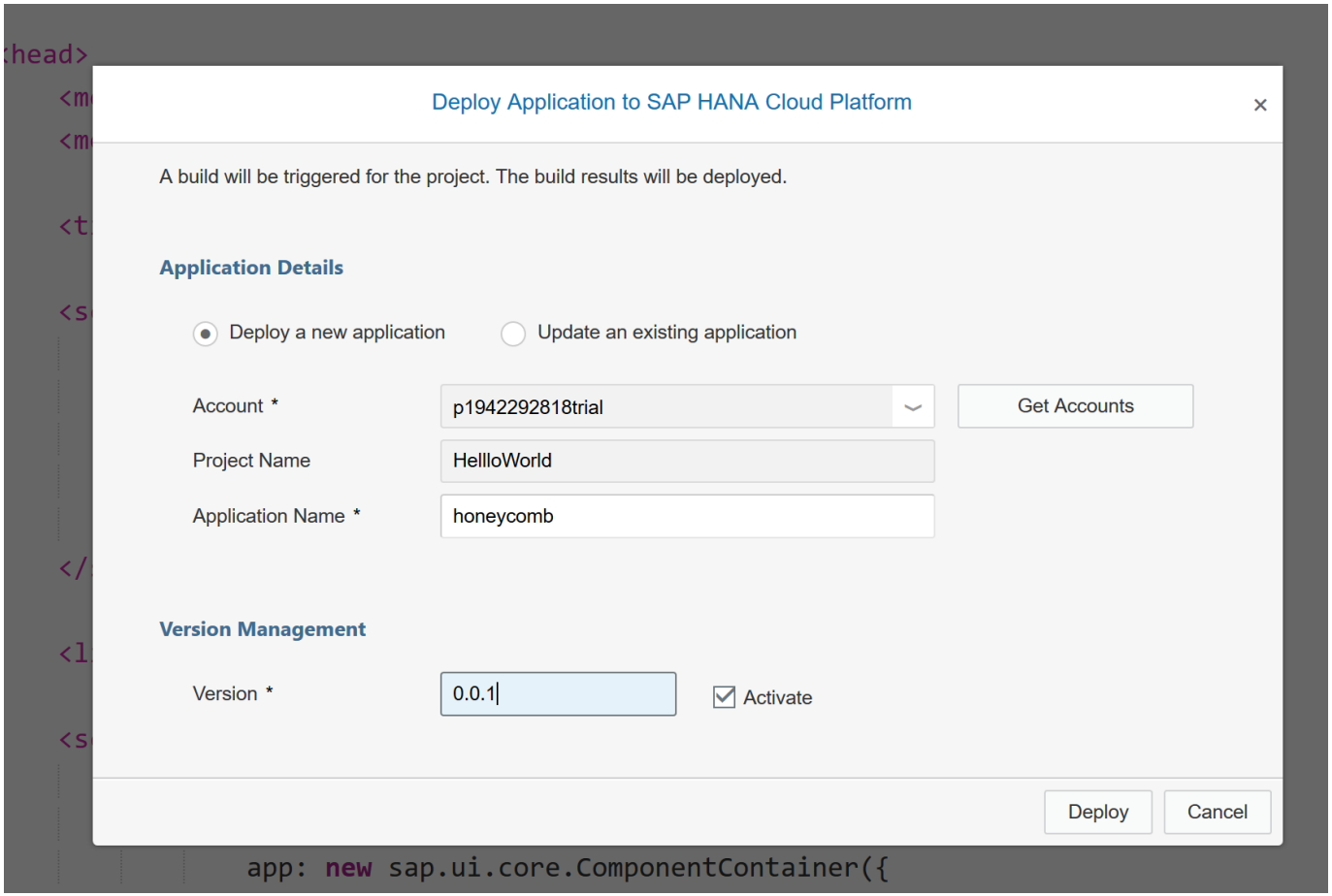
Launchpad HANA



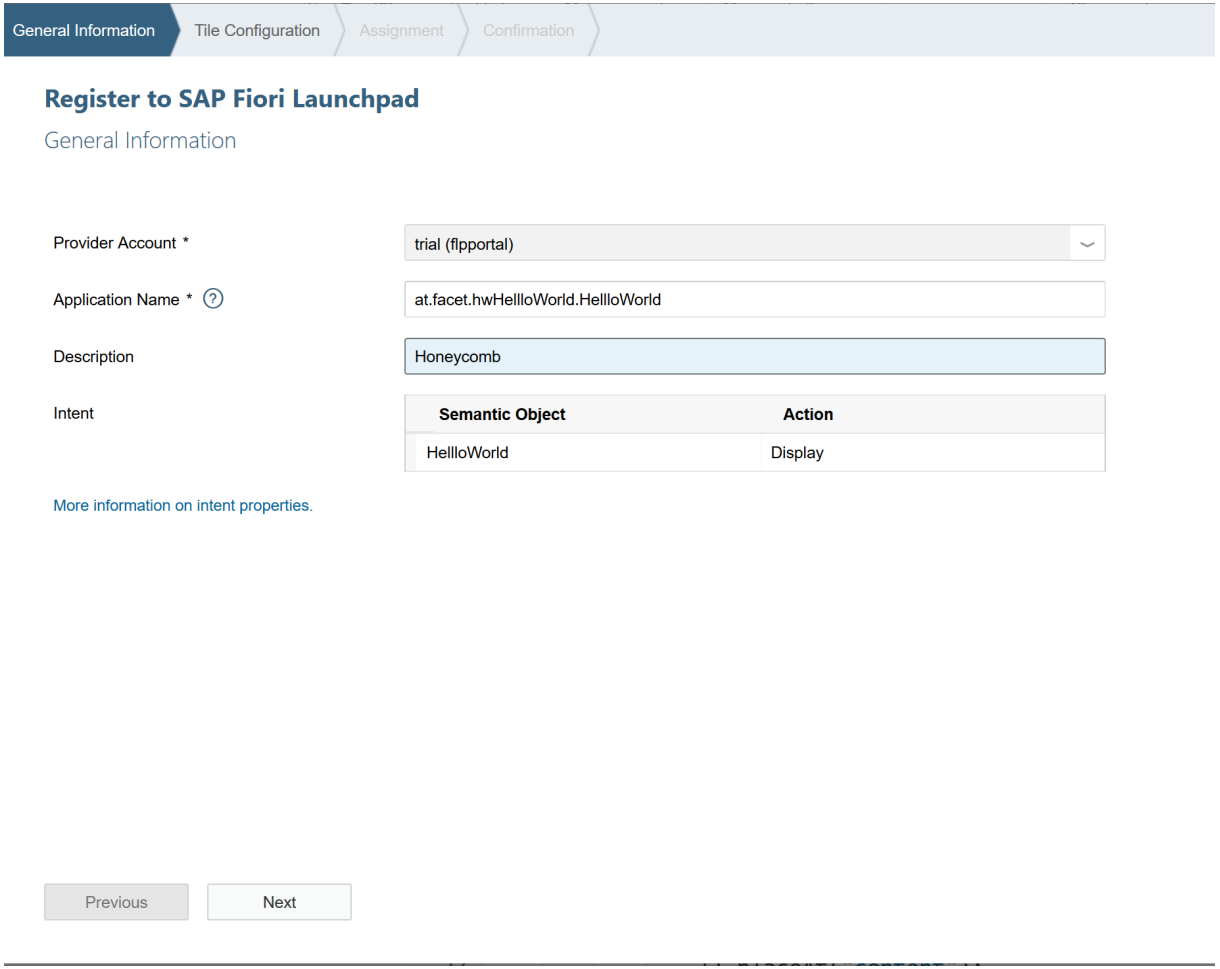
Launchpad HANA



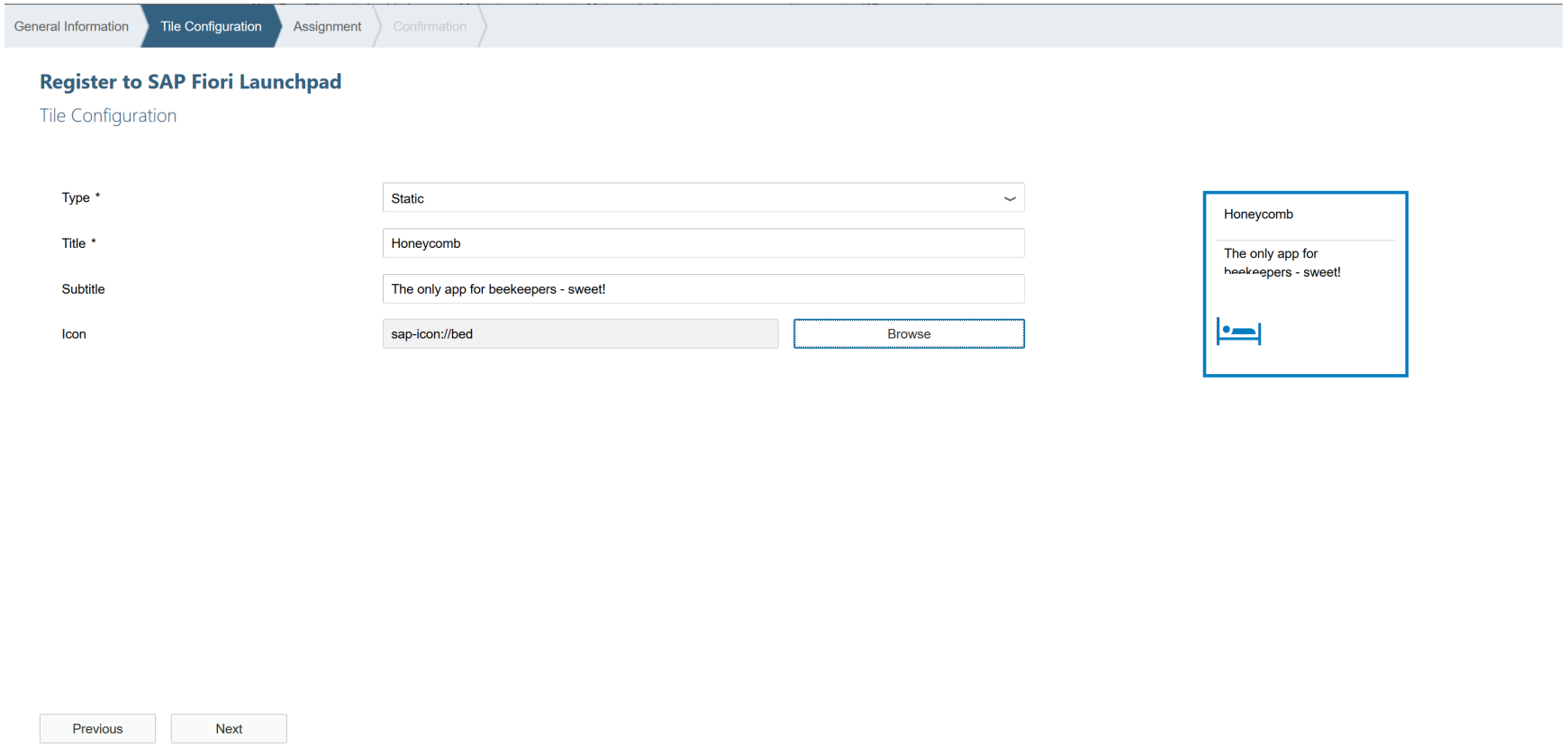
Launchpad HANA



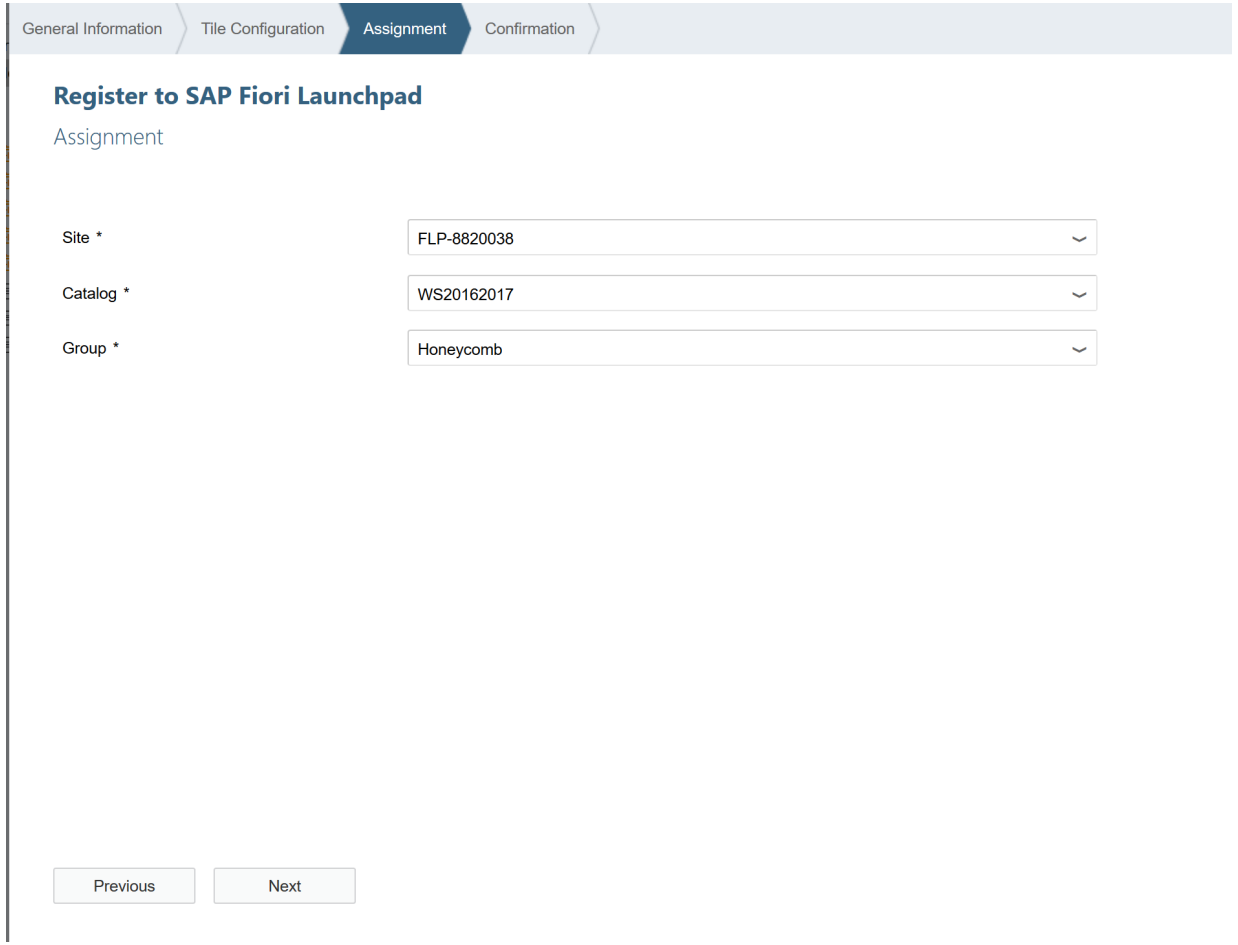
Launchpad HANA



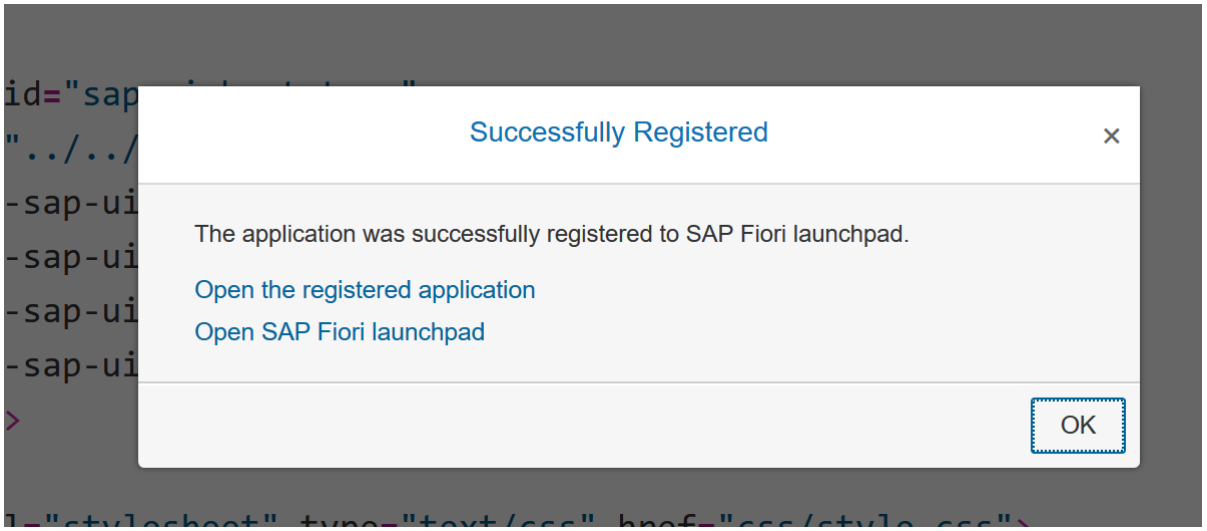
Launchpad HANA



Launchpad HANA

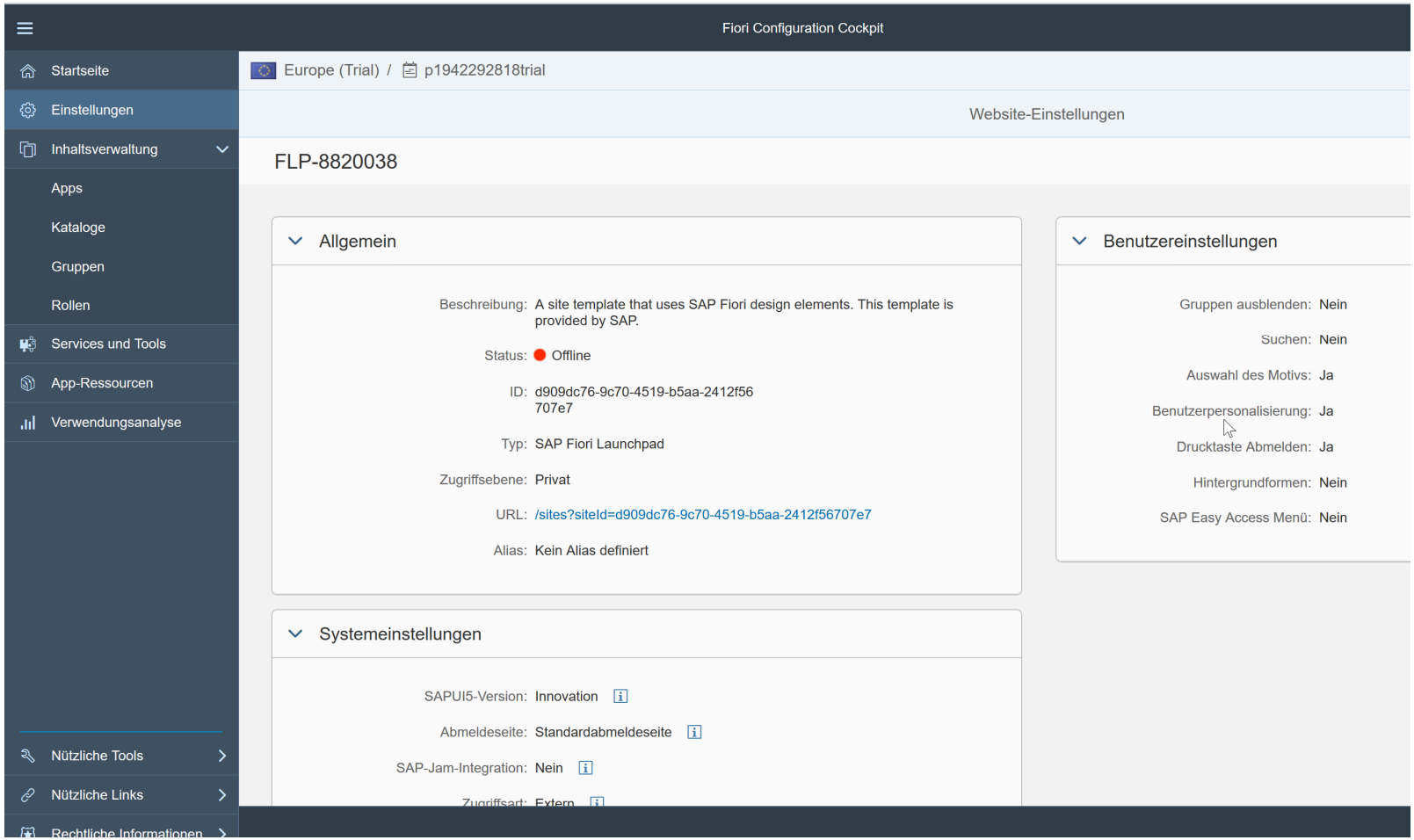


Launchpad HANA



Launchpad HANA

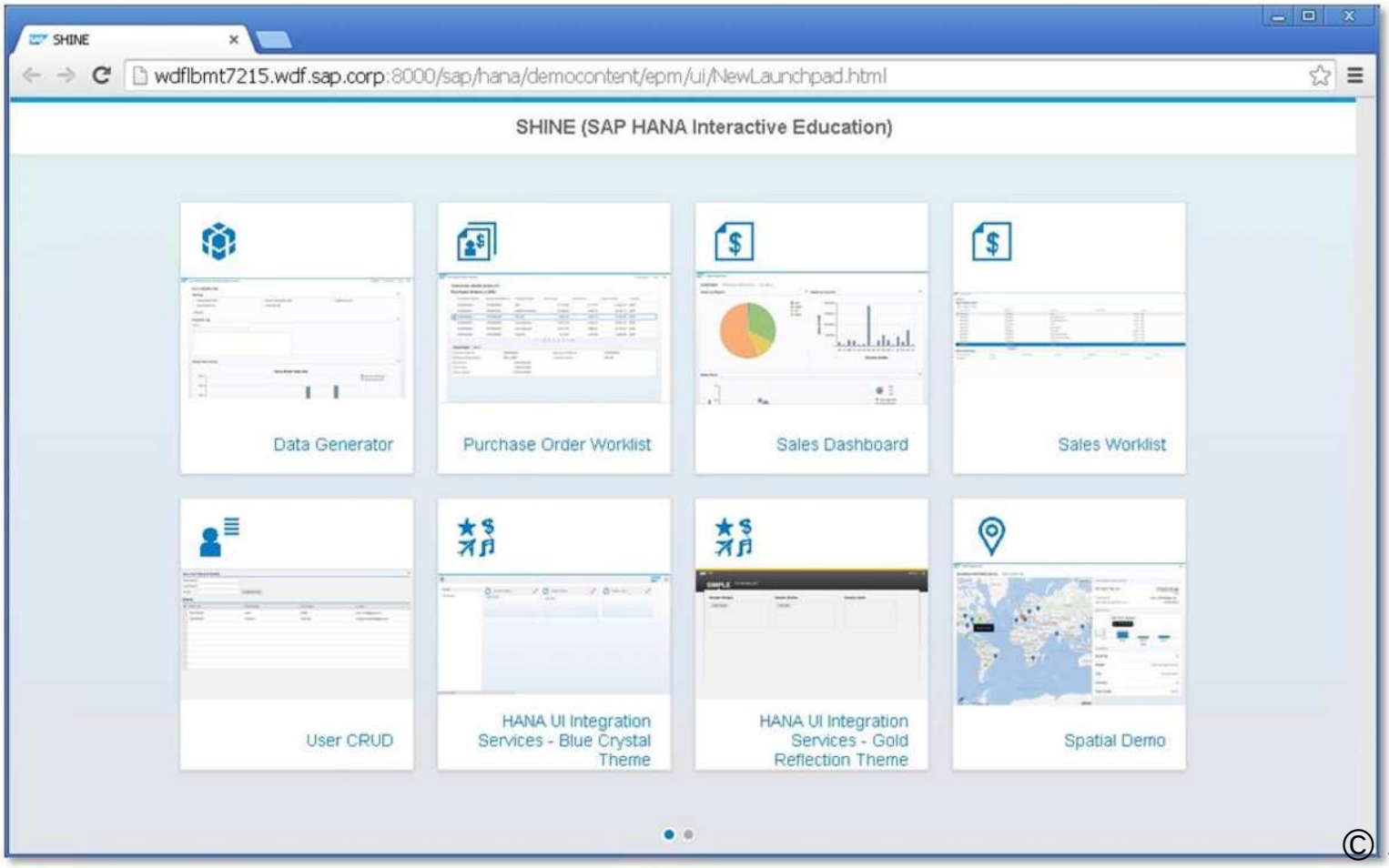
Launchpad HANA



THE PROJECT IN SAP HANA XS CLASSIC AND ADVANCED

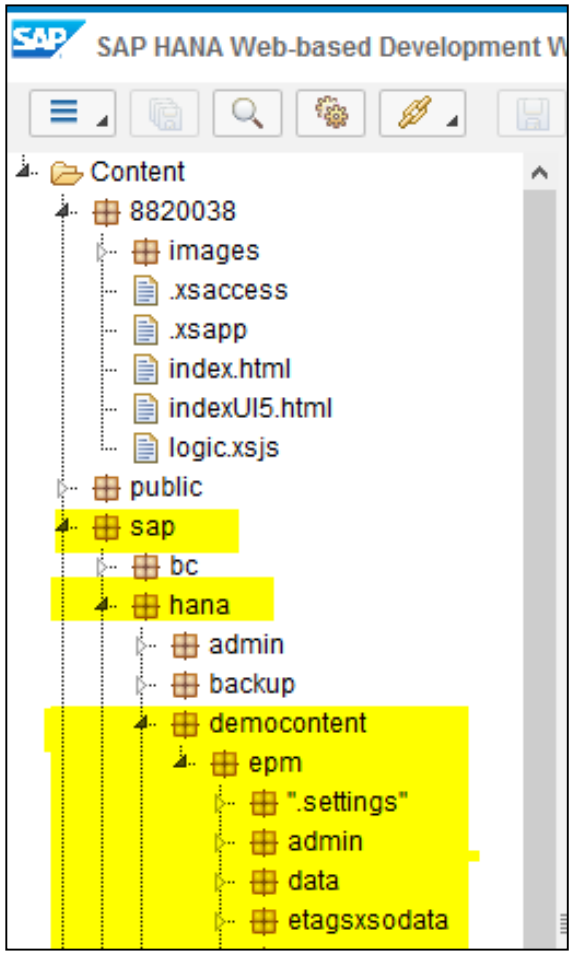
- Give an overview about the SHINE application
- Comparing the XS Classic and XS Advanced development process
- Create a XS Classic Project
- XS Advanced Multi Target Application
- Create a XS Advanced Project

SHINE Demo App



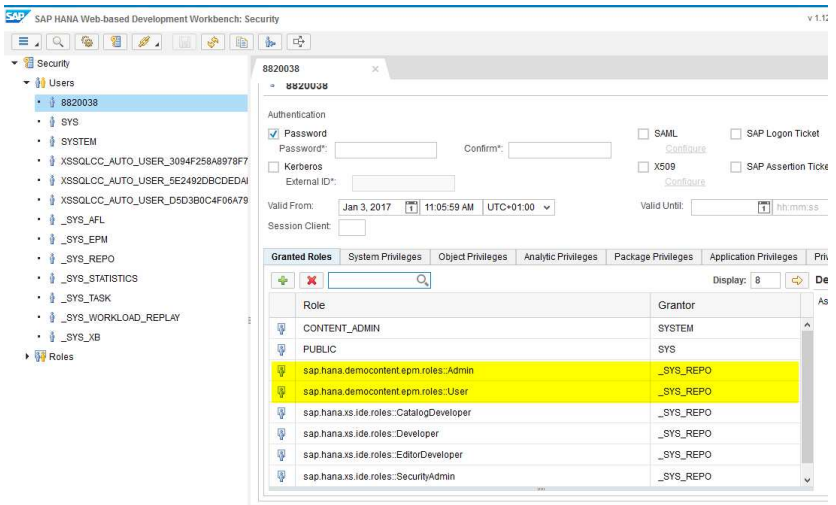
© SAP

SHINE: SAP HANA
Web-based Dev Workbench: Editor

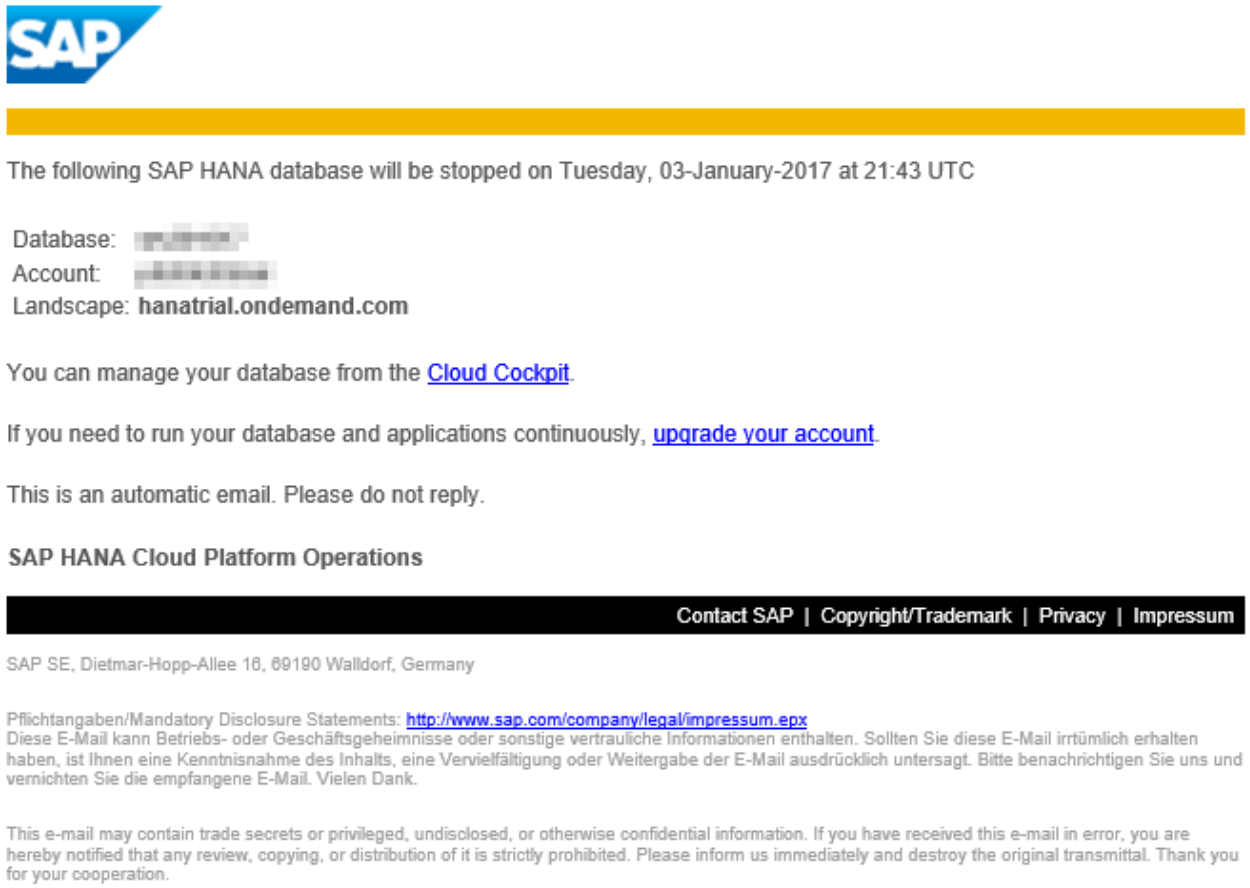


SHINE: Auth Preparation

- To login to SHINE Launchpad, the user should have either **sap.hana.democontent.epm.roles:Admin** or **sap.hana.democontent.epm.roles:User**



Stop after 12 hours



The following SAP HANA database will be stopped on Tuesday, 03-January-2017 at 21:43 UTC

Database: [redacted]
Account: [redacted]
Landscape: **hanatrial.ondemand.com**

You can manage your database from the Cloud Cockpit.

If you need to run your database and applications continuously, upgrade your account.

This is an automatic email. Please do not reply.

SAP HANA Cloud Platform Operations

Contact SAP | Copyright/Trademark | Privacy | Impressum

SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany

Pflichtangaben/Mandatory Disclosure Statements: http://www.sap.com/company/legal/impressum.epx
Diese E-Mail kann Betriebs- oder Geschäftsgeheimnisse oder sonstige vertrauliche Informationen enthalten. Sollten Sie diese E-Mail irrtümlich erhalten haben, ist Ihnen eine Kenntnisnahme des Inhalts, eine Vervielfältigung oder Weitergabe der E-Mail ausdrücklich untersagt. Bitte benachrichtigen Sie uns und vernichten Sie die empfangene E-Mail. Vielen Dank.

This e-mail may contain trade secrets or privileged, undisclosed, or otherwise confidential information. If you have received this e-mail in error, you are hereby notified that any review, copying, or distribution of it is strictly prohibited. Please inform us immediately and destroy the original transmittal. Thank you for your cooperation.

XSC Project

Initially we create a proper “space” on the server for you to develop.

To do this, we use user <MATNR> (with security administration authorizations).

1. Create a **package** in the repository to store development artifacts (that is, JavaScript sources).
2. Create a **schema** in the catalog to store database objects (that is, database tables).
3. Create a security **Role** with authorization to develop in the package and in the schema
4. Assign the role to your user.
5. Generate and run a test application.

Then we create the project in SAP HANA Web-based Development Workbench

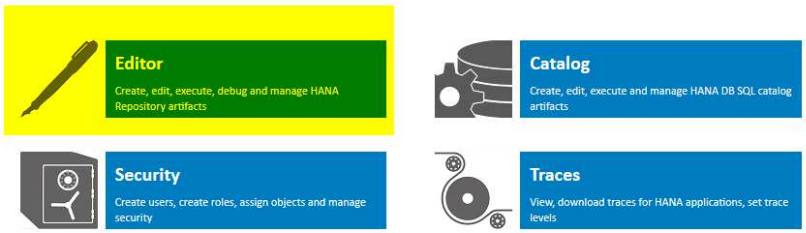
6. Create an XS Project in SAP HANA Web-based Development Workbench

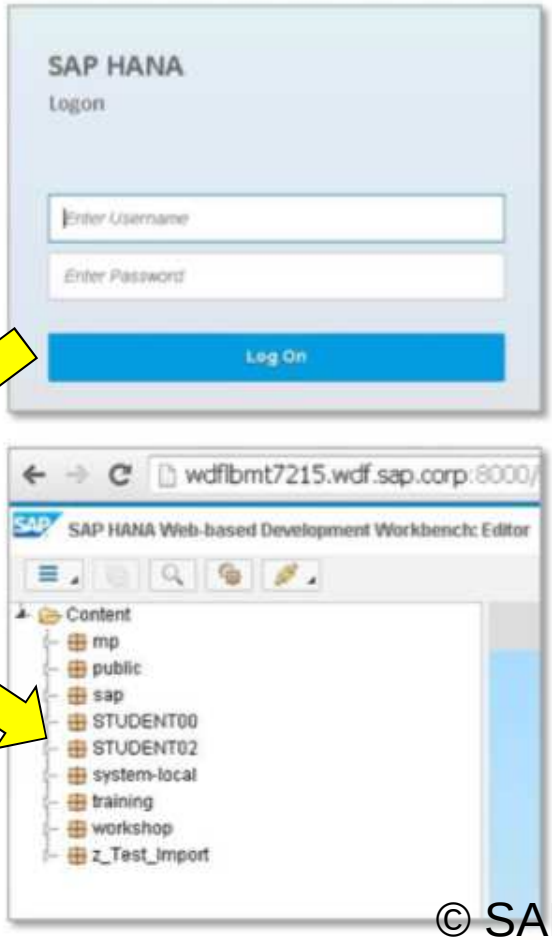
XSC Project

All of your design-time development artifacts will be stored in a package within the SAP HANA Repository.

To create the package, execute following actions:

- Using a web browser, open the **SAP HANA Web-based Development Workbench editor**

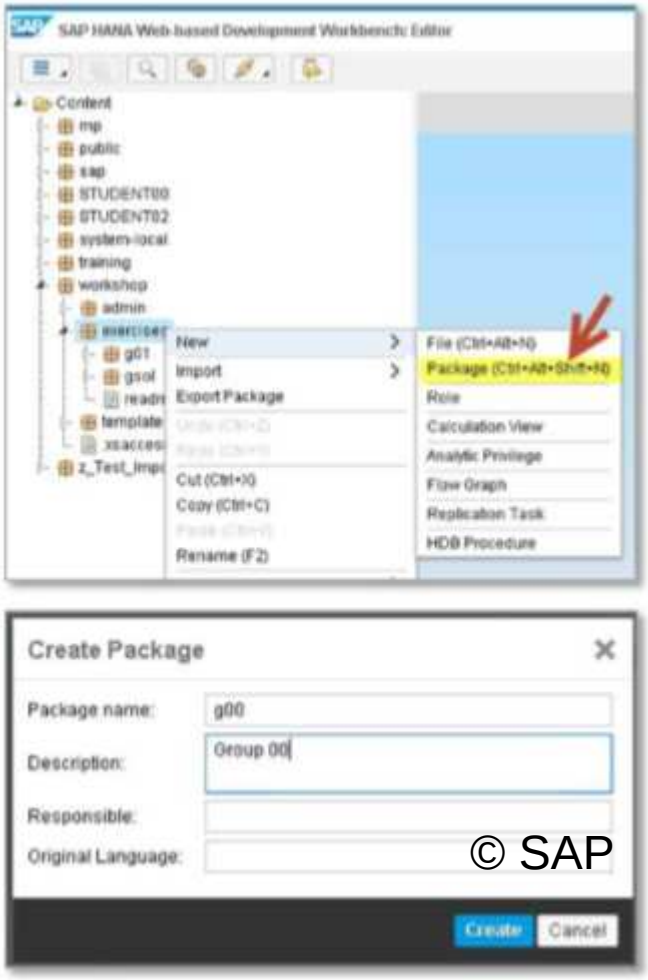




© SAP

XSC Project
Create Package

- Switch to the Editor Hierarchy
- Use the Context Menu -> New -> Package menu entry
- Provide the data for the package
- Use the button Create to insert the package



Exercise
XSC Project
Create Package

- Create packages right below the Content node and call it <MATNR>, given the following hierarchy:
- Content
  - playground
  - honeycomb

Solution
XSC Project
Create Package

- Just follow the steps explained above.

XSC Project Schema

- **Relational databases** contain a **catalog** that describes the various **elements** in the system. The catalog divides the database into **sub-databases** known as **schema**. A database schema enables you to logically group together objects such as **tables**, **views**, and **stored procedures**. Without a defined schema, you cannot write to the catalog.
- SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database schema as a transportable design-time file in the repository. Repository files can be read by applications that you develop.

XSC Project
Create Schema

Repository File .hdbschema

Activate

Schema created in the catalog

© SAP

XSC Project
Create Schema

- Navigate package hierarchy to *workshop.exercises.g<group number>* (for example, `g00`).
- Press the right mouse button and choose *New → File*.
- Name the file **`hana_workshop_`***<group number>*`.hdbschema` (for example, `hana_workshop_00.hdbschema`).
- Choose *Insert Snipplet* button from the bar.
- Replace the default MY_SCHEMA_NAME with `hana_workshop_`*<group number>* (for example, `hana_workshop_00`).
- Save the file.
- The schema is automatically created in the Catalog.

© SAP

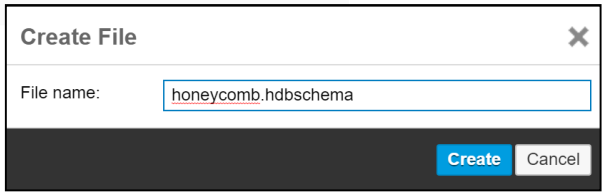
Exercise
XSC Project
Create Schema

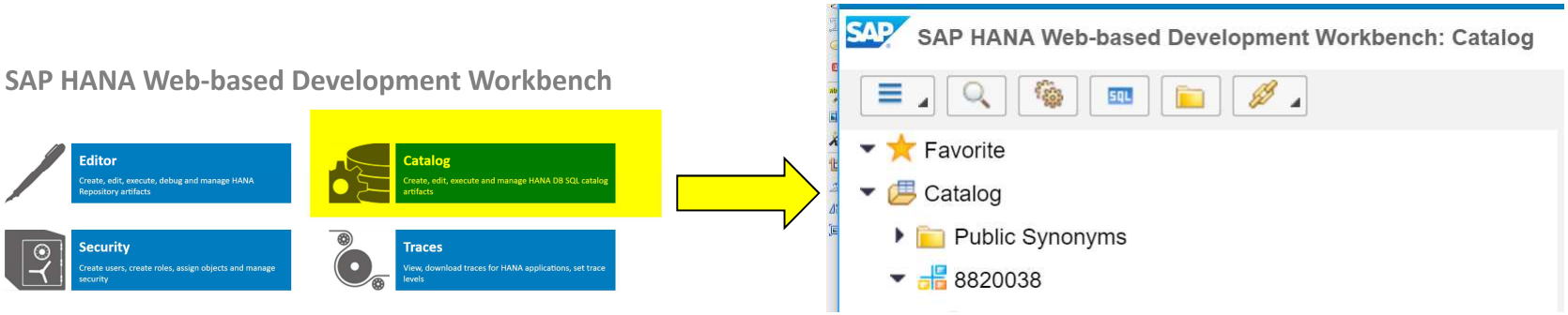
- Create a schema file and call it <MATNR>.hdbschema.
- The name of the schema is <MATNR>

Solution
XSC Project
Create Schema

- Just follow the steps explained above.

- schema_name=„<MATNR>“;





XSC Project Roles

- SAP HANA uses roles to **control access** to the Web-based tool that enable you to maintain important parts of the application-development environment, for example, security and authentication methods.
- When using the Web-based tools provided by SAP HANA XS, the availability of features, screens, tabs, and UI controls (for example, Add, Edit, or Save, or Delete buttons) is based on privileges. **For the sake of convenience**, the specific privileges required to use the features provided with a particular tool have been **collected into a selection of specific roles**, which you assign to the user who wants to use a tool.
- For example,
  - a user assigned the **role sap.hana.xs.admin.roles::HTTPDestViewer** can display HTTP destinations but not change them in any way;
  - a user assigned the role **sap.hana.xs.admin.roles::SQLCCAdministrator** can view SQL connection configurations and modify them, too.

XSC Project Roles

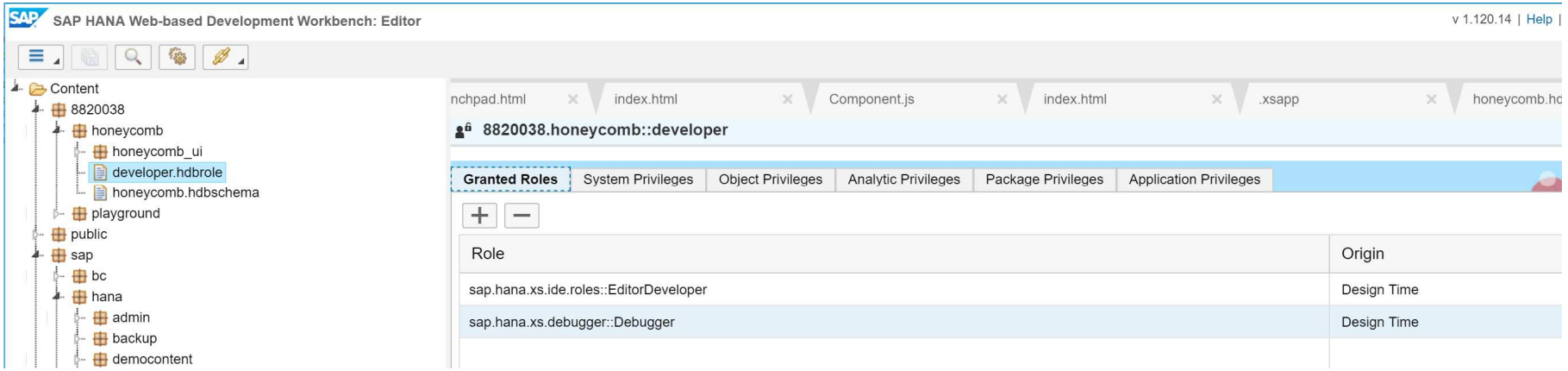
Privileges Types	Description
System Privileges	It controls normal system activity.
	System Privileges are mainly used for –
	Creating and Deleting Schema in SAP HANA Database
	Managing user and role in SAP HANA Database
	Monitoring and tracing of SAP HANA database
	Performing data backups
	Managing license
	Managing version
	Managing Audit
	Importing and Exporting content
	Maintaining Delivery Units
Object Privileges	Object Privileges are SQL privileges that are used to give authorization to read and modify database objects. To access database objects user needs object privileges on database objects or on the schema in which database object exists. Object privileges can be granted to catalog objects (table, view, etc.) or non-catalog objects (development objects). Object Privileges are as below –
	CREATE ANY
	UPDATE, INSERT, SELECT, DELETE, DROP, ALTER, EXECUTE
	INDEX, TRIGGER, DEBUG, REFERENCES

XSC Project Roles

Privileges Types	Description
Analytic Privileges	Analytic Privileges are used to allow read access on data of SAP HANA Information model (attribute view, Analytic View, calculation View).
	This privilege is evaluated during query processing.
	Analytic Privileges grants different user access on different part of data in the
	Same information view based on user role.
	Analytic Privileges are used in SAP HANA database to provide row level data
	Control for individual users to see the data is in the same view.
Package Privileges	Package Privileges are used to provide authorization for actions on individual packages in SAP HANA Repository.
Application Privileges	Application Privileges are required in In SAP HANA Extended Application Services (SAP HANA XS) for access application.
	Application privileges are granted and revoked through the proceduresGRANT_APPLICATION_PRIVILEGE and REVOKE_APPLICATION_PRIVILEGE procedure in the _SYS_REPO schema.
Privileges on User	It is an SQL Privileges, which can grant by the user on own user.
	ATTACH DEBUGGER is the only privilege that can be granted to a user.

XSC Project
Create Dev Role

- Create a new role in the honeycomb folder: developer.hdbrole
- Use the authorizations in the next slide



For Example
XSC Project
Auth for Dev Role 1

Granted Roles

- `sap.hana.xs.ide.roles::EditorDeveloper`
- `sap.hana.xs.debugger::Debugger`

System Privileges

- `repo.export`
- `repo.import`
- `repo.maintain_delivery_units`
- `repo.modify_change`
- `repo.modify_foreign_contribution`
- `repo.modify_own_contribution`
- `repo.work_in_foreign_workspace`

Exercise
XSC Project

Execute the following actions:

1. Create a Repository package in <MATNR> to store your development artifacts.
2. Create a schema to store database objects.
3. Create and run the test HelloWorld application from our Project.

Solution
XSC Project

Execute the following actions:

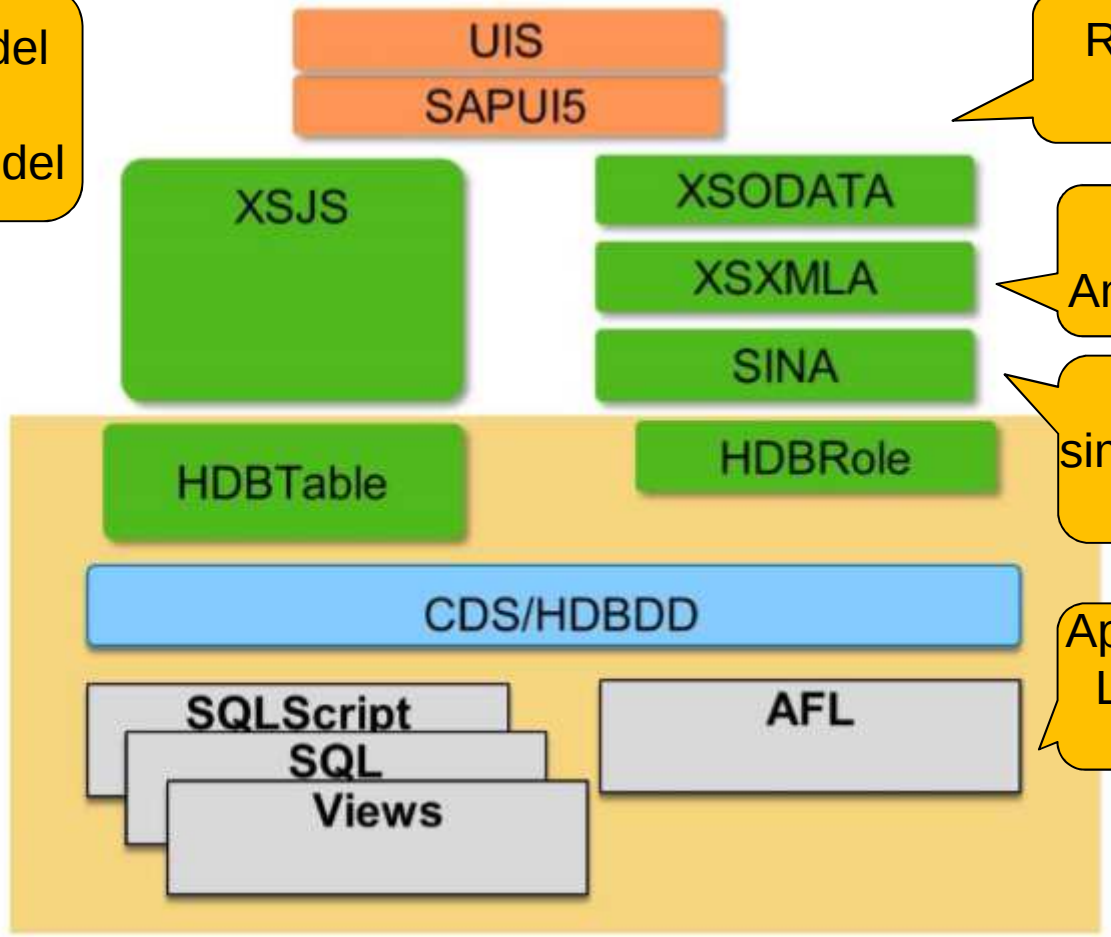
1. Create a Repository package in <MATNR> to store your development artifacts.
2. Create a schema to store database objects.
3. Create and run the test HelloWorld application from our Project.

THE PERSISTENCE MODEL

- Explain basic concepts of Core Data Services

Persistence Model

Persistence Model
vs
Consumption Model



RestService
(oData)

XML for
Analysis (XMLA)

SAP HANA
simple info access
(SINA)

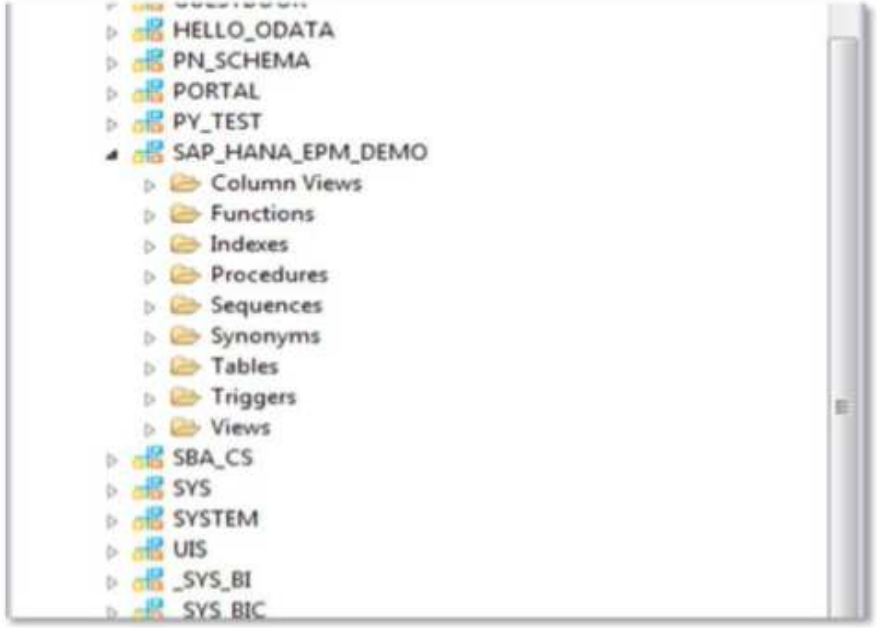
Application Function
Library (AFL), e.g.
PAL, BFL, …

© SAP

Persistence Model
Data Persistence in SAP HANA

Data persistence objects

- **Schemas**
- **Tables**
- **SQL views**
- **Sequences**
- **Procedures**
- **.....**



© SAP

Persistence Model Create

To create objects:

At design time: the developer creates a descriptive representation of the DB object in the Repository.

On activation: the corresponding runtime object is created.

```
namespace workshop.exercises.gsql.data;
@Schema: 'HANA_WORKSHOP_SQL'
context User {
    @Catalog.tableType: #COLUMN
    Entity Details {
        key PERS_NO: String(10);
        FIRSTNAME: String(40);
        LASTNAME: String(40);
        E_MAIL: String(255);
    };
```

Table Name:
workshop.exercises.gsql.data::User.Details

Columns | Indexes | Further Properties | Runtime Information

	Name	SQL Data Type	Di...	Column St
1	PERS_NO	NVARCHAR	10	STRING
2	FIRSTNAME	NVARCHAR	40	STRING
3	LASTNAME	NVARCHAR	40	STRING
4	E_MAIL	NVARCHAR	255	STRING

Persistence Model
Core Data Service

artifact Type	CDS	HDBTable (old)
Schema	.hdbschema	.hdbschema
Table	.hdbdd	.hdbtable
Table Type	.hdbdd	.hdbstructure
View	.hdbdd	.hdbview
Association	.hdbdd	-
Sequence	.hdbsequence	.hdbsequence
Structured Types	.hdbdd	
Data import	.hdbti	.hdbti

Persistence Model
DB Schema for HANA

A schema defines the container that holds database objects such as tables, views, and stored procedures.

Database object

Mandatory

Contains logical groups:

- Tables
- Views
- Procedures
- Sequences
- .....

© SAP

Persistence Model
Creating a Schema

Repository File .hdbschema

Activate

Schema created in the catalog

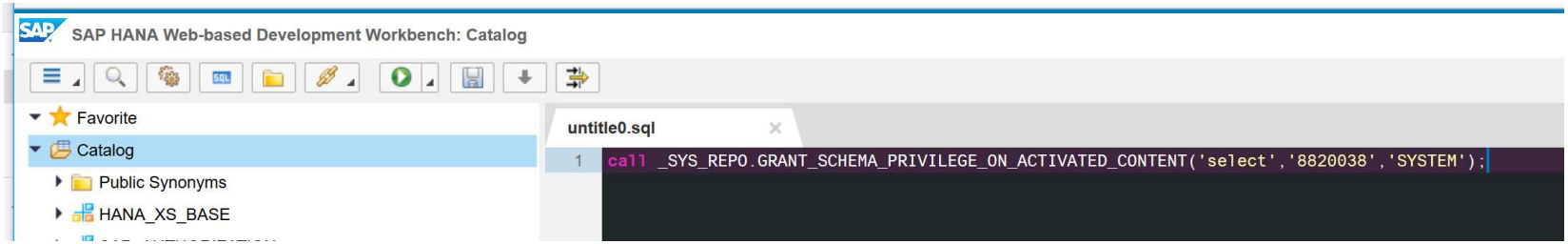
© SAP

Persistence Model Authorizing Schema

- In order to see the schema in the Catalog you have to authorize the viewing user.
- Open the SQL console in the catalog and provide the command:

call _SYS_REPO.GRANT_SCHEMA_PRIVILEGE_ON_ACTIVATED_CONTENT('select',‘<Schema>',‘<User>');

- Execute the command
- Refresh the catalog through the use of the context menu on the Catalog folder



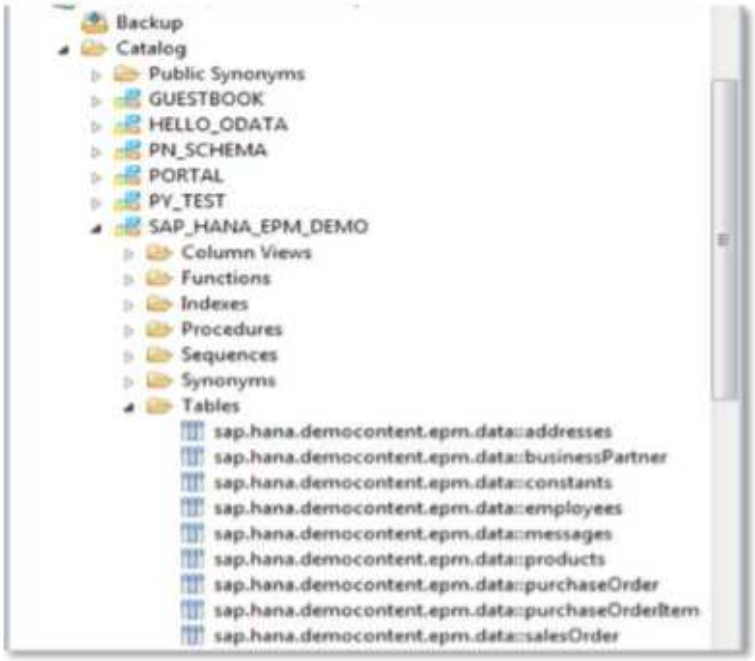
Persistence Model HDBTable

Database object
<package.path>::<TableName>

A set of data elements

Type:
- Column
- Row

Metadata:
- Constraints on table
- Constraints on values in particular columns…



© SAP

Persistence Model
Creating a Table with HDBTable

Repository File .hdbtable

```
table.schemaName = "HANA_WORKSHOP_SOL";
table.tableType = COLUMNSTORE;
table.columns =
 [
    (name = "MY_COL1";  sqlType = VARCHAR;  length = 200; defau
    (name = "MY_COL2";  sqlType = INTEGER;  defaultValue = "1";
    (name = "MY_COL3";  sqlType = NVARCHAR; length = 200; defau
    (name = "MY_COL4";  sqlType = DECIMAL;  precision = 4; scal
    (name = "MY_COL5";  sqlType = DATE; nullable = false; defau
    (name = "MY_COL6";  sqlType = TIME; nullable = false; defau
    (name = "MY_COL7";  sqlType = TIMESTAMP; nullable = false;
 ];
table.primaryKey.pkcolumns = ["MY_COL1", "MY_COL2"];
```

Activate

Table created in the catalog

Table Name:

workshop.exercises.gsol.data::table

Columns | Indexes | Further Properties | Runtime Information

	Name	SQL Data Type	Di...	Column Store Data Type
1	MY_COL1	VARCHAR	200	STRING
2	MY_COL2	INTEGER		INT
3	MY_COL3	NVARCHAR	200	STRING
4	MY_COL4	DECIMAL	4,3	FIXED
5	MY_COL5	DATE		DAYDATE
6	MY_COL6	TIME		SECONDTIME
7	MY_COL7	TIMESTAMP		LONGDATE

Persistence Model
Creating a Sequence

Repository File .hdbsequence

```
schema= "HANA_WORKSHOP_SOL";
start_with= 1000000100;
maxvalue= 1999999999;
nomaxvalue=false;
minvalue= 1;
nominvalue=true;
cycles= false;
depends_on_table= "workshop.exercises.gsol.data::User.Details";
```

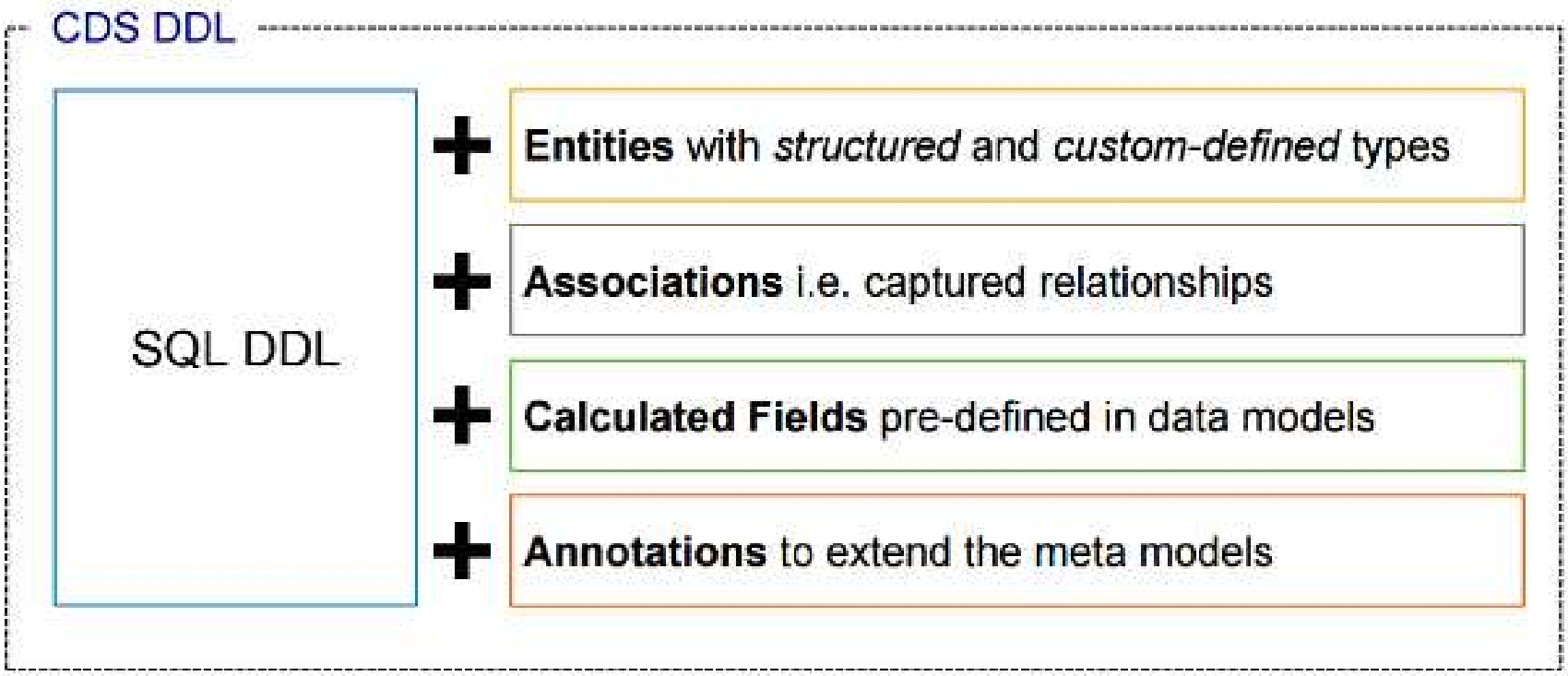
Activate

Sequence created in the catalog

Persistence Model
Core Data Services

- To take advantage of SAP HANA for application development, SAP introduced a new **data modeling infrastructure** known as **core data services**.
- With CDS, data models are defined and **consumed** on the **database** rather than on the application server.
- CDS also offers capabilities beyond the traditional data modeling tools, including support for **conceptual modeling** and **relationship definitions**, **built-in functions**, and **extensions**.

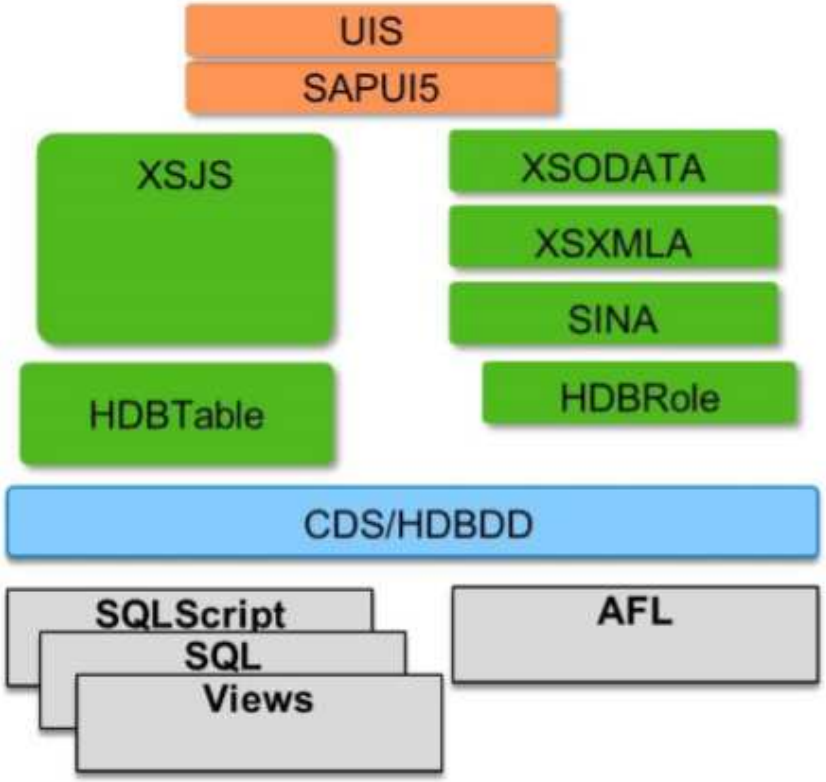
Persistence Model
CDS Dev Model

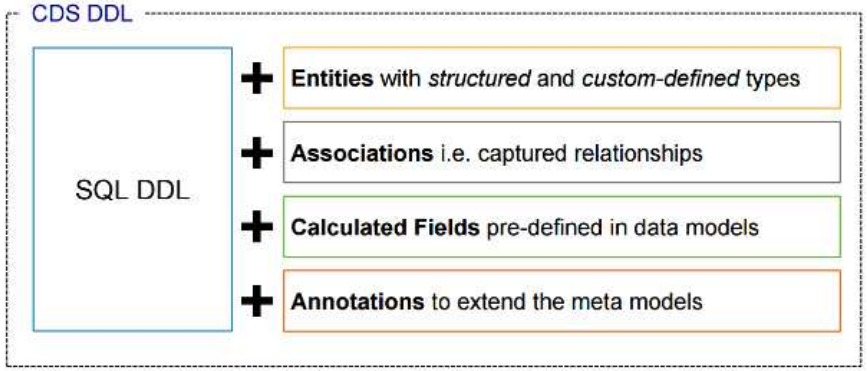


Persistence Model
CDS Dev Model

CDS – Core Data Services

- Data Definition Language
- Query Language
- Expression Language





Persistence Model
CDS Dev Artifacts

- New development artifact extension – hdbdd (Data Dictionary)
- Define the schema to use, reusable types, and multiple entities (tables), and views all within one source file
- Multiple catalog objects are generated upon activation

EmployeeModel.hdbdd

```
namespace demo.cds.CoreDataServicesDemo;

@Schema: 'SAPUIA'
context EmployeeModel {

    type Name : String(80);

    type FullName {
        firstName  : Name;
        middleName : Name;
        lastName   : Name;
    };

    type Street {
        street : String(80);
        number : String(8);
    };

    entity Address {
        key id : Integer;
        street : EmployeeModel.Street;
        city   : String(40);
        zip    : String(16);
    };
```

Persistence Model
CDS Associations

- Associations define relationships between Entities (tables)
- Not yet integrated into the database/SQL but can be referenced when defining views within the same CDS object

```
entity Address {
   owner : Association to Employee;
   key streeAddress : String(77);
   key zipCode : String(11);
   city : String(44);
   kind : String(10);
};

entity OrganizationUnit {
   key name : String(111);
   costcenter : String(44);
   manager : Association [0..1] to Employee;
   parent : Association to OrganizationUnit;
};

entity Employee {
   key ID : String(36); //UUID;
   name : String(77);
   salary : Amount;       // Amount is a struct
   addresses : Association [0..*] to Address;
   orgunit : Association to OrganizationUnit;
```

Persistence Model
CDS Views

- CDS syntax expands to include the definition of views
- Views can use the defined associations between entities

```
define view EmployeesView as SELECT from Employee {
   /*key*/ ID, name,
   orgunit.name as orgunitname
};


define view OrgUnitSalaries //with parameters
as SELECT from Employee {
   orgunit.name as orgunit,
   SUM (salary) as totalSalaries
} group by orgunit;
```

Persistence Model
Table Import

File .hdbti

```
import =
[{
    cdstable = "workshop.exercises.gsol.data::User.Details";
    file = "workshop.exercises.gsol.data.loads:UserDetails.csv";
    header = false;
}];
```

File .csv

```
1000000000,Akira,Kurosawa,akira.kurosawa@unknown.com
1000000001,Federico,Fellini,federico.fellini@unknown.com
1000000002,Igmar,Bergman,igmar.bergman@unknown.com
1000000003,John,Ford,john.ford@unknown.com
```

DB Table filled with data

Raw Data | Distinct values | Analysis

Filter pattern — 4 rows retrieved - 21 ms

PERS_NO	FIRSTNAME	LASTNAME	E_MAIL
1000000000	Akira	Kurosawa	akira.kurosa...
1000000001	Federico	Fellini	federico.felli...
1000000002	Igmar	Bergman	igmar.bergm...
1000000003	John	Ford	john.ford@u...

Excercise
Create CDS Tables in XSC

1. In the exercises-<MATNR> project, create a new package **honeycomb_data** and within create a data definition file named **honeycomb.hdbdd**.
2. In the file, within the honeycomb context, define an entity named **Measures**, with the columns defined on the next slide.
3. Activate the file.
4. Check that the DB table was correctly created in the SB schema.

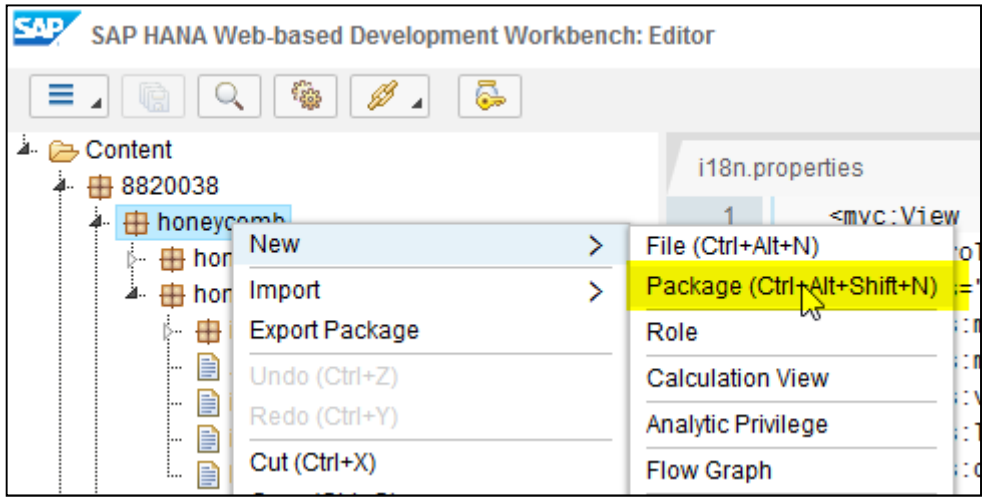
Excercise
Column Details

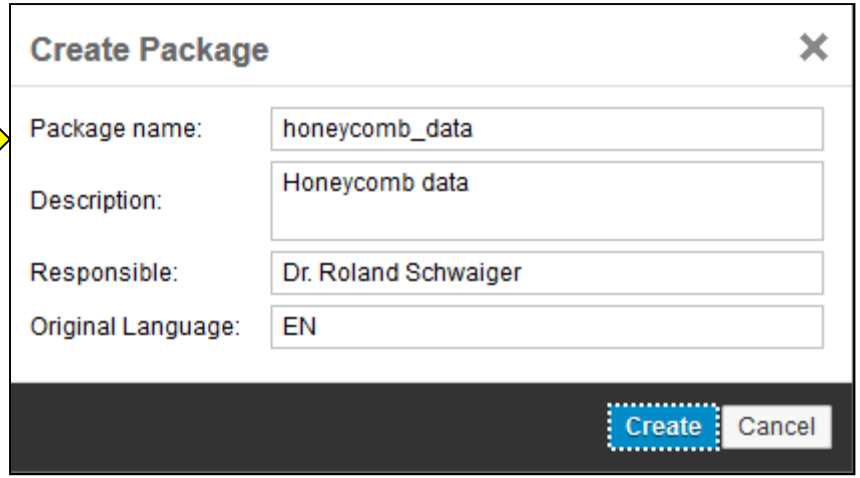
- BEEKEEPERID: String(10); -> OK
- HONEYCOMBID:String(4); -> OK
- MEASUREDATE: LocalDate; -> Format yyyy-mm-dd, e.g. 2017-01-07
- MEASURETIME: LocalTime; -> Format hh:mm:ss, e.g. 09:05:00
- WEIGHT: Decimal(5,2); -> e.g. 33.00
- WEIGHTUNIT: String(2); -> Empty or unit, e.g. kg (SI-Writing)
- TEMPERATURE: Decimal(5,2); -> e.g. 27.50
- TEMPERATUREUNIT: String(2); -> Empty or ([C|F], e.g. C
- BEEKEEPERNAME: String(40); -> Optional

Excercise
Hints

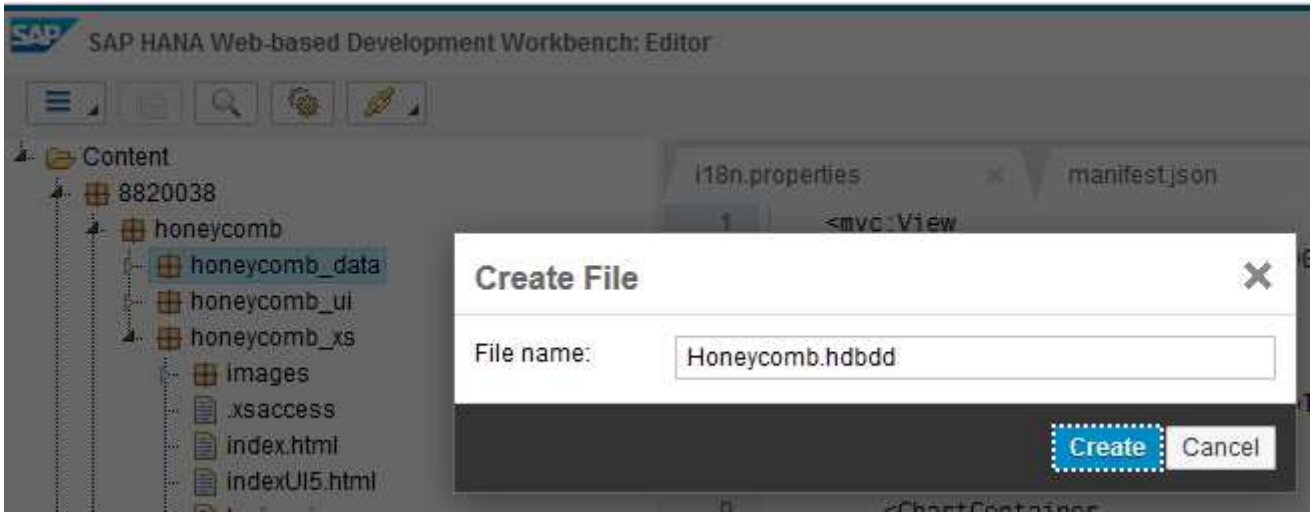
- Numbers have to be quoted in the namespace
- CDS Primitive Types: https://help.sap.com/saphelp_hanaplatform/helpdata/en/cf/394efd3fb4400f9c09d10315028515/content.htm

Solution
Honeycomb data package





Solution
Honeycomb CDS file



Solution
Honeycomb CDS file

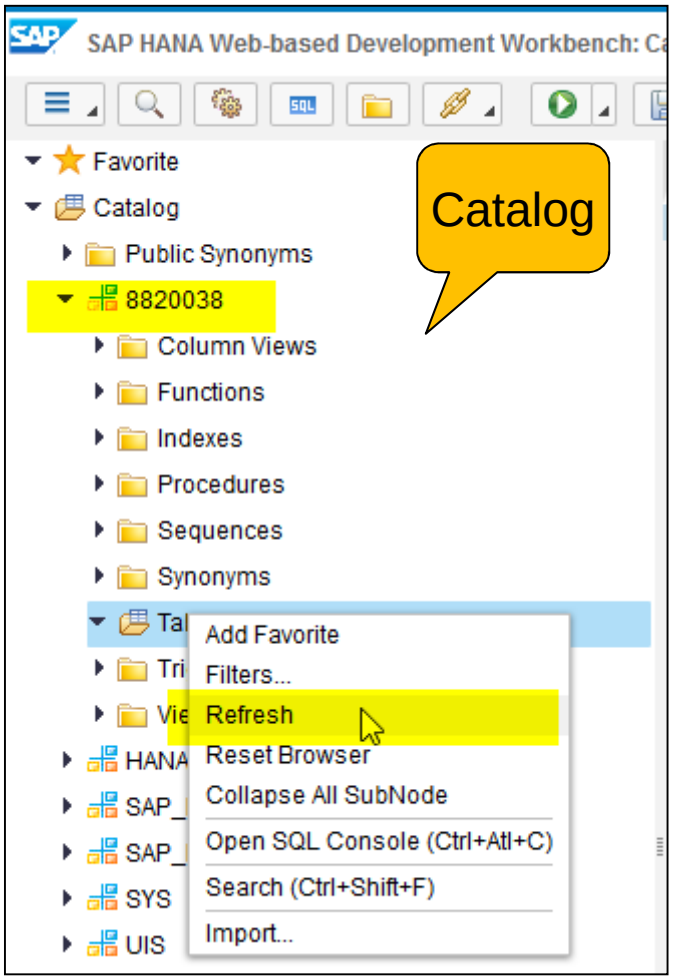
```
namespace 8820038.honeycomb.honeycomb_data;
@Schema: '8820038'
context Honeycomb {
   @Catalog.tableType: #COLUMN
   Entity Measures {
      KEY ID: Integer;
      BEEKEEPERID: String(10);
      HONEYCOMBID:String(4);
      MEASUREDATE: LocalDate;
      MEASURETIME: LocalTime;
      WEIGHT: Decimal(5,2);
      WEIGHTUNIT: String(2);
      TEMPERATURE: Decimal(5,2);
      TEMPERATUREUNIT: String(2);
      BEEKEEPERNAME: String(10);
   };
};
```

Path in the editor.
!Note the quoted number!

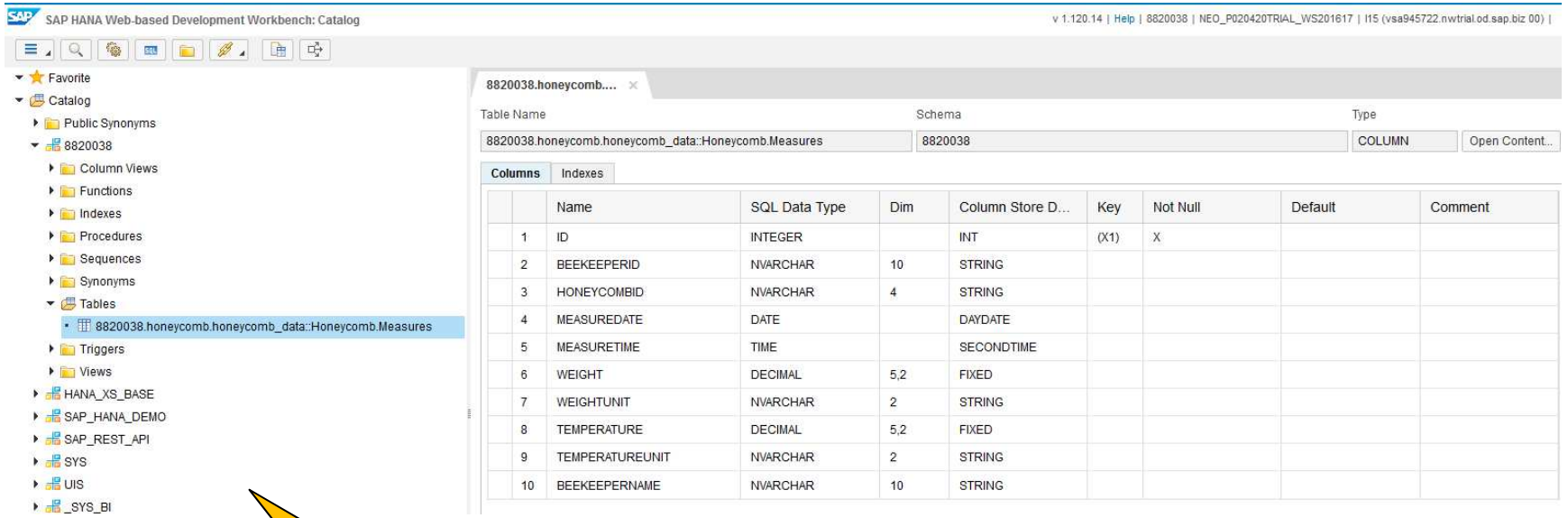
The schema from the catalog view

Context.Entity
A means to cluster the tables

Solution
CDS Activation 1



Solution
CDS Activation 2



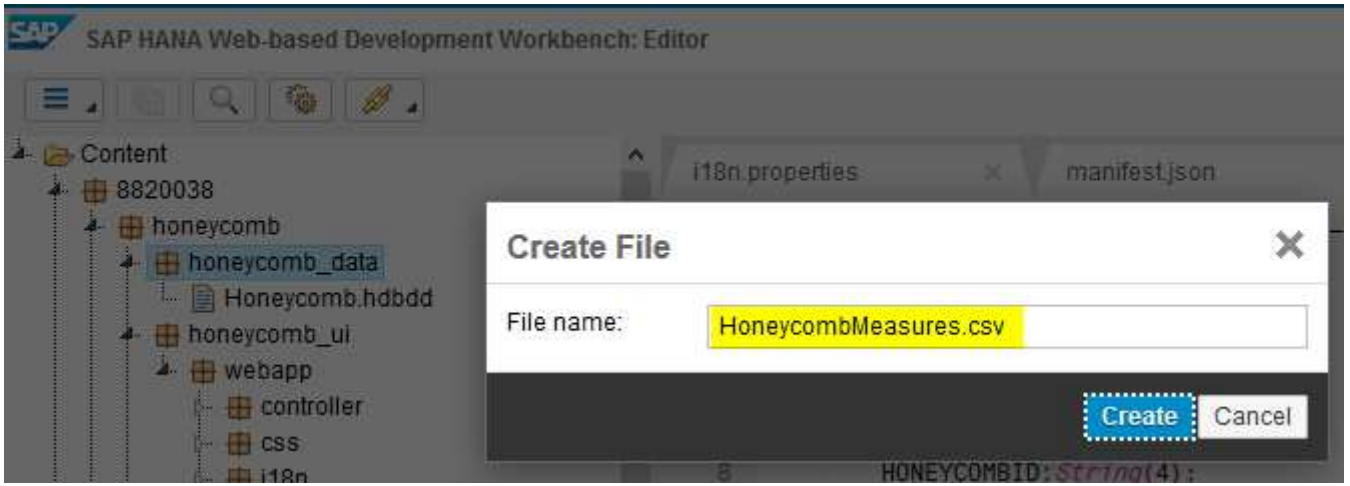
Solution
Import Data

- Create a new file **honeycomb_data/HoneycombMeasures.csv** containing the lines defined in the next slide.
- Execute a table import of the file in the **Honeycomb.Measures** table that you created in the previous exercise. Define the import definition **Honeycomb.hdbti**

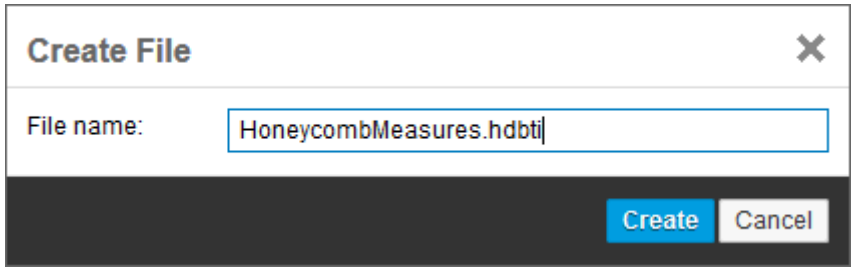
Solution
Data File

1,1,0001,2017-01-01,10:00:00,30.0,kg,27.00,C,lois
2,1,0001,2017-01-02,10:00:00,31.0,kg,27.00,C,lois
3,1,0001,2017-01-03,10:00:00,33.0,kg,27.00,C,lois
4,1,0001,2017-01-04,10:00:00,37.0,kg,27.00,C,lois
5,1,0001,2017-01-05,10:00:00,29.0,kg,27.00,C,lois
6,1,0001,2017-01-06,10:00:00,35.0,kg,27.00,C,lois

Solution
Data File



Solution
Import File



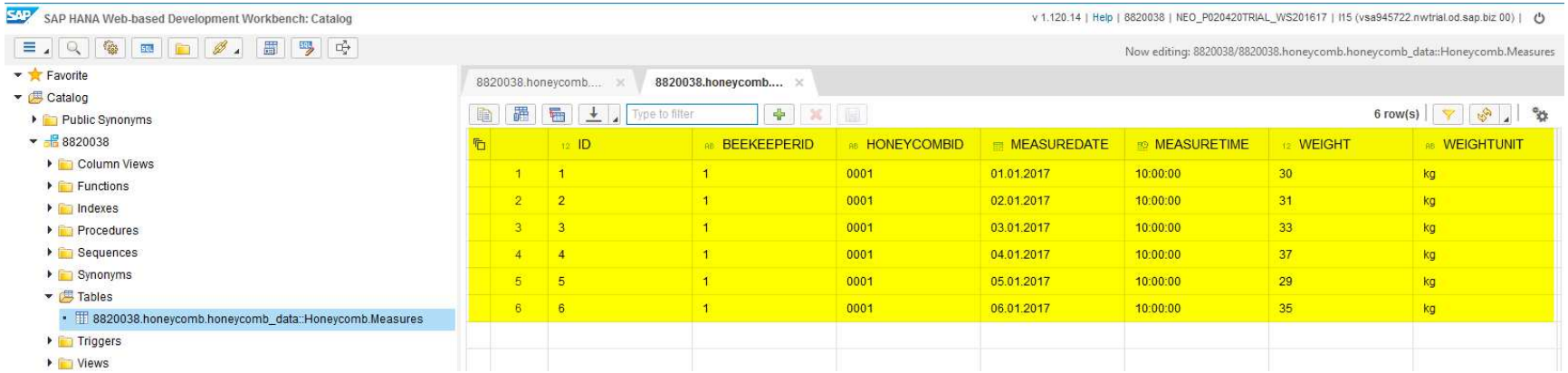

```
import =
[{
    cdstable = 8820038.honeycomb.honeycomb_data::Honeycomb.Measures;
    file = 8820038.honeycomb.honeycomb_data:HoneycombMeasures.csv;
    header = false;
}];
```

Save File = Activation

Editor Console

```
[09:41:59] File /8820038/honeycomb/honeycomb_data/Honeycomb.hdbti saved & activated successfully.
[09:44:15] File /8820038/honeycomb/honeycomb_data/HoneycombMeasures.csv saved & activated successfully.
[09:44:39] File /8820038/honeycomb/honeycomb_data/Honeycomb.hdbti saved & activated successfully.
```

Solution
Import File



Catalog
Show Content

Notes

- Once the database tables are activated again the order of the columns may change in the table randonmly. Thus you have to check the order and possibly change it in the csv file.
- Use Integer for foreign keys! String does not work.

ODATA SERVICES

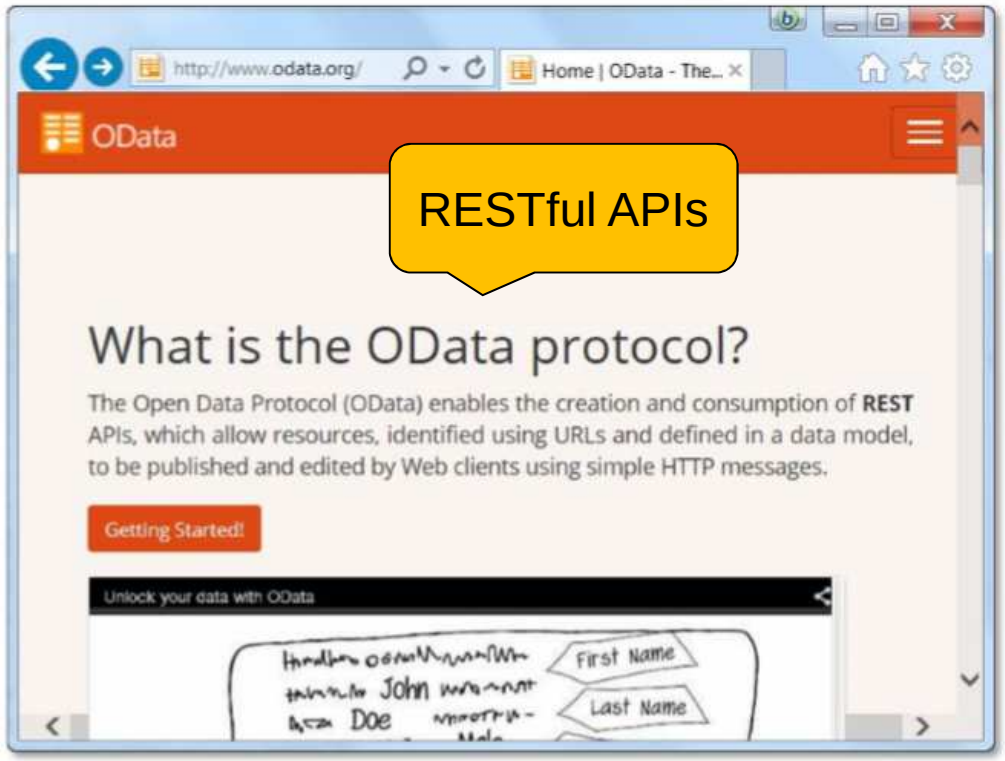
- Explain basic concepts of Odata in SAP HANA
- OData Services on XSC

oData Intro

The Open Data Protocol (OData) enables the creation and consumption of REST APIs, which allow resources to be published and edited by Web clients using simple HTTP messages.

It is an open standard, by the OASIS consortium.

Detailed reference information at http://www.odata.org/

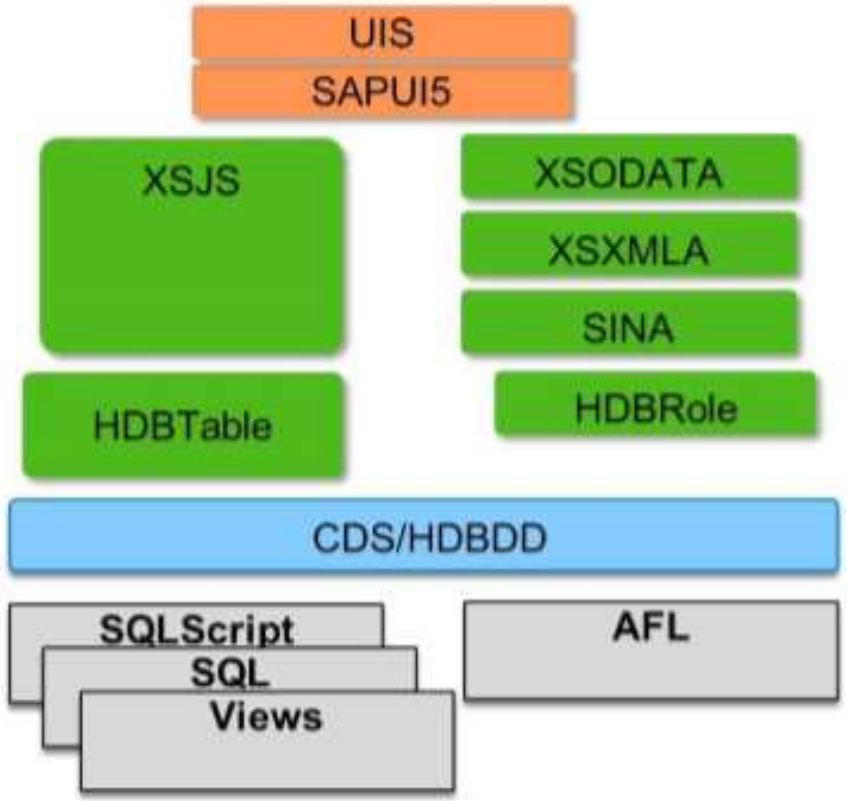


© SAP

oData
oData in SAP HANA

XSODATA

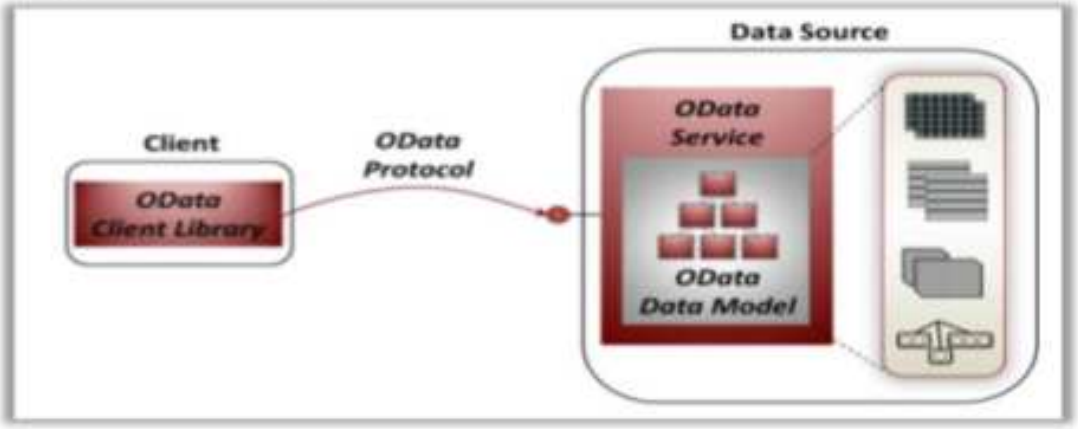
- Low coding OData REST service generation framework
- Based on existing tables and views
- Create/Update/Deletion operations support added in SPS 06



© SAP

oData Basics

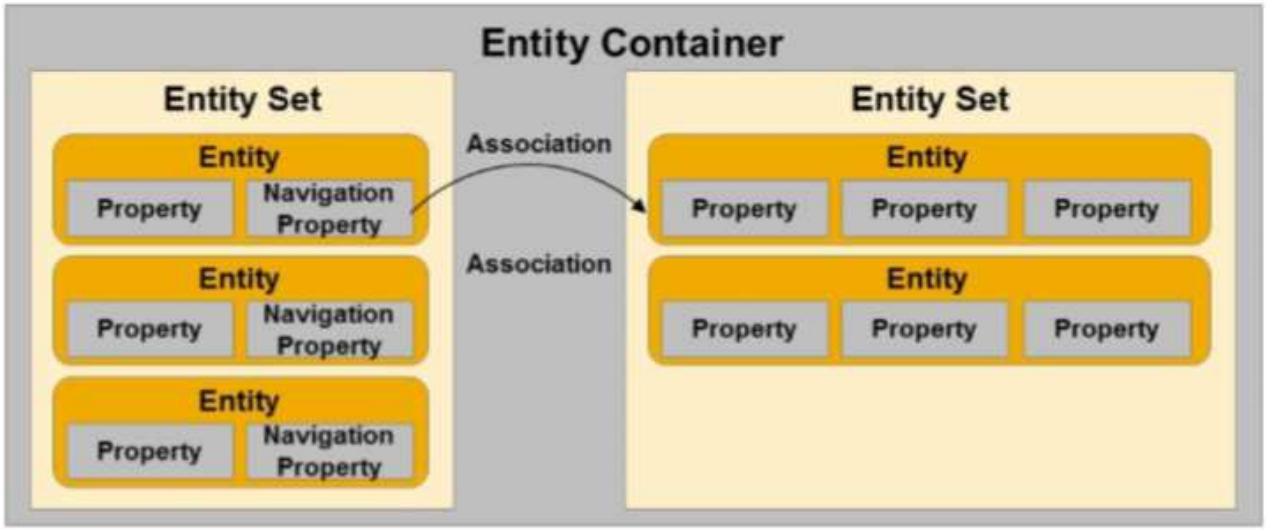
- **OData Data Model**
  Organize/describe data with Entity Data Model (EDM)
- **OData Protocol**
  REST-based create, read, update, and delete + OData-defined query language
- **Client Libraries**
  Pre-built libraries to request OData and display results
- **OData Services**
  Exposes an end point that allows access to data in the SAP HANA database



© SAP

oData
Funcs in SAP HANA

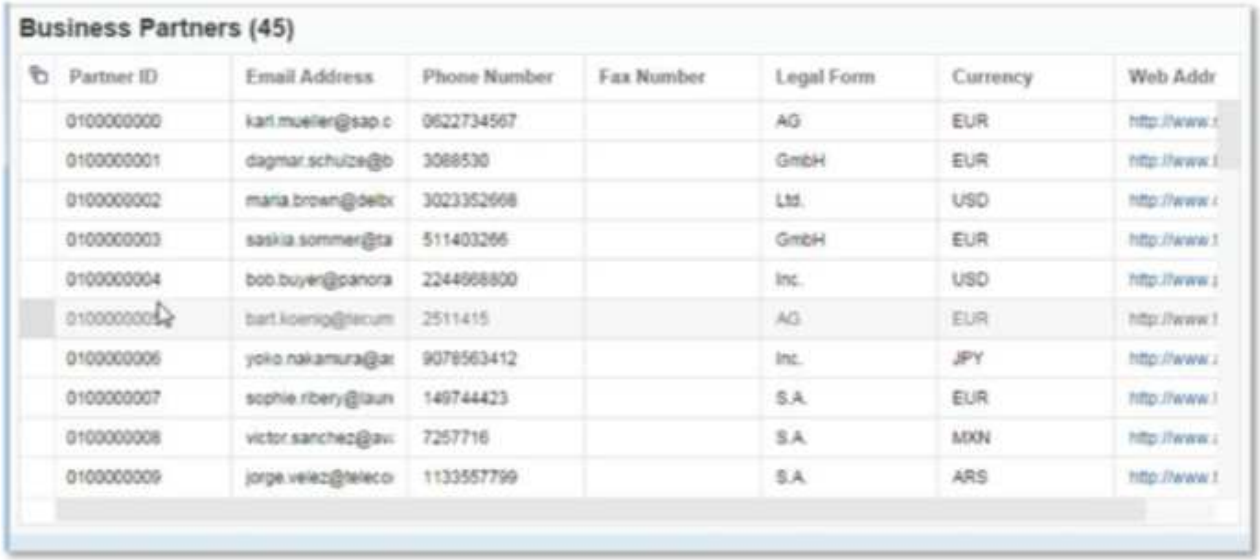
- **Aggregation**
  Use functions to aggregate columns
- **Associations**
  Express relationships between entities
- **Key Specification**
  EntityType denotes a set of properties forming a unique key
- **Parameter Entity Sets**
  Input parameters for SAP HANA calculation views and analytic views
- **Property Projection**
  Restrict the number of selected columns to expose



© SAP

oData
Dev Process

- **Data End Point**
  Define which data to expose (for example, table)
- **OData Service Definition**
  Define mechanism to expose authorized data
- **UI Call to OData Service**
  Link UI table view to the OData service
- **OData Display/Test**
  URL filters: $top, $format, $orderby, $filter, $select…



Business Partners (45)

Partner ID	Email Address	Phone Number	Fax Number	Legal Form	Currency	Web Addr
0100000000	kari.mueller@sap.c	0622734567		AG	EUR	http://www.s
0100000001	dagmar.schulze@b	3088530		GmbH	EUR	http://www.b
0100000002	maria.brown@delbx	3023352668		Ltd.	USD	http://www.c
0100000003	saskia.sommer@ta	511403266		GmbH	EUR	http://www.t
0100000004	bob.buyer@panora	2244868800		Inc.	USD	http://www.p
0100000005	bart.koenig@lecum	2511415		AG	EUR	http://www.t
0100000006	yoko.nakamura@ac	9078563412		Inc.	JPY	http://www.a
0100000007	sophie.ribery@laun	149744423		S.A.	EUR	http://www.l
0100000008	victor.sanchez@av	7257716		S.A.	MXN	http://www.a
0100000009	jorge.velez@teleco	1133557799		S.A.	ARS	http://www.t

© SAP

oData
Service Def

An OData service is defined with the activation of an .xsodata file in the Repository.

```
service {
        "sample.odata::table" as "MyTable";
}
```

Resulting service at:

```
<service xml:base="http://localhost:8002/sample/odata/repo.xsodata/"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app" xmlns="http://www.w3.org/2007/app">
    <workspace>
      <atom:title>Default</atom:title>
      <collection href="MyTable">
        <atom:title>MyTable</atom:title>
      </collection>
    </workspace>
</service>
```

© SAP

oData
Service Def

Resulting service metadata:

`http://myHost:myPort/sample/odata/repo.xsodata/$metadata`

```
<edmx:Edmx Version="1.0"
xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <edmx:DataServices m:DataServiceVersion="2.0"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <Schema Namespace="sample.odata.repo"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
      <EntityType Name="MyTableType">
        <Key>
          <PropertyRef Name="ID"/>
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
        <Property Name="Text" Type="Edm.String" Nullable="true"
        MaxLength="1000"/>
        <Property Name="Time" Type="Edm.DateTime" Nullable="true"/>
      </EntityType>
      <EntityContainer Name="repo" m:IsDefaultEntityContainer="true">
        <EntitySet Name="MyTable"
        EntityType="sample.odata.repo.MyTableType"/>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

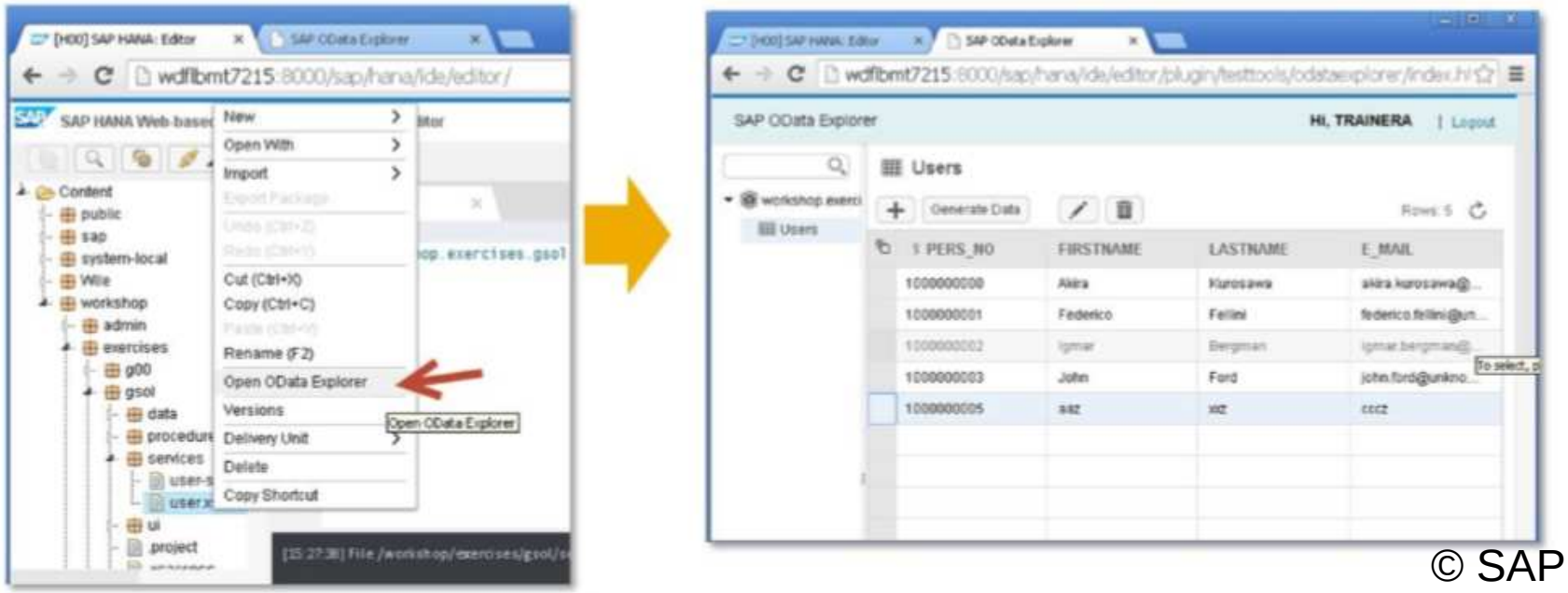
© SAP

oData Explorer

SAP OData Explorer allows you to browse the data accessible via an OData service.

In addition, it allows you to execute *Create*, *Update*, and *Delete* actions on the service content.

It is accessible via the Web-based IDE Editor, running Open OData Explorer from the context menu.



© SAP

oData
Read / Write

By default, the created OData service is write-enabled, so a user with proper authorizations on the destination table can use it to create, change, or delete table lines.

It is possible to forbid data modification via explicit options:

```
service { "sap.test::myTable"
    create forbidden
    update forbidden
    delete forbidden;
}
```

© SAP

oData
DataBinding in SAPUI5

- Read-only binding with OData is similar to binding with JSON.
- You just need to use a different data motel type.

JSON-Model:

```
var oModel = new sap.ui.model.json.JSONModel(dataUrlOrData);
```

OData-Model:

```
var oModel = new sap.ui.model.odata.ODataModel(dataUrl [, useJSON, user,  pass] );
```

oData
Two-Way Binding

To realize a **write-enabled** OData binding, you need to:

- Activate two-ways binding in the model:

```
oModel = new sap.ui.model.odata.ODataModel(
        "/workshop/exercises/gsol/services/user.xsodata/", true
);
oModel.setDefaultBindingMode(sap.ui.model.BindingMode.TwoWay);
```

- Call properly, when required, the three model methods:

```
model.create()
model.remove()
Model.submitChanges()
```

© SAP

oData
oModel.create()

```
var oEntry = {};
oEntry.PERS_NO       =
oEntry.FIRSTNAME     =
oEntry.LASTNAME      =
oEntry.E_MAIL        =


oModel.create('/Users', oEntry, null,
    function() {alert("Create successful"); },
    function() {alert("Create failed");} );
```

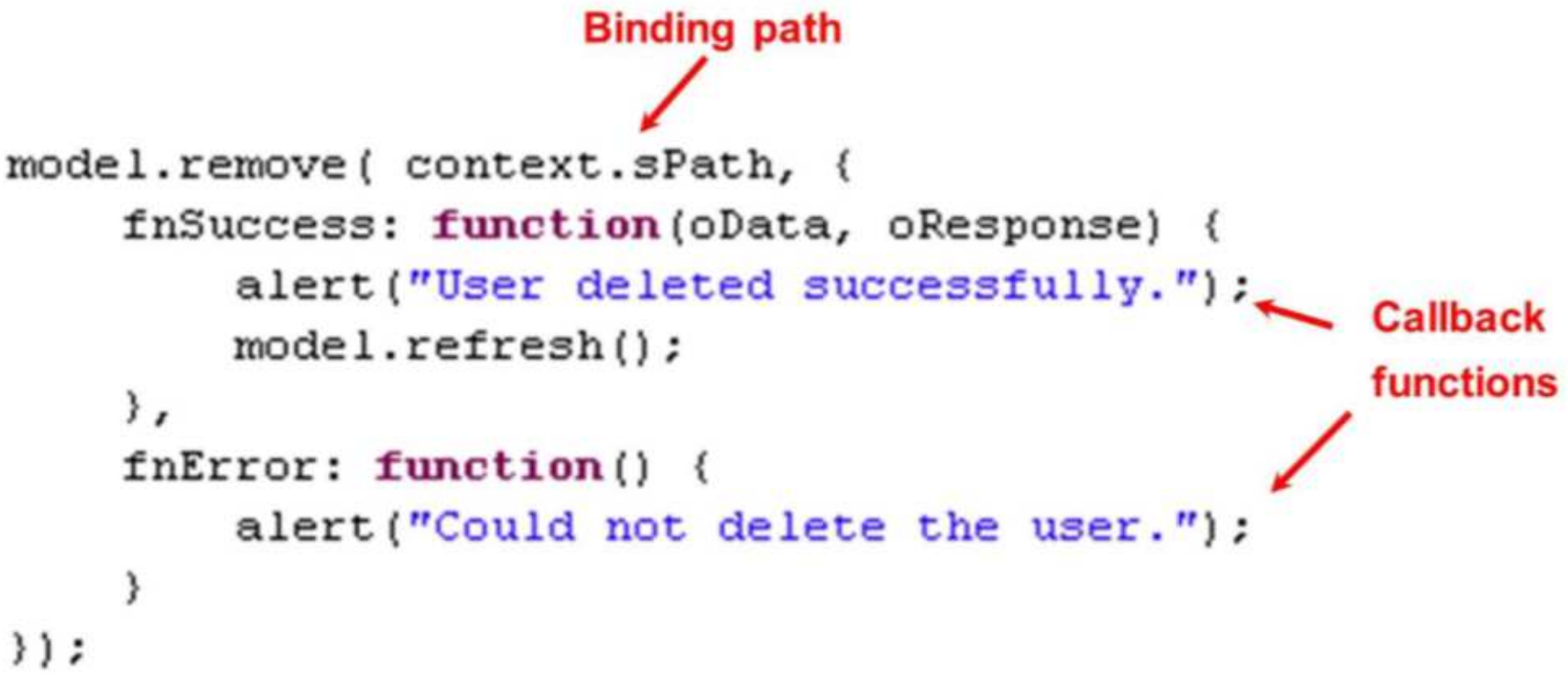
New table entry

Binding path

Callback functions

© SAP

oData
oModel.remove()



© SAP

oData
oModel.submitChanges()

```
oModel.submitChanges(
    function() {alert("Update successful");},
    function() {alert("Update failed");} );
```

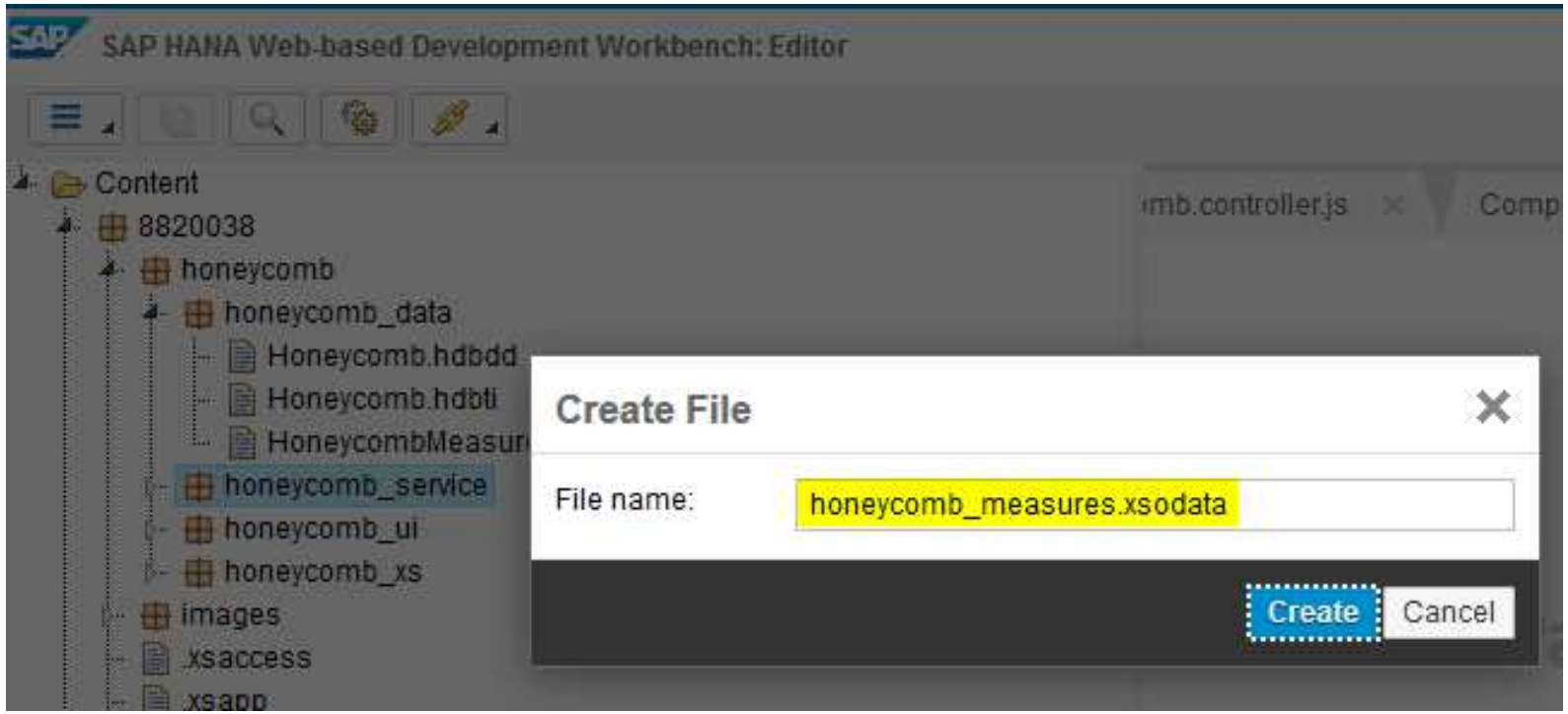
Callback functions

© SAP

Exercise oData

- Create an **honeycomb_service/honeycomb_measures.xsodata** service for the table **Honeycomb:Measures**.
- Test your service (oData Explorer).
- Modify the **honeycomb ui** (**Honeycomb.controller.js**) so that it displays the data from the h**oneycomb_measures.xsodata** service created in the previous step.
- Change the JSON data binding of the table to a **data binding** with the oData service created in the previous exercise.
- Activate and **run** the files.
- Create and set a formatter for the Date typed columns.
- Activate and run the files.

Solution honeycomb_measures.xsodata



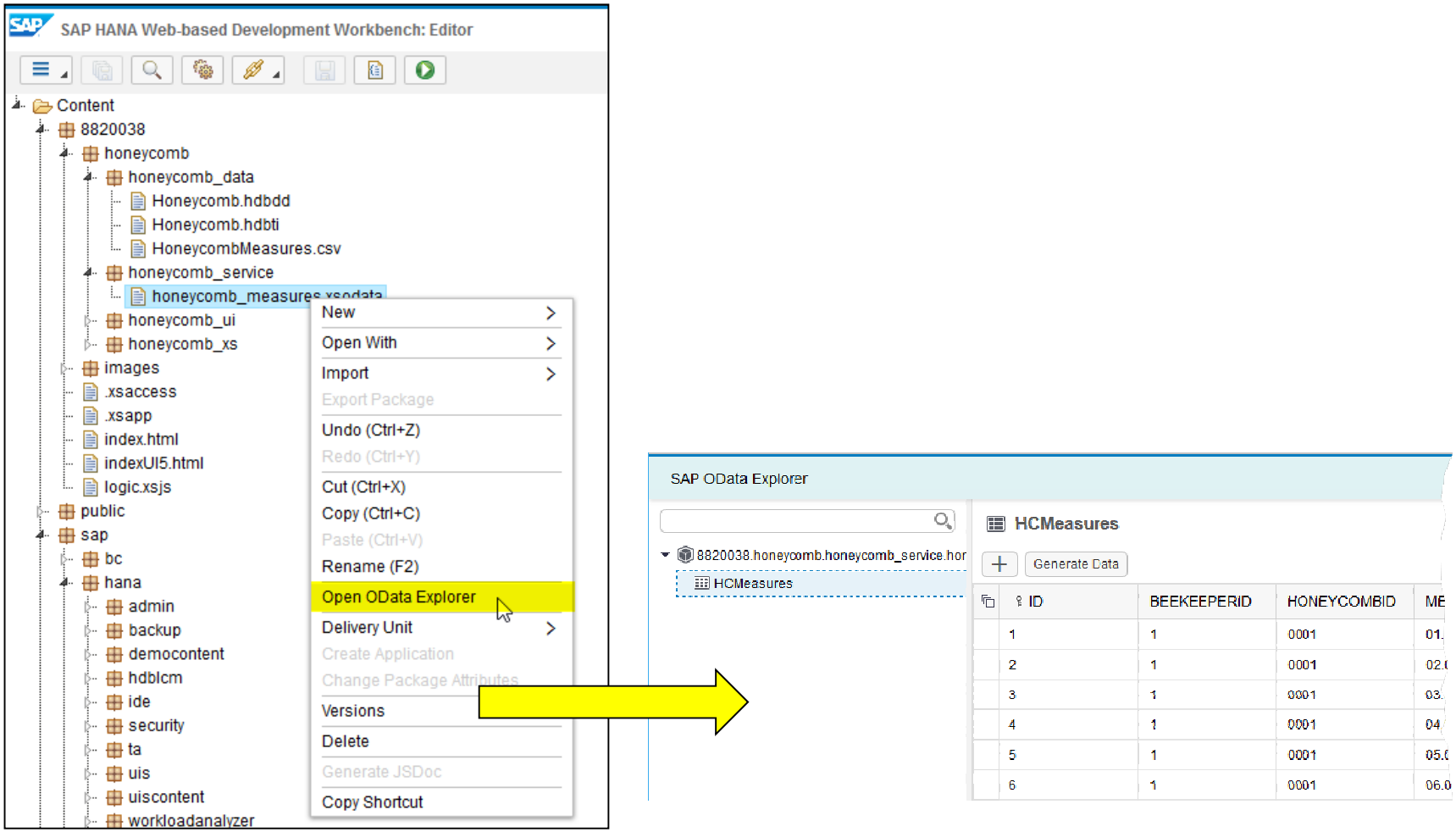
Solution honeycomb_measures.xsodata

```
service{
        "8820038.honeycomb.honeycomb_data::Honeycomb.Measures" as
"HCMeasures";
}
```

Pfad:

https://ws20162017p1942292818trial.hanatrial.ondemand.com/8820038/honeycomb/honeycomb_service/honeycomb_measures.xsodata/HCMeasures/$format=json

Solution
Test oData



Solution
Change App for oData

1) Change manifest.json for the new data source in the „sap.app“ section.

```
"dataSources": {
  „hcmeasureRemote": {
                "uri": "8820038/honeycomb/honeycomb_service/honeycomb_measures.xsodata/",
                "type": "OData",
                "settings": {
                  "odataVersion": "2.0"
                }
  }
}
```

2) Add a new model in the manifest.json for the new data source in the „sap.ui5/models“ section.

```
"hcmeasure": {
                "dataSource": "hcmeasureRemote
}
```

3) In Honeycomb.controller.js insert in init method the creation of the model.

Request URL
https://ws201617p020420trial.hanatrial.ondemand.com/8820038/honeycomb/honeycomb_service/honeycomb_measures.xsodata/HCMeasures

Solution
Change App for oData (B)

- **Problem**: manifest.json instantiation does not work, Component instantiation does not work, App.controller instantiation does not work
- **Solution**: Instantiate oData JSON model in **init method** of Honeycomb.controller and store it it this.getOwnerComponent().setModel(„measures“)
- **Change**: Binding of measures table in Honeycomb.controller
- **Problem**: oData service gives back JS Date object for Date column
- **Solution**: Create Formatter for corresponding columns

SQLSCRIPT

- Explain basic concepts about SQLScript
- Use the SQLScript Wizard and Editor
- Use the SQLScript Debugger
- Explain main SQLScript language features
- Use Functions in SQLScript
- Use Triggers in SQLScript
- Explain how the R language can be used in SAP HANA

http://help.sap.com/hana/SAP_HANA_SQL_Script_Reference_en.pdf

SQLScript Definition

SQLScript is:

- An interface for applications to access SAP HANA.
- An extension of ANSI Standard SQL.
- A language for creating stored procedures and user defined functions in HANA.
- A combination set based on declarative SQL with imperative control flow constructs, also suitable for mass data processing and OLTP scenarios.

Two reasons why this is required to achieve the best performance:

- Eliminates data transfer between database and application tiers.
- Calculations need to be executed in the database layer to gain maximum benefit from SAP HANA features, such as fast column operations, query optimization, and parallel execution.

SQLScript
SQL vs. SQLScript

Compared to plain SQL queries, SQLScript has the following advantages:

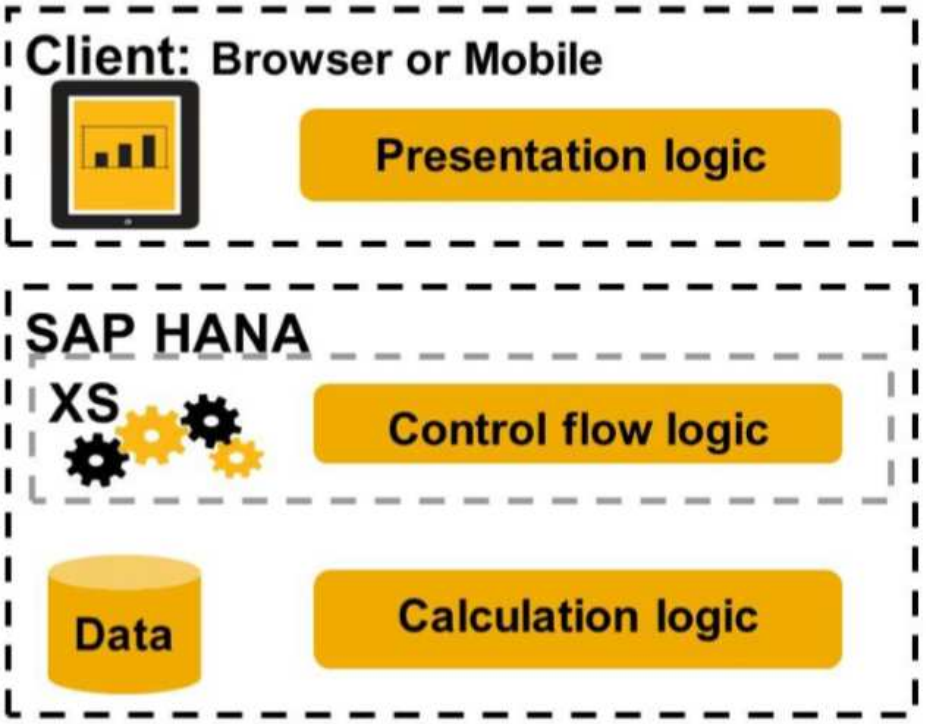
- Returns multiple results, while a SQL query returns only one result set.
- Complex logic can be broken down into smaller chunks of code. This enables modular programming, reuse and a better understanding by functional abstraction. For structuring complex queries, standard SQL only allows the definition of SQL views. However, SQL views have no parameters.
- Complex SQL statement can be well structured, using table variables, improving maintainability, and gaining performance by parallel execution based on statement dependencies. With standard SQL, it would be necessary to define globally visible views, even for intermediate steps.
- SQL Script has control logic, such as if/else, which is not available in SQL.

SQLScript Technologies

Front-end Technologies
- http/s
- HTML5 / SAPUI5
- Client-side JavaScript

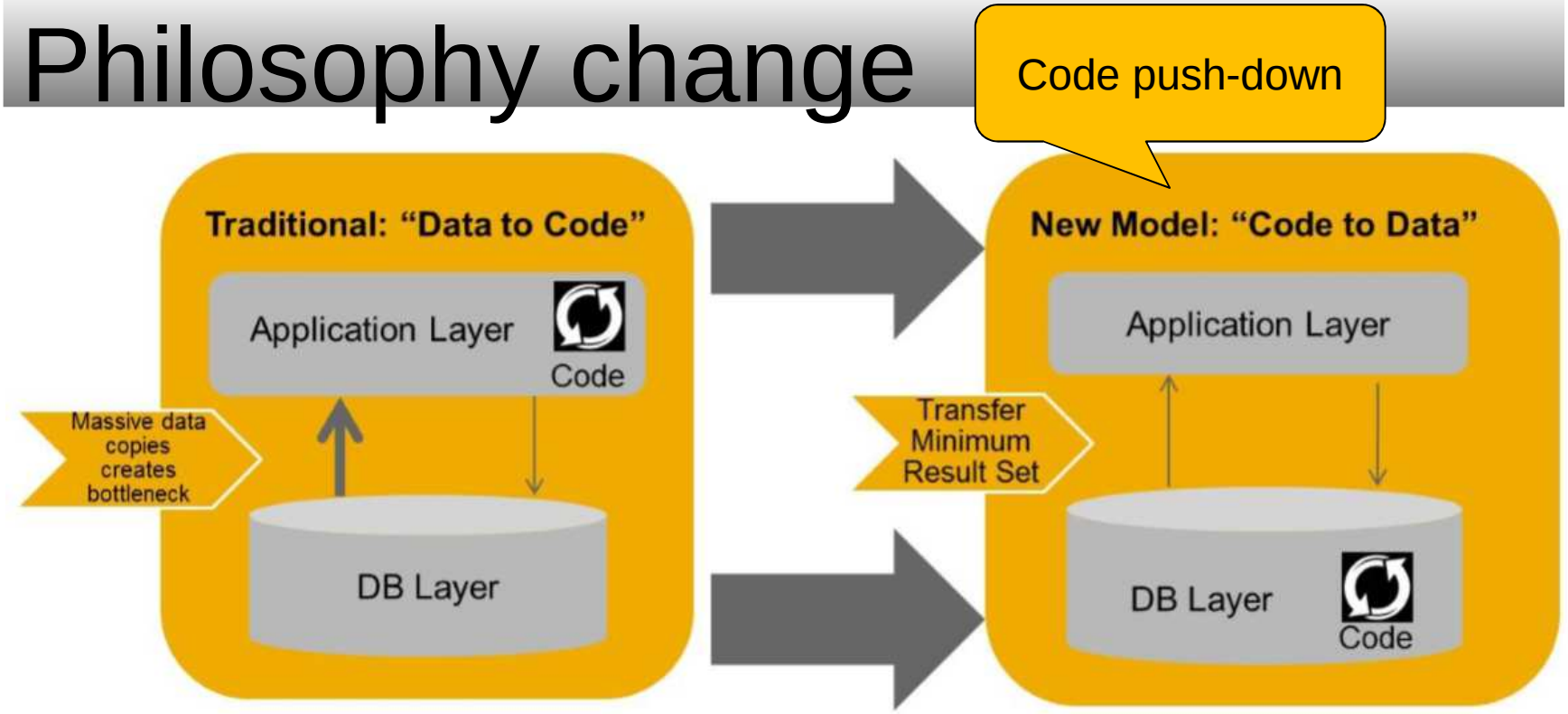
Control Flow Technologies
- OData
- Server-Side JavaScript
- XMLA

Data Processing Technologies
- SQL / SqlScript
- Application Function Library (AFL)



© SAP

SQLScript Philosophy change



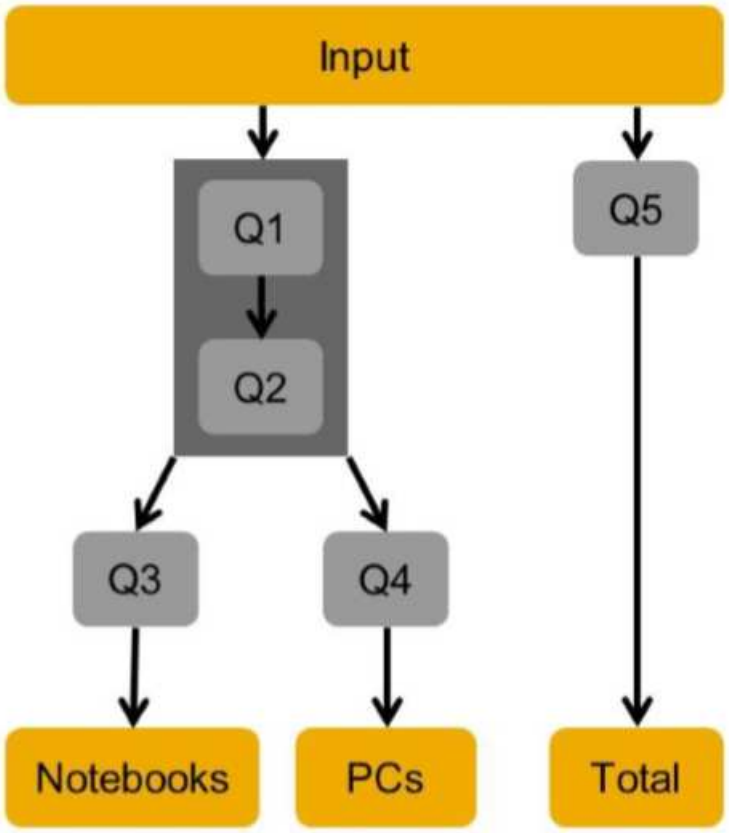
Code push-down: The SAP Business Suite on HANA does this in a couple of ways, for example:

- in ABAP, we can now leverage SAP HANA views by exposing them to the ABAP dictionary via **External Views**.
- We can also expose **SQLScript procedures** and call them directly from ABAP using the CALL DATABASE PROCEDURE statement

© SAP

SQLScript
Code example

```
BEGIN
 ...

  -- Query 1
  product_ids = select "ProductId", "Category", "DescId"
      from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::products"
        where "Category" = 'Notebooks' or "Category" = 'PC';

  -- Query 2
  product_texts = select "ProductId", "Category", "DescId", "Text"
      from :product_ids as prod_ids
     inner join "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::texts"
          as texts on prod_ids."DescId" = texts."TextId";

  -- Query 3
  out_notebook_count = select count(*) as cnt from
                       :product_texts where "Category" = 'Notebooks';

  -- Query 4
  out_pc_count = select count(*) as cnt from
                       :product_texts where "Category" = 'PC';

  -- Query 5
  out_total_count = select count(*) as cnt
      from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::products";
  ...

END;
```



© SAP

SQLScript Select

SELECT statements are executed in parallel unless:

- Any local scalar parameters and variables are used in the procedure
- Any read/write procedures or DML/DDL operations are executed
- Any imperative logic is used within the procedure, such as IF statements of FOR loops
- Any SQL statements are used that are not assigned to an intermediate variable or parameter

SQLScript Schema

- Schema definition – developer can define which schema in which the run-time object of the procedure will be created.

SAP HANA SQLScript

```
1  PROCEDURE SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.procedures::get_bp_addresses_by_role" (
2                        IN im_partnerrole NVARCHAR(3) DEFAULT '01',
3                        OUT ex_bp_addresses SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.data::EPM.Procedures.tt_bp_addresses" )
4      LANGUAGE SQLSCRIPT
5      SQL SECURITY INVOKER
6      DEFAULT SCHEMA SAP_HANA_EPM_NEXT
7      READS SQL DATA AS
8  BEGIN
9  /*****************************
10     Write your procedure logic
11  *****************************/
12 ex_bp_addresses =
13      select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS", a."COMPANYNAME",
14             a."ADDRESSES.ADDRESSID" as "ADDRESSID", b."CITY", b."POSTALCODE", b."STREET"
15               from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.BusinessPartner" as a
16                  inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Addresses" as b
17                      on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
18                               where a."PARTNERROLE" = :im_partnerrole;
19
20 END;
```

SQLScript
Package Hierarchy

- Package hierarchy and procedure name – contains the complete package hierarchy as well as the name of the procedure, separated by double colon(::).

SAP HANA SQLScript

```
1  PROCEDURE SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.procedures::get_bp_addresses_by_role"
2                          IN im_partnerrole NVARCHAR(3) DEFAULT '01',
3                          OUT ex_bp_addresses SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.data::EPM.Procedures.tt_bp_addresses" )
4      LANGUAGE SQLSCRIPT
5      SQL SECURITY INVOKER
6      DEFAULT SCHEMA SAP_HANA_EPM_NEXT
7      READS SQL DATA AS
8  BEGIN
9  /*****************************
10     Write your procedure logic
11  *****************************/
12 ex_bp_addresses =
13      select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS", a."COMPANYNAME",
14             a."ADDRESSES.ADDRESSID" as "ADDRESSID", b."CITY", b."POSTALCODE", b."STREET"
15               from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.BusinessPartner" as a
16                  inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Addresses" as b
17                      on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
18                              where a."PARTNERROLE" = :im_partnerrole;
19
20 END;
```

SQLScript
Input/Output parameter

- Input/output parameter definition – developer can define both input parameters with default values as well as output parameters. Parameters can reference simple types, or global types defined via CDS.

SAP HANA SQLScript

```
1  PROCEDURE SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.procedures::get_bp_addresses_by_role" (
2                          IN im_partnerrole NVARCHAR(3) DEFAULT '01',
3                          OUT ex_bp_addresses SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.data::EPM.Procedures.tt_bp_addresses" )
4      LANGUAGE SQLSCRIPT
5      SQL SECURITY INVOKER
6      DEFAULT SCHEMA SAP_HANA_EPM_NEXT
7      READS SQL DATA AS
8  BEGIN
9  /*****************************
10     Write your procedure logic
11  *****************************/
12 ex_bp_addresses =
13      select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS", a."COMPANYNAME",
14             a."ADDRESSES.ADDRESSID" as "ADDRESSID", b."CITY", b."POSTALCODE", b."STREET"
15               from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.BusinessPartner" as a
16                  inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Addresses" as b
17                      on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
18                              where a."PARTNERROLE" = :im_partnerrole;
19
20 END;
```

SQLScript Metadata

- Metadata declarations: developer can set language(SQLScript/R), security(invoker/definer), default schema, and read/write access.

SAP HANA SQLScript

```
1  PROCEDURE SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.procedures::get_bp_addresses_by_role" (
2                        IN im_partnerrole NVARCHAR(3) DEFAULT '01',
3                        OUT ex_bp_addresses SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.data::EPM.Procedures.tt_bp_addresses" )
4      LANGUAGE SQLSCRIPT
5      SQL SECURITY INVOKER
6      DEFAULT SCHEMA SAP_HANA_EPM_NEXT
7      READS SQL DATA AS
8  BEGIN
9  /*****************************
10     Write your procedure logic
11 *****************************/
12 ex_bp_addresses =
13      select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS", a."COMPANYNAME",
14             a."ADDRESSES.ADDRESSID" as "ADDRESSID", b."CITY", b."POSTALCODE", b."STREET"
15               from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.BusinessPartner" as a
16                  inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Addresses" as b
17                      on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
18                               where a."PARTNERROLE" = :im_partnerrole;
19
20 END;
```

SQLScript
Script Body

- Script body – developer writes the body of the script between the BEGIN and END statements.

SAP HANA SQLScript

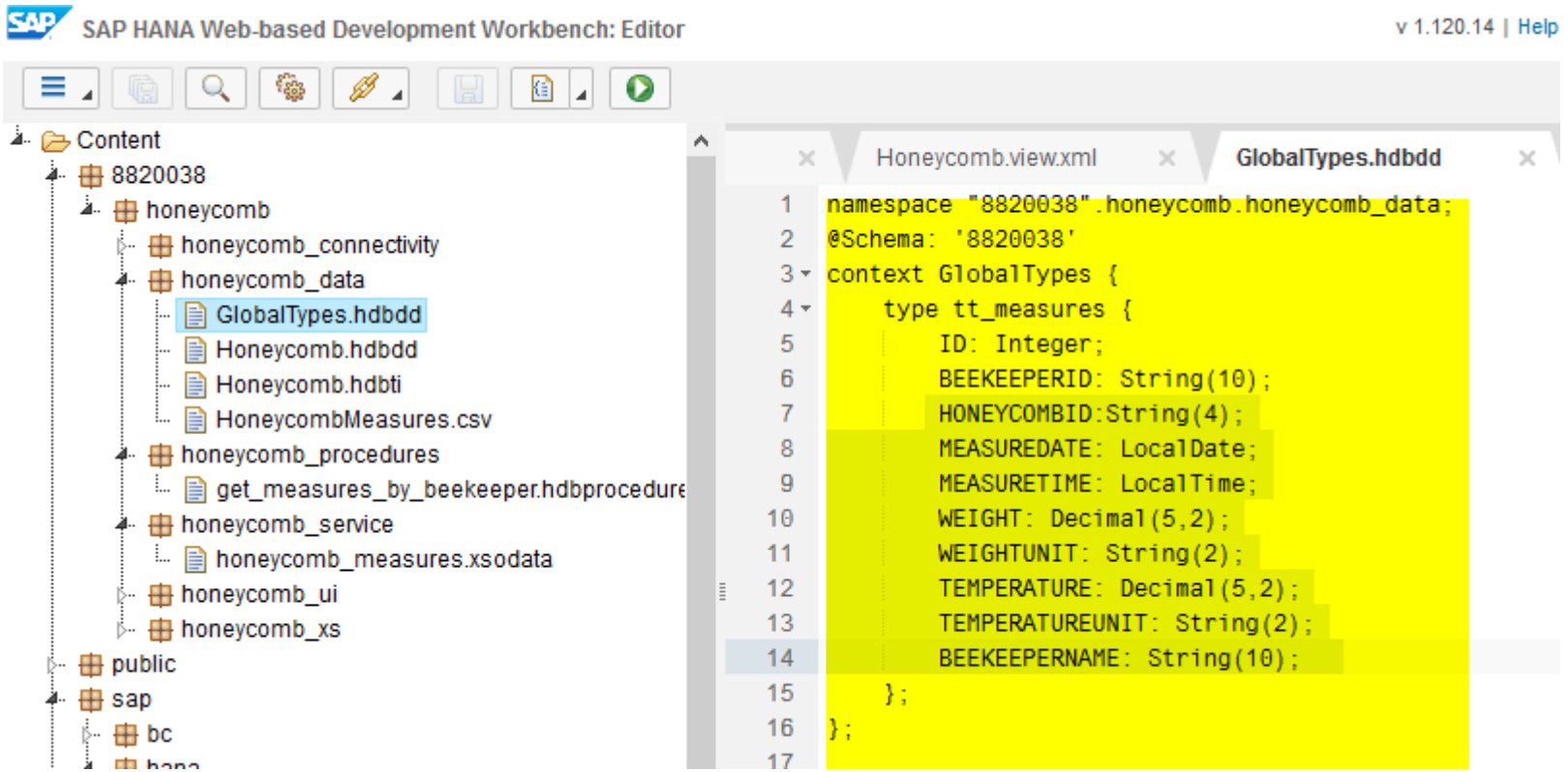
```
1  PROCEDURE SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.procedures::get_bp_addresses_by_role" (
2                        IN im_partnerrole NVARCHAR(3) DEFAULT '01',
3                        OUT ex_bp_addresses SAP_HANA_EPM_NEXT."sap.hana.democontent.epmNext.data::EPM.Procedures.tt_bp_addresses" )
4      LANGUAGE SQLSCRIPT
5      SQL SECURITY INVOKER
6      DEFAULT SCHEMA SAP_HANA_EPM_NEXT
7      READS SQL DATA AS
8  BEGIN
9  /*****************************
10     Write your procedure logic
11  *****************************/
12 ex_bp_addresses =
13      select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS", a."COMPANYNAME",
14             a."ADDRESSES.ADDRESSID" as "ADDRESSID", b."CITY", b."POSTALCODE", b."STREET"
15               from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.BusinessPartner" as a
16                  inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Addresses" as b
17                      on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
18                               where a."PARTNERROLE" = :im_partnerrole;
19
20 END;
```

Exercise SQLScript

1. In the data sub package, create a data definition file **GlobalTypes.hdbdd** defining the following type in the schema <MATNR>.

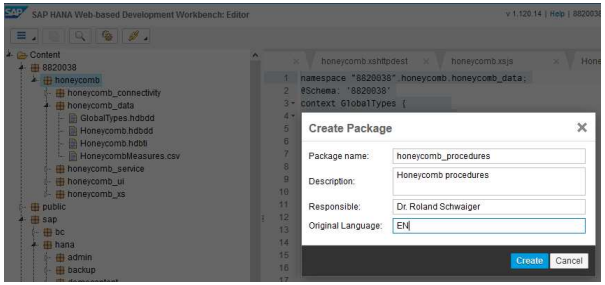
```
namespace "8820038".honeycomb.honeycomb_data;
@Schema: '8820038'
context GlobalTypes {
    type tt_measures {
        ID: Integer;
        BEEKEEPERID: String(10);
        HONEYCOMBID:String(4);
        MEASUREDATE: LocalDate;
        MEASURETIME: LocalTime;
        WEIGHT: Decimal(5,2);
        WEIGHTUNIT: String(2);
        TEMPERATURE: Decimal(5,2);
        TEMPERATUREUNIT: String(2);
        BEEKEEPERNAME: String(10);
    };
};
```

Solution SQLScript



Solution SQLScript

2. Create a folder named **honeycomb_procedures**.



3. Inside the folder, create an SQLScript procedure named **get_measures_by_beekeeper.hdbprocedure**, with the following behavior:

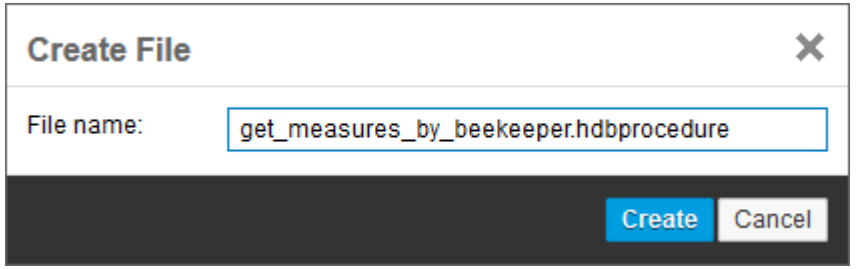
Input : variable named im_beekeeperid of type nvarchar(10)

Output: variable named ex_measures of type tt_measures (just defined)

Procedure:

```
ex_measures =
select ID, BEEKEEPERID,  HONEYCOMBID, MEASUREDATE, MEASURETIME, WEIGHT,
WEIGHTUNIT,
     TEMPERATURE, TEMPERATUREUNIT, BEEKEEPERNAME
    from "8820038"."8820038.honeycomb.honeycomb_data::Honeycomb.Measures"
    where "BEEKEEPERID" = :im_beekeeperid;
```

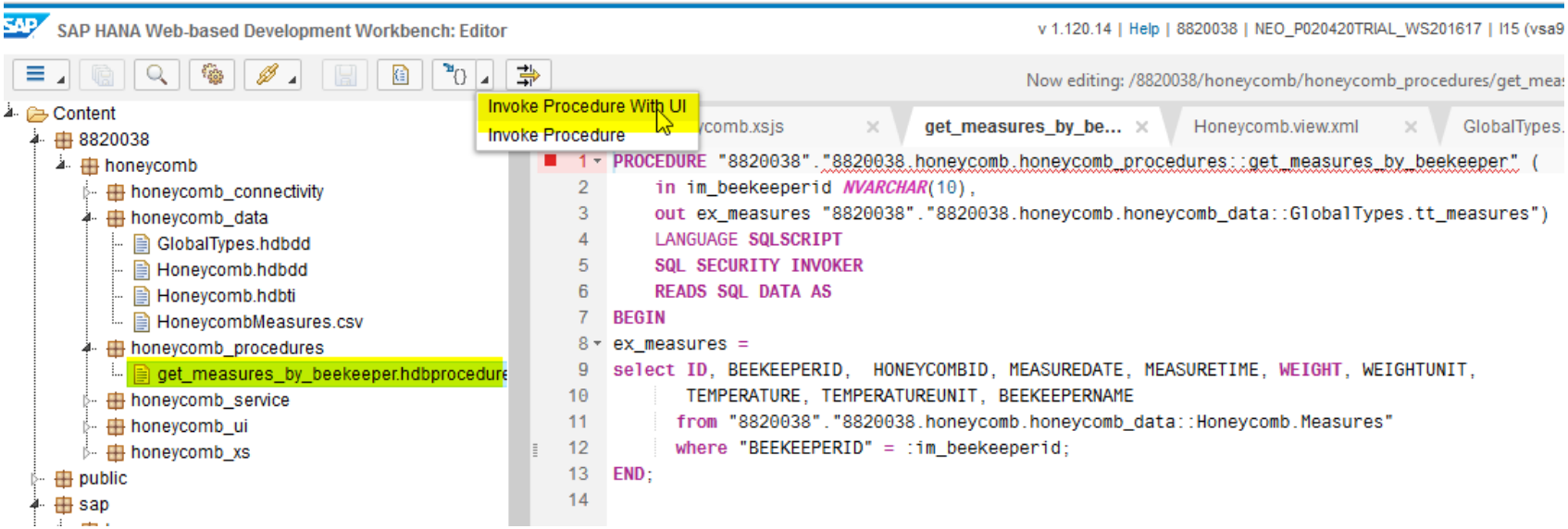
Solution SQLScript



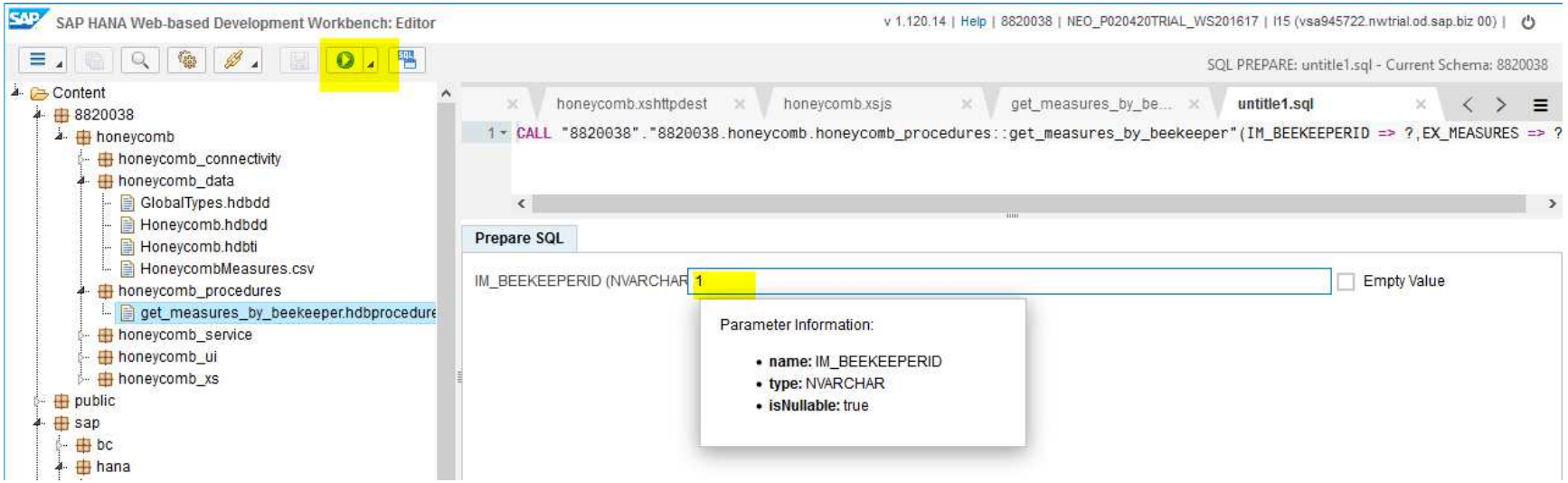

```
PROCEDURE
"8820038"."8820038.honeycomb.honeycomb_procedures::get_measures_by_beekeeper" (
    in im_beekeeperid NVARCHAR(10),
    out ex_measures
"8820038"."8820038.honeycomb.honeycomb_data::GlobalTypes.tt_measures")
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS
BEGIN
ex_measures =
select ID, BEEKEEPERID,  HONEYCOMBID, MEASUREDATE, MEASURETIME, WEIGHT,
WEIGHTUNIT,
       TEMPERATURE, TEMPERATUREUNIT, BEEKEEPERNAME
      from "8820038"."8820038.honeycomb.honeycomb_data::Honeycomb.Measures"
      where "BEEKEEPERID" = :im_beekeeperid;
END;
```

Solution SQLScript

4. Run the procedure, setting the input parameter im_beekeeperid to 1.



Solution SQLScript




```
CALL "8820038"."8820038.honeycomb.honeycomb_procedures::get_measures_by_beekeeper"(IM_BEEKEEPERID => ?,EX_MEASURES => ?)
```

Problem with missing permissions

- Try: GRANT SELECT ON SCHEMA 8820038 to _SYS_REPO WITH GRANT OPTION;
- Not working
- Created User 8820038D, provided the authorizations as described at the beginning of the course. Changed the code to schema 8820038D.
- Now
  - GRANT SELECT ON SCHEMA 8820038D to _SYS_REPO WITH GRANT OPTION; works.
  - The procedure call via UI works
  - XSODATA does not work
- Create .xaccess in 8820038D
  - Everything seems to work … WHY?

SQLScript Features
Declarative Logic

- Allows the developer to declare the data selection via SELECT statements
  - Developer defines the what
  - Engine defines the how, and executes accordingly
  - Massive parallelization

```
19 ***************************/
20 ex_bp_addresses =
21     select a."PARTNERID", a."PARTNERROLE", a."EMAILADDRESS",
22         a."COMPANYNAME",a."ADDRESSES.ADDRESSID" as "ADDRESSID",
23         b."CITY", b."POSTALCODE", b."STREET"
24           from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.BusinessPartner" as a
25             inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Addresses" as b
26               on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
27                 where a."PARTNERROLE" = :im_partnerrole;
28
29 END;
```

© SAP

SQLScript Features
Imperative Logic

- Allows developer to control the flow of the logic within SQLScript.
  - Scalar variable manipulation
  - DDL/DML logic
  - WHILE loops
  - Branching logic based on some conditions, for example IF/ELSE
  - Executed exactly as scripted, procedurally

```
17 declare lv_category varchar(40) := null;
18 declare lv_discount decimal(15,2) := 0;
19
20 lt_product = select PRODUCTID, CATEGORY, PRICE
21                 from "sap.hana.democontent.epmNext.data::MD.Products
22                     where PRODUCTID = :im_productid;
23
24 select CATEGORY into lv_category from :lt_product;
25
26 if :lv_category = 'Notebooks' then
27     lv_discount := .20;
28 elseif :lv_category = 'Handhelds' then
29     lv_discount := .25;
30 elseif :lv_category = 'Flat screens' then
31     lv_discount := .30;
32 elseif :lv_category like '%printers%' then
33     lv_discount := .30;
34 else
35     lv_discount := 0.00;  -- No discount
36 end if;
37
38 ex_product_sale_price =
39     select PRODUCTID, CATEGORY, PRICE,
40         PRICE - (PRICE * :lv_discount) as "SALEPRICE"
41             from :lt_product;
42 END;
```

© SAP

SQLScript Features
Arrays

- Allows the developer to define and construct arrays within SQLScript
  - Set elements
  - Return elements
  - Remove elements
  - Concatenate two arrays
  - Turn array into a table
  - Turn a column of a table into an array
  - Return cardinality of an array

```
11   declare productid nvarchar(10) ARRAY;
12   declare category nvarchar(40) ARRAY;
13   declare price decimal(15,2) ARRAY;
14   declare saleprice decimal(15,2) ARRAY;
15
16   productid[1] := 'ProductA';
17   productid[2] := 'ProductB';
18   productid[3] := 'ProductC';
19
20   category[1] := 'CategoryA';
21   category[2] := 'CategoryB';
22   category[3] := 'CategoryC';
23
24   price[1] := 19.99;
25   price[2] := 29.99;
26   price[3] := 39.99;
27
28   saleprice[1] := 15.99;
29   saleprice[2] := 25.99;
30   saleprice[3] := 35.99;
31
32   ex_products = unnest(:productId, :category, :price, :saleprice)
33                      as ("PRODUCTID", "CATEGORY", "PRICE", "SALEPRICE");
34
35 END;
```

SQLScript Features
Dynamic Filter

- Allows developers to apply a dynamic WHERE clause to SELECT statements by using the APPLY_FILTER statement
  - Both database tables and intermediate table variables are supported

```
6     --DEFAULT SCHEMA <default_schema_name>
7     READS SQL DATA AS
8   BEGIN
9   /***************************
10      Write your procedure logic
11    ***************************/
12  ex_products = APPLY_FILTER("SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Products",
13                             :im_filter_string) ;
14
15  END
```

get_products_by_filter... | untitle0.sql

```
1 CALL "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.procedures::get_products_by_filter"(
2         IM_FILTER_STRING => 'CATEGORY like ''Soft%''',
3         EX_PRODUCTS => ?
4     );
5
```

Result1 — 8 row(s)

	PRODUCTID	TYPECODE	CATEGORY	HISTORY.CR...
1	HT-1100	PR	Software	0000000033
2	HT-1101	PR	Software	0000000033
3	HT-1102	PR	Software	0000000033

SQLScript Features
Exception Handling 1/4

- Exception handling is a method for handling exception and completion conditions in an SQLScript procedures.
- The DECLARE EXIT HANDLER statement allows you to define exception handlers to process exception conditions in your procedures.
- You use the DECLARE CONDITION parameter to name exception conditions, and, optionally, their associated SQL state values.
- You can use SIGNAL or RESIGNAL with specified error code in user-defined error code range. A user-defined exception can be handled by the handler declared in the procedure. Also, it can be handled by the caller, which can be another procedure or client.

SQLScript Features
Exception Handling 2/4

- Declare exit handlers for generic SQL exceptions

```
23 DECLARE EXIT HANDLER FOR lcon_10001
24 BEGIN
25   ex_message :=  'Error Code: ' || ::SQL_ERROR_CODE || ' ' || ::SQL_ERROR_MESSAGE;
26 END;

28 DECLARE EXIT HANDLER FOR SQLEXCEPTION
29 BEGIN
30   ex_message := 'SQL Exception occurred!!';
31 END;

33 if :im_product = ' ' or :im_category = ' ' or :im_price = 0 then
34   SIGNAL lcon_10001 SET MESSAGE_TEXT = 'Input parameter  can not be empty';
```

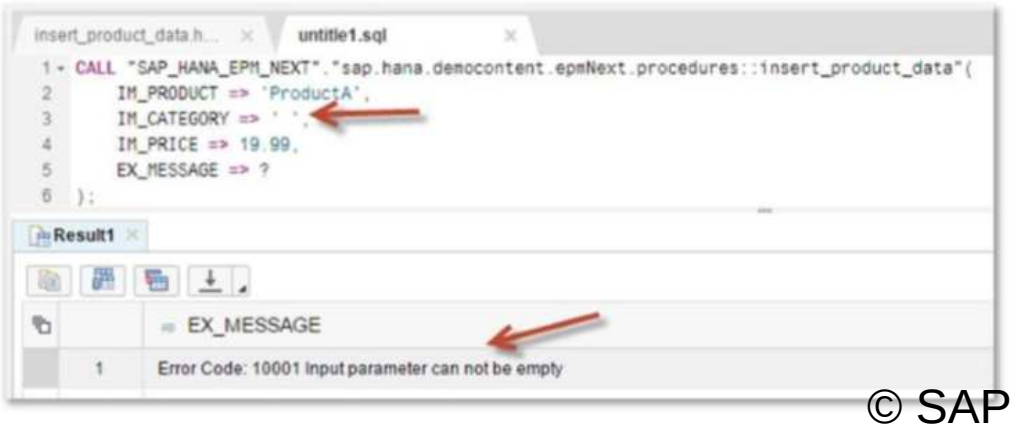
- Declare exit handlers for specific SQL error codes

```
23 DECLARE EXIT HANDLER FOR lcon_10001
24 BEGIN
25   ex_message :=  'Error Code: ' || ::SQL_ERROR_CODE || ' ' || ::SQL_ERROR_MESSAGE;
26 END;

28 DECLARE EXIT HANDLER FOR SQL_ERROR_CODE 301
29 BEGIN
30   ex_message := 'SQL Exception occurred!!';
31 END;

33 if :im_product = ' ' or :im_category = ' ' or :im_price = 0 then
34   SIGNAL lcon_10001 SET MESSAGE_TEXT = 'Input parameter can not be empty';
```

SQLScript Features
Exception Handling 3/4

- Signaling and catching user-defined conditions

```
19
20  DECLARE Icon_10001 CONDITION FOR SQL_ERROR_CODE 10001;
21  DECLARE EXIT HANDLER FOR Icon_10001    -- RESIGNAL
22  BEGIN
23    ex_message :=  'Error Code: ' || ::SQL_ERROR_CODE || ' ' || ::SQL_ERROR_MESSAGE;
24  END;
25
26  if :im_product = ' ' or
27     :im_category = ' ' or
28     :im_price = 0 THEN
29    SIGNAL Icon_10001 SET MESSAGE_TEXT = 'Input parameter can not be empty';
30  end if;
31
```



© SAP

SQLScript Features
Exception Handling 4/4

- RESIGNAL user defined exceptions to the caller
- RESIGNAL can also be executed explicitly within the body of the exit handler

```
20
21 DECLARE Icon_10001 CONDITION FOR SQL_ERROR_CODE 10001;
22 DECLARE EXIT HANDLER FOR Icon_10001 RESIGNAL;
23 --BEGIN
24 --  ex_message :=  'Error Code: ' || ::SQL_ERROR_CODE || ' ' || ::SQL_ERROR_MESSAGE
25 --END;
26
27 if :im_product = ' ' or
28    :im_category = ' ' or
29    :im_price = 0 THEN
30   SIGNAL Icon_10001 SET MESSAGE_TEXT = 'Input parameter can not be empty';
31 end if;
32
33
```

insert_product_data.h... | untitle1.sql

```
1 CALL "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.procedures::insert_product_data"(
2     IM_PRODUCT => 'ProductA',
3     IM_CATEGORY => ' ',
4     IM_PRICE => 19.99,
5     EX_MESSAGE => ?
6 );
7
```

[13:30:47] (SQL Editor) Could not execute 'CALL "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.procedures::insert_product_data"(...'
Error: (dberror) 10001 - user-defined error: "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.procedures::insert_product_data": line 22

SQLScript
Autonomous Transaction

- Allows developer to create an isolated block of code which runs as an independent transaction.
- BEGIN AUTONOMOUS TRANSACTION … END statement block.
- Committed statements inside autonomous transaction block will be persistent regardless of a rollback of the main transaction.
- For tables updated within the main procedure body, access to those tables is not allowed in the autonomous transaction block.
- Used commonly for logging tasks.

```
31  DECLARE EXIT HANDLER FOR SQLEXCEPTION   -- SQL_ERROR_CODE 301
32  BEGIN
33   ex_message := 'SQL Exception occurred!! ' || ::SQL_ERROR_CODE || ' ' || ::SQL_ERROR_MESSAGE;
34   BEGIN AUTONOMOUS TRANSACTION
35    INSERT INTO "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
36            VALUES(:im_product,
37                   (select IFNULL(MAX(logid), 0) + 1
38                      from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog
39                       where productid = :im_product),
40                   now(), CURRENT_USER, ex_message );
41
42    COMMIT;
43    RESIGNAL;
44   END;
45  END;
```

© SAP

SQLScript Cursors

- Allows developers to iterate over a result set and perform row-based processing and calculations.

```
12  ********************/
13  declare v_new_price decimal(15,2);
14  declare CURSOR c_products FOR
15        SELECT PRODUCTID, CATEGORY, PRICE
16          from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Products";
17
18  FOR cur_row as c_products DO
19
20   if :im_direction = 'INCREASE' then
21     v_new_price := cur_row.PRICE + (cur_row.PRICE * :im_rate);
22   elseif :im_direction = 'DECREASE' then
23     v_new_price := cur_row.PRICE - (cur_row.PRICE * :im_rate);
24   end if;
25   UPDATE "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Products"
26        SET PRICE = V_new_price where PRODUCTID = cur_row.PRODUCTID;
27
28  END FOR;
29
30  ex_products = select * from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Products";
31
32  END;
```

SQLScript Commit/Rollback 1/2

User defined functions

- Supported in procedures only, not supported for scalar or table UDFs.
- COMMIT command commits the current transaction and all changes before the COMMIT command.
- ROLLBACK command rolls back the current transaction and undoes all changes since the last COMMIT.
- Transaction boundary is not tied to the procedure block, so, if there are nested procedures that contain COMMIT/ROLLBACK, then all statements in the top-level procedure are affected.
- If dynamic SQL was used in the past to execute COMMIT and ROLLBACK statements, we recommend that you replace all occurrences with the native command, because they are more secure.

SQLScript Commit/Rollback 2/2

- The first and third INSERT statements affect the productLog table, but not the second, since it was rolled back.
- With the first COMMIT, the first transaction is written to persistence, and a new transaction is started.
- By triggering ROLLBACK, all changes in the second transaction are reverted, and a new transaction is started.

```
10  ****************************/
11  declare lv_message varchar(100);
12
13  lv_message := 'First log entry for Product ' || :im_product || ' inserted successfully';
14  INSERT INTO "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
15          VALUES(:im_product,
16                 (select IFNULL(MAX(logid), 0) + 1
17                     from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
18                      where productid = :im_product),
19                  now(), CURRENT_USER, lv_message );
20  COMMIT;
21
22  lv_message := 'Second log entry for Product ' || :im_product || ' inserted successfully';
23  INSERT INTO "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
24          VALUES(:im_product,
25                 (select IFNULL(MAX(logid), 0) + 1
26                     from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
27                      where productid = :im_product),
28                  now(), CURRENT_USER, lv_message );
29  ROLLBACK;
30
31  lv_message := 'Third log entry for Product ' || :im_product || ' inserted successfully';
32  INSERT INTO "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
33          VALUES(:im_product,
34                 (select IFNULL(MAX(logid), 0) + 1
35                     from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.productLog"
36                      where productid = :im_product),
37                  now(), CURRENT_USER, lv_message );
38  COMMIT;
39
```

SQLScript Functions

- Language used within the body is SQLScript, other languages are not supported.
- Structure is similar to procedures.
- Functions are read-only, side effect free.
- Two types, both have a wizard and associated editor:
  - Table User Defined Function(.hdbtablefunction artifact)
  - Scalar User Defined Function(.hdbscalarfunction artifact)

apply_discount.hdbs...

```
FUNCTION "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.functions::apply_discount" (
                im_price decimal(15,2),
                im_discount decimal(15,2))
    RETURNS ex_result decimal(15,2)
  LANGUAGE SQLSCRIPT
  SQL SECURITY INVOKER AS
BEGIN
/*****************************
    Write your function logic
*****************************/
  ex_result := :im_price - ( :im_price * :im_discount );
END;
```

get_employees_by_n...

```
FUNCTION "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.functions::get_employees_by_name_filter"
  (im_lastNameFilter nvarchar(40))
  RETURNS table ( EMPLOYEEID NVARCHAR(10),
                  "NAME.FIRST" NVARCHAR(40),
                  "NAME.LAST" NVARCHAR(40),
                  EMAILADDRESS NVARCHAR(255),
                  ADDRESSID NVARCHAR(10), CITY NVARCHAR(40),
                  POSTALCODE NVARCHAR(10), STREET NVARCHAR(60))
  LANGUAGE SQLSCRIPT
  SQL SECURITY INVOKER AS
BEGIN
/*****************************
    Write your function logic
*****************************/

 RETURN
    select a.EMPLOYEEID, a."NAME.FIRST",
        a."NAME.LAST", a.EMAILADDRESS,
        a."ADDRESSES.ADDRESSID" as ADDRESSID, b.CITY, b.POSTALCODE, b.STREET
          from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Employees"  as a
            inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Addresses"  as b
              on a."ADDRESSES.ADDRESSID" = b.ADDRESSID
                where contains("NAME.LAST", :im_lastNameFilter, FUZZY(0.9));

END;
```

SQLScript Functions
Table User defined functions (UDFs)

- Can have any number of input parameters.
- Returns exactly one table.
- Table operations are allowed within the body.
- Consumed in the FROM clause of a SELECT statement.

```
get_employees_by_n...
1  FUNCTION "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.functions::get_employees_by_name_filter"
2     (im_lastNameFilter nvarchar(40))
3     RETURNS table ( EMPLOYEEID NVARCHAR(10),
4                     "NAME.FIRST" NVARCHAR(40),
5                     "NAME.LAST" NVARCHAR(40),
6                     EMAILADDRESS NVARCHAR(255),
7                     ADDRESSID NVARCHAR(10), CITY NVARCHAR(40),
8                     POSTALCODE NVARCHAR(10), STREET NVARCHAR(60))
9     LANGUAGE SQLSCRIPT
10    SQL SECURITY INVOKER AS
11 BEGIN
12 /*****************************
13    Write your function logic
14 *****************************/
15
16  RETURN
17    select a.EMPLOYEEID, a."NAME.FIRST",
18         a."NAME.LAST", a.EMAILADDRESS,
19         a."ADDRESSES.ADDRESSID" as ADDRESSID, b.CITY, b.POSTALCODE, b.STREET
20           from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Employees"  as a
21              inner join "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::MD.Addresses"  as b
22                   on a."ADDRESSES.ADDRESSID" = b.ADDRESSID
23                        where contains("NAME.LAST", :im_lastNameFilter, FUZZY(0.9));
24
25 END;
```

```
get_employees_by...   untitle2.sql
1  select * from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.functions::get_employees_by_name_filter"( '*ll*' );

Result1                                                                          4 row(s)
```

	EMPLOYEEID	NAME.FIRST	NAME.LAST	EMAILADDRESS	ADDRESSID	
1	0000000003	Franco	Fall	franco.fall@itelo.info	1000000004	Sa
2	0000000010	Michael	Miller	michael.miller@itelo.info	1000000011	Lo
3	0000000018	Al	Wallace	al.wallace@itelo.info	1000000019	Sa
4	0000000027	Barbara	Ingalls	barbara.ingalls@itelo.in	1000000028	Sa

SQLScript Functions
Scalar UDFs

- Can have any number of input parameters.
- Can return multiple values.
- Input parameters cannot be table type.
- Consumed from the field list or the WHERE clause of the SELECT statement.
- Also callable via direct assignment(x := my_scalar_func()), multiple parameter assignments also supported.

apply_discount.hdbs...

```
1  FUNCTION "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.functions::apply_discount" (
2                  im_price decimal(15,2),
3                  im_discount decimal(15,2))
4      RETURNS ex_result decimal(15,2)
5    LANGUAGE SQLSCRIPT
6    SQL SECURITY INVOKER AS
7  BEGIN
8  /*****************************
9      Write your function logic
10  *****************************/
11   ex_result := :im_price - ( :im_price * :im_discount );
12 END;
```

apply_discount.hdbsc... | untitle3.sql

```
1 select PRODUCTID, CATEGORY, PRICE,
2   "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.functions::apply_discount"(PRICE, 0.33 )
3       as "SALEPRICE" from "sap.hana.democontent.epmNext.data::MD.Products";
4
```

Result1 — 106 row(s)

	PRODUCTID	CATEGORY	PRICE	SALEP
1	HT-1000	Notebooks	956	640.52
2	HT-1001	Notebooks	1249	836.83
3	HT-1002	Notebooks	1570	1051.9
4	HT-1003	Notebooks	1650	

SQLScript Triggers

- Special type of stored procedure that automatically execute when an event occurs in the database server.
- Triggers can be executed BEFORE or AFTER an event on a given table, such as INSERT, UPDATE, or DELETE.
- Management of objects can only be done via SQL Console. Support for storing the artifacts in the repository is coming in a future support package.

facet.at
Plan-Do-Check-Act

SERVER SIDE JAVASCRIPT

- Explain basic concepts of XSJS
- Explain how requests to XSJS programs are processed
- Access SAP HANA Database using XSJS
- Use XSJS Debugger
- Server Side JavaScript changes in XS Advanced
- Access server side XSJS scripts from SAPUI5 applications

http://help.sap.com/hana/SAP_HANA_XS_JavaScript_API_Reference_en/

XSJS

Server Side JavaScript (XSJS)

Lightweight procedural logic

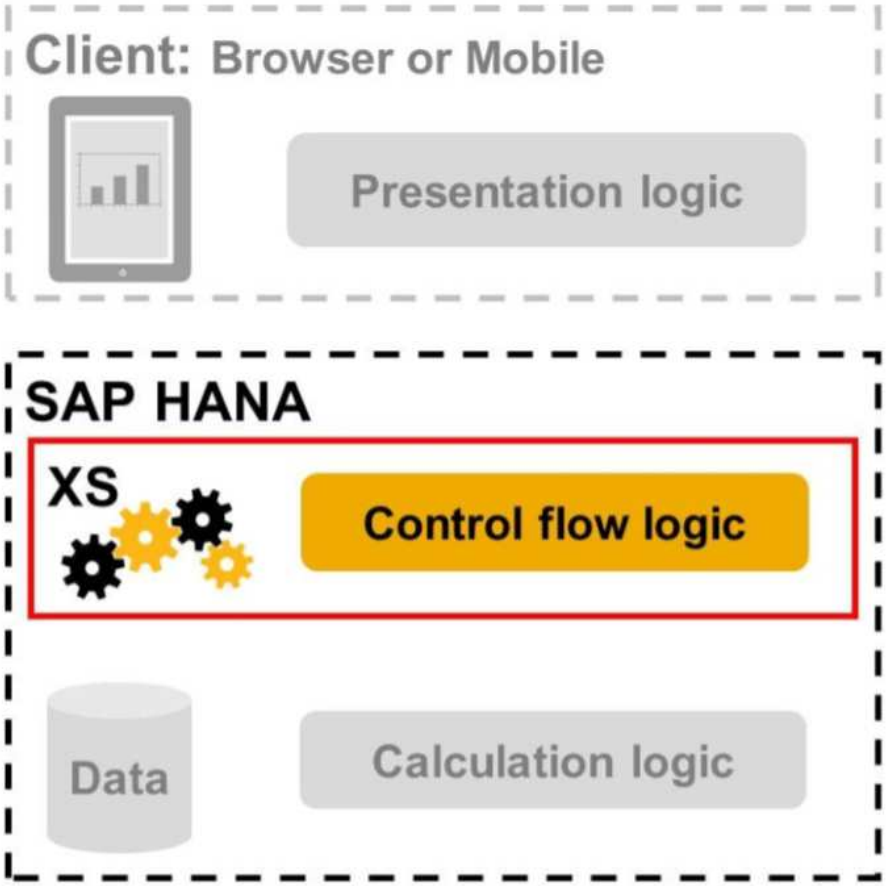
Reusable

One language:

- Client
- Server side

Widespread knowledge

Rapid development



© SAP

XSJS

Native application-specific code

Control data flow logic

Store XS JavaScript code in the repository

JavaScript editor/debugger

*retrieveData.xsjs

```
$.response.contentType = "text/html";
var output = "Hello, World !<br><br>";

var conn = $.db.getConnection();
var pstmt = conn.prepareStatement( "select * from DUMMY" );
var rs = pstmt.executeQuery();

if (!rs.next()) {
    $.response.setBody( "Failed to retrieve data" );
    $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
    }
else {
    output = output + "This is the response from my SQL: "
    + rs.getString(1);
    }
rs.close();
pstmt.close();
conn.close();

$.response.setBody(output);
```

XSJS

Interact with the SAP HANA XS runtime environment

Access SAP HANA database capabilities directly

Expose, update, insert, and delete data

Request API

- Control HTTP request/response

Database API

- Open DB connection, roll back changes in SAP HANA

XSJS

Reuse program elements

Reuse 3rd party or open source code

Perform repetitive tasks

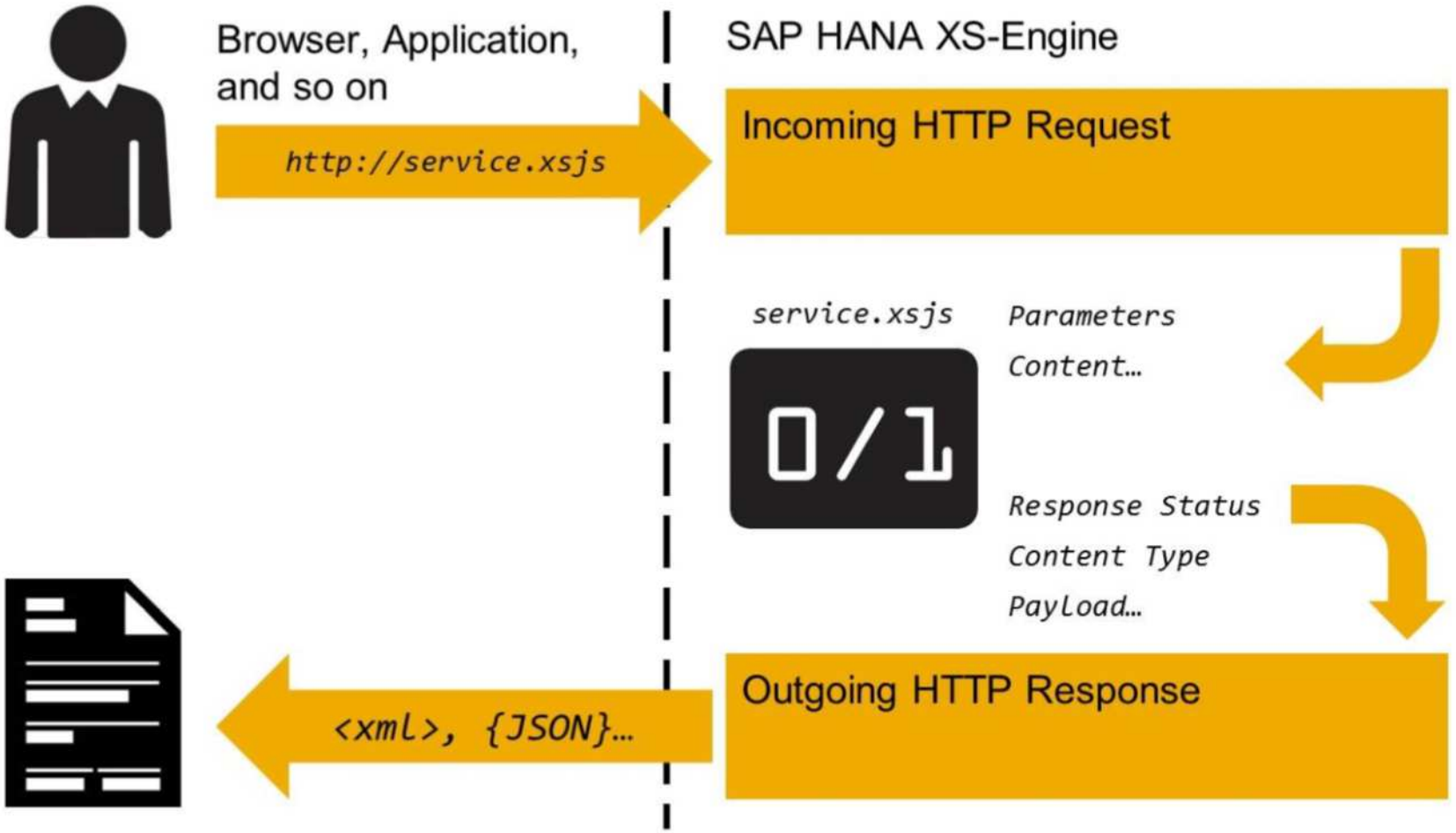
- Handle forms
- Manage date/time strings
- Parse URLs

Import into other XSJS programs

Call from other XSJS programs

```
// import math lib
$.import("sap.myapp.lib","math");

// use math lib
var max_res = $.sap.myapp.lib.math.max(3, 7);
```

XSJS
Request Processing



XSJS
Request processing API

Namespace: web

Access via: `$.web`

Important Classes

- **`WebRequest`**
  - **Current Request: `$.request`**
- **`WebResponse`**
  - **Current Response: `$.response`**

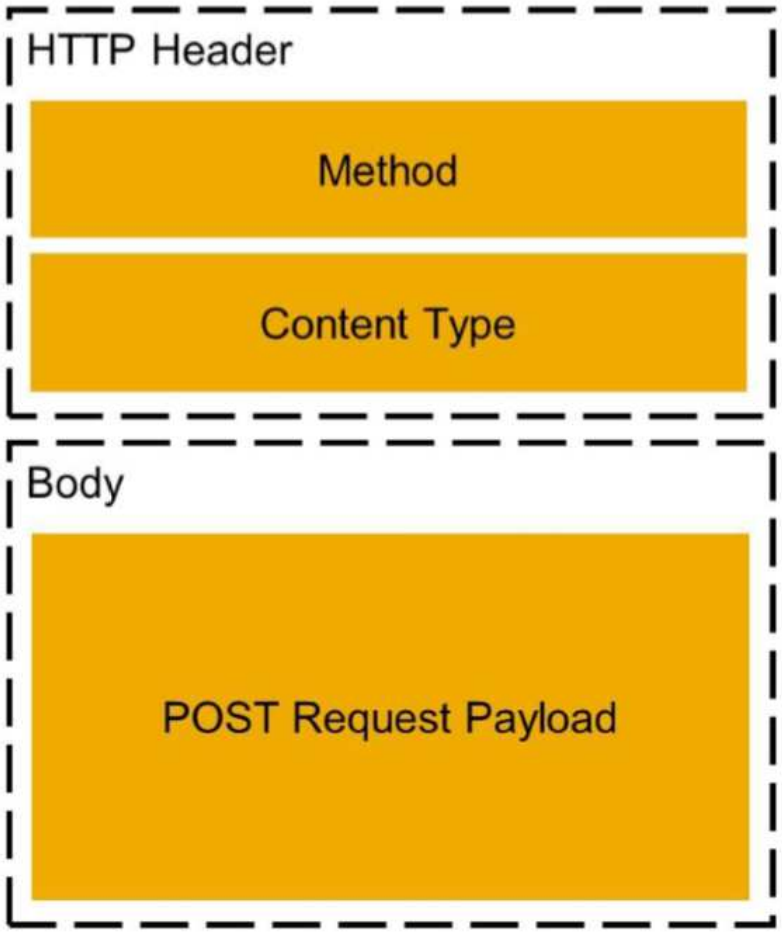
XSJS WebRequest

Represents the client HTTP request currently being processed.

Access via: `$.request`

Important members:

- *method*
- *contentType*
- *body*
- *parameters*



XSJS WebResponse

Represents the HTTP response currently being populated.

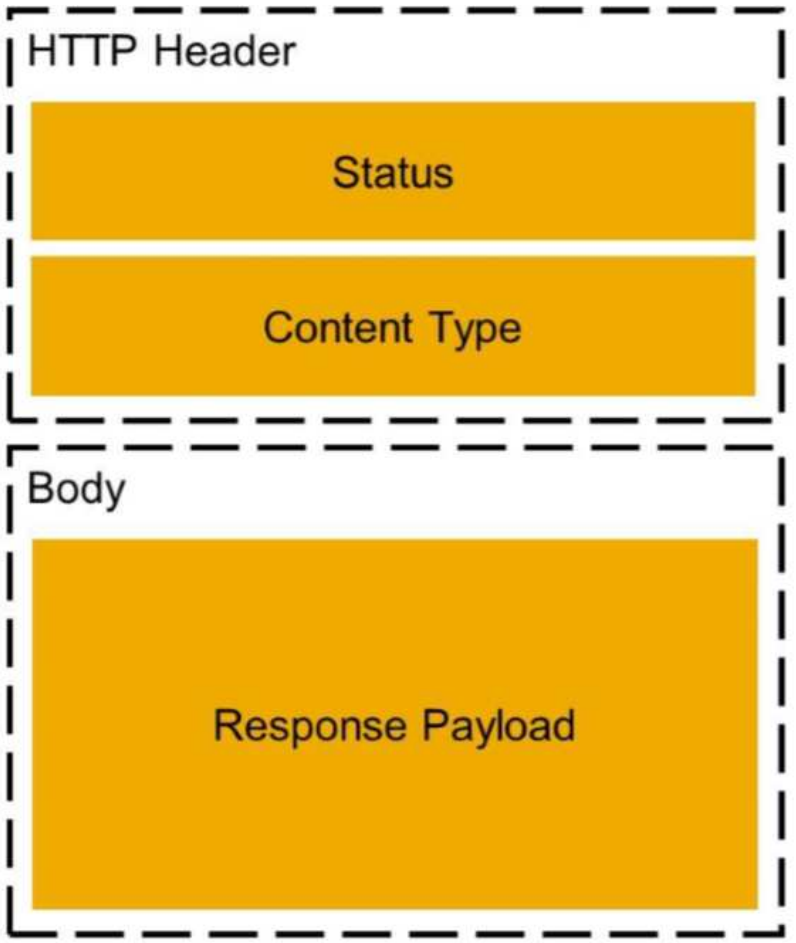
Access via: `$.response`

Important members:

- `contentType`
- `Status`

Important method:

- `setBody()`



XSJS
HTTP Status Codes

Represents the HTTP constants for status codes, inserted into the response.

Access via: *$.net.http*

Important properties:

- *OK*
- *BAD_REQUEST*
- *INTERNAL_SERVER_ERROR*
- *NOT_YET_IMPLEMENTED*

<static, readonly> **http**

HTTP constants for status codes

Properties:

Name	Type	Default	Description
CONTINUE	number	100	
SWITCH_PROTOCOL	number	101	
OK	number	200	
CREATED	number	201	
ACCEPTED	number	202	
NON_AUTHORITATIVE	number	203	
NO_CONTENT	number	204	
RESET_CONTENT	number	205	
PARTIAL_CONTENT	number	206	
MULTIPLE_CHOICES	number	300	
MOVED_PERMANENTLY	number	301	
FOUND	number	302	
SEE_OTHER	number	303	
NOT_MODIFIED	number	304	
USE_PROXY	number	305	
TEMPORARY_REDIRECT	number	307	
BAD_REQUEST	number	400	
UNAUTHORIZED	number	401	
PAYMENT_REQUIRED	number	402	
FORBIDDEN	number	403	
NOT_FOUND	number	404	

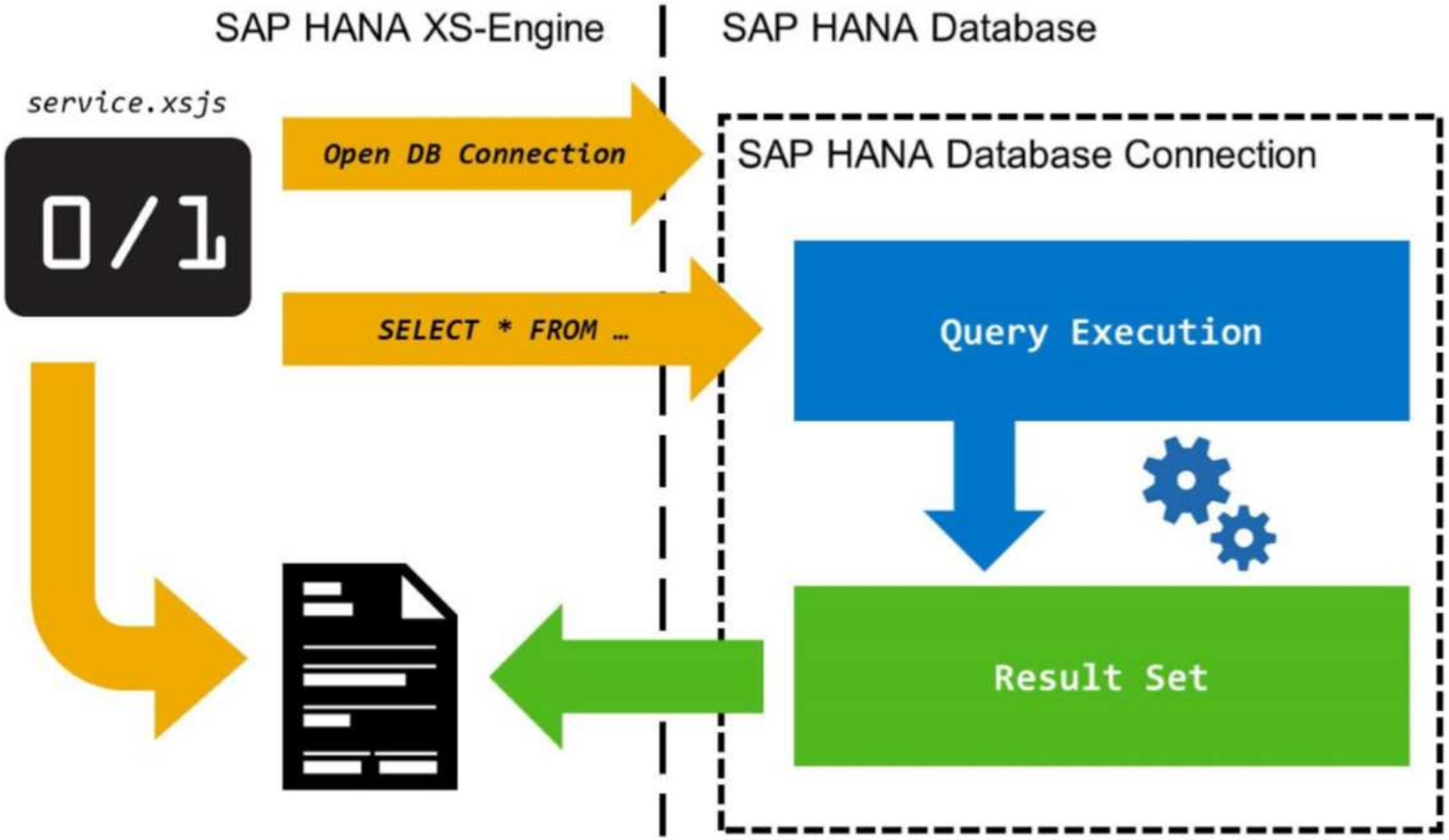
Exercise
XSJS

- Write a request handler which sends „Hello World“

Solution
XSJS

```
function getUsername(){
   var username =  $.session.getUsername();
   return username;
}
var result = "Hello World from User " +
getUsername();
$.response.setBody(result);
```

XSJS
Accessing the HANA DB



XSJS
DB API

This namespace provides means for seamless SAP HANA database access.

Namespace: hdb

Access via: `$.hdb`

Important Classes

- `Connection`
- `ResultSet`
- `ProcedureResult`

Important Methods

- `getConnection()`

XSJS
Establish connection

Represents the SAP HANA database connection.

Access via: `$.hdb.getConnection()`

Important Methods

- `executeQuery(query [,arguments])`
- `executeUpdate(statement [,arguments])`
- `loadProcedure(schema, procedure)`
- `commit()`
- `rollback()`
- `close()`

XSJS
Execute Statement

SQL Queries to the database are executed on the connection object:

```
connection.executeUpdate(query, arguments);
```

```
var oConnection= $.hdb.getConnection();

var sQuery =  'SELECT * FROM "workshop.exercises.g<group>.data::User.Details"  ' +
              'LIMIT ?';
var oResultSet = oConnection.executeQuery(sQuery, vLimit);
```

The result set can be accessed like any JavaScript via the *ResultSet* object.

```
for(var i = 0; i < oResultSet.length; i++){

    sBody +=    oResultSet[i].PERS_NO +      "," +
                oResultSet[i].FIRSTNAME +    "," +
                oResultSet[i].LASTNAME +     "," +
                oResultSet[i].E_MAIL +       "\n";
}
```

It can also be used easily by a JavaScript frontend by sending it as a JSON string, using the method JSON.stringify().

XSJS
Call a procedure

The Database API provides mechanisms to call a database procedure in an XSJS program like a JavaScript procedure.

The database procedure is loaded via the connection object:

```
connection.loadProcedure(schema, procedure);
```

```
var oConnection = $.hdb.getConnection();

var fnCreateUser = oConnection.loadProcedure("HANA_WORKSHOP_SOL",
                          "workshop.exercises.g<group>.procedures::createUser");
```

Subsequently, the procedure is called using its JavaScript representation. The function has the same parameters as the database procedure.

```
var oResult = fnIncreasePrice(1.5);
```

Exercise
XSJS

- Accessing the database
- downLoadMeasures.xsjs

Solution XSJS

```
function downLoadMeasures() {
		var vLimit = $.request.parameters.get("limit");
		if (isNaN(parseInt(vLimit, 10)) || vLimit < 0) {
			$.response.status = $.net.http.BAD_REQUEST;
			$.response.contentType = "text/json";
			var oResponse = {
				message: "The Parameter 'limit' must be set!"
			};
			$.response.setBody(JSON.stringify(oResponse));
			return;
		}
		var oConnection = $.hdb.getConnection();
		var sQuery = "SELECT * FROM \"8820038D.honeycomb.honeycomb_data::honeycomb8820038.Measures\"" + " LIMIT ?";
		var oResultSet = oConnection.executeQuery(sQuery, vLimit);
		var sBody = "";
		for (var i = 0; i < oResultSet.length; i++) {
			sBody += oResultSet[i].ID + "," + oResultSet[i].HONEYCOMBID + "," + oResultSet[i].MEASUREDATE + "," +
oResultSet[i].WEIGHT + "\n";
		}
		$.response.status = $.net.http.OK;
		$.response.contentType = "application/csv";
		$.response.headers.set("Content-Disposition", "attachment;filename=list.csv");
		$.response.setBody(sBody);
		oConnection.close();
}
downLoadMeasures();
```

Exercise XSJS

- Call procedure
- createMeasure.xsjs

Solution XSJS

- Create a procedure **create_measure.hdbprocedure**

```
PROCEDURE "8820038D"."8820038D.honeycomb.honeycomb_procedures::create_measure" (
    in im_measures "8820038D"."8820038D.honeycomb.honeycomb_data::GlobalTypes.tt_measures" )
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS
BEGIN
 // here goes your code
END;
```

Solution XSJS

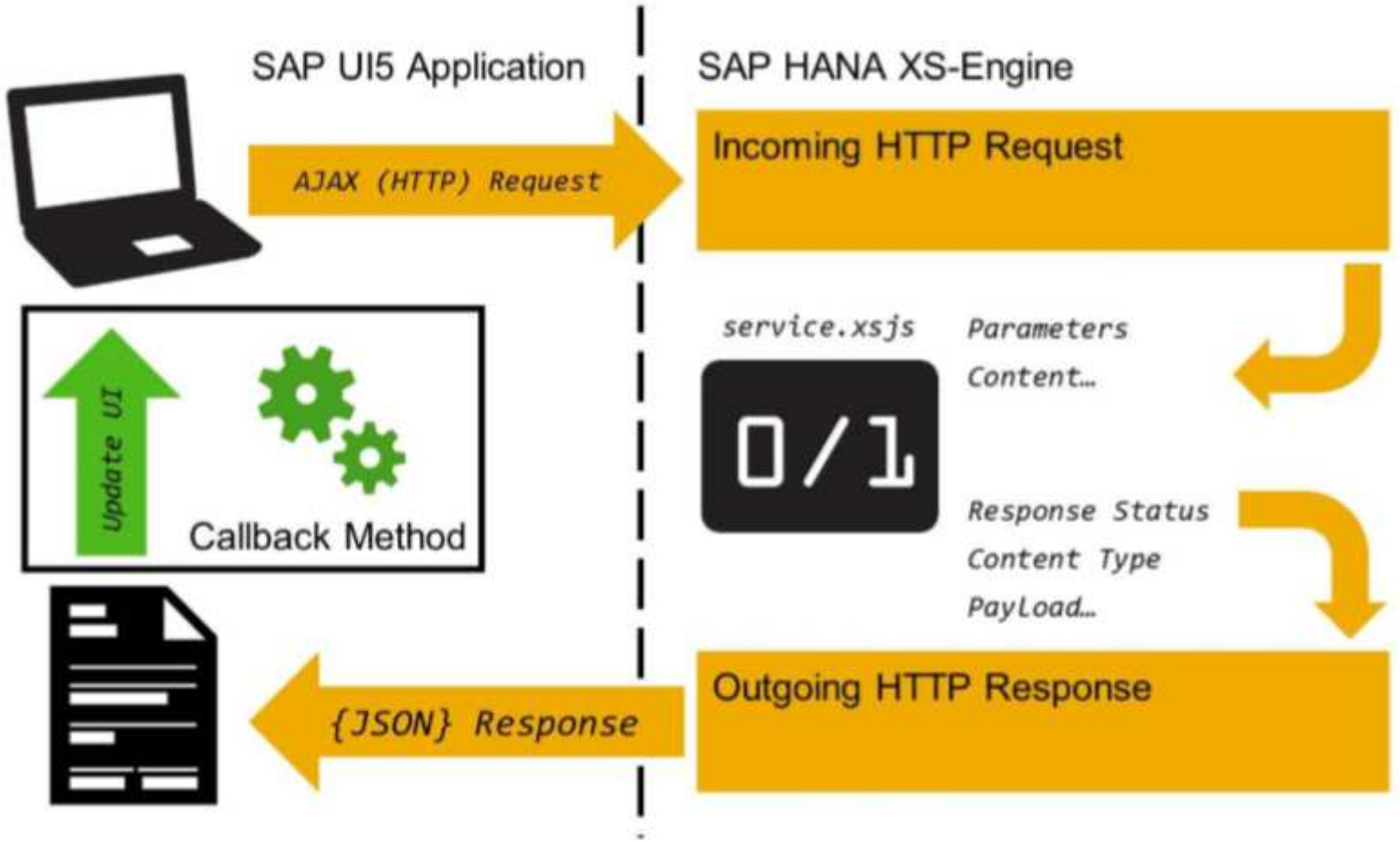
- Create a XSJS service **createMeasure.xsjs**

```
function createMeasure() {
		$.response.contentType = "text/json";
		var oConnection = $.hdb.getConnection();
		var fnCreateMeasure = oConnection.loadProcedure("8820038D", "8820038D.honeycomb.honeycomb_procedures::create_measure");
		var oNewMeasure = [{
				ID: "00000",
				HONEYCOMBID: $.request.parameters.get("honeycombID"),
				MEASUREDATE: $.request.parameters.get("measureDate"),
				WEIGHT: $.request.parameters.get("measureWeight")
		}];
		var oResult;
		try {
				oResult = fnCreateMeasure(oNewMeasure);
		} catch (oException) {
				$.response.status = $.net.http.INTERNAL_SERVER_ERROR;
				$.response.setBody(JSON.stringify({
						message: oException.message
				}));
				return;
		}
		if (oResult.ERROR[0] !== undefined) {
				$.response.status = parseInt(oResult.ERROR[0].HTTP_STATUS_CODE, 10);
				$.response.setBody(JSON.stringify({
						message: oResult.ERROR[0].ERROR_MESSAGE
				}));
		} else {
				$.response.status = $.net.http.OK;
				$.response.setBody(JSON.stringify({
						message: "The procedure call was successful..."
				}));
		}
		oConnection.commit();
		oConnection.close();
}
createMeasure();
```

Solution XSJS

Test the URL, which will look like the following one:
https://<domain>:<port>/<path>/createMeasure.xsjs?id=1&measuredate=2017-02-03&weight=34

XSJS - UI5


SAP UI5 Application
SAP HANA XS-Engine
AJAX (HTTP) Request
Incoming HTTP Request
service.xsjs
Parameters
Content…
0/1
Update UI
Callback Method
Response Status
Content Type
Payload…
{JSON} Response
Outgoing HTTP Response

XSJS – UI5
jQuery

Namespace: jQuery

Access via: *jQuery.**ajax(settings)***

Important settings

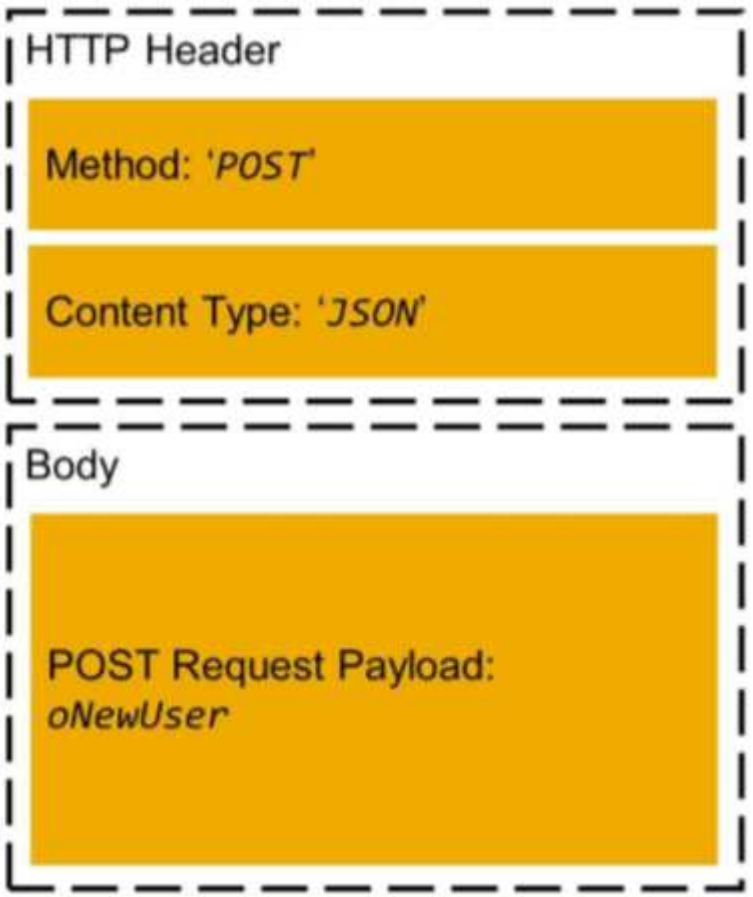
- ***url***
- ***method***
  GET, POST, PUT
- ***data***
  Request payload
- ***dataType***
  xml, html, script, json, text
- ***context***
  Callback method context
- ***success***
  Callback function if success
- ***error***
  Callback function if error

http://api.jquery.com/jquery.ajax/

XSJS – UI5
AJAX request

A typical AJAX request, which creates a POST HTTP Request to the service CreateUser.xsjs, sending the JSON object oNewUser.

```
jQuery.ajax({

    url: "../../services/CreateUser.xsjs",
    method: 'POST',
    data: oNewUser,
    dataType: 'JSON',
    context: this,

    success: function(oResponse, sStatus, oXHR){
        // Process the server response
    },
    error: function(oXHR, sStatus, sError){
        // Signal an error
    }
});
```



XSJS – UI5
AJAX response

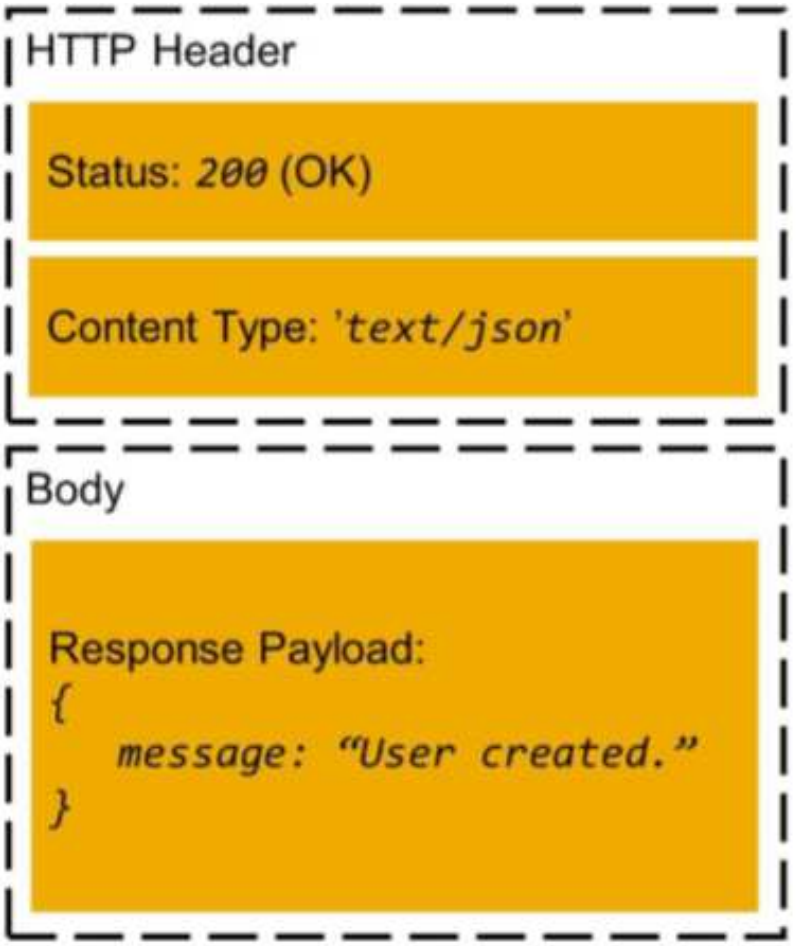
As the AJAX request is done asynchronously, the servers response is handled with the callback methods.

Depending on the response status (you set them, for example, in XSJS programs), the success or the error function is executed.

- success: status = 2xx / 304
- error: everything else

The response content type should correspond to the data type you specified to expect in the call. E.g. JSON objects are parsed by jQuery and can be accessed directly in the callback method.

The callback methods are executed with an own context. If you need access to variables of another context you need to set the context setting of the AJAX call accordingly.



CONNECTIVITY 4 HANA XS

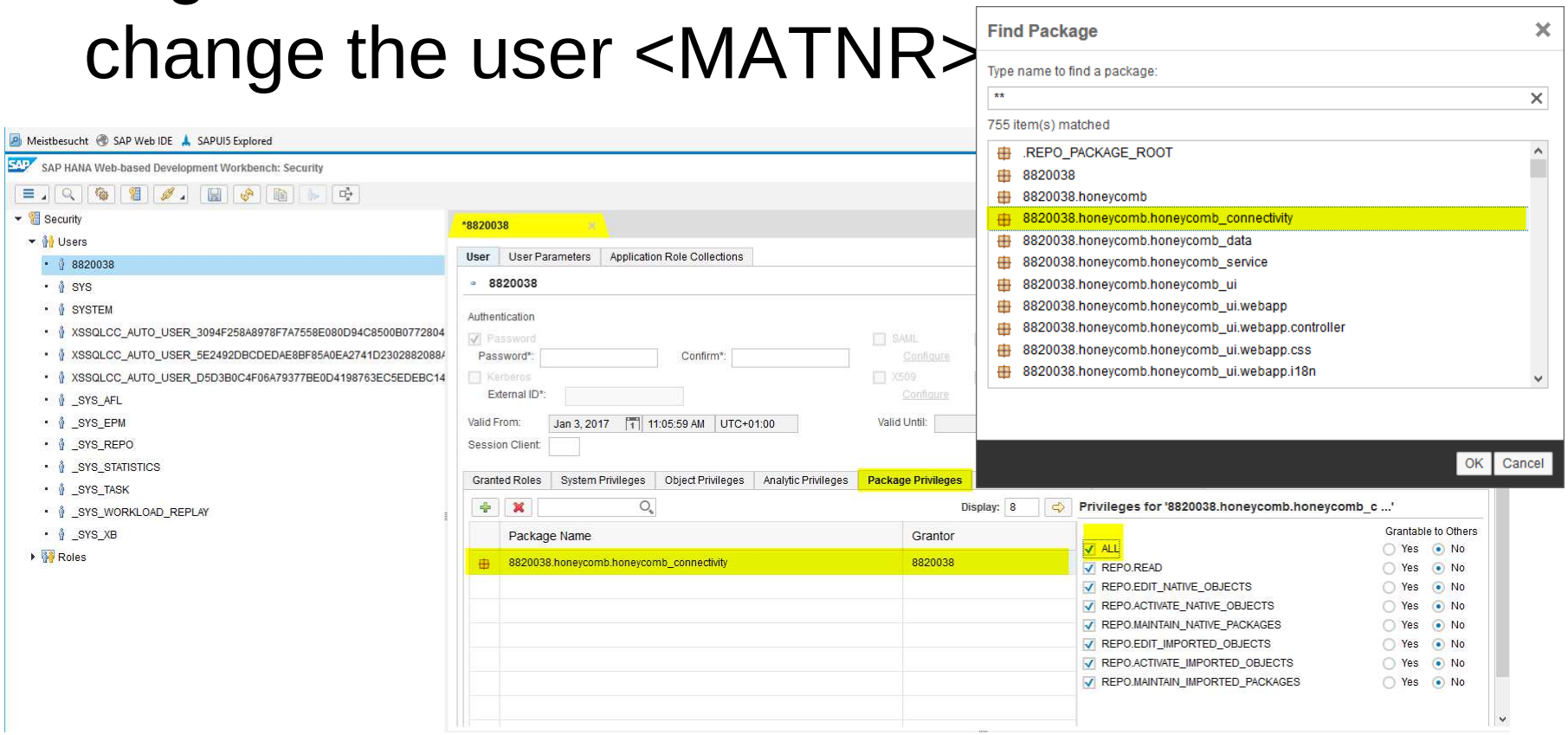
- Make XS destinations for consuming HTTP Internet services
- Use existing services (Google and Honeycomb.Measures)

Exercise Connectivity

- Create package honeycomb_connectivity

Connectivity
Grant access to package 4 user

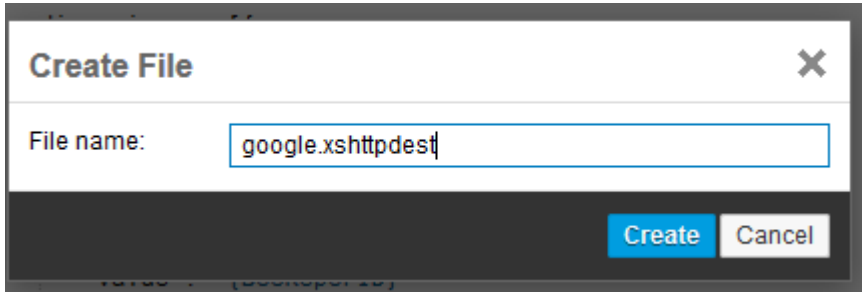
- Logon as user SYSTEM in order to change the user <MATNR>



Connectivity
XS Destination File

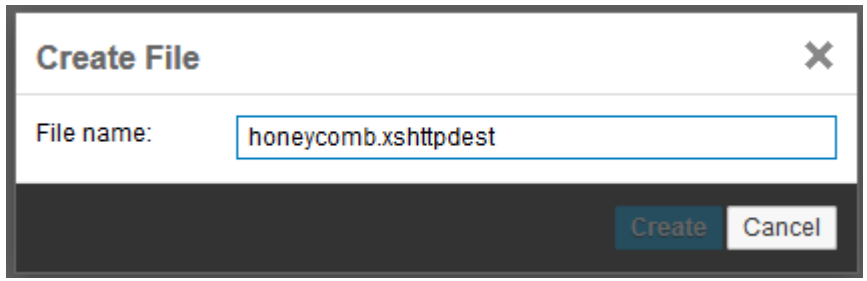
Google

```
host = "maps.googleapis.com";
port = 80;
pathPrefix = "/maps/api/distancematrix/json";
useProxy = true;
proxyHost = "proxy-trial";
proxyPort = 8080;
authType = none;
useSSL = false;
timeout = 30000;
```



Honeycomb

```
host = "www.niun.org";
port = 80;
pathPrefix = "/beelog";
useProxy = false;
authType = none;
useSSL = false;
timeout = 30000;
```



Connectivity
XSJS File

honeycomb_connectivity/honeycomb.xsjs

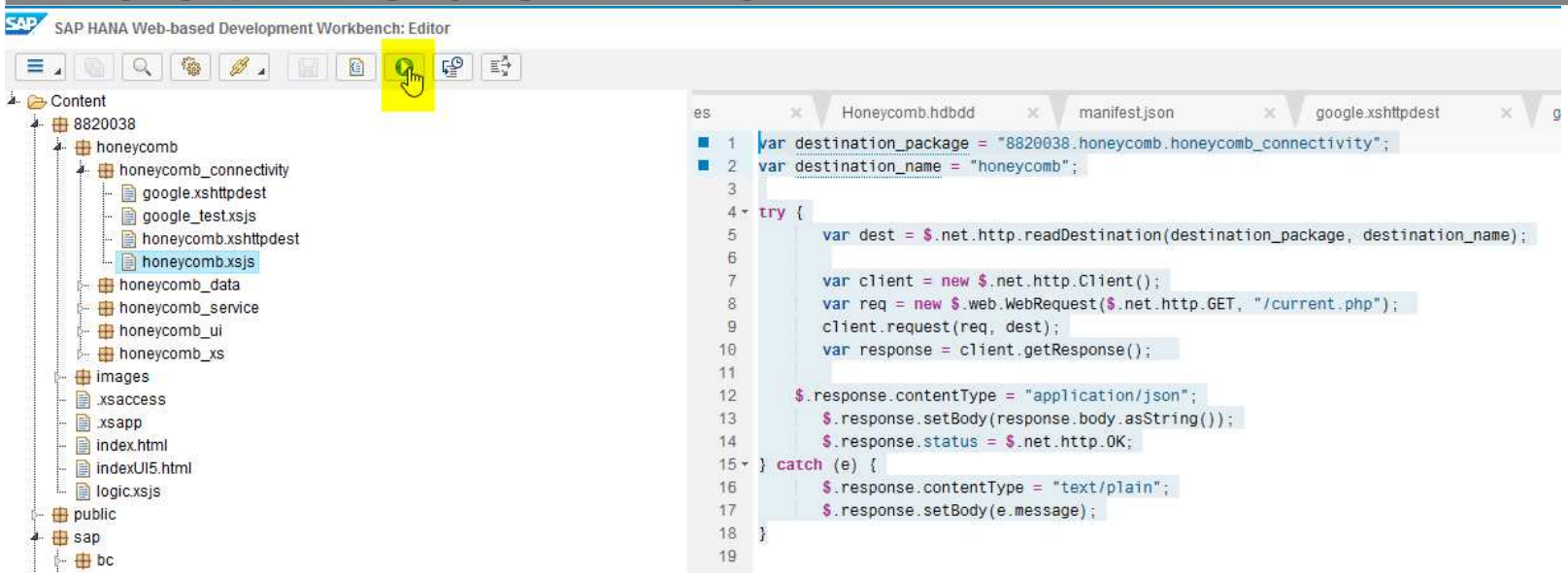
```
var destination_package = "8820038.honeycomb.honeycomb_connectivity";
var destination_name = "honeycomb";

try {
	var dest = $.net.http.readDestination(destination_package, destination_name);

	var client = new $.net.http.Client();
	var req = new $.web.WebRequest($.net.http.GET, "/current.php");
	client.request(req, dest);
	var response = client.getResponse();

   $.response.contentType = "application/json";
	$.response.setBody(response.body.asString());
	$.response.status = $.net.http.OK;
} catch (e) {
	$.response.contentType = "text/plain";
	$.response.setBody(e.message);
}
```

Connectivity
XSJS File

honeycomb_connectivity/google_test.xsjs

```
var destination_package = "8820038.honeycomb.honeycomb_connectivity";
var destination_name = "google";

try {
	var dest = $.net.http.readDestination(destination_package, destination_name);
	var client = new $.net.http.Client();
	var req = new $.web.WebRequest($.net.http.GET,
"?origins=Frankfurt&destinations=Cologne&mode=driving&language=en-US&sensor=false");
	client.request(req, dest);
	var response = client.getResponse();

   $.response.contentType = "application/json";
	$.response.setBody(response.body.asString());
	$.response.status = $.net.http.OK;
} catch (e) {
	$.response.contentType = "text/plain";
	$.response.setBody(e.message);
}
```

Connectivity
Test XSJS File




```
{
        "Weights": {
                "BeekeeperName": "inkerlois",
                "BeekeeperID": "inkerlois",
                "HoneycombID": "1",
                "MeasureDate": "2017-01-05",
                "MeasureTime": "15:23:24",
                "Weight": "24.10",
                "Temperature": "0.10"
        }
}
```

Connectivity
Test XSJS File

Google


```
{
        "destination_addresses": ["Cologne, Germany"],
        "origin_addresses": ["Frankfurt, Germany"],
        "rows": [{
                "elements": [{
                        "distance": {
                                "text": "190 km",
                                "value": 190447
                        },
                        "duration": {
                                "text": "2 hours 1 min",
                                "value": 7283
                        },
                        "status": "OK"
                }]
        }],
        "status": "OK"
}
```

Abschluss LV

- Der **Abschluss** der LV findet im Rahmen des LV Blocks statt. Dieser besteht aus:
  - **Implementierung** von SAP Entwicklungs Objekten im Rahmen der LV
  - Abschlussprüfung
    - **Theoretische** Prüfung mit zehn Fragen aus den Inhalten der LV
    - **Entwicklung** von SAP Entwicklungs Objekten

Literaturliste

Create XS Application https://help.hana.ondemand.com/help/frameset.htm?4959458552574c77b62fe27b0eb363ef.html

SAP HANA Multitenant Database Containers (MDC)
https://blogs.sap.com/2016/01/13/sap-hana-multitenant-database-containers-mdc-scenarios-now-on-trial-landscape/

Install SHINE
https://blogs.sap.com/2016/05/20/shine-for-sap-hana-extended-services-classic-model-xsc-as-self-service/

VizChart
https://archive.sap.com/discussions/thread/3745279

HAMA SBS 11
https://blogs.sap.com/2016/03/31/sap-hana-sps-11-new-developer-features-sap-web-ide-for-sap-hana/

DataSource
http://www.niun.org/beelog/current.php
http://www.niun.org/beelog/all.php

CDS
https://help.sap.com/saphelp_hanaplatform/helpdata/en/b5/23afd66f5a40469573d9c47d7af831/content.htm?frameset=/en/ad/036c56b5e545ae8b31ece0ab95379f/frameset.htm&current_toc=/en/8f/796f93e6c3419c9b8029197aa61874/plain.htm&node_id=50&show_children=false

Connectivity
https://help.hana.ondemand.com/help/frameset.htm?2eb13e1217a74561af5950e006fd1e36.html

Debugging
https://help.hana.ondemand.com/help/frameset.htm?1beaa7aaadc743568c8144066d005dab.html

JobSchedule
https://help.sap.com/saphelp_hanaplatform/helpdata/en/62/15446213334d9fa96c662c18fb66f7/content.htm

Arduino to HANA mit IoT
https://create.arduino.cc/projecthub/derapados/arduino-iot-with-sap-hana-cloud-platform-031988

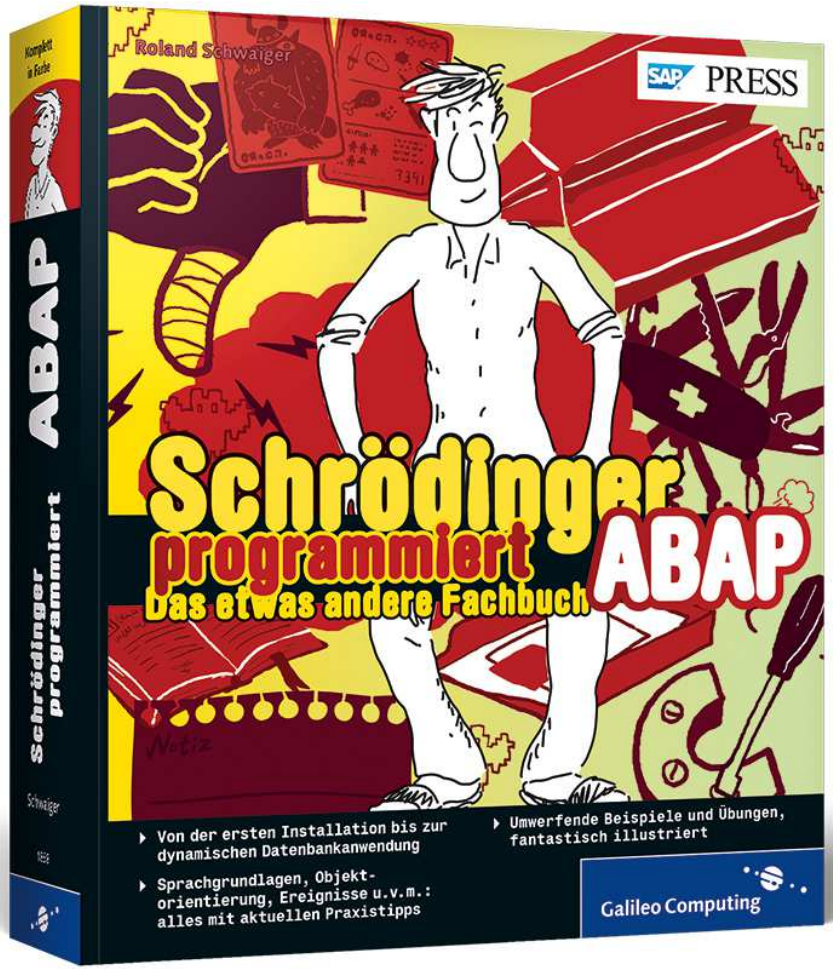
IoT FAQ HANA
https://help.hana.ondemand.com/iot/frameset.htm?a012a3788b6e498a8971dd27c97ce6bf.html

sdn.sap.com
help.sap.com
www.facet.at

Glossary

Abbreviation	Explanation
MDC	SAP HANA Multitenant Database Containers

Empfehlenswert



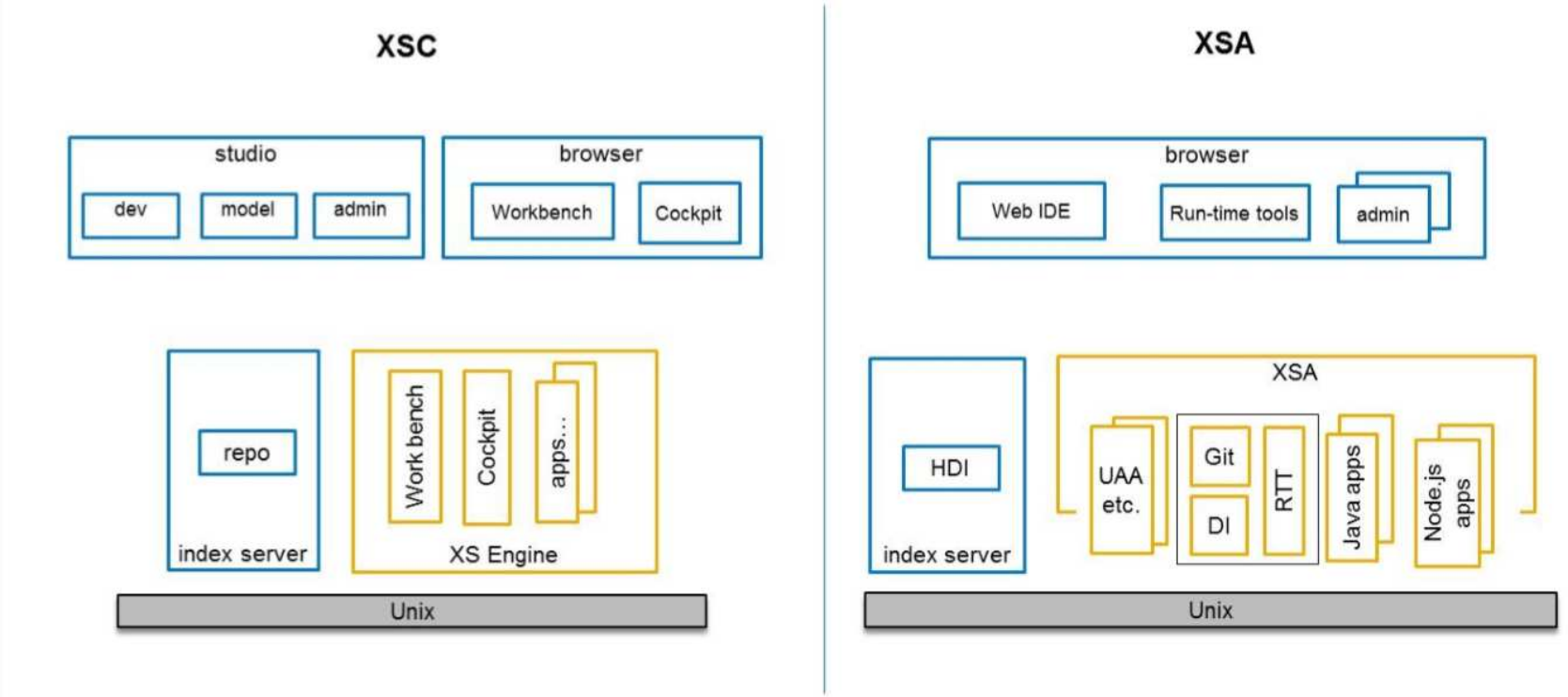
APPENDIX

All the things that did not make it into the main release

XSA

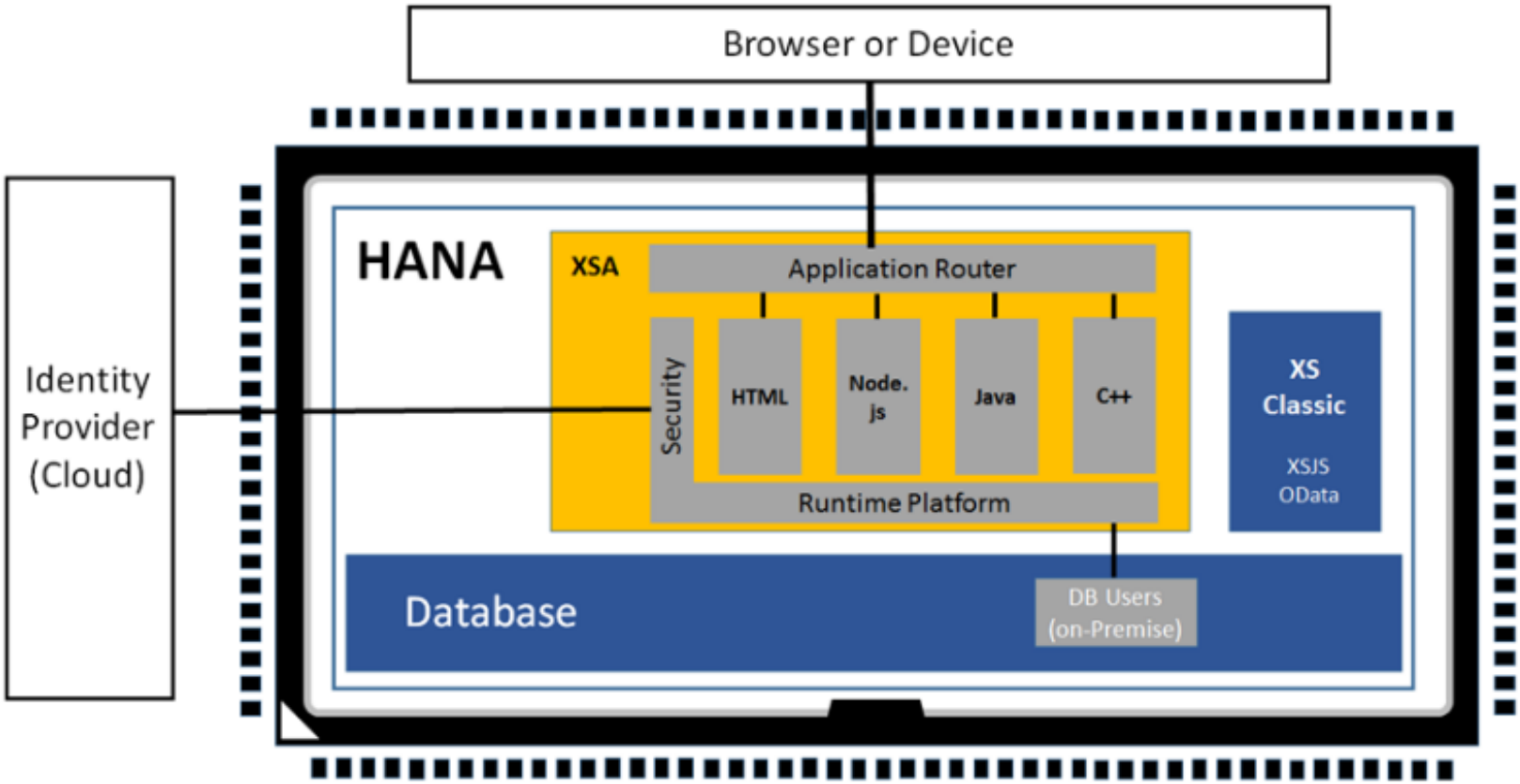
Because XSA (SPS 11) is not available in the HANA Trial at the moment of writing.

Introduction to HANA
XS Classic vs. XS Advanced



© SAP

Introduction to HANA
XSA - Arch



XSA - Arch

- SAP HANA XS advanced (XSA) is a completely re-engineered application server for native development of applications in SAP HANA environment which is available with **SP11**. It brings dramatic improvements in terms of architecture with microservices.

- XSA has an **application router** to route browser requests to relevant applications. XSA supports **multiple language** runtimes like Node.js, Java, C++ and so on, as well as a runtime platform. One application can use one or more of these runtimes by combining them as **microservices** running side by side in one application.

- In SAP HANA XS, if one service running on HANA server fails, it impacts the dependent processes. In XS Advanced on the other hand, these language runtimes run separately for every app by using a copy of the runtime version or SDKs as required. They also run independently of one another, which reduces the impact of a failed service and high scalability.

- When it comes to security, no HANA DB User is needed for accessing the HANA applications. SAP HANA **XSA** uses an **identity provider**, which provides user authentication info using SAP Cloud Identity Service or SAP ID Service (IDS) as in SCN, SAP.com or SAP store.

- **Note:** SAP HANA XS and SAP HANA XSA co-exist in HANA, but are completely separated and work independently of one another. SAP HANA XSA supersedes SAP HANA XS in HANA native development.

XSC vs XSA

XSC

+ Familiar development environment (HANA Studio, Web IDE and the HANA repository)
+ System in a Box
+ Supports JavaScript as application run-time
+ Direct access to database objects
+ Integrated transportation management
- Single source code repository per system
- Limited scalability due to tight coupling with DB
- Requires DB authorization management to be used
- Requires additional systems for production emergency fix delivery

XSA

+ Application layer separable from DB
+ Support of different run-time containers, e.g. Node.js (JavaScript), Java, C++
+ Highly scalable due to container concept
+ Advanced Git/Gerrit Source Code Management
+ Advanced resource management per container
- Additional HANA software components to be installed
- External Source Code Repository to be set up
- Additional development knowledge necessary to create database objects

❖ Web-based tools for developers and administrators (**no Eclipse**/HANA Studio support!)
❖ Development: SAP Web IDE + server-side Development Infrastructure (DI)
❖ HANA container isolation and activation: HDI
❖ Deploy MTA archives

XSC vs XSA

XS Classic

- HANA repository contains design-time objects
- Objects are organized in repository packages
- Database run-time objects are created upon activation of design-time objects
- Objects can be activated individually
- User _SYS_REPO owns all run-time objects

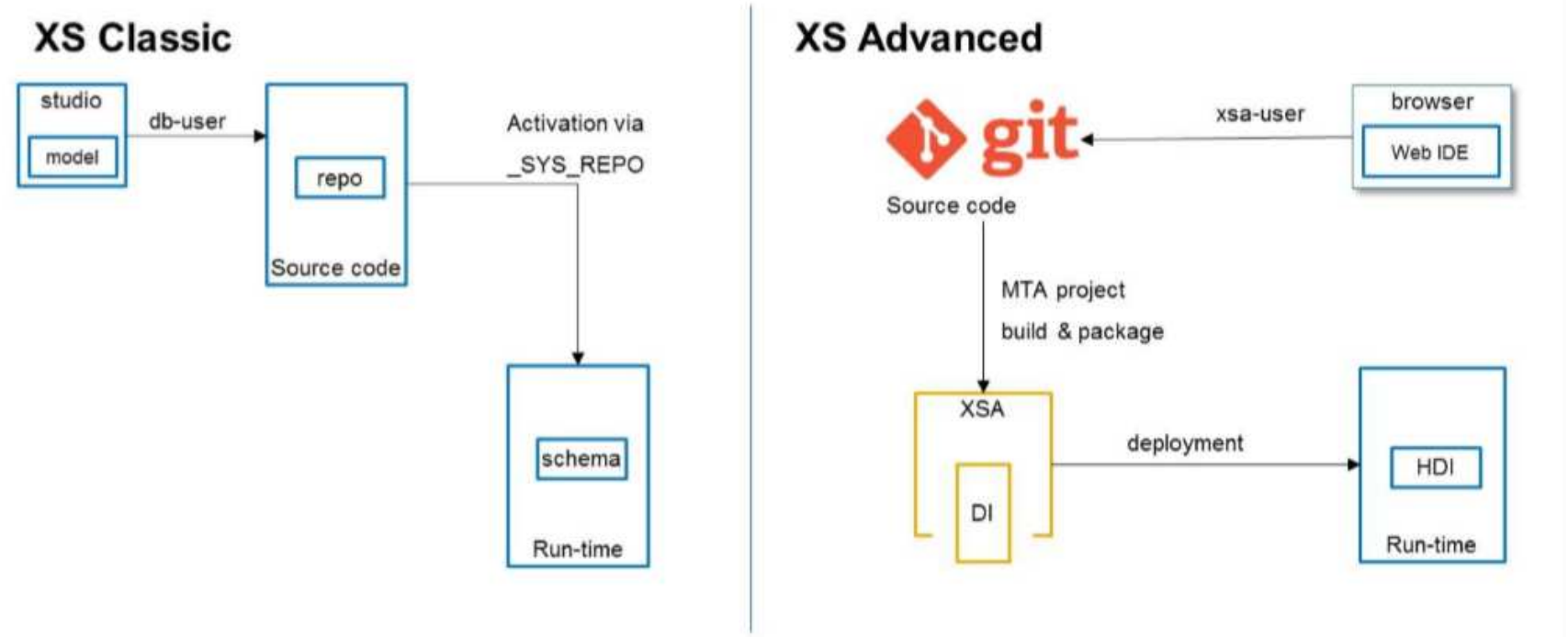
XS Advanced

- External GIT repository contains design-time objects
- Objects are organized in MTA Projects
- Projects have to be built by the XSA infrastructure
- Database run-time objects are created upon deployment of the built HDI Container
- A generated …#OO user owns the run-time objects for a specific HDI container

Multi Target Application (MTA)

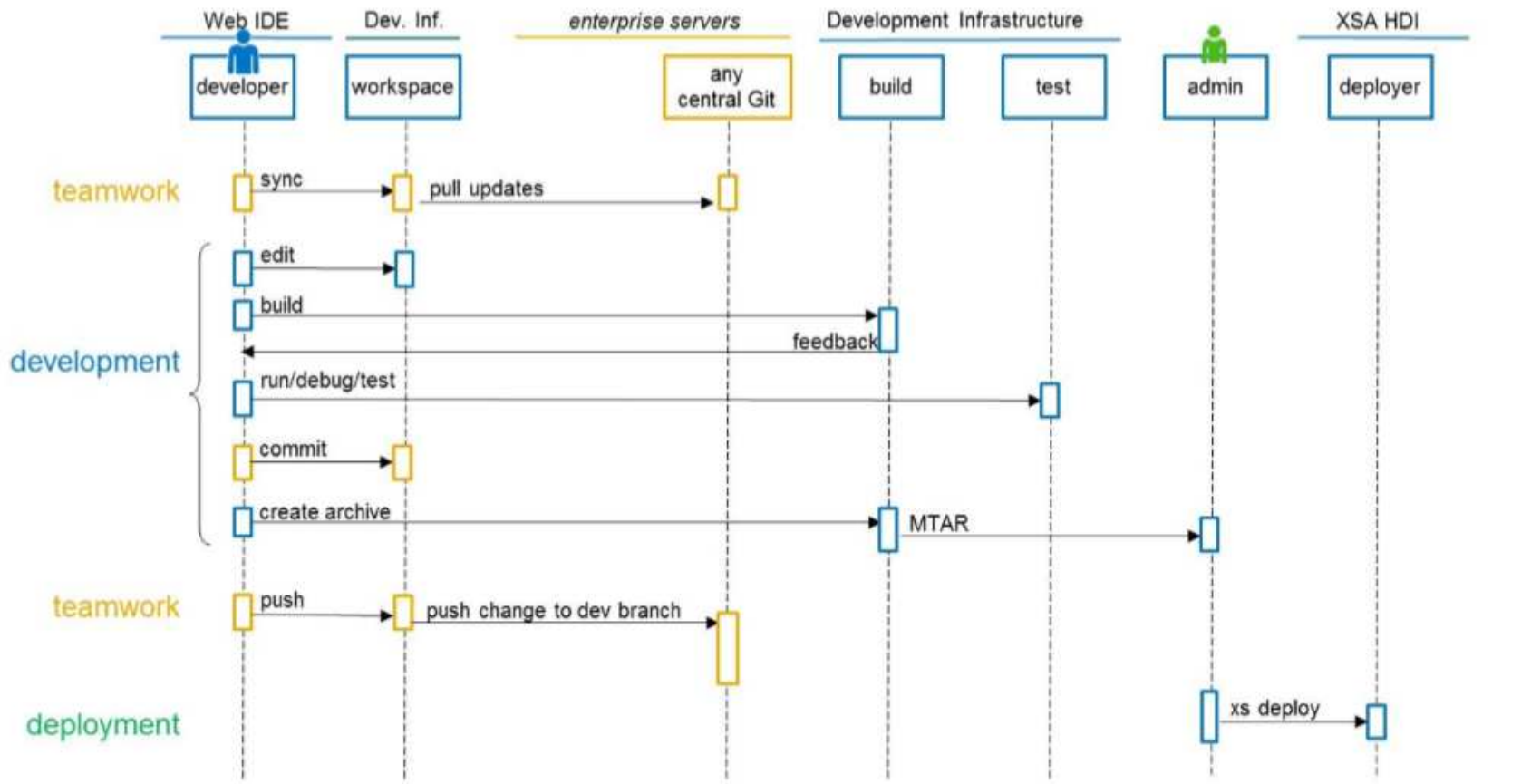
HANA Development Infrastructure (HDI)

XSC vs XSA
Dev Process



© SAP

XSA Dev Process



© SAP

XSA Dev Process

Usually used components
•SAP HANA Web IDE
•Development Infrastructure (DI)
•GIT source code repository
•HANA Deployment Infrastrucute (HDI)

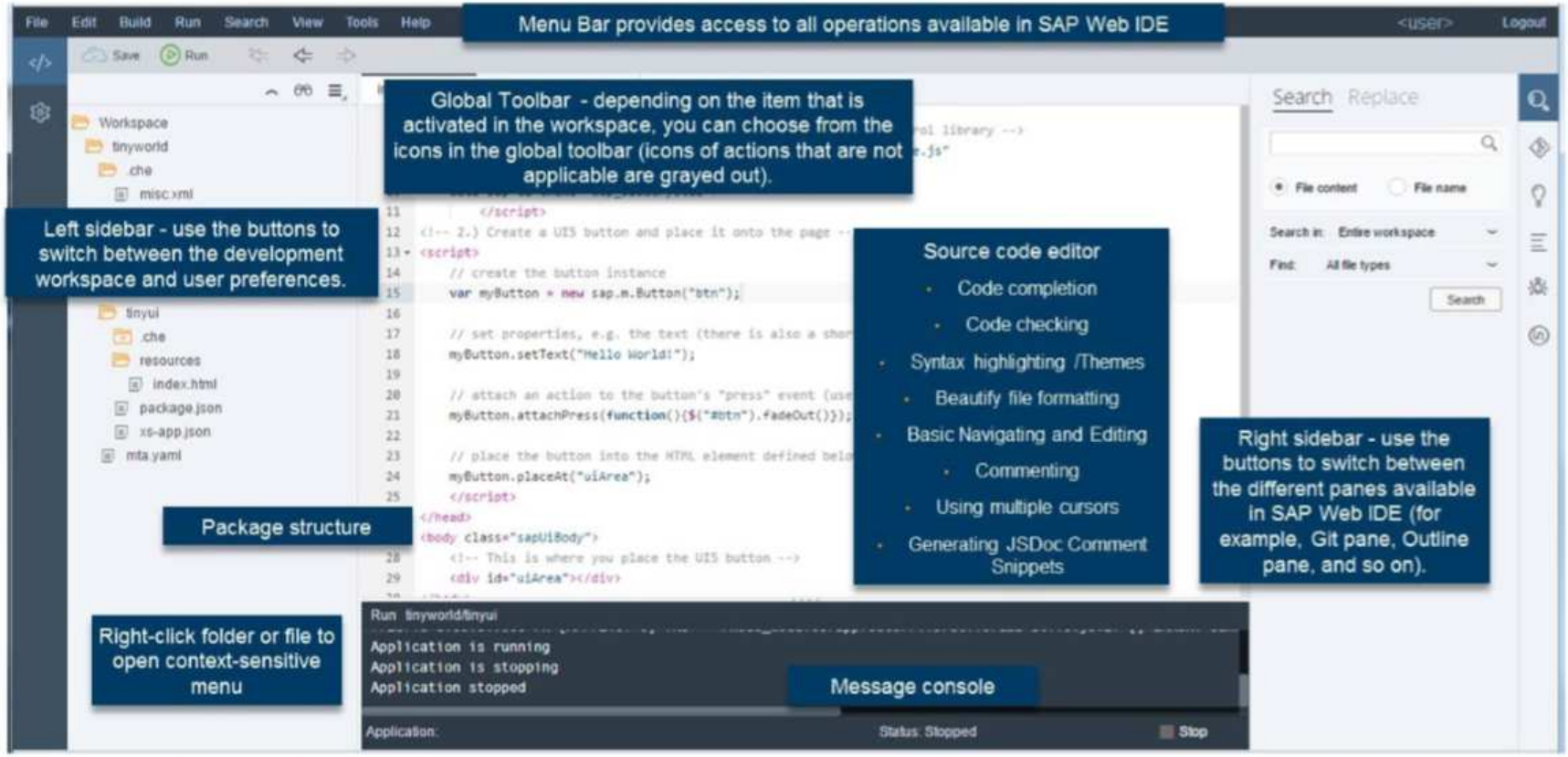
The process itself works the following way:
1.The developer pulls the latest sources from the GIT repository into his Web IDE development workspace
2.He makes some changes to the code
3.He builds his local version of the project which is done by the Development Infrastructure
4.In order to run/test/debug his version he uses again the DI
5.The developer commits the changes to the local workspace
6.When he is satisfied with the changes he creates the MTAR archive which contains the built objects
7.From the local workspace the changes are pushed to the central GIT repository
8.The MTAR archive will be deployed by an administrator in the target system and made available

XSA Application Dev Tools

- **XS Advanced Development Tools**
  - Code editors
  - Debugging tools
  - Unit Test tools
  - Version Control tools
  - Code visualization tool
- **XS Advanced Run-Time Tools**
  - SAP HANA database catalog browser
  - SQL Console
  - SQL Debugger
  - Job Log Viewer
- **XS Advanced Administration Tools**
  - Application Monitor
  - Organization and Space Management
  - Application Role Builder
  - SAML Identity Providers Configuration
  - User Management
  - SAP HANA Logical Database Setup
  - SAP HANA Service Broker Configuration
  - Job Scheduler Service Dashboard

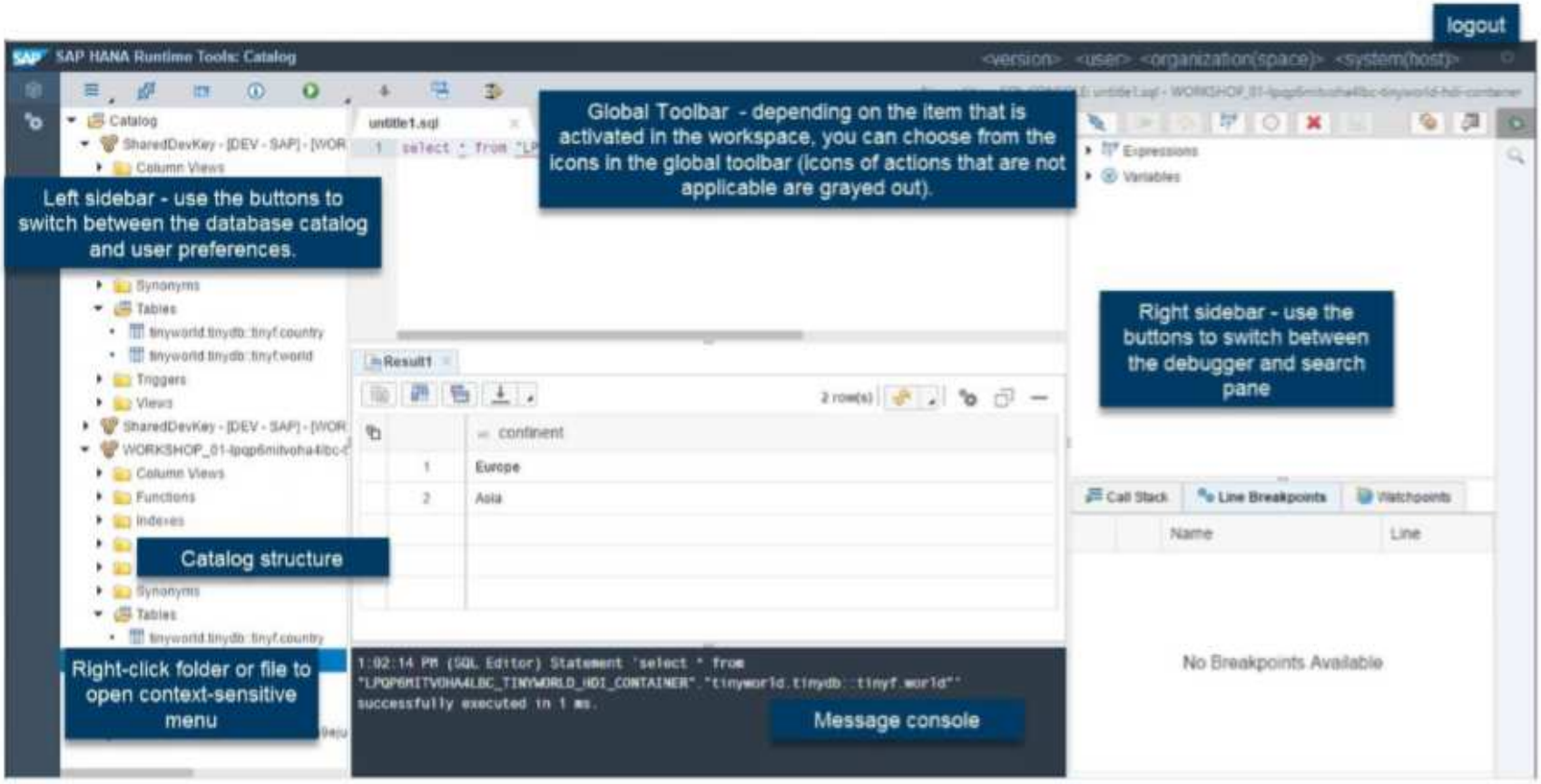
© SAP

SAP Web IDE: Code Editor Tool



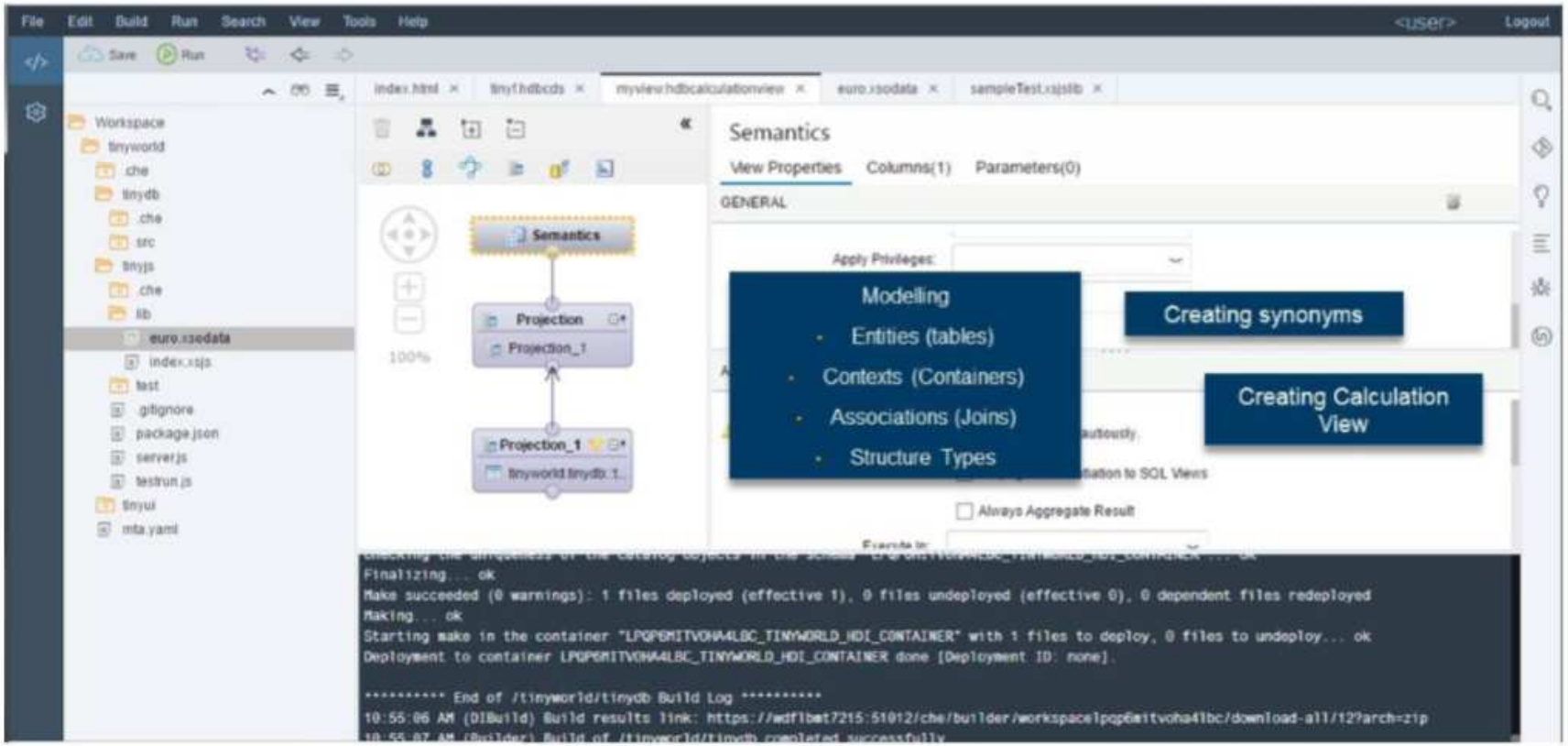
© SAP

SAP HANA
Runtime Tools: Catalog



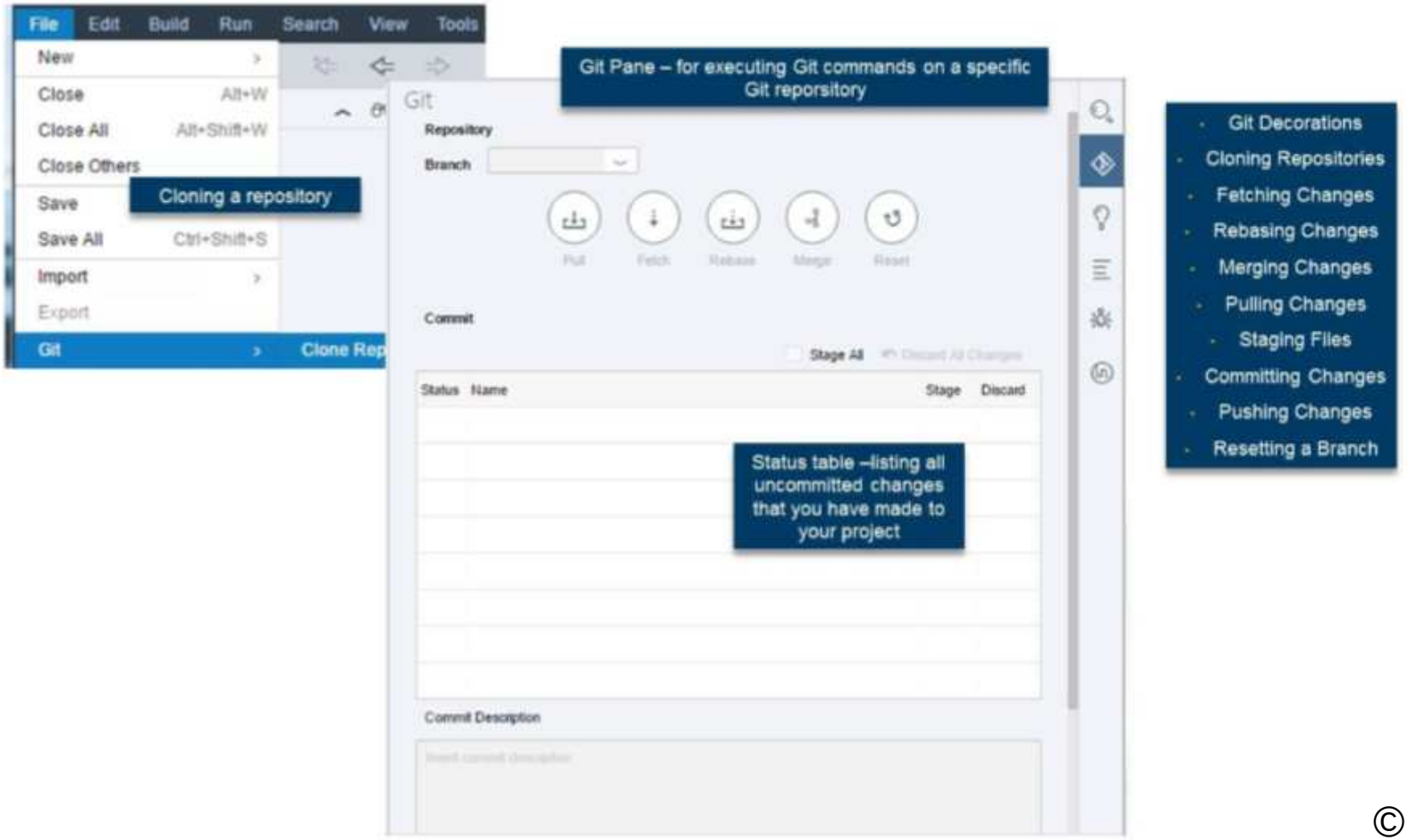
© SAP

SAP Web IDE: Graphical Editor



© SAP

SAP Web IDE:
Git Source Control System

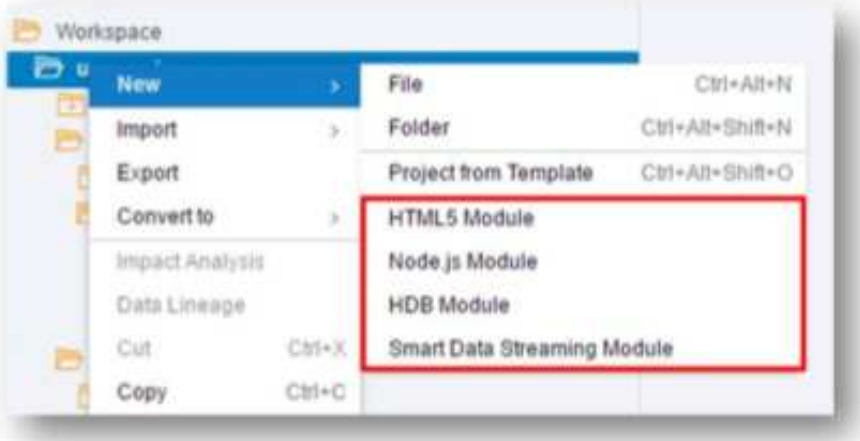


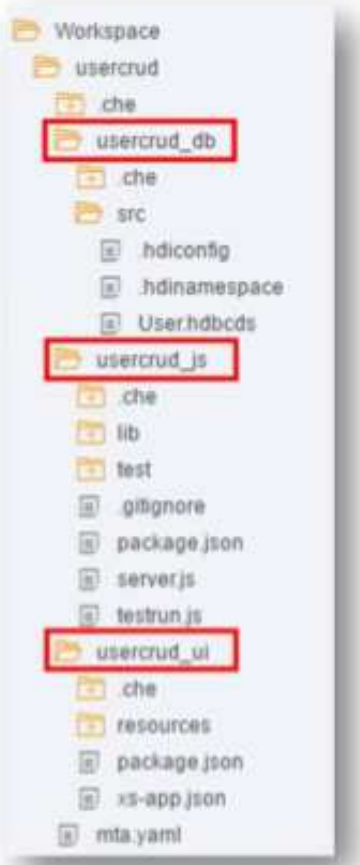
© SAP

XSA Multi Target App Example

Example of an MTA project in the WEB IDE

- Sub-structures for
  - HDB Module
  - JS Module
  - Web Module
- Configuration files
  - .hdiconfig
  - package.json
  - mta.yaml



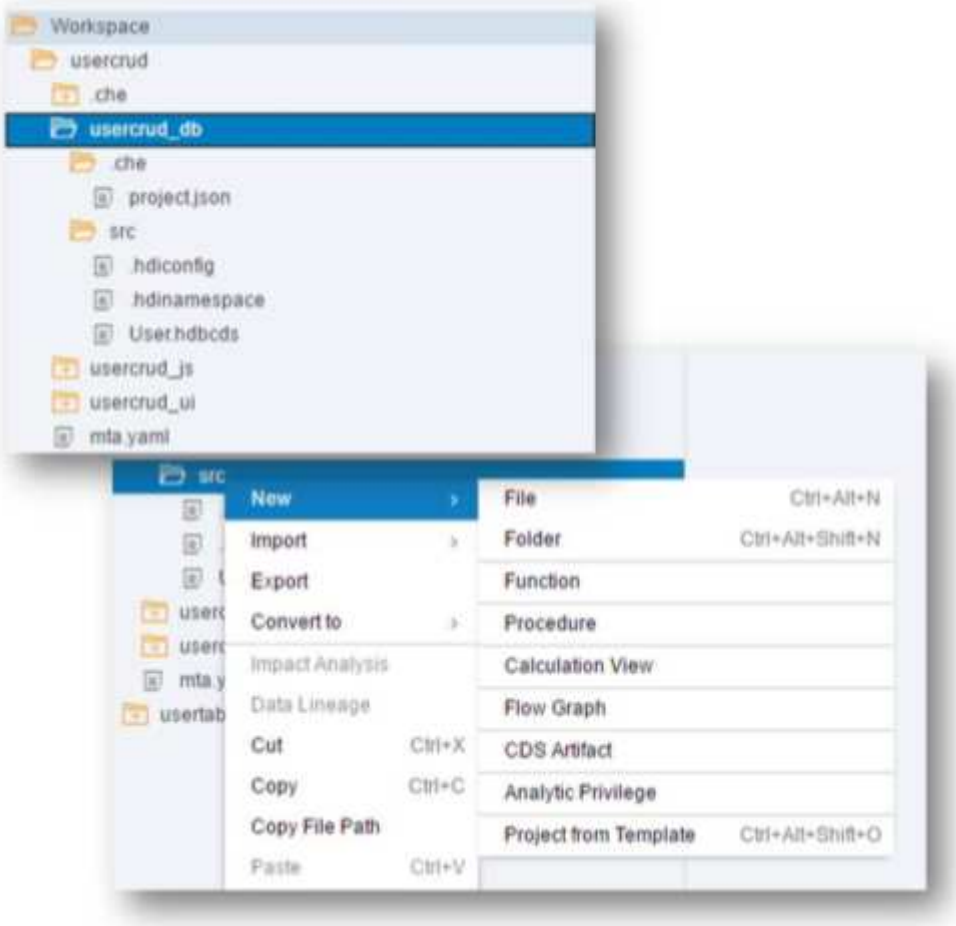


© SAP

XSA Multi Target App
HDB Module (to create DB content)

Module for Database Content

- Standard files
  - project.json
  - .hdiconfig
  - .hdinamespace
- Web IDE templates
  - Function
  - Procedure
  - Calculation View
  - Flow Graph
  - CDS Artefact
  - Analytic Privilege



© SAP

XSA Multi Target App .hdiconfig

.hdiconfig

- Contains the plugin name and version for every supported file suffix
- Filled automatically by the Web IDE
- Versions can be adapted to match different HANA DB Versions

XSA Multi Target App .hdinamespace

.hdinamespace

- Contains namespace configuration for the HDB module
- name
- subfolder:
  - append
  - ignore

.hdinamespace

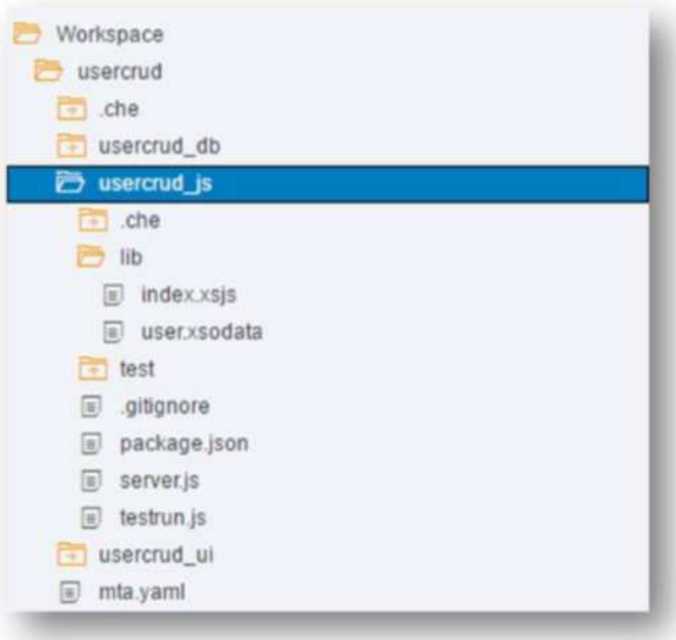
```
{
    "name" : "com.sap.hana.example",
    "subfolder" : "<[append | ignore]>"
}
```

Design-time Resource Path	Name of Run-Time Object (append)
/src/	<namespace>::<objectName>
/src/load	<namespace>.load::<objectName>
/src/load/data	<namespace>.load.data::<objectName>

XSA Multi Target App Node.js (JS) Module

Module for Server-Side JavaScript

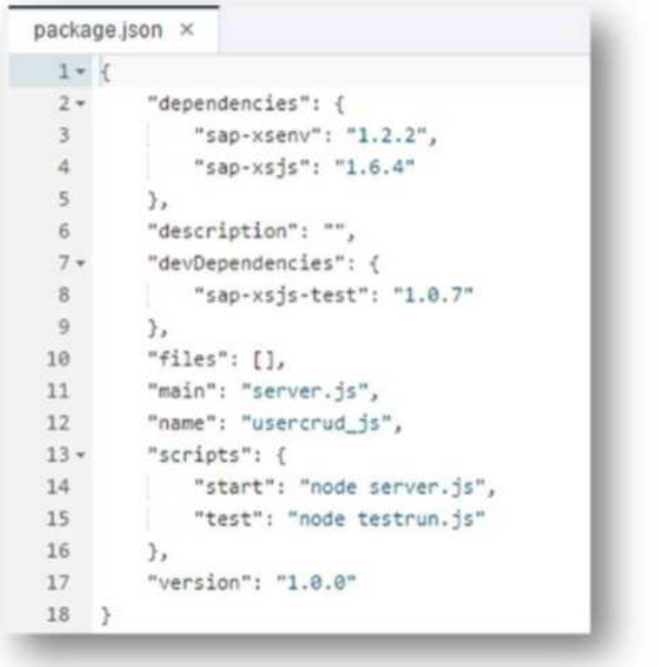
- Standard files
  - package.json
  - server.js
  - testrun.js
  - /lib/index.xsjs



XSA Multi Target App package.json

XS Advanced Application Package Descriptor

- Used by the Node Package Manager (NMP)
- name
- version
- dependencies
- engines (Node.js version)
- scripts
- main (main Program)



XSA Multi Target App server.js

Main JavaScript file

- Started by default when running the Node.js module
- XSJS compatibility layer
- XSENV variables
- HANA DB connection
- XSUAA security service
- Webserver start

XS Advanced Environment Variables (XSENV)

XS Advanced User Account and Authorization Service (XSUAA)

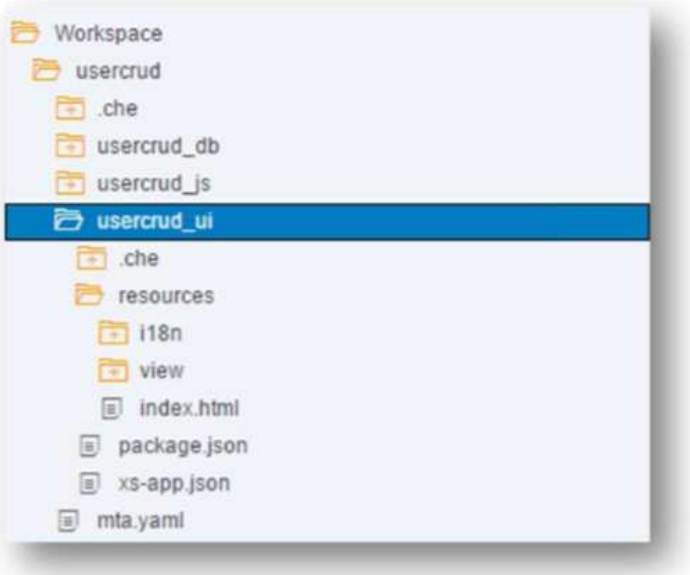
server.js

```
/*eslint no-console: 0, no-unused-vars: 0*/
"use strict";

var xsjs  = require("sap-xsjs");
var xsenv = require("sap-xsenv");
var port  = process.env.PORT || 3000;

var options = xsjs.extend({
    anonymous : true, // remove to authenticate calls
    redirectUrl : "/index.xsjs"
});

//configure HANA
try {
    options = xsjs.extend(options, xsenv.getServices({ hana: {tag: "hana"} }));
} catch (err) {
    console.error(err);
}

// configure UAA
try {
    options = xsjs.extend(options, xsenv.getServices({ uaa: {tag: "xsuaa"} }));
} catch (err) {
    console.error(err);
}

// start server
xsjs(options).listen(port);

console.log("Server listening on port %d", port);
```

XSA Multi Target App
HTML5 Module

HTML5 Module for frontend/user interface

- Contains static content to be delivered to the client side
- Standard files
  - package.json
  - xs-app.json



XSA Multi Target App mta.yaml

Multi Target Application Development Descriptor

- Parts
  - Global elements
  - Modules
  - Resources
  - Properties
  - Parameters

The development descriptor (mta.yaml) is used to define the elements and dependencies of an XS advanced-compliant multi-target application (MTA)

mta.yaml

```
_schema-version: 2.0.0
ID: usercrud
version: 0.0.1

modules:
- name: usercrud_db
  type: hdb
  path: usercrud_db
  requires:
   - name: hdi-container

- name: usercrud_js
  type: nodejs
  path: usercrud_js

# ------- dependency on usercrud_db
  requires:
    - name: usercrud_db
    - name: hdi-container

#---------------- exposes SERVICE URL to consumers
  provides:
    - name: usercrud_js_api
      properties:
        service_url: ${default-url}
```

XSA Multi Target App
mta.yaml - Global elements

Global elements

- _schema_version*
- ID*
- description
- version*
- provider
- copyright

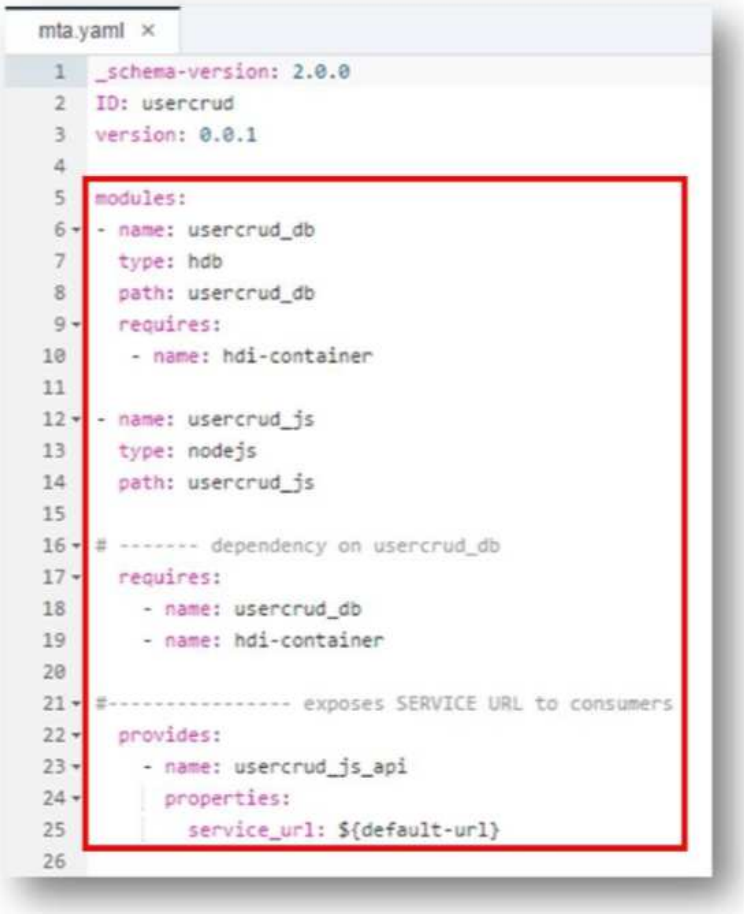
mta.yaml

```
1  _schema-version: 2.0.0
2  ID: usercrud
3  version: 0.0.1
4
5  modules:
6  - name: usercrud_db
7    type: hdb
8    path: usercrud_db
9    requires:
10    - name: hdi-container
11
12 - name: usercrud_js
13   type: nodejs
14   path: usercrud_js
15
16 # ------- dependency on usercrud_db
17   requires:
18     - name: usercrud_db
19     - name: hdi-container
20
21 #---------------- exposes SERVICE URL to consumers
22   provides:
23     - name: usercrud_js_api
24       properties:
25         service_url: ${default-url}
26
```

XSA Multi Target App mta.yaml Modules

Modules

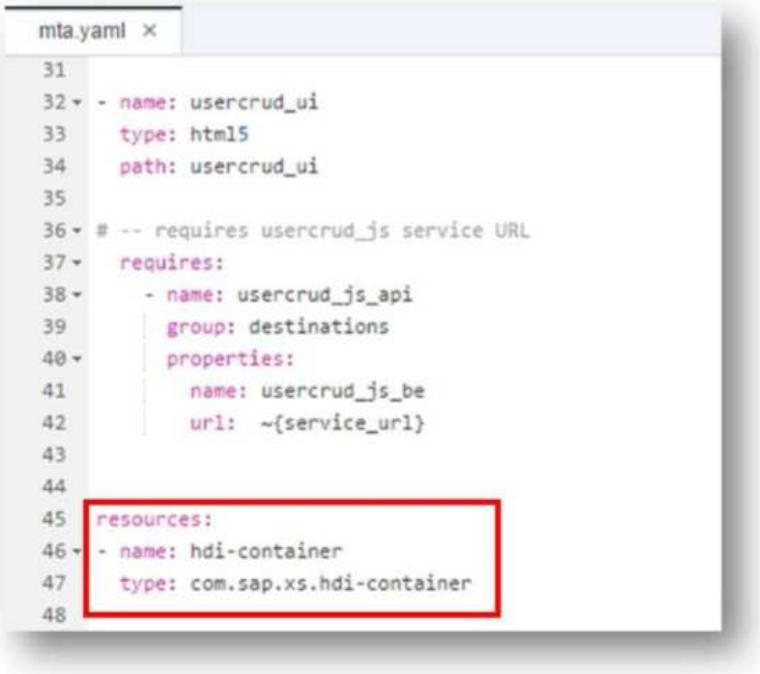
- Source modules of the MTA project
  - name*
  - type*
  - path*
  - description
  - requires
  - provides
  - properties
  - parameters



XSA Multi Target App mta.yaml Resources

Resources

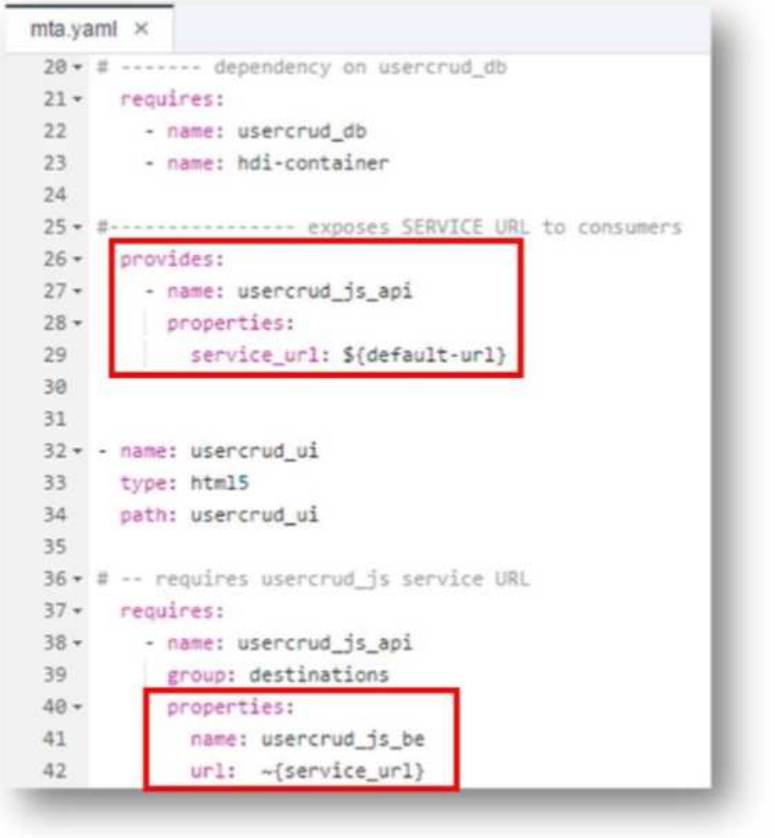
- Dependency declaration on external resources
  - name*
  - type*
  - description
  - properties
  - parameters



XSA Multi Target App mta.yaml

Properties

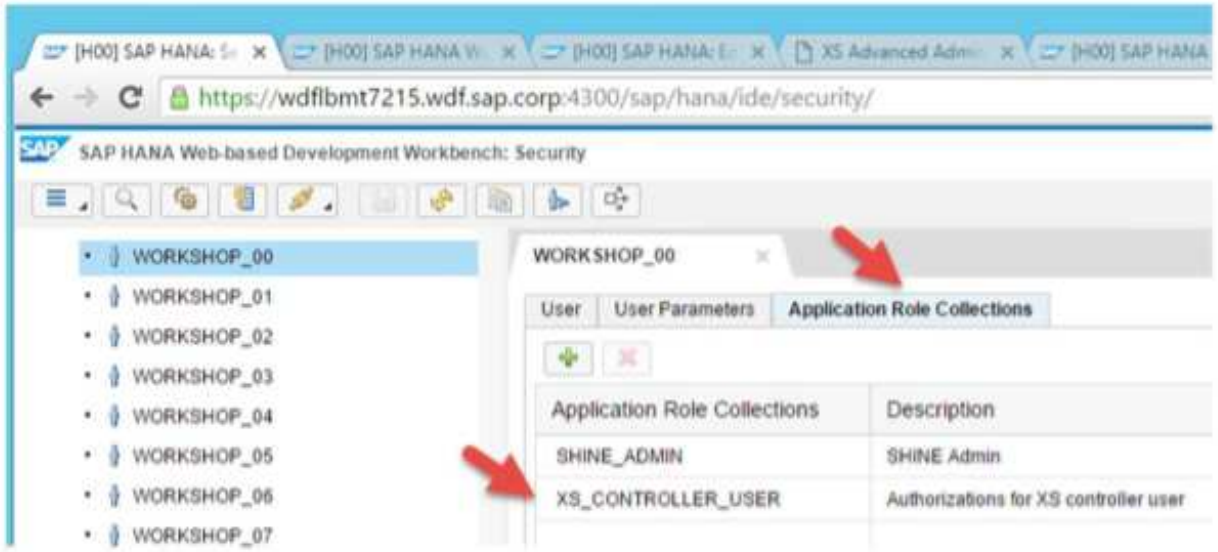
- Variables specified during deployment
  - <key>: <value>
  - use defined placeholders via *${<parameter>}*
  - refer to property values via *~{<property>}*



Creating an XSA Project Auth

For HANA users to be able to access XS Advance you need:

- To have the XS_CONTROLLER_USER collection assigned in the usre maintenance screen.



SQLScript Triggers Example 1/2

SQL

```
1  CREATE TRIGGER "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.sql/add_insert_to_log"
2      AFTER INSERT ON "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Products"
3      REFERENCING NEW ROW newrow FOR EACH ROW
4  BEGIN
5
6    INSERT INTO "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.ProductLog"
7            VALUES(:newrow.PRODUCTID, now(), CURRENT_USER,
8                   :newrow.PRODUCTID || ' has been created');
9  END;
```

SQL

```
1  INSERT into "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Products"
2    values( 'ProductA', 'PR', 'Handheld', '0000000033', '20121003', '0000000033', '20121003',
3           '1000000149', '1000000150', '0100000029', 1, 'EA', 0.5, 'KG', 'CAD', 2490,
4           '/sap/hana/democontent/epmSP6/data/images/HT-7030.jpg', 0.09, 0.15, 0.1, 'M');
```

SQL | Result | Result

```
select PRODUCTID, CATEGORY, PRICE from "SAP_HANA_
        where PRODUCTID = 'ProductA'
```

	PRODUCTID	CATEGORY	PRICE
1	ProductA	Handheld	2,490

SQL | Result | Result

```
select * from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.ProductLog"
```

	PRODUCTID	DATETIME	USERNAME	LOGTEXT
1	ProductA	Jun 27, 2014 10:40:10.0 AM	SYSTEM	ProductA has been created

SQLScript Triggers Example 2/2

SQL

```
1  CREATE TRIGGER "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.sql/add_update_price_to_log"
2      AFTER UPDATE ON "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Products"
3      REFERENCING NEW ROW newrow, OLD ROW oldrow FOR EACH ROW
4  BEGIN
5
6    declare lv_price_difference decimal(15,2) := 0;
7    if :oldrow.price <> :newrow.price then
8        lv_price_difference := :newrow.price - :oldrow.price;
9        INSERT INTO "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.ProductLog"
10             VALUES(:newrow.PRODUCTID, now(), CURRENT_USER,
11                  :newrow.PRODUCTID || ' has been updated with price difference of ' ||
12                   to_decimal(lv_price_difference, 15, 2));
13   end if;
14
15 END;
```

SQL

```
1  UPDATE "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.Products"
2                  SET PRICE = 2690.00 WHERE PRODUCTID = 'ProductA';
3
```

SQL | Result | Result

```
select PRODUCTID, CATEGORY, PRICE from "SAP_HANA_EPM_NEXT"."s
          where PRODUCTID = 'ProductA'
```

	PRODUCTID	CATEGORY	PRICE
1	ProductA	Handheld	2,690

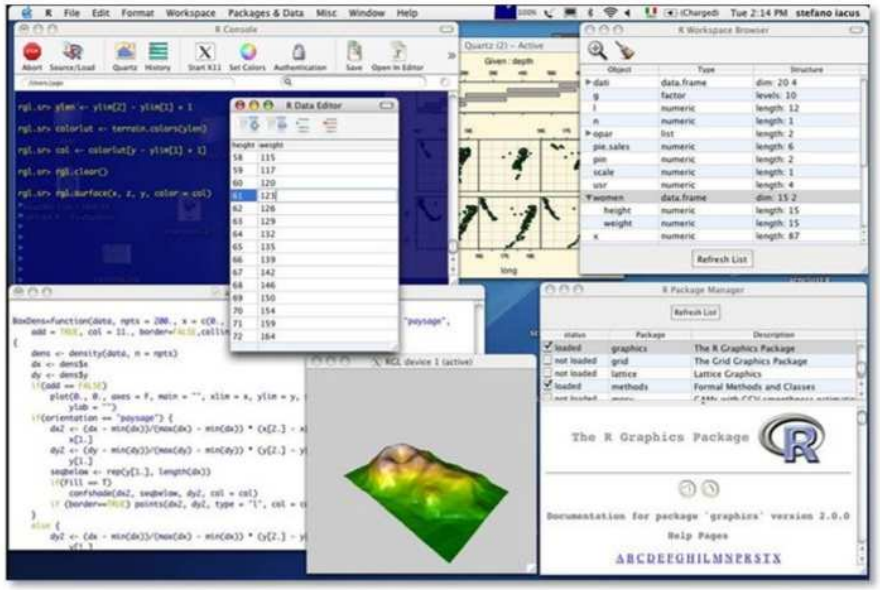
SQL | Result | Result

```
select * from "SAP_HANA_EPM_NEXT"."sap.hana.democontent.epmNext.data::EPM.MasterData.ProductLog"
```

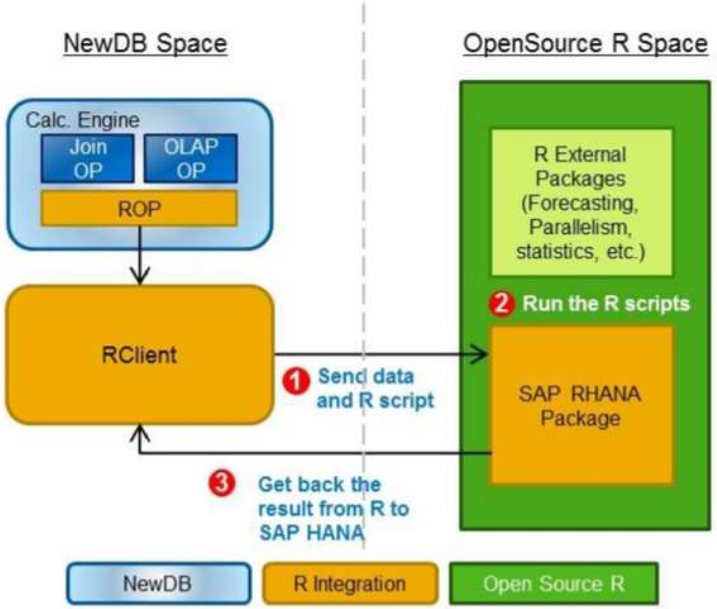
	PRODUCTID	DATETIME	USERNAME	LOGTEXT
1	ProductA	Jun 27, 2014 10:40:10.0 AM	SYSTEM	ProductA has been created
2	ProductA	Jun 27, 2014 10:42:37.0 AM	SYSTEM	ProductA has been updated with price difference of 200.00

R

- An open source software language and environment for statistical computing and graphics with over 3000 add-on packages.
- http://www.r-project.org/
- The packages cover a wide range of topics:
  - Cluster Analysis & Finite Mixture Models
  - Probability Distributions
  - Computational Econometrics
  - Empirical Finance
  - Statistical Genetics
  - Graphic Displays, Dynamic Graphics, Graphic Devices, & Visualisation
  - Machine Learning & Statistical Learning
  - Medical Image Analysis
  - Multivariate Statistics
  - Natural Language Processing
  - Statistics for the Social Sciences
  - Time Series Analysis



R
HANA Integration



*build_product_list.procedure

ORG (SYSTEM)

SAP HANA SQLScript | Local Table Types

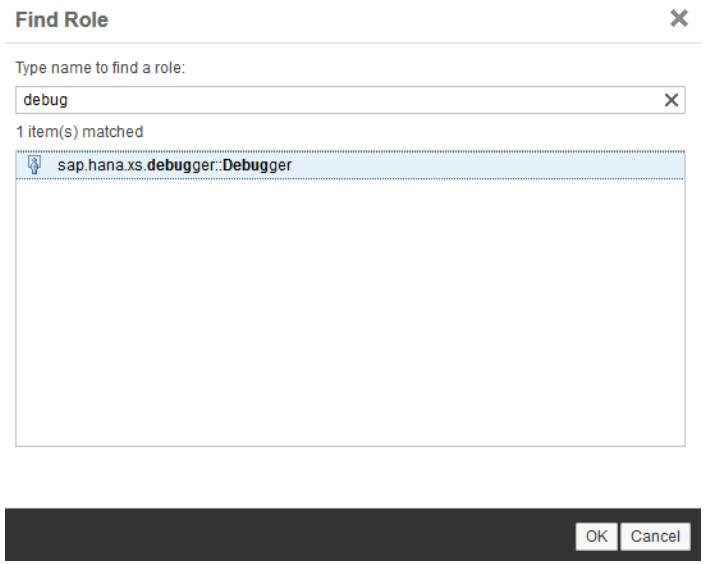
```
1  CREATE PROCEDURE build_product_list ( )
2      LANGUAGE RLANG
3      SQL SECURITY INVOKER
4      --DEFAULT SCHEMA <default_schema_name>
5      READS SQL DATA AS
6  BEGIN
7  /*****************************
8      Write your procedure logic
9   *****************************/
10     ProductId <- c('HT-1000', 'HT-1095', 'HT-1100')
11     Category <- c('Notebooks', 'Headset', 'Software')
12     Price <- c(199.99, 299.99, 399.99)
13     result <- as.data.frame(cbind(ProductId,Category,Price))
14
15 END;
```

© SAP

XSJS

Debugging Precondition for
HANA Web-based Development Workbench

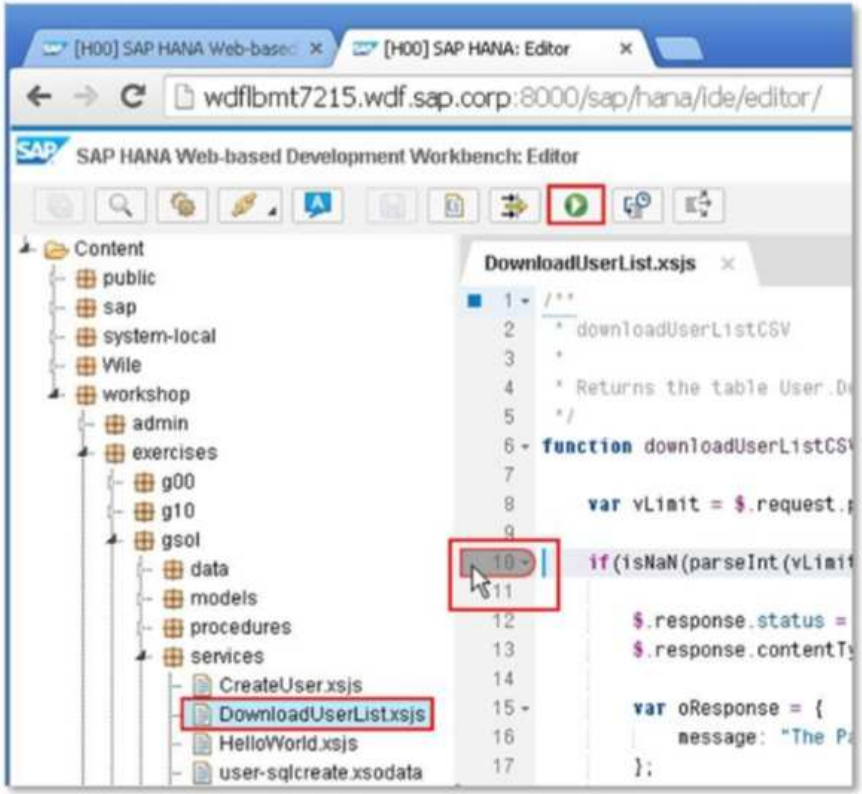
- Assign the database user
  the s*ap.hana.xs.debugger::Debugger* role.



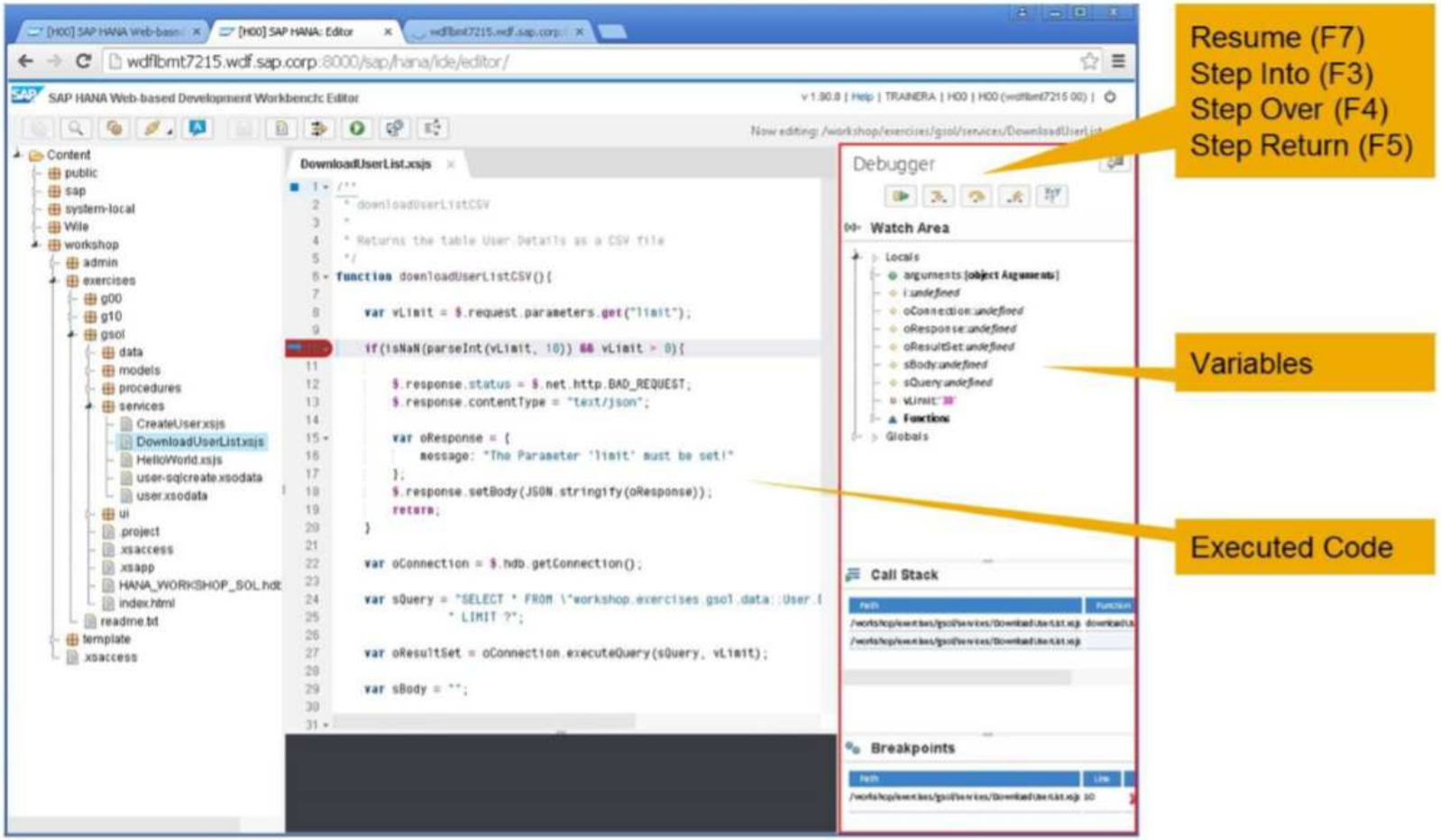
XSJS Debugging

The debug functionality is also included in the SAP HANA WEB IDE.

- Open the XSJS Program that you want to debug.
- Set a breakpoint by clicking the line number.
- Execute the program.
- A new browser window will appear, executing the program.
- If you switch back to the WEB IDE window, you can access the debugger.



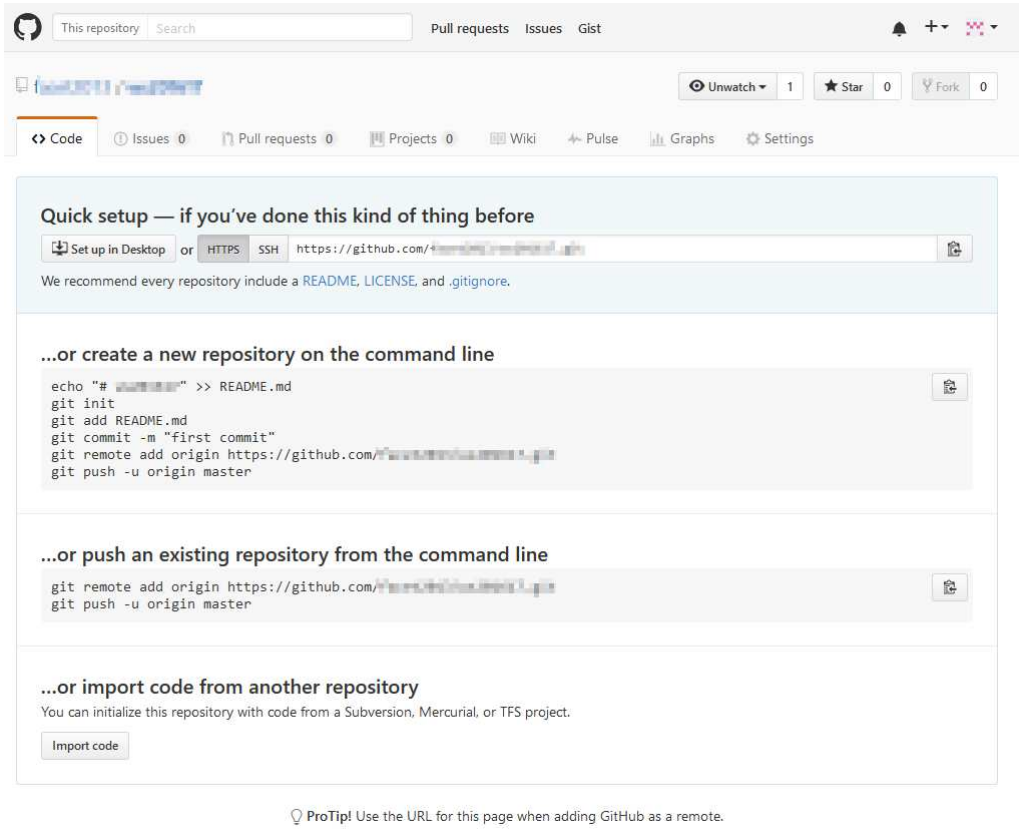
XSJS Debugging



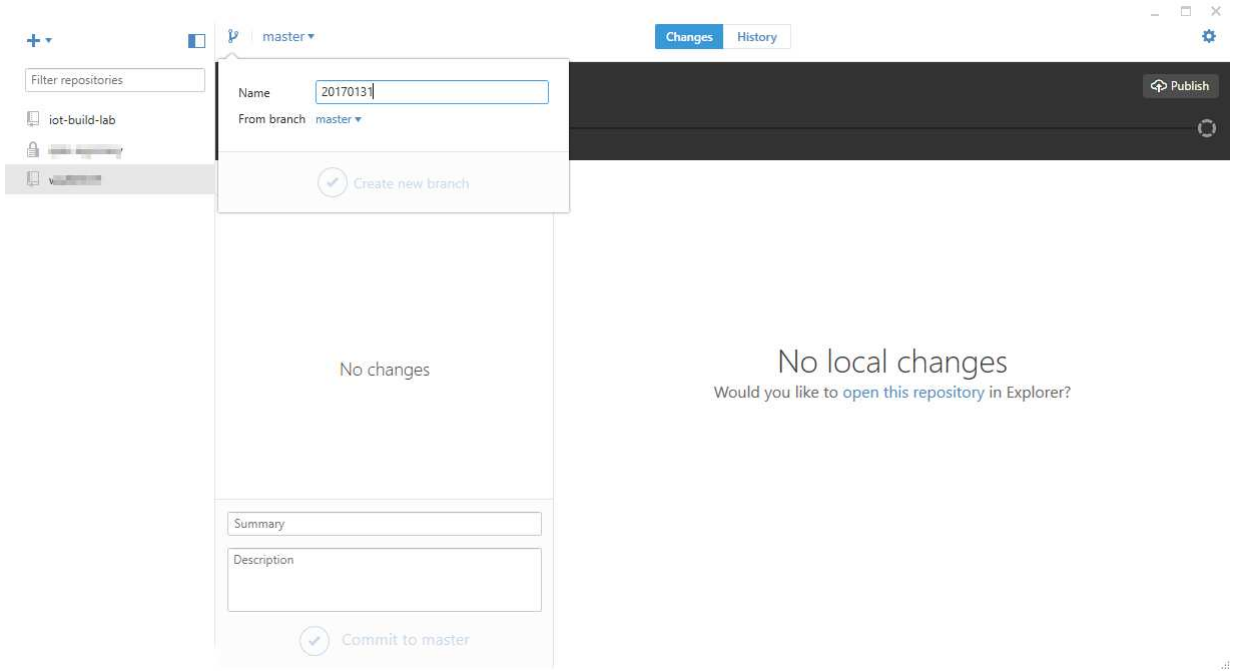
GIT

Git

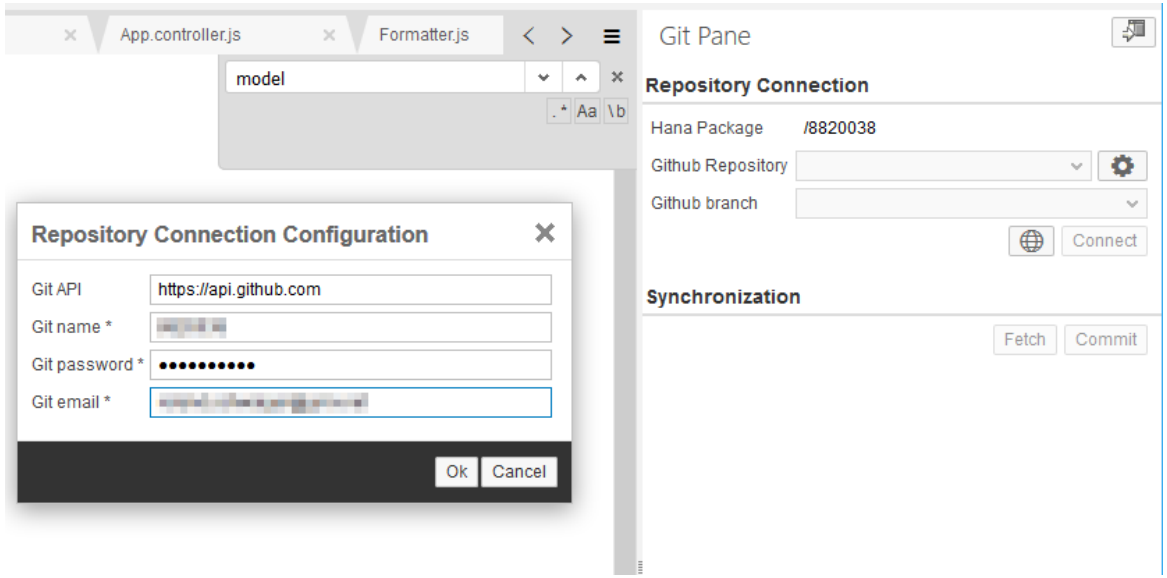
Create Repo



Git
Create Branch



Git in HANA


App.controller.js
Formatter.js
model
Repository Connection Configuration
Git API
https://api.github.com
Git name *
Git password *
Git email *
Ok
Cancel
Git Pane
Repository Connection
Hana Package
/8820038
Github Repository
Github branch
Connect
Synchronization
Fetch
Commit

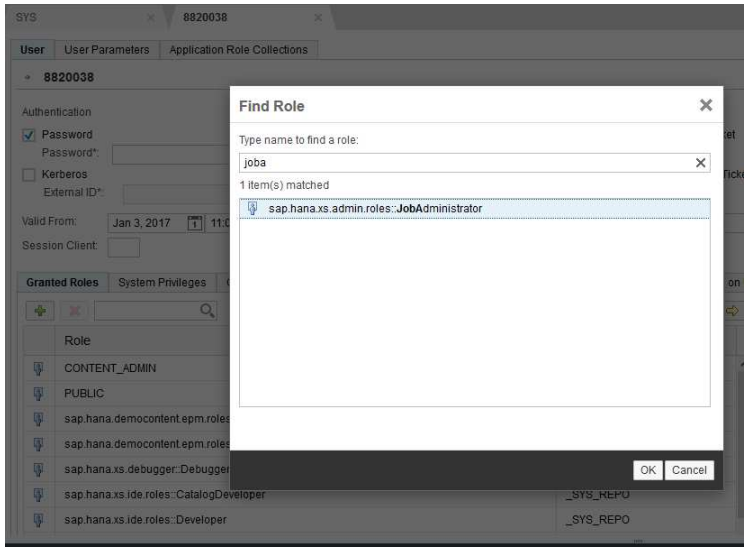
PREDICTION(?)

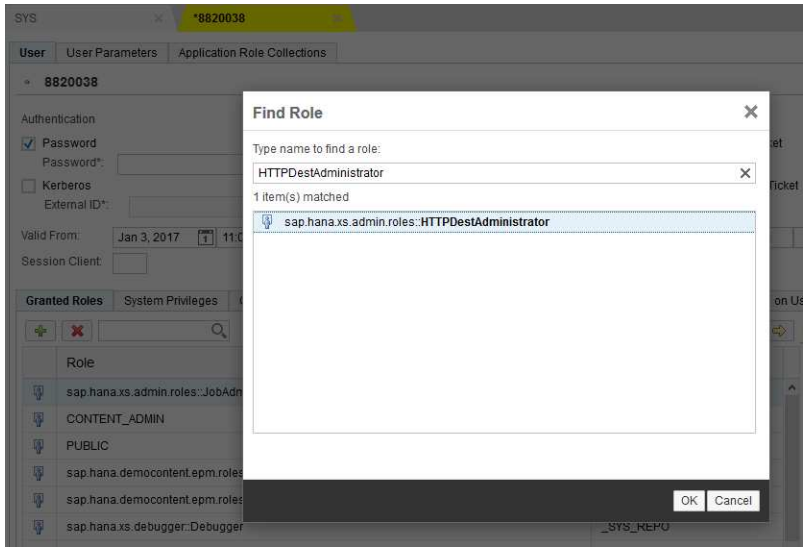
JOB SCHEDULE

- **Schedule a job** that triggers an **XS JavaScript application** that reads the latest value from the public **honeycomb service** available on the Internet.
- You also see how to **check** that the XS **job** is **working** and running on **schedule**.

Job Schedule Prerequisites

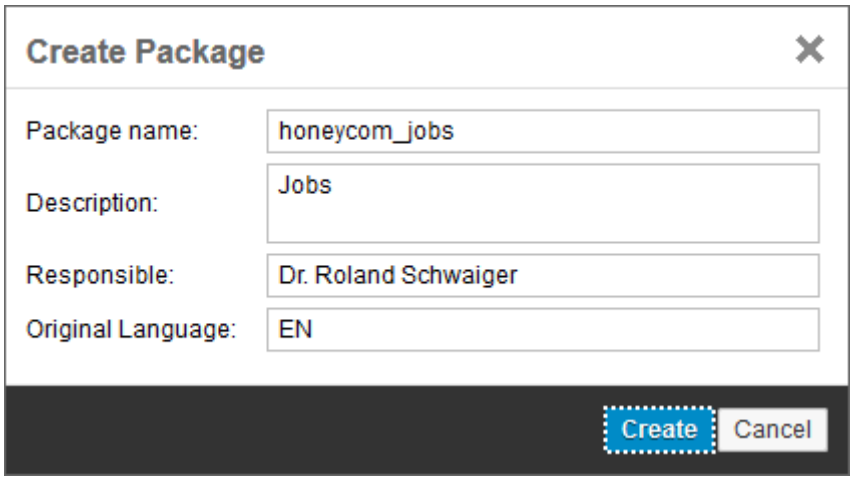
- You have the privileges granted in the SAP HANA user role sap.hana.xs.admin.roles::JobAdministrator
- You have the privileges granted in the SAP HANA user rolesap.hana.xs.admin.roles::HTTPDestAdministrator





Job Schedule

- Create new package for the job definitions



Job Schedule yahoo.xsjs

```
function readStock(input) {
    var stock = input.stock;

    var dest = $.net.http.readDestination("yahoo", "yahoo");
    var client = new $.net.http.Client();
    var req = new $.web.WebRequest($.net.http.GET, "/d/quotes.csv?f=a&s=" + stock);

    client.request(req, dest);
    var response = client.getResponse();

    var stockValue;
    if(response.body)
        stockValue = parseInt(response.body.asString(), 10);

    var sql = "INSERT INTO stock_values VALUES (NOW(), ?)";
    var conn = $.db.getConnection();
    var pstmt = conn.prepareStatement(sql);
    pstmt.setDouble(1, stockValue);
    pstmt.execute();
    conn.commit();
    conn.close();
}
```

Job Schedule

HTTP destination (sap.hana.xs.admin.roles::HTTPDestAdministrator)

yahoo.xshttpdest

host = "download.finance.yahoo.com";

port = 80;

Job Schedule
XS job file

- The XS job file uses a cron-like syntax to define a schedule at which the XS JavaScript must run
- This job file triggers the script yahoo.xsjs on the 59th second of every minute and provides the name “SAP.DE” as the parameter for the stock value to check.

Job Schedule yahoo.xsjob

```
{
    "description": "Read stock value",
    "action": "yahoo:yahoo.xsjs::readStock",
    "schedules": [
      {
        "description": "Read current stock value",
        "xscron": "* * * * * * 59",
        "parameter": {
          "stock": "SAP.DE"
          }
      }
    ]
}
```

Job Schedule

XS job's runtime configuration

(sap.hana.xs.admin.roles::JobAdministrator)

XS Job Dashboard

- Start the SAP HANA XS Administration Tool at the following URL: http://<*WebServerHost*>:80<*SAPHANAinstance*>/sap/hana/xs/admin/.

You need to specify the following details:

- User

  The user account in which the job runs, for example, SYSTEM
- Password

  The password required for user, whose account is used to run the job.
- Locale

  The language encoding required for the locale in which the job runs, for example, en_US
- Start/Stop time

  An optional value to set the period of time during which the job runs. Enter the values using the syntax used for the SAP HANA data type LocalDate and LocalTime, for example, 2014-11-05 00:30:00 (thirty minutes past midnight on the 5th of November 2014).
- Active

  Enable or disable the job schedule

Save

Job Schedule

- Enable the job-scheduling feature in SAP HANA XS
- In the XS Job Dashboard set the Scheduler Enabled toggle button to YES

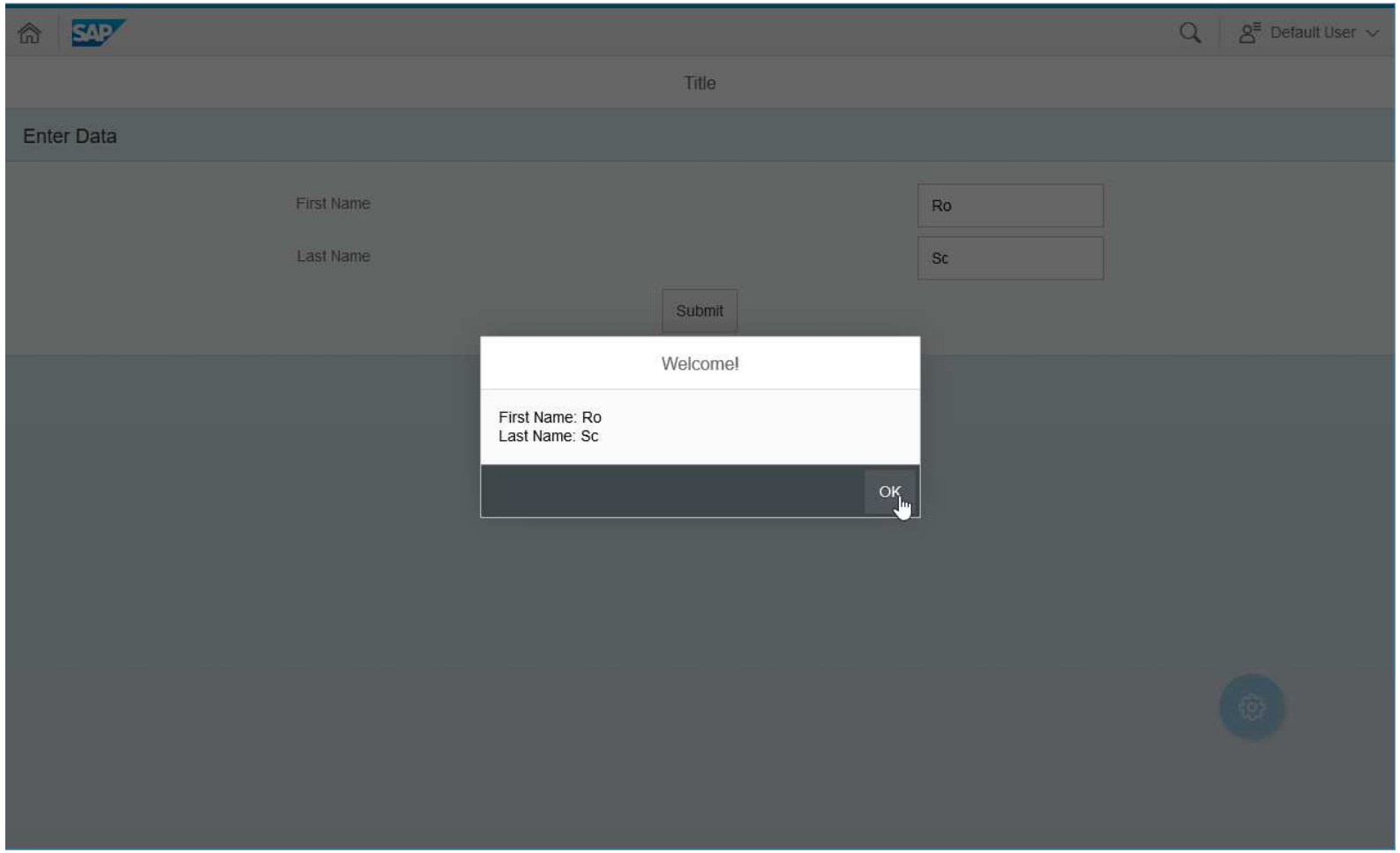
  -> Which sets the SAP HANA configuration variable: xsengine.ini -> scheduler -> enabled
- Check the job logs to ensure the XS job is active and running according to the defined schedule.You can view the xsjob logs in the XS Job Dashboard tab of the SAP HANA XS Administration Tool.

Job Schedule

Exercise
UI5

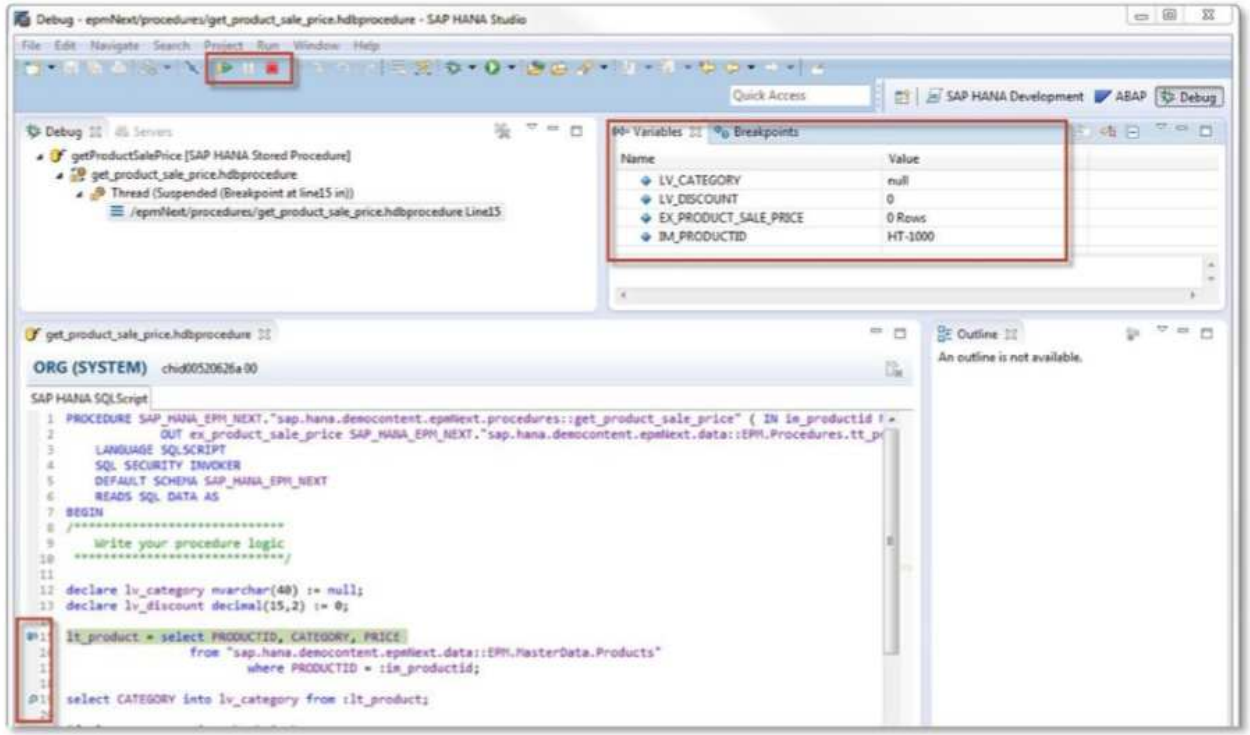
1. Create new SAPUI5 project called panel-basic with an initial XML view called **Main**.
2. Insert a **sap.m.Panel** with headerText **Enter Data**.
3. Create tree **sap.m.FlexBox** layouts with **alignItems Center and justifyContent SpaceAround**.
4. Add **sap.m.Label** objects **with text FirstName / LastName and ID idInputFirstName / idInputLastName** to the two first FlexBoxes.
5. Add the sap.m.Input objects with type text behind each.
6. Add the sap.m.Button to the last FlexBox and assign a callback method **onButtonPressed** to the press event.
7. Include the **sap.m.MessageBox** in the view controllers define array **sap/m/MessageBox** and **MessageBox** in the parameter list of the function;
8. Implement the buttons callback method **onButtonPressed**.
9. Read the entered value from the two input fields FirstName and LastName.
10. Show a message on the screen with the retrieved values.

Exercise
UI5



SQLScript
Debugging in HANA Dev Studio

- Resume/Terminate
- Variable evaluation
- Breakpoint management
- Break on break-points
- Basic step debugging



We are not using HANA Dev Studio

Exercise
SQLScript & oData

- (optional) First create a table PRODUCT using SQLScript in the SQL console of the catalog view
- Create an oData validation exit based on SQLScript for the update operation
  - (optional) honeycomb_data:product.hdbtable
  - Create a table type honeycomb_data:error.hdbtabletype

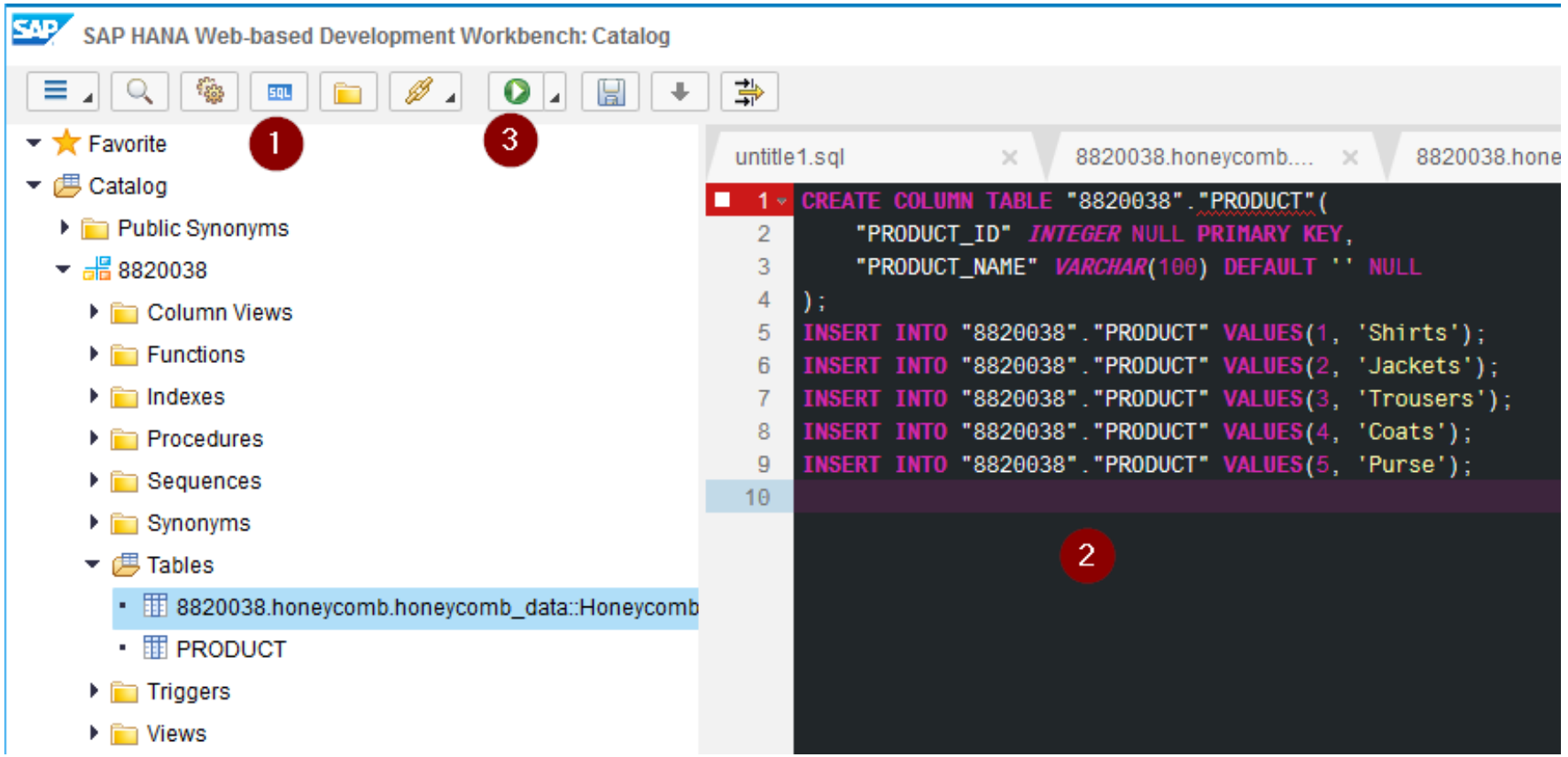
Hint
SQLScript & oData

```
--REPLACE <YOUR SCHEMA> WITH YOUR SCHEMA NAME

-- Create Product table
create column table "<YOUR SCHEMA>"."PRODUCT"(
      "PRODUCT_ID" INTEGER null primary key,
      "PRODUCT_NAME" VARCHAR (100) null default ''
);

insert into "<YOUR SCHEMA>"."PRODUCT" values(1,'Shirts');
insert into "<YOUR SCHEMA>"."PRODUCT" values(2,'Jackets');
insert into "<YOUR SCHEMA>"."PRODUCT" values(3,'Trousers');
insert into "<YOUR SCHEMA>"."PRODUCT" values(4,'Coats');
insert into "<YOUR SCHEMA>"."PRODUCT" values(5,'Purse');
```

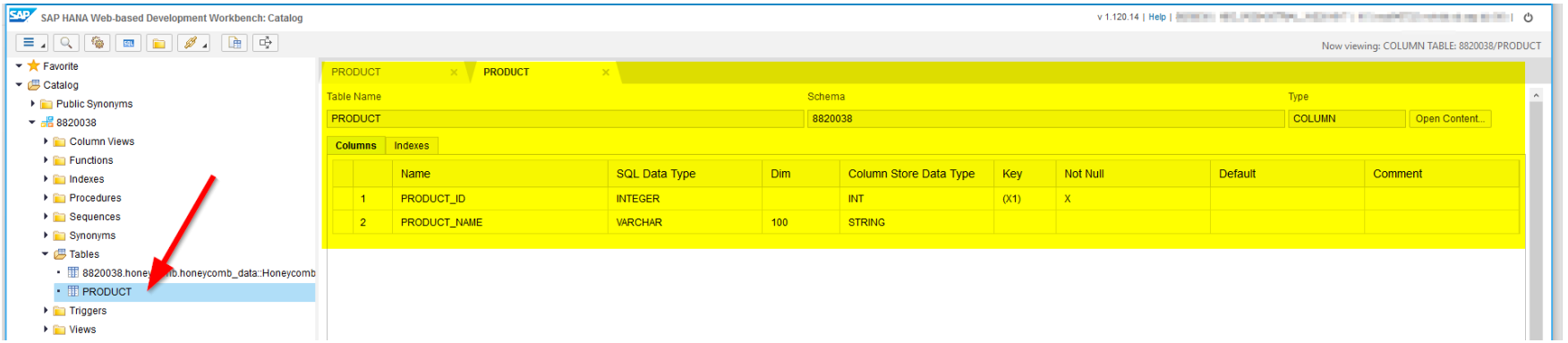
Hint
SQLScript & oData

```
table.schemaName = "8820038";
table.tableType = COLUMNSTORE;
table.columns = [
    {name = "PRODUCT_ID"; sqlType = INTEGER; nullable = false;},
    {name = "PRODUCT_NAME"; sqlType = NVARCHAR; nullable =
true; length = 100;
defaultValue = "";}];
table.indexes =  [
    {name = "MYINDEX1"; unique = true; indexColumns =
["PRODUCT_NAME"];}];
table.primaryKey.pkcolumns = ["PRODUCT_ID"];
```

Solution
SQLScript & oData



Solution
SQLScript & oData



Solution
SQLScript & oData

product.hdbtable

```
1  table.schemaName = "8820038";
2  table.tableType = COLUMNSTORE;
3  table.columns = [
4      {name = "PRODUCT_ID"; sqlType = INTEGER; nullable = false;},
5      {name = "PRODUCT_NAME"; sqlType = NVARCHAR; nullable = true; length = 100;
6  defaultValue = "";}];
7  table.indexes =  [
8      {name = "MYINDEX1"; unique = true; indexColumns = ["PRODUCT_NAME"];}];
9  table.primaryKey.pkcolumns = ["PRODUCT_ID"];
```

Solution
SQLScript & oData

File error.hdbdd

```
namespace "8820038".honeycomb.honeycomb_data;
@Schema: '8820038‘
context error {
        type error{
        "HTTP_STATUS_CODE": Integer;
        "ERROR_MESSAGE": String(100);
        "DETAIL": String(100);
        };
};
```

Solution
SQLScript & oData

File sample.odata:beforeupdate.hdbprocedure

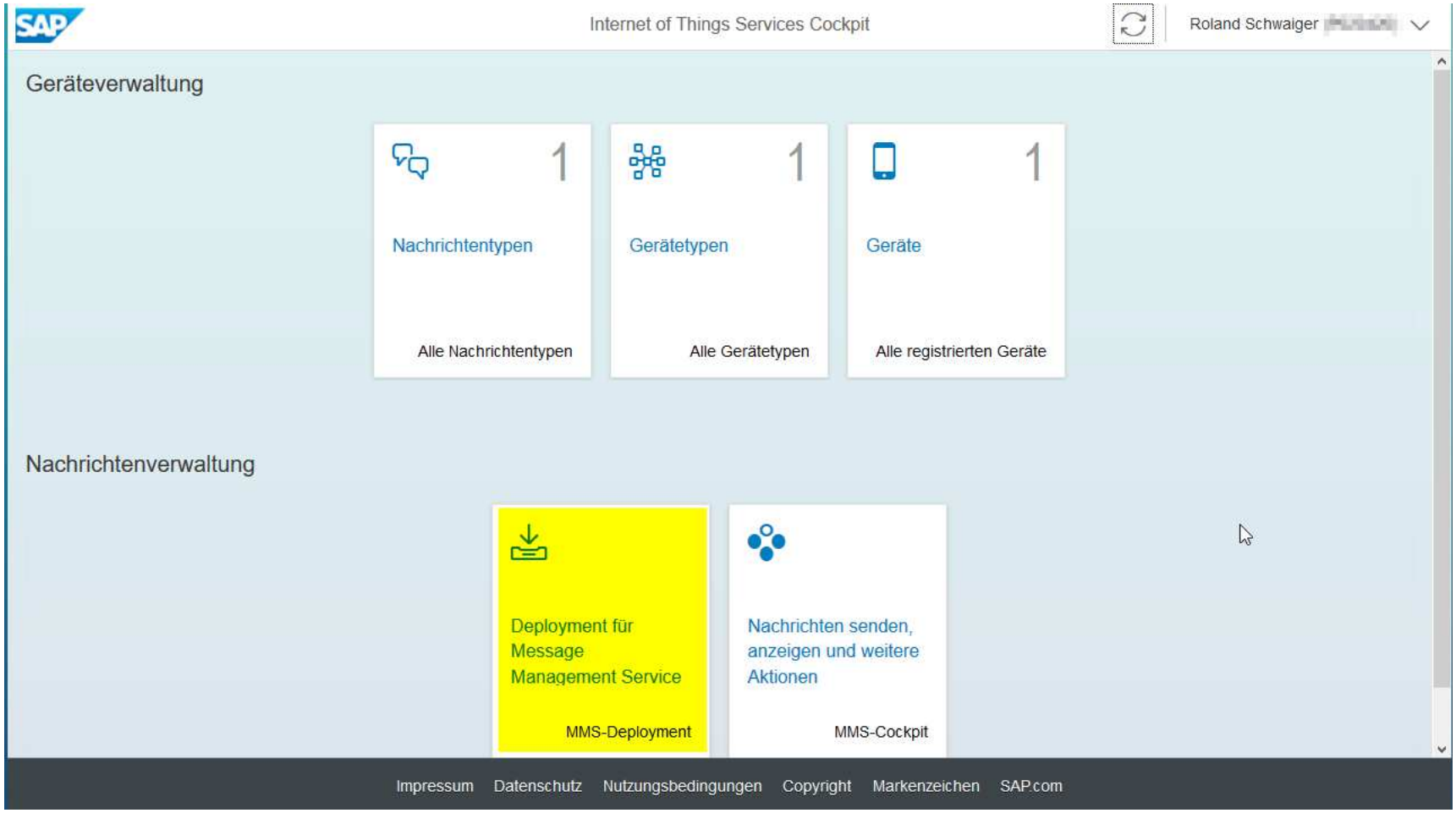
```
procedure "sample.odata::beforeupdate"
    (IN new "sample.odata::table", IN old "sample.odata::table", OUT error
"sample.odata::error")
language sqlscript
sql security invoker as
    idnew INT;
    idold INT;
begin
    select ID into idnew from :new;
    select ID into idold from :old;
if :idnew <= :idold then
error = select 400 as http_status_code,
    'invalid ID' error_message,
    'the new value must be larger than the previous' detail from dummy;
end if;
end;
```

Solution
SQLScript & oData

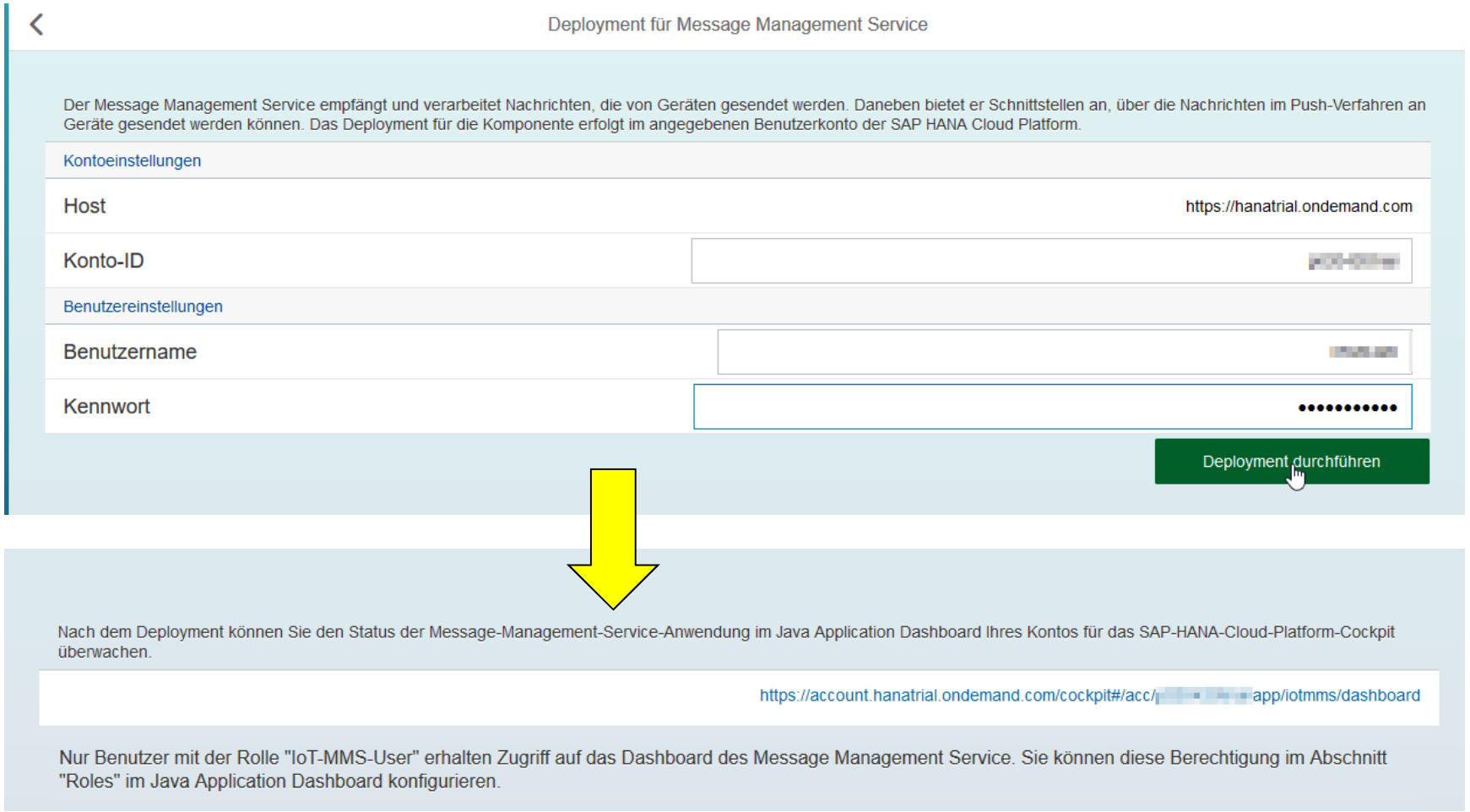
```
service {
    “sample.odata::table”
        update events (before
“sample.odata::beforeupdate”);
}
```

IOT INTEGRATION (?)

IoT
Deploy

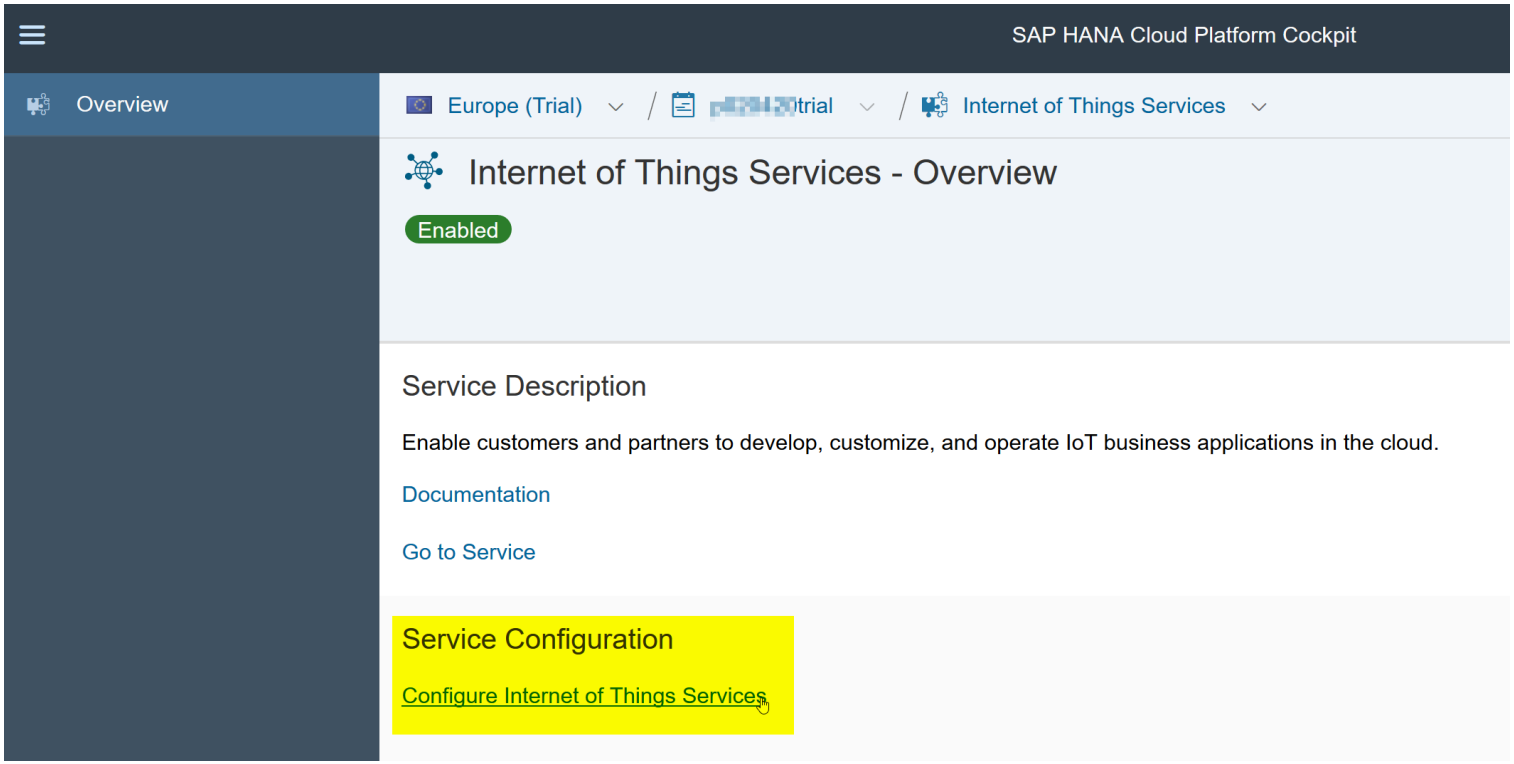

Internet of Things Services Cockpit
Roland Schwaiger
Geräteverwaltung
1
Nachrichtentypen
Alle Nachrichtentypen
1
Gerätetypen
Alle Gerätetypen
1
Geräte
Alle registrierten Geräte
Nachrichtenverwaltung
Deployment für Message Management Service
MMS-Deployment
Nachrichten senden, anzeigen und weitere Aktionen
MMS-Cockpit
Impressum
Datenschutz
Nutzungsbedingungen
Copyright
Markenzeichen
SAP.com

IoT
Deploy


Deployment für Message Management Service
Der Message Management Service empfängt und verarbeitet Nachrichten, die von Geräten gesendet werden. Daneben bietet er Schnittstellen an, über die Nachrichten im Push-Verfahren an Geräte gesendet werden können. Das Deployment für die Komponente erfolgt im angegebenen Benutzerkonto der SAP HANA Cloud Platform.
Kontoeinstellungen
Host
https://hanatrial.ondemand.com
Konto-ID
Benutzereinstellungen
Benutzername
Kennwort
Deployment durchführen
Nach dem Deployment können Sie den Status der Message-Management-Service-Anwendung im Java Application Dashboard Ihres Kontos für das SAP-HANA-Cloud-Platform-Cockpit überwachen.
https://account.hanatrial.ondemand.com/cockpit#/acc/ app/iotmms/dashboard
Nur Benutzer mit der Rolle "IoT-MMS-User" erhalten Zugriff auf das Dashboard des Message Management Service. Sie können diese Berechtigung im Abschnitt "Roles" im Java Application Dashboard konfigurieren.

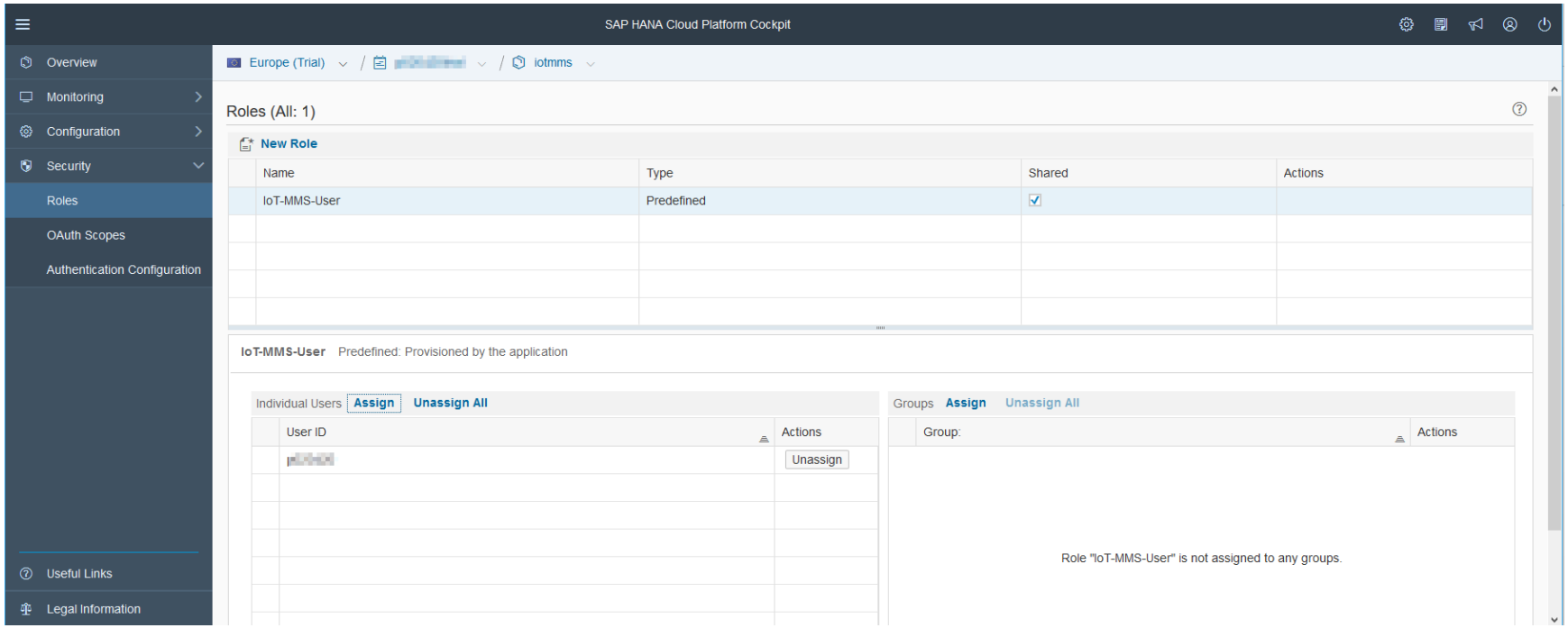
IoT
Authorization 1/2

- Within the IoT Service: Configure …

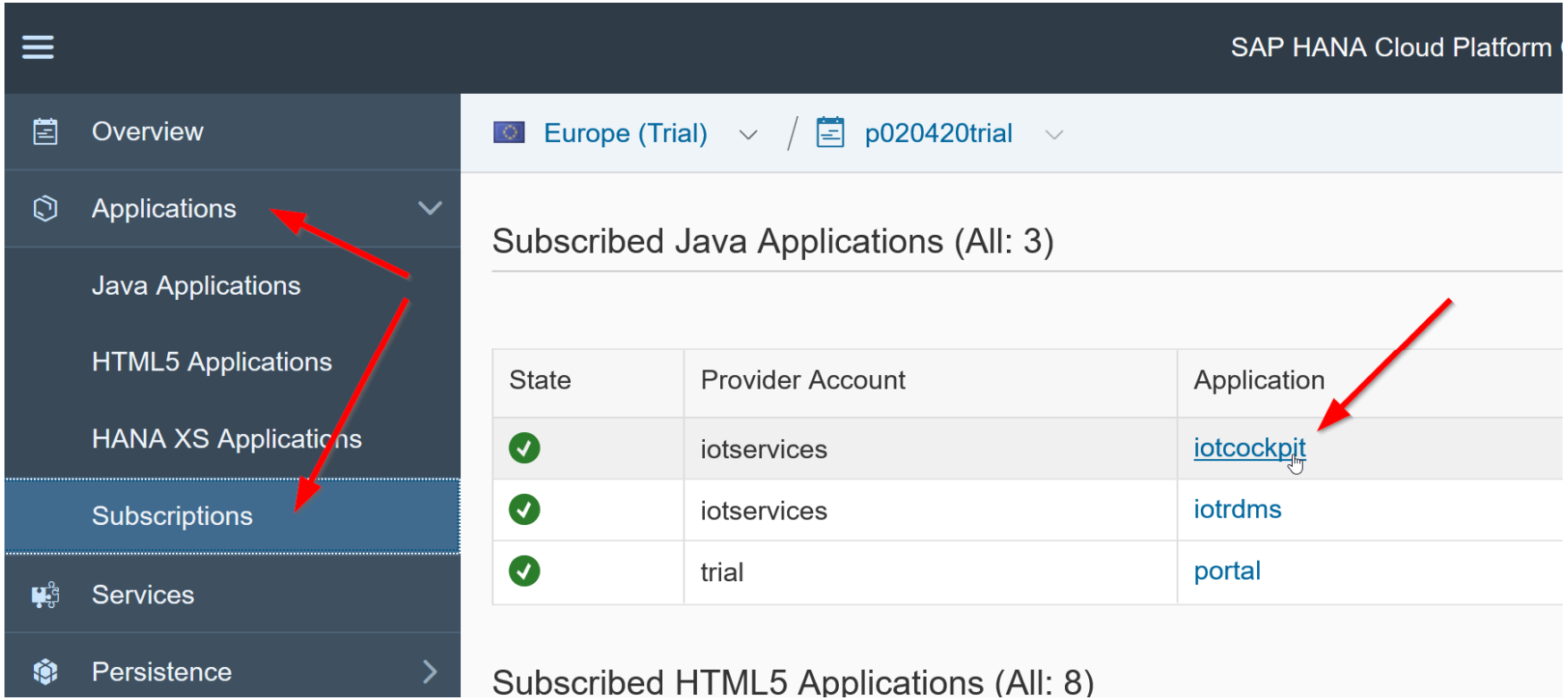


IoT
Authorization 2/2

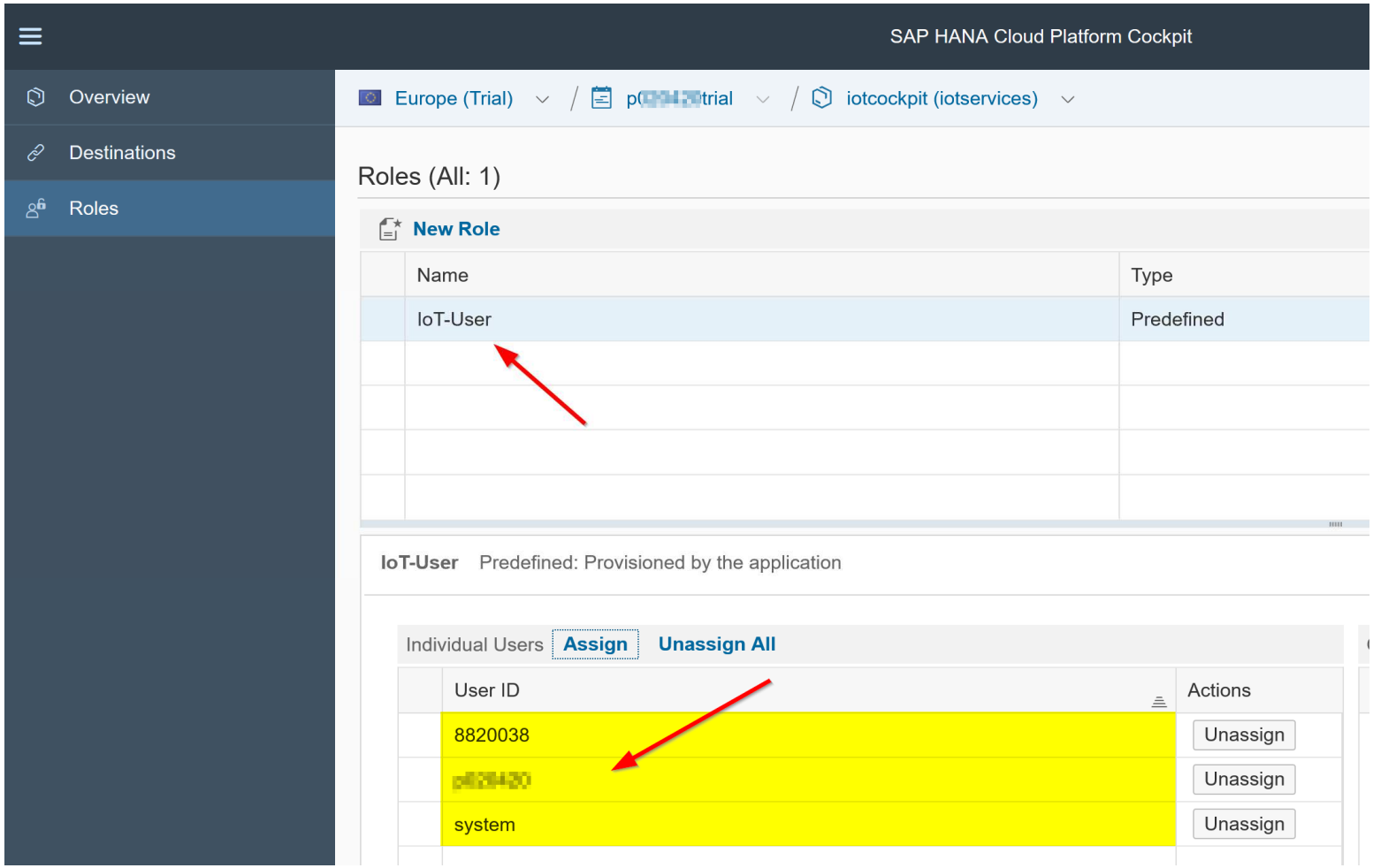
- Assign the role



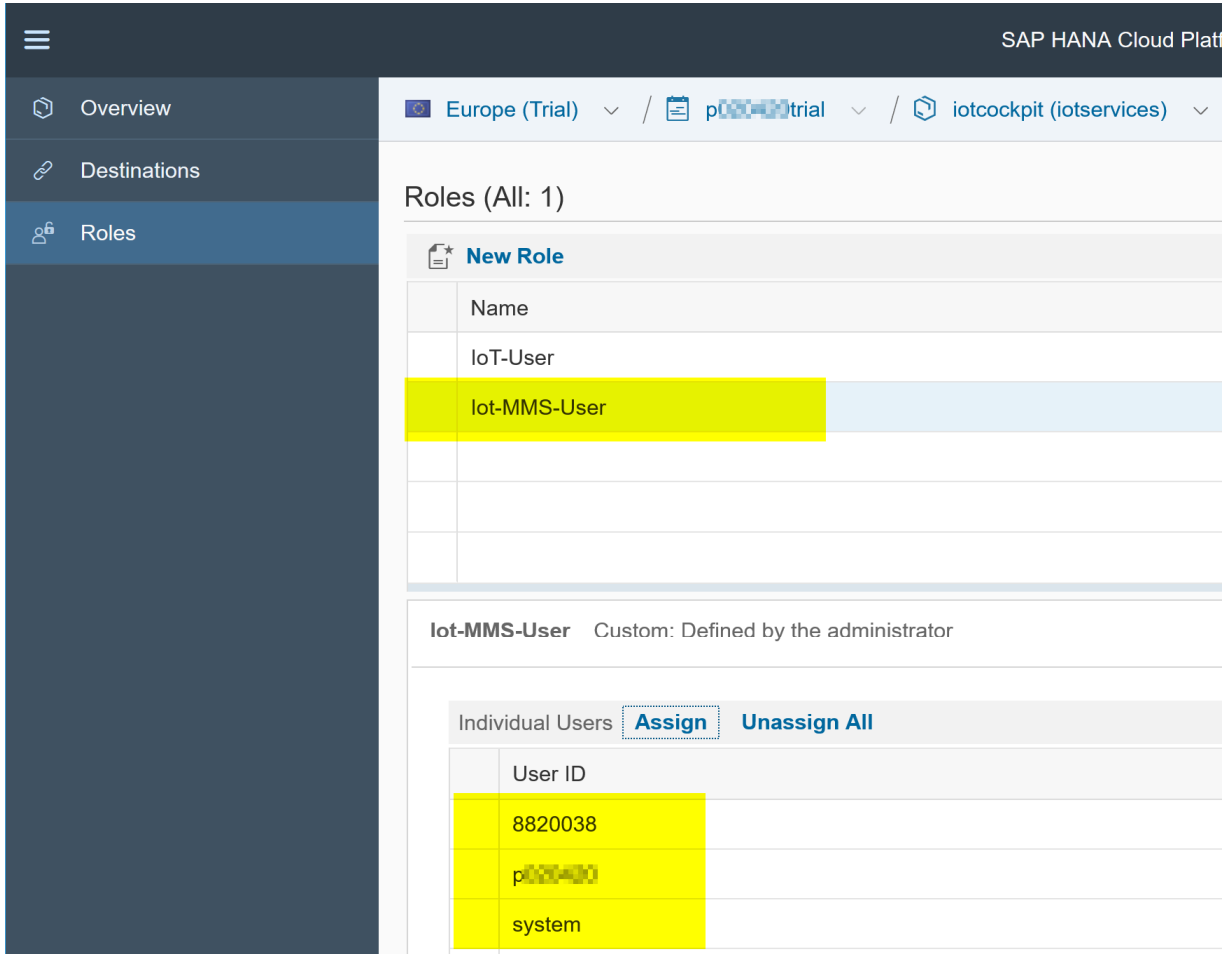
IoT Authorization



IoT Authorization

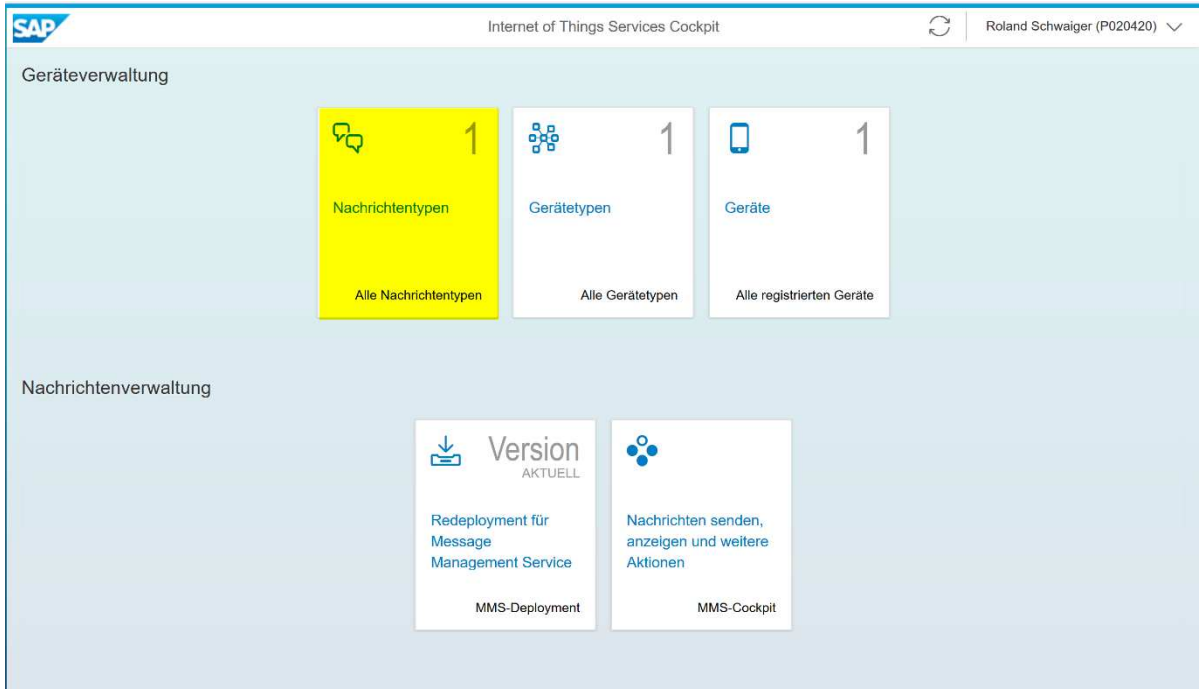


IoT Authorization

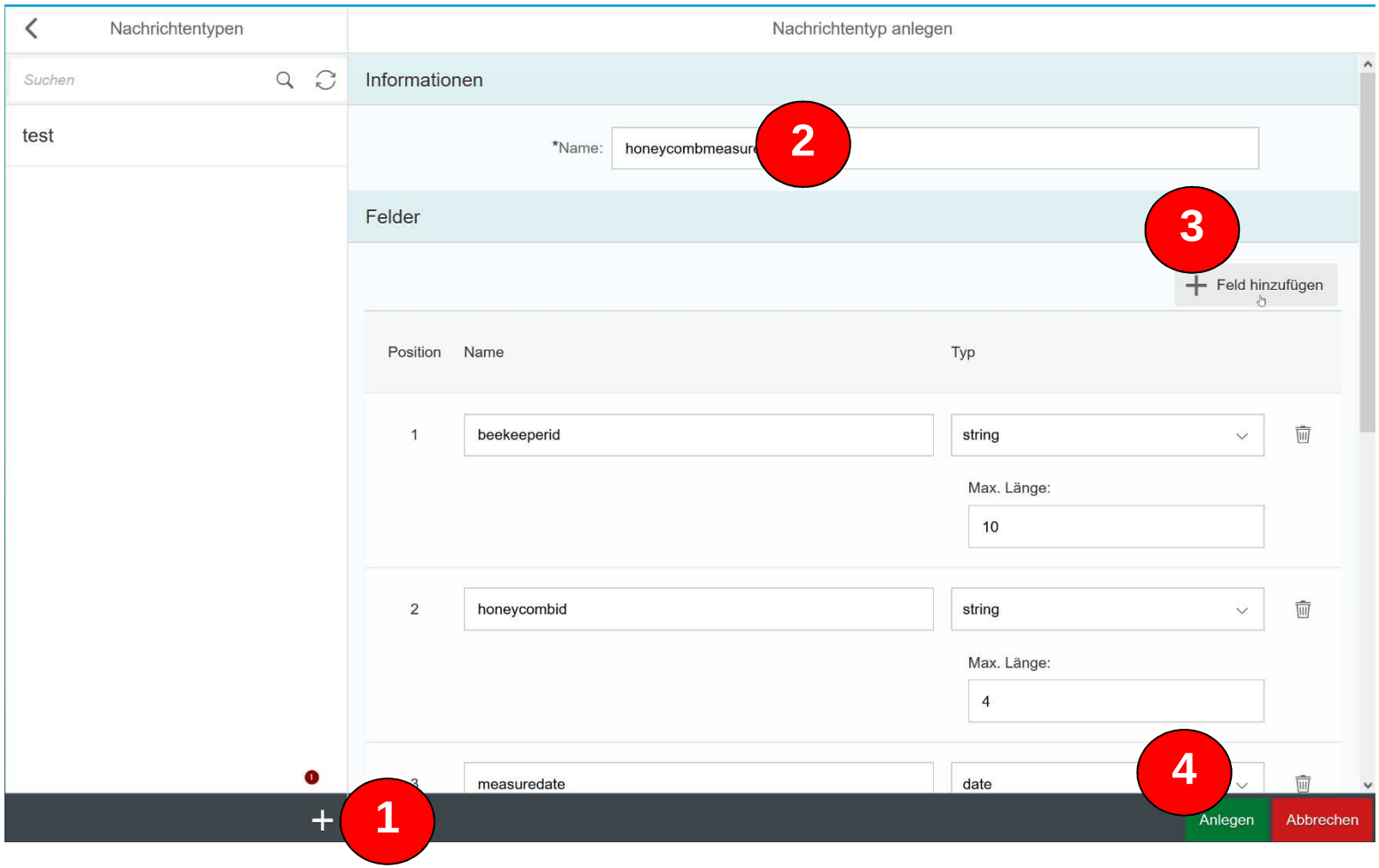


IoT Service Cockpit Message Type

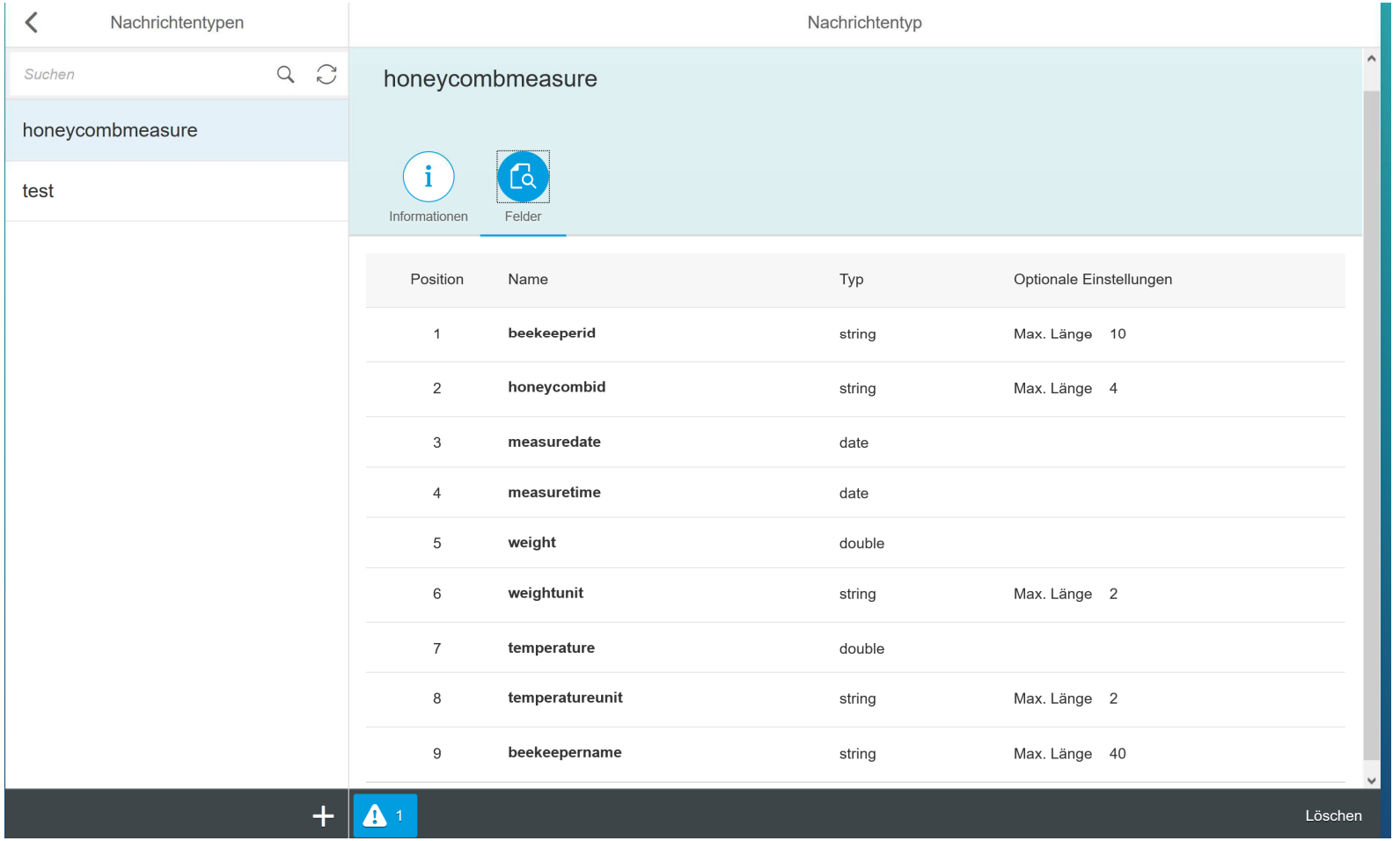
- Hana Account -> Services -> IoT Service -> Go to Service



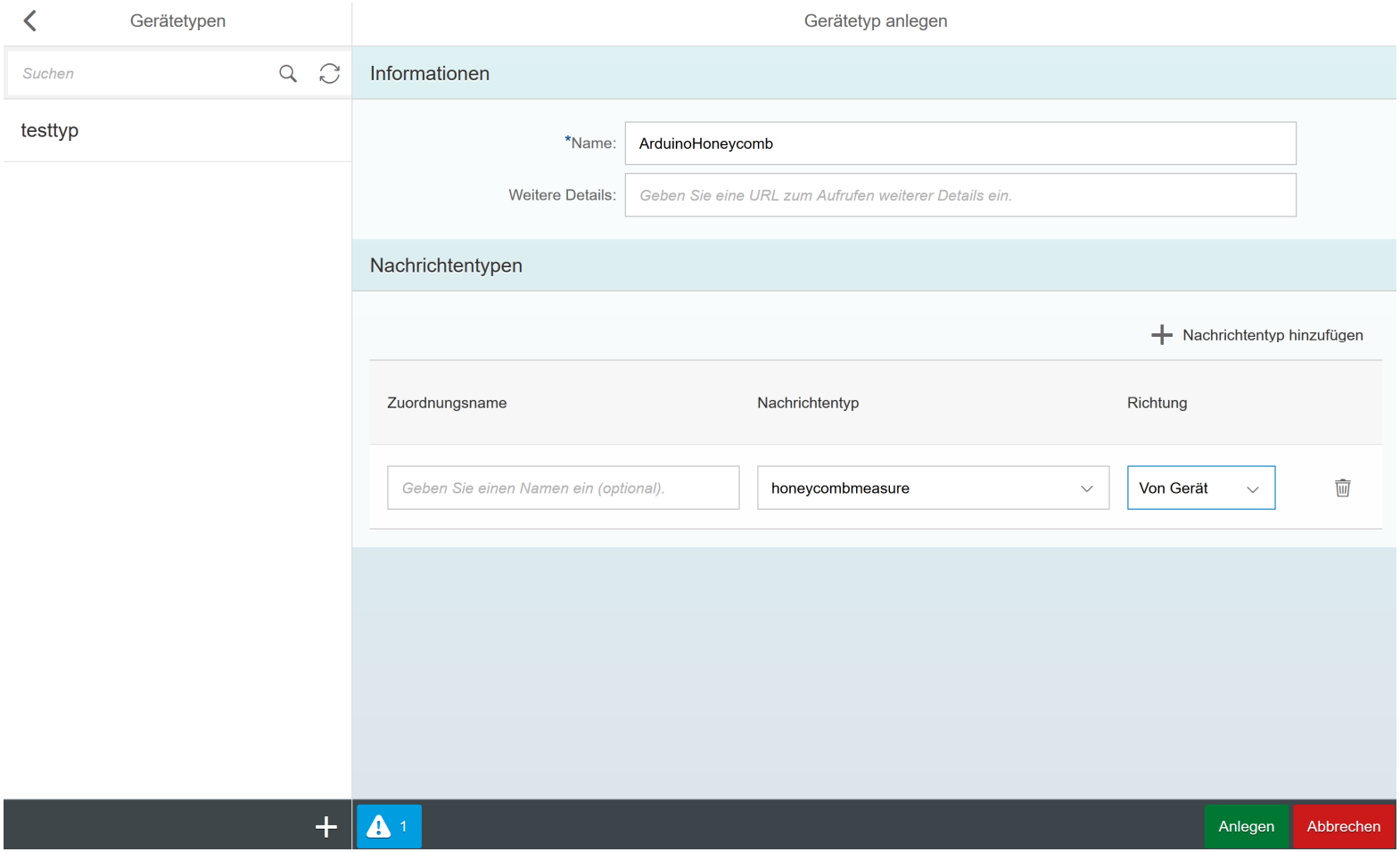
IoT Service Cockpit
Nachrichtentyp anlegen



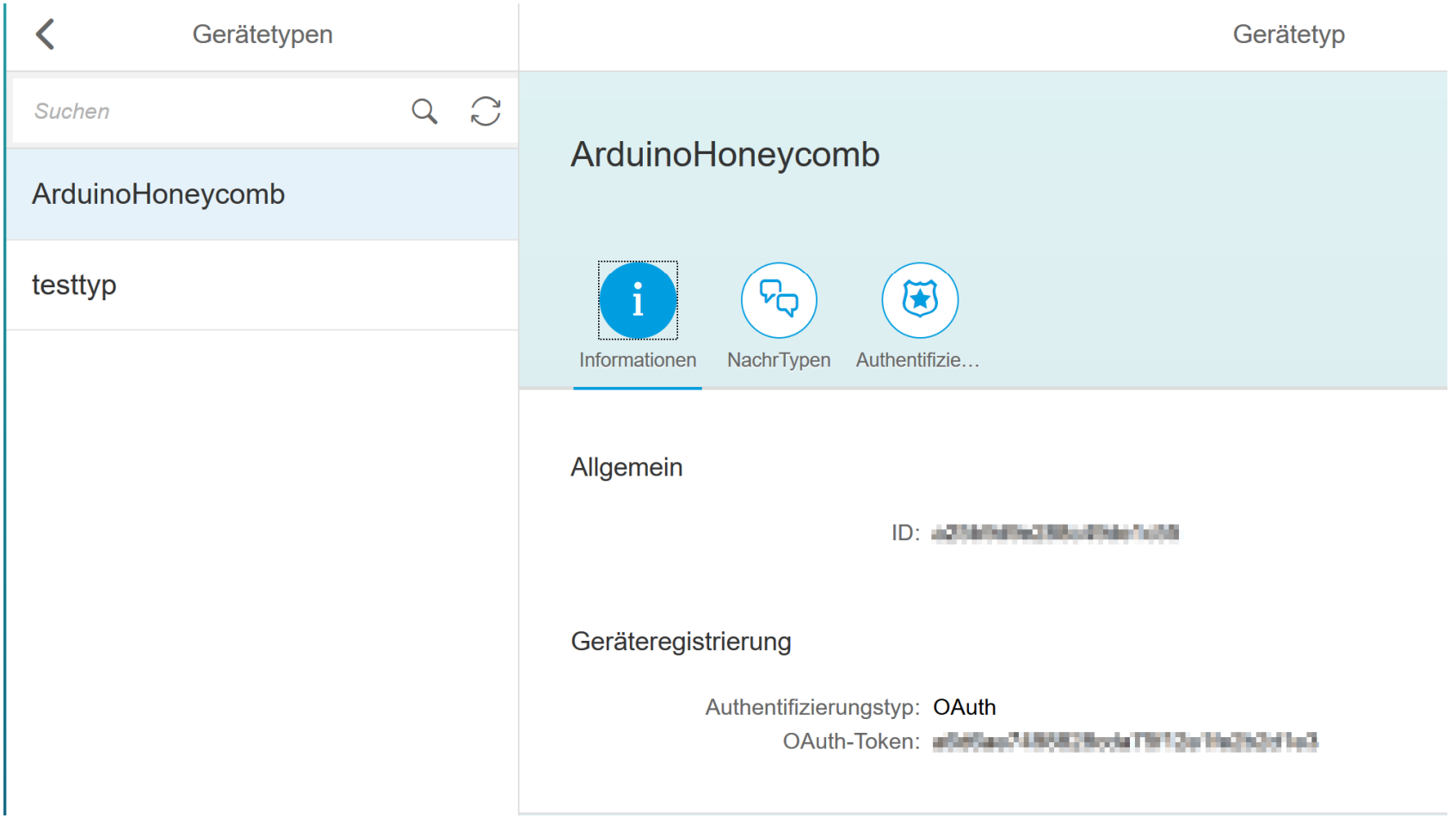
IoT Service Cockpit
Nachrichtentyp anlegen



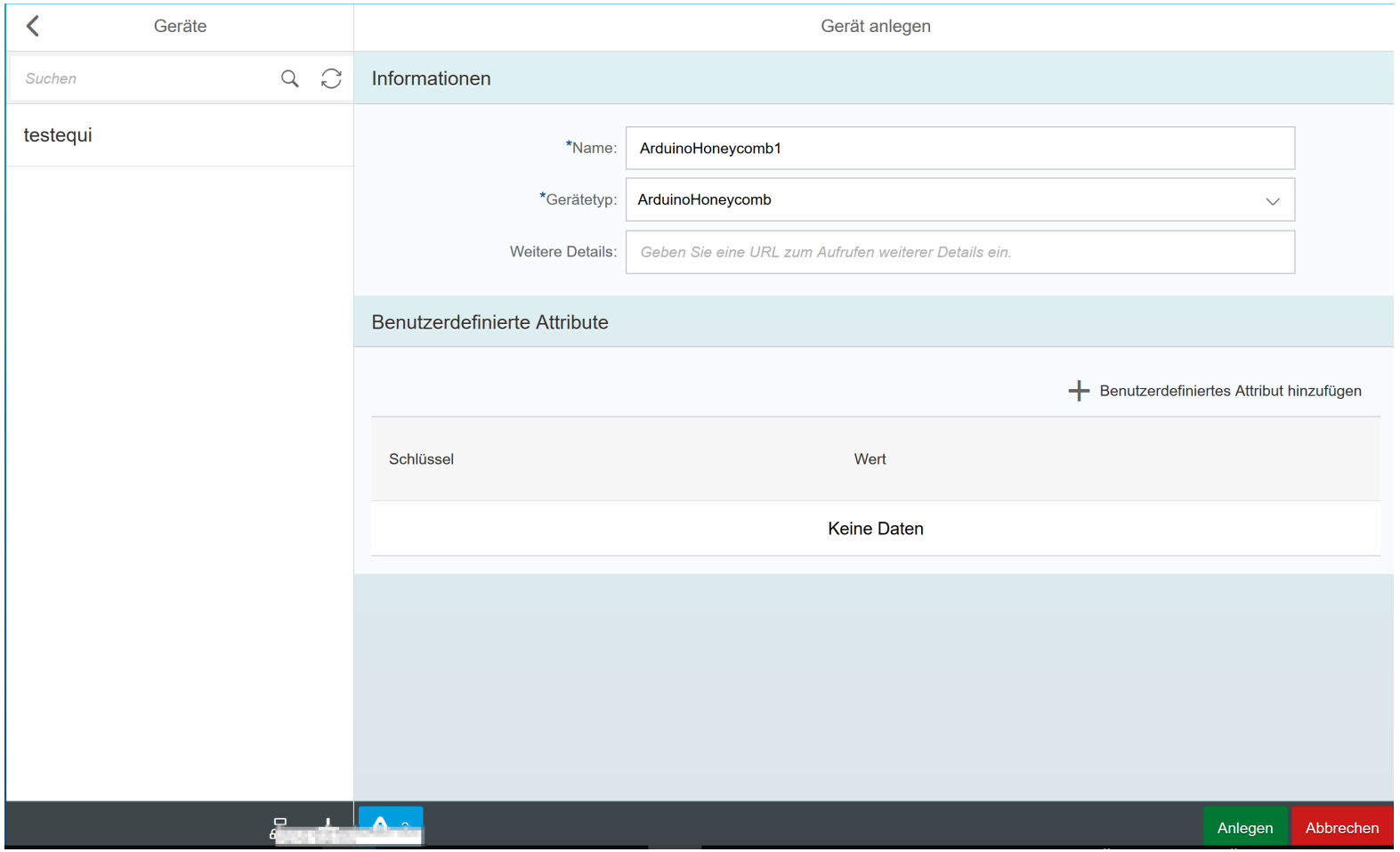
IoT Service Cockpit
Gerätetyp anlegen



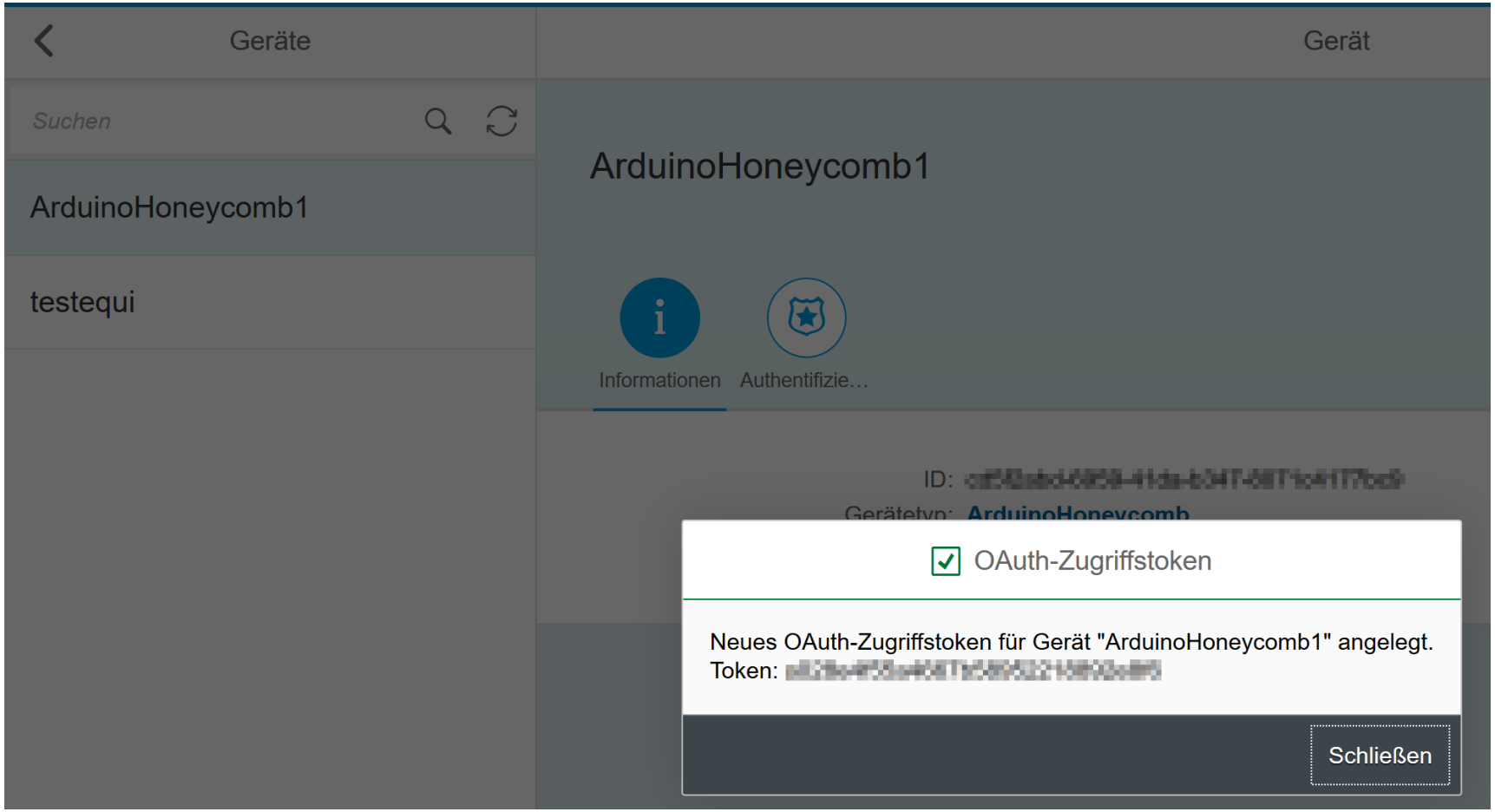
IoT Service Cockpit
Gerätetyp anlegen



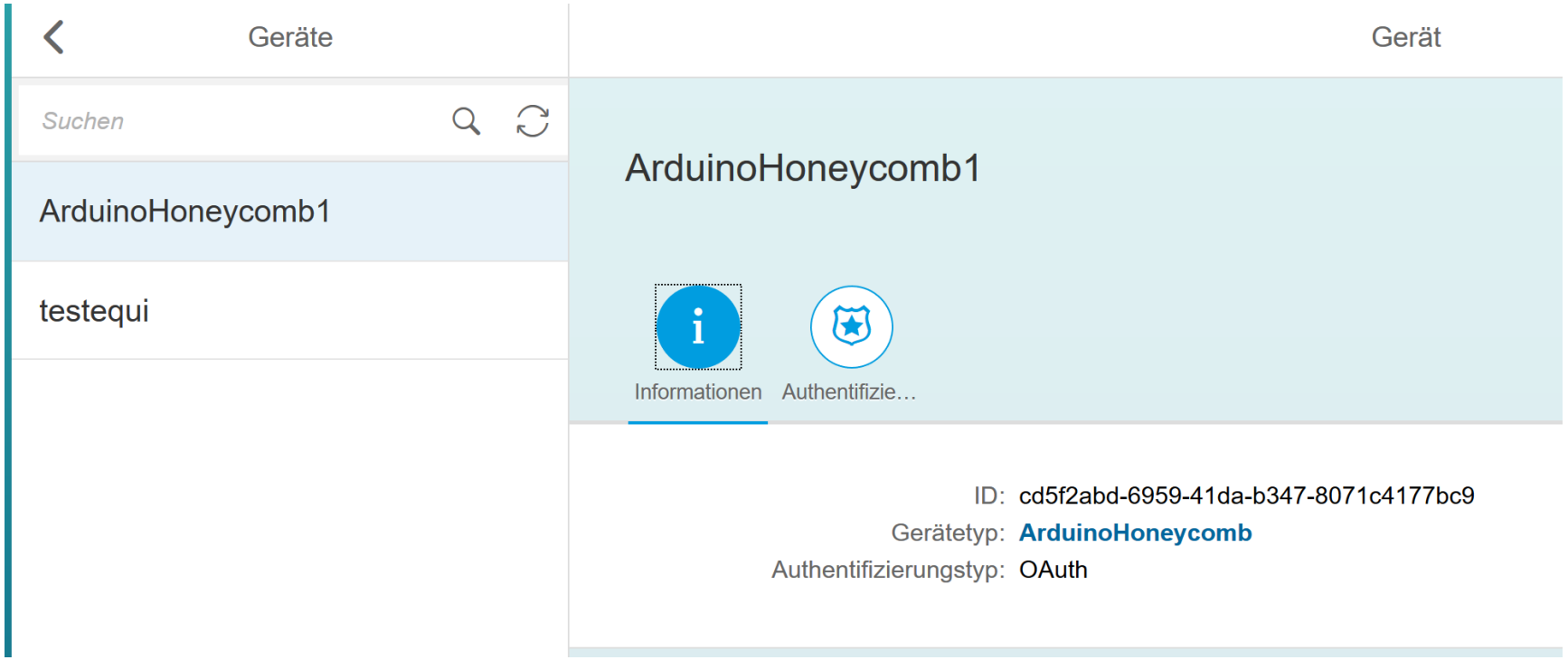
IoT Service Cockpit
Gerät anlegen



IoT Service Cockpit
Gerät anlegen

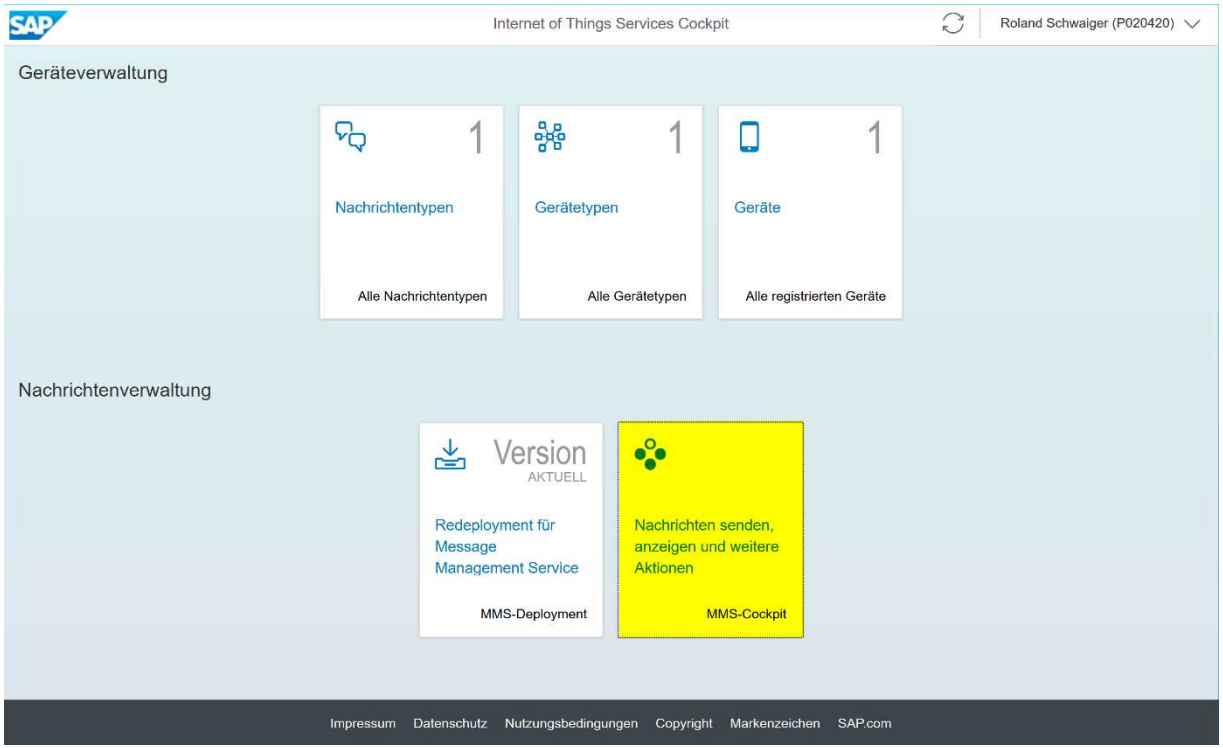


IoT Service Cockpit
Gerät anlegen



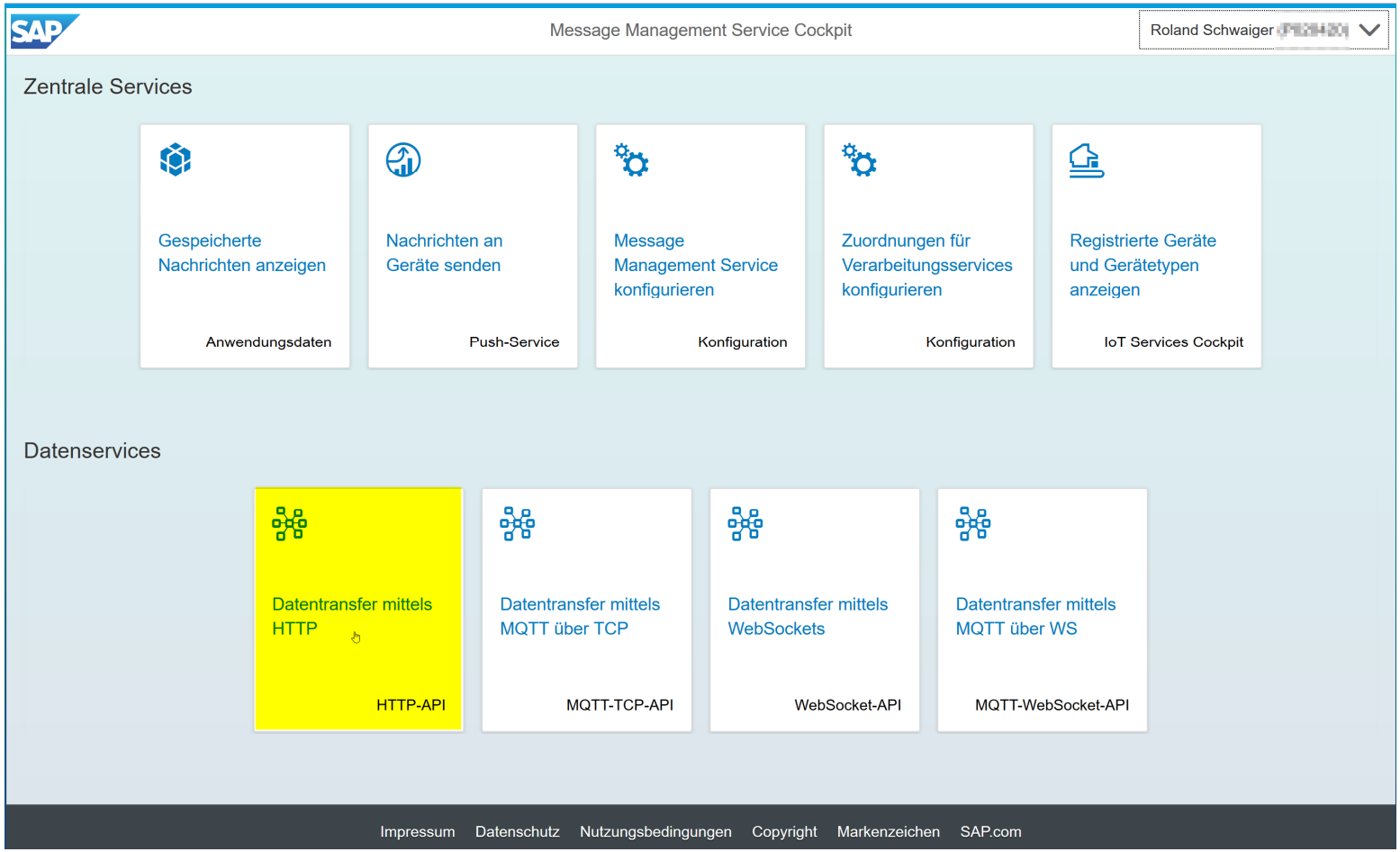
IoT Service Cockpit
Find Details for Arduino Sketch

- Hana Account -> Services -> IoT Service -> Go to Service



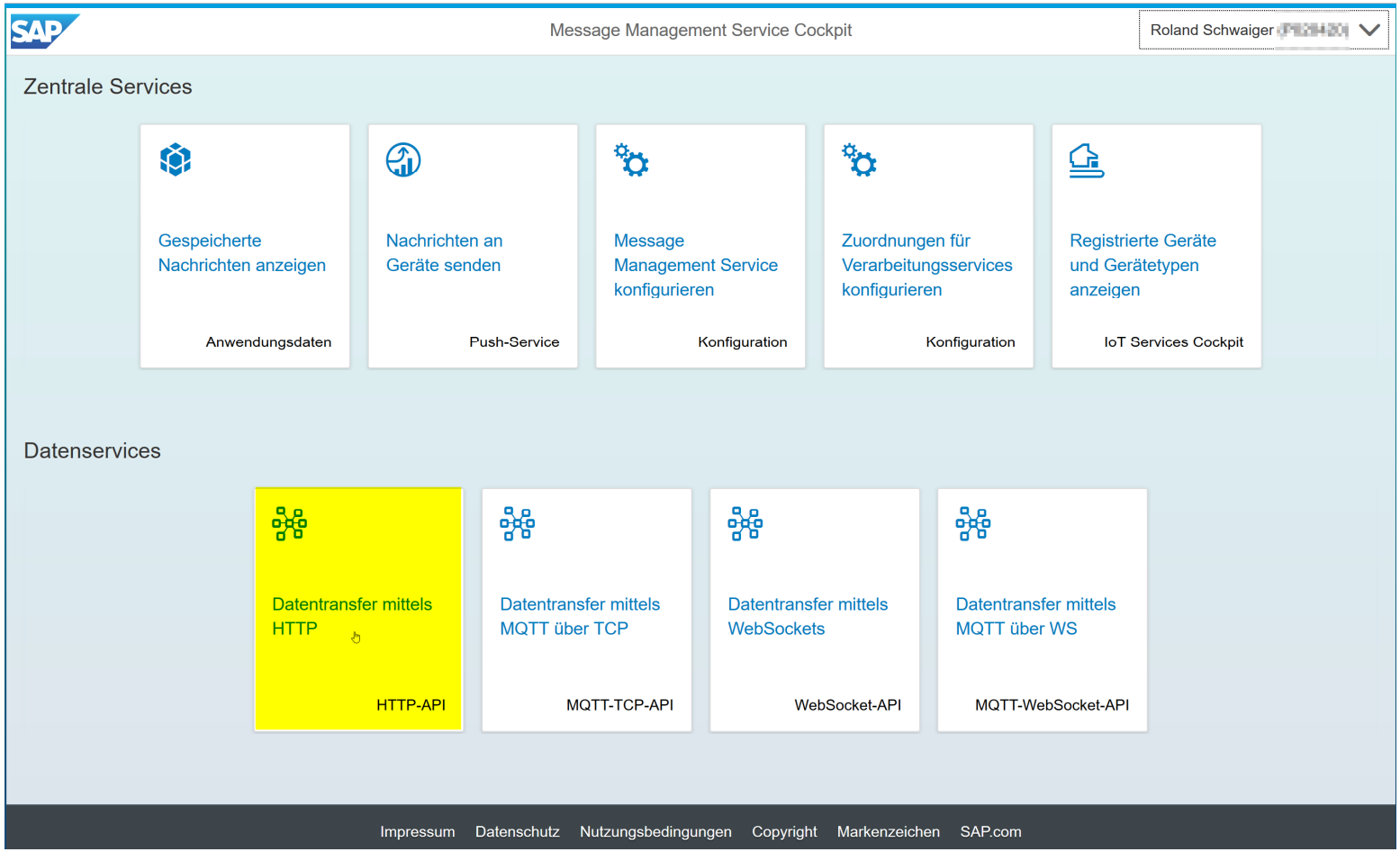
Message Management Service Cockpit

Find Details for Arduino Sketch

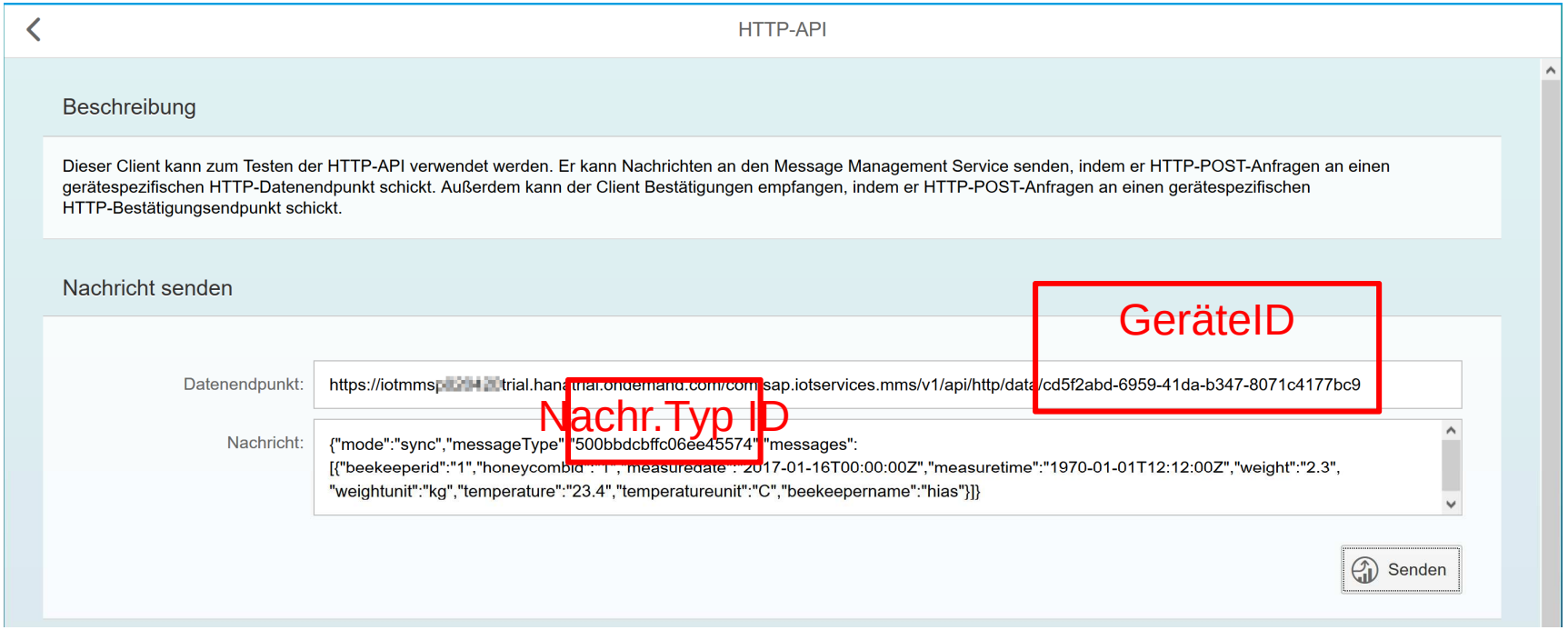


Message Management Service Cockpit

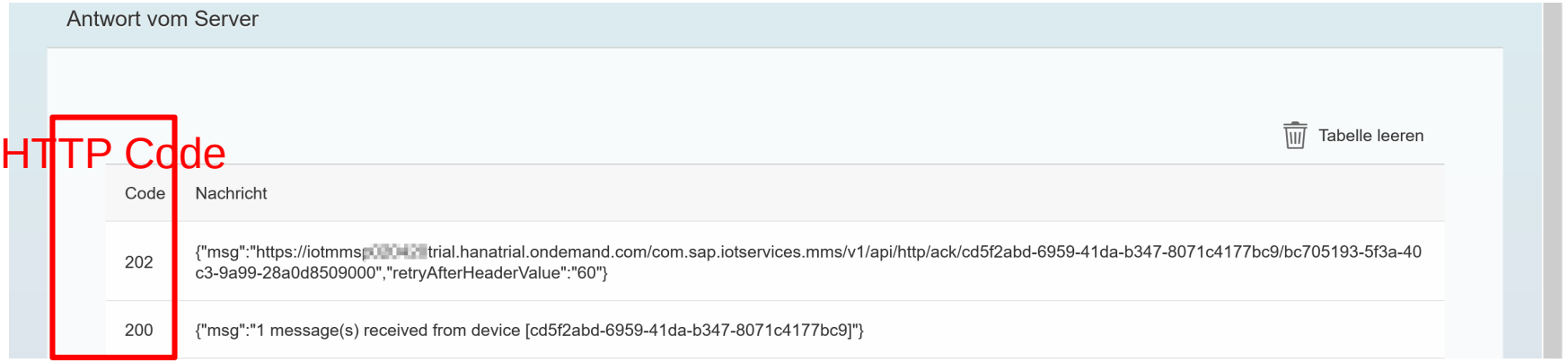
Find Details for Arduino Sketch



HTTP-API Request



HTTP-API Response



HTTP-API

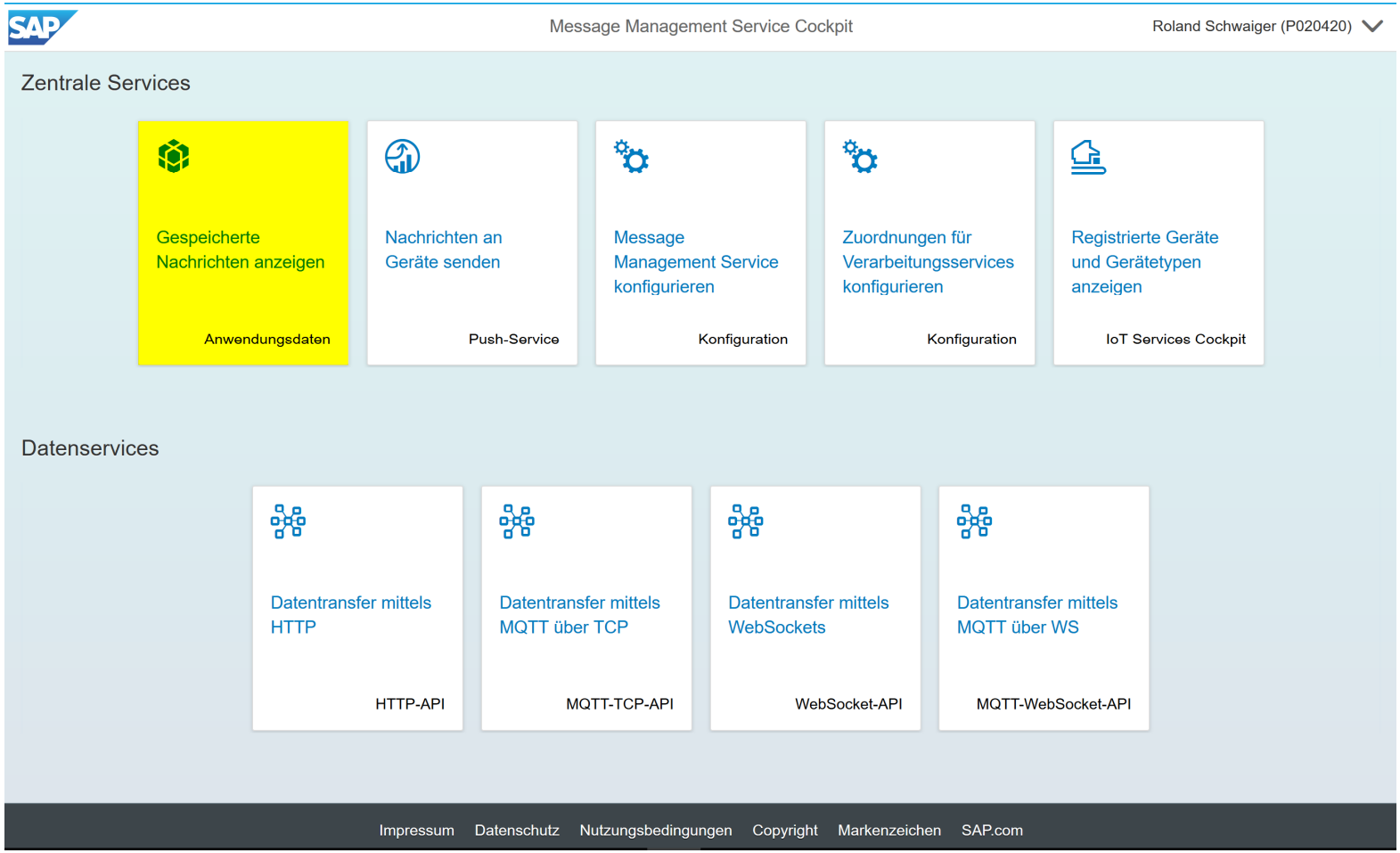
- GeräteID
- NachrichtentypID
- Nachrichtentyp Felder
- Date = Timestamps and dates can be sent as ISO 8601 compatible string or in Unix time, represented as seconds since January 1, 1970. (For example 2016-01-14T08:00:00Z).
-

Test:

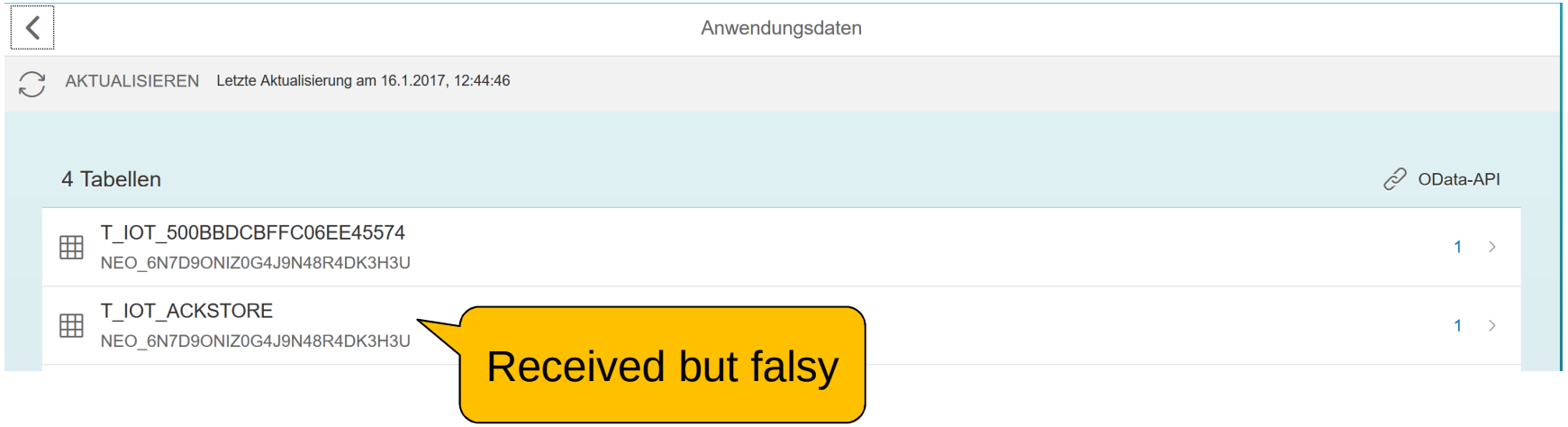
- https://iotmms<User>trial.hanatrial.ondemand.com/com.sap.iotservices.mms/v1/api/http/data/cd5f2abd-6959-41da-b347-8071c4177bc9
- {"mode":"sync","messageType":"500bbdcbffc06ee45574","messages":[{"beekeeperid":"1","honeycombid":"1","measuredate":"2017-01-16T00:00:00Z","measuretime":„1970-01-01T12:12:00Z","weight":"2.3", "weightunit":"kg","temperature":"23.4","temperatureunit":"C","beekeepername":"hias"}]}

Response:

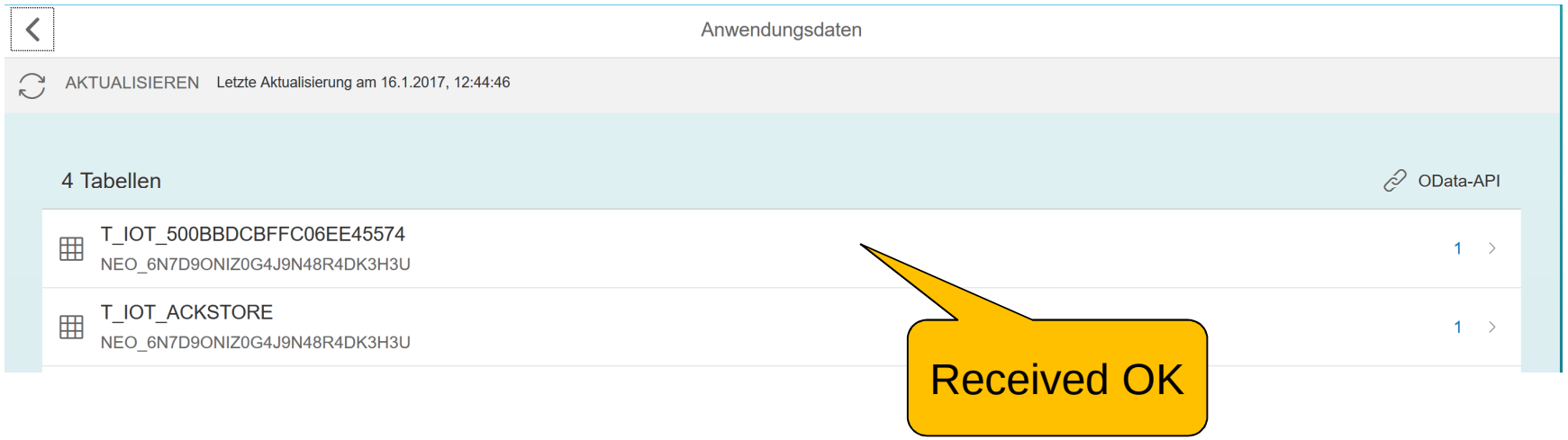
- 202
- {"msg":"https://iotmms<User>trial.hanatrial.ondemand.com/com.sap.iotservices.mms/v1/api/http/ack/cd5f2abd-6959-41da-b347-8071c4177bc9/bc705193-5f3a-40c3-9a99-28a0d8509000","retryAfterHeaderValue":"60"}

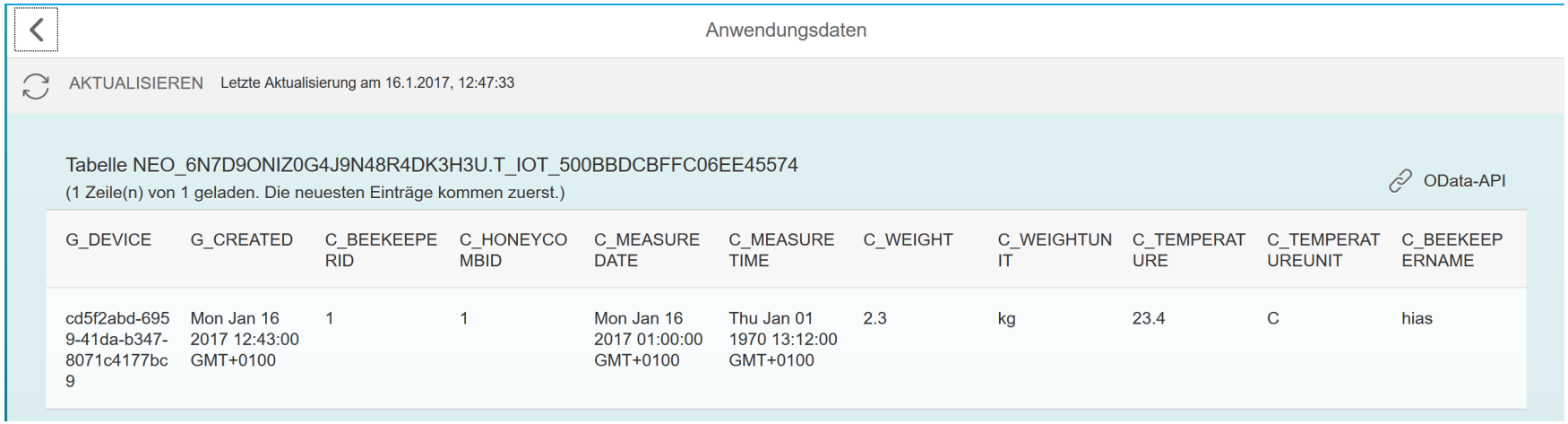

SAP
Message Management Service Cockpit
Roland Schwaiger (P020420)
Zentrale Services
Gespeicherte Nachrichten anzeigen
Anwendungsdaten
Nachrichten an Geräte senden
Push-Service
Message Management Service konfigurieren
Konfiguration
Zuordnungen für Verarbeitungsservices konfigurieren
Konfiguration
Registrierte Geräte und Gerätetypen anzeigen
IoT Services Cockpit
Datenservices
Datentransfer mittels HTTP
HTTP-API
Datentransfer mittels MQTT über TCP
MQTT-TCP-API
Datentransfer mittels WebSockets
WebSocket-API
Datentransfer mittels MQTT über WS
MQTT-WebSocket-API
Impressum
Datenschutz
Nutzungsbedingungen
Copyright
Markenzeichen
SAP.com

HTTP-API
Test Result



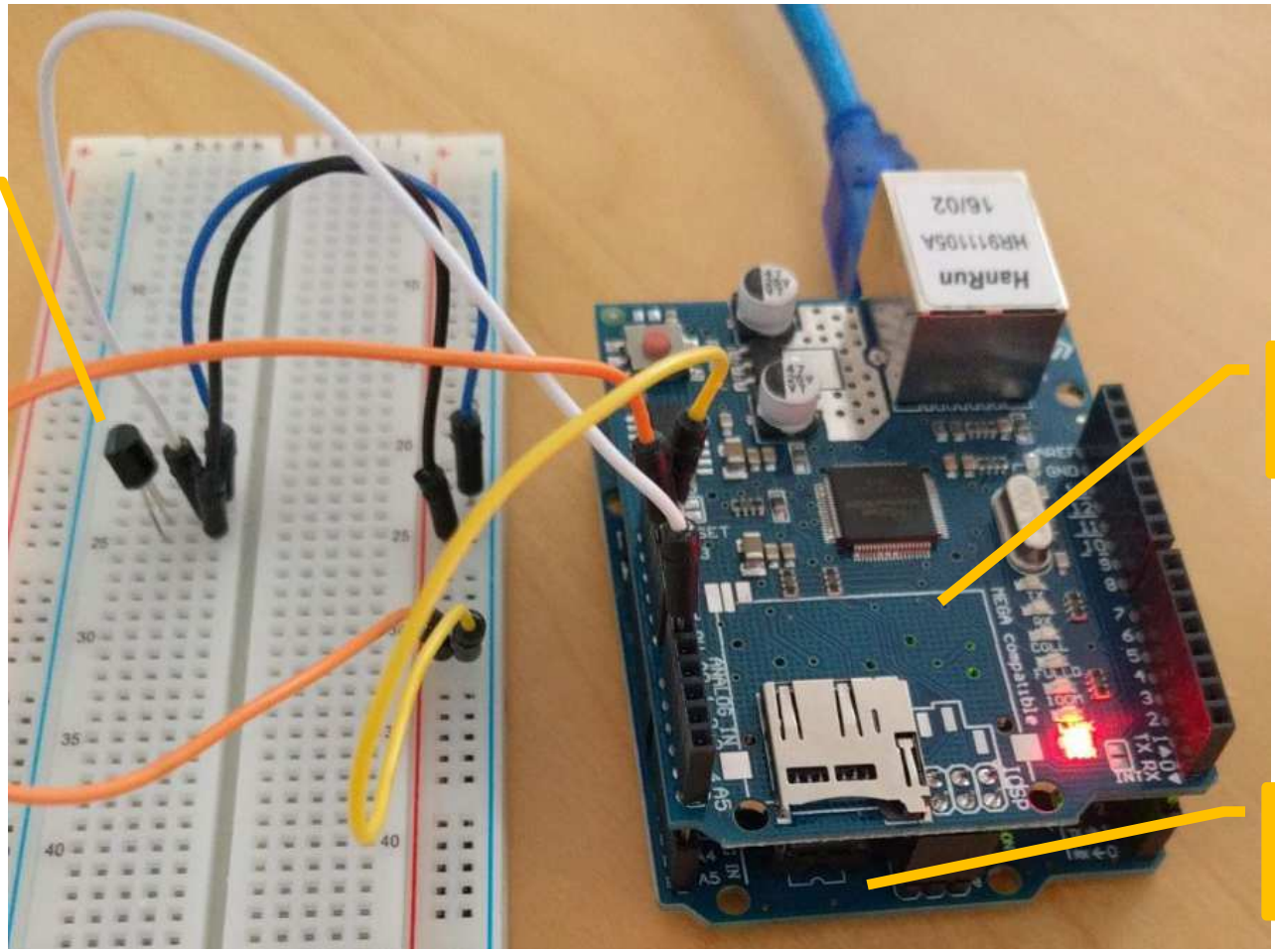
HTTP-API
Test Result





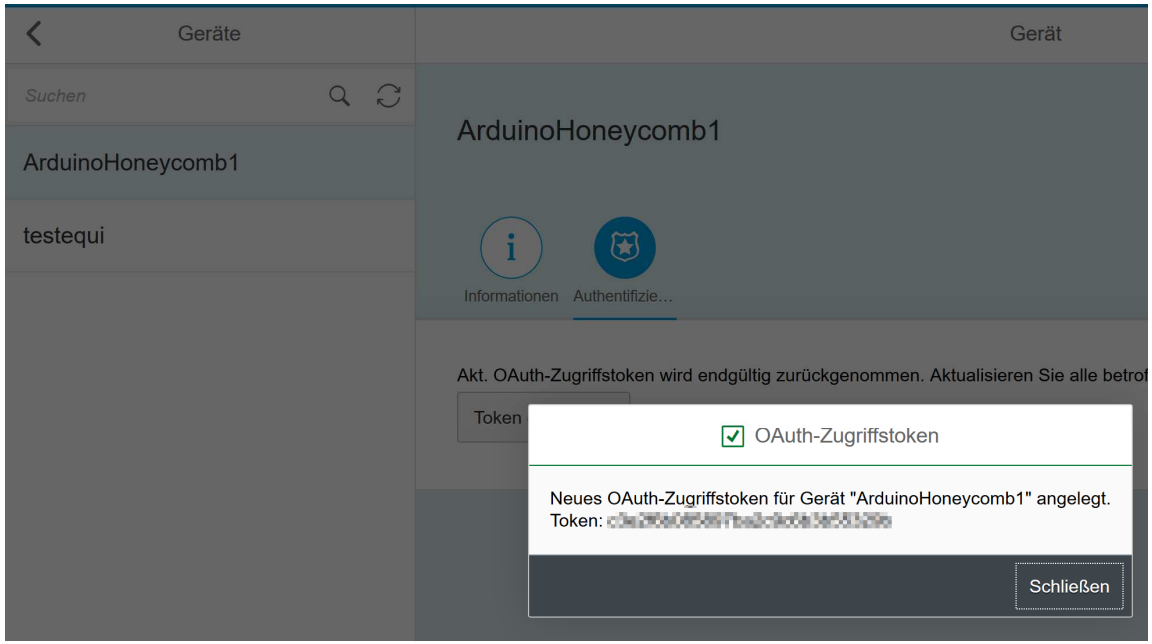
G_DEVICE	G_CREATED	C_BEEKEEPERID	C_HONEYCOMBID	C_MEASUREDATE	C_MEASURETIME	C_WEIGHT	C_WEIGHTUNIT	C_TEMPERATURE	C_TEMPERATUREUNIT	C_BEEKEEPERNAME
cd5f2abd-6959-41da-b347-8071c4177bc9	Mon Jan 16 2017 12:43:00 GMT+0100	1	1	Mon Jan 16 2017 01:00:00 GMT+0100	Thu Jan 01 1970 13:12:00 GMT+0100	2.3	kg	23.4	C	hias

Arduino Hardware



Arduino
Sketch - Preparation

- oAuthToken: Internet of Things Services Cockpit -> Geräte -> <Gerät> -> Authentifizieren -> Token generieren



Arduino Sketch

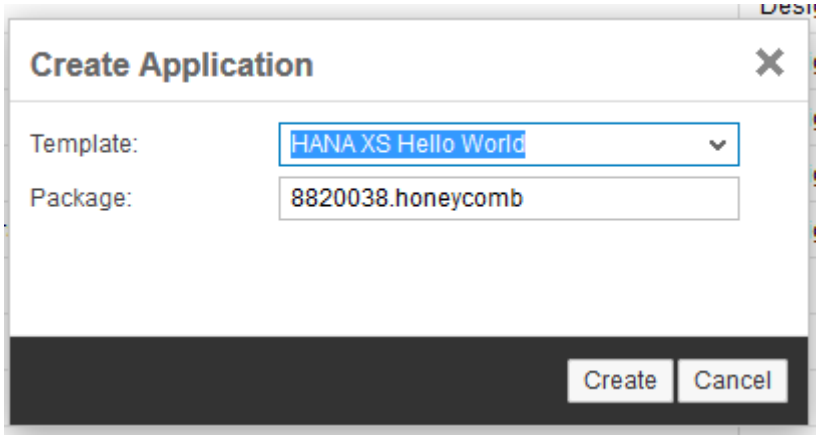
HONEYCOMB

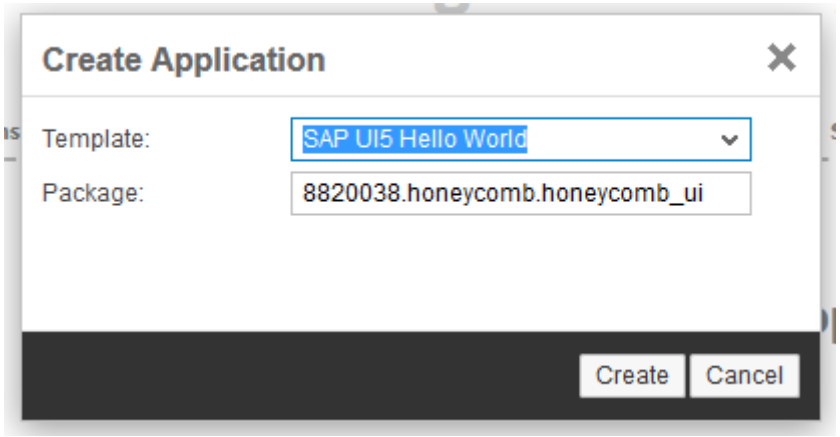
A smart example for IoT and SAP HANA

Honeycomb Package

- SAP HANA Web-based Development Workbench: Editor

Honeycomb
XSC Application





Honeycomb
SAPUI5 - View

- Create a view App.view.xml

Conventions

- View names are capitalized
- All views are stored in the view folder
- All controller are stored in the controller folder
- Names of XML views always end with *.view.xml
- The default XML namespace is sap.m
- Other XML namespaces use the last part of the SAP namespace as alias (for example, mvc for sap.ui.core.mvc)

Honeycomb
SAPUI5 - View

```
<?xml version=1.0 encoding=UTF-8?>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv=X-UA-Compatible content=IE=edge />
    <meta charset=utf-8 />
    <title>Honeycomb</title>
    <!-- 1.) Load SAPUI5 (from local server), select theme and control library -->
    <script id=sap-ui-bootstrap type=text/javascript src=/sap/ui5/1/resources/sap-ui-core.js data-sap-ui-
theme=sap_bluecrystal data-sap-ui-libs=sap.m data-sap-ui-compatVersion=edge data-sap-ui-preload=async
data-sap-ui-resourceroots={            at.facet.8820038.honeycomb: ./         } />
    <script>sap.ui.getCore().attachInit(function () {
        sap.ui.xmlview({
          viewName : at.facet.8820038.honeycomb.view.App
        }).placeAt(content);
      });</script>
  </head>
  <body class=sapUiBody>
    <!-- This is where you place the UI5  -->
    <div id=content />
  </body>
</html>
```

Honeycomb
Pitfall index.html

- Resource path from Web

src=/sap/ui5/1/resources/sap-ui-core.js

- Resource path from Web IDE

src=resources/sap-ui-core.js

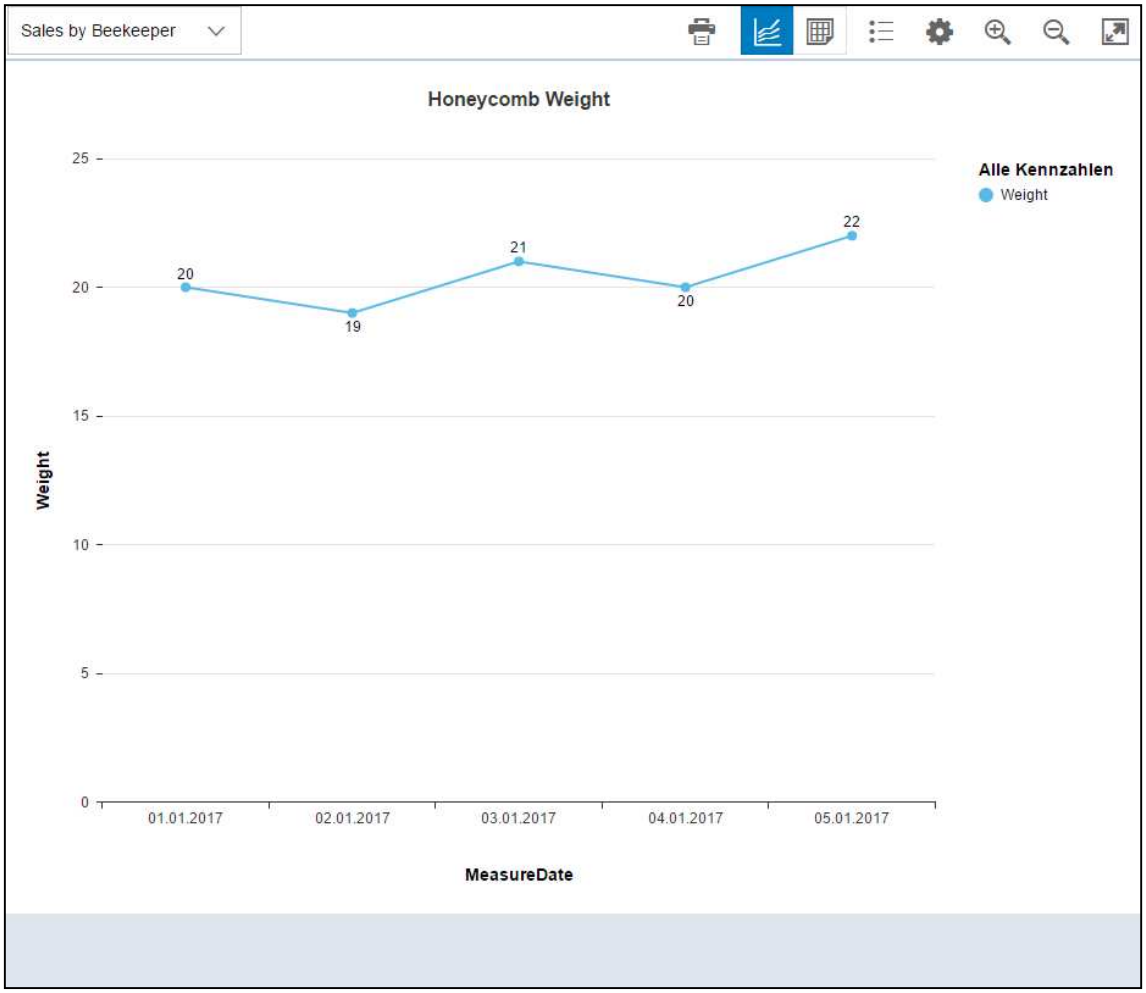
- Alternative:

src=https://sapui5.hana.ondemand.com/1.30.6/resources/sap-ui-core.js

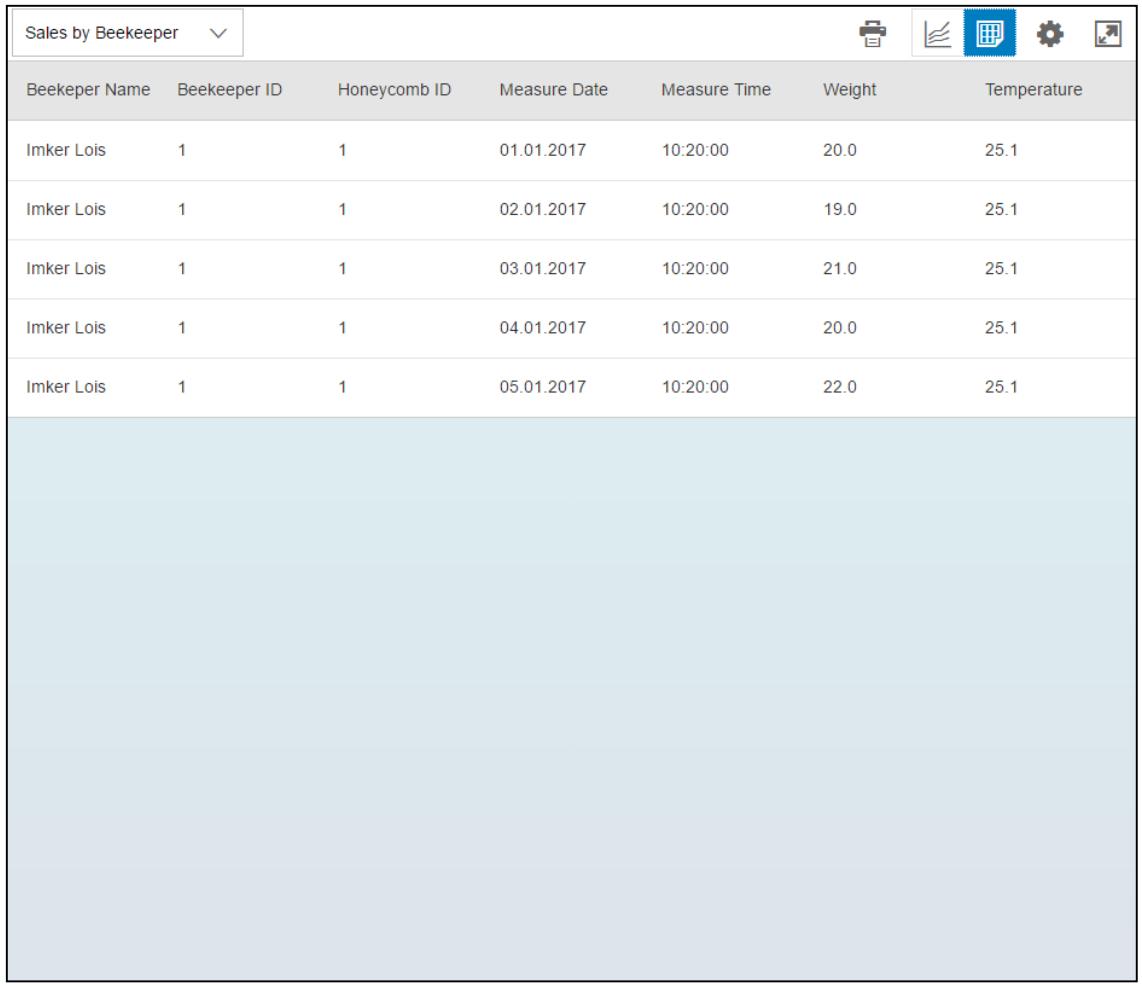
Honeycomb
SAPUI5 - View

- Now take the code from https://sapui5.netweaver.ondemand.com/sdk/explored.html#/sample/sap.suite.ui.commons.sample.ChartContainerFixFlexLayout/code

Honeycomb Graph



Honeycomb Table

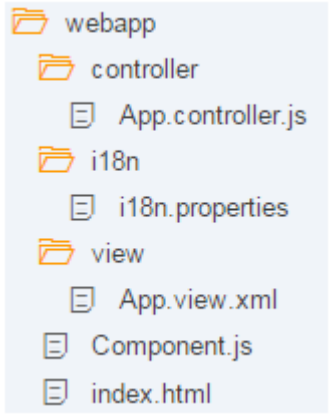


Beekeper Name	Beekeeper ID	Honeycomb ID	Measure Date	Measure Time	Weight	Temperature
Imker Lois	1	1	01.01.2017	10:20:00	20.0	25.1
Imker Lois	1	1	02.01.2017	10:20:00	19.0	25.1
Imker Lois	1	1	03.01.2017	10:20:00	21.0	25.1
Imker Lois	1	1	04.01.2017	10:20:00	20.0	25.1
Imker Lois	1	1	05.01.2017	10:20:00	22.0	25.1

Honeycomb Refactor

- TODO: Follow the tutorial

Honeycomb Refactor



Honeycomb manifest.json Conventions

- The descriptor file is named manifest.json and located in the webapp folder.
- Use translatable strings for the title and the description of the app.

Honeycomb Result