BUILD "DELIGHTFUL" USER INTERFACES IN THE CLOUD WITH THE SAP UI 5 FRAMEWORK

CD360

Exercises / Solutions

Ben Aflalo / SAP Labs,LLC.

TABLE OF CONTENT

Introduction

Welcome

Welcome to the developer hands-on workshop for SAP NetWeaver Cloud Portal and SAPUI5. Over the next two hours, you will build rich, branded and customizable application user interfaces and deploy them to the SAP cloud. After this session, you will be equipped with the knowledge and skills to quickly build and design your own application site composed of widgets and pages, using the SAPUI5 html5 libraries, the standard Open Social component model and Open Social features implemented by NetWeaver Cloud Portal and the NetWeaver Cloud Portal authoring environment.

Application Description

In this workshop, you will enhance the user interface of a simple application that allows a user to schedule events, choose venues and assign participants. The main focus will be "widgetizing" this application by writing Open Social specification XMLs that include JavaScript and HTML code that in turn rely on SAPUI5 and NW Cloud Portal libraries. The application will leverage the key services of the NetWeaver Cloud Portal including branding, persistence, context, personalization and embedded support for SAPUI5.

Workshop Agenda

This workshop is structured into 9 exercises, as follows:

1. Configure development environment and load the initial web application;
2. "Widgetize" the *Events* Section with UI5 controls – Table widget, Details widget, Map widget;
3. "Widgetize" the *Event Venues* Section – Table widget with alternative expanded view, Map widget;
4. "Widgetize" the *My Team* Section with UI5 controls - Table widget, Details widget;
5. Add Cloud Portal Open Social Features;
6. Deploy the Widgets to NW Cloud and expose in NW Cloud Portal;
7. Authoring the Application Site;
8. Theming and Branding the Site;
9. Publishing the Site.

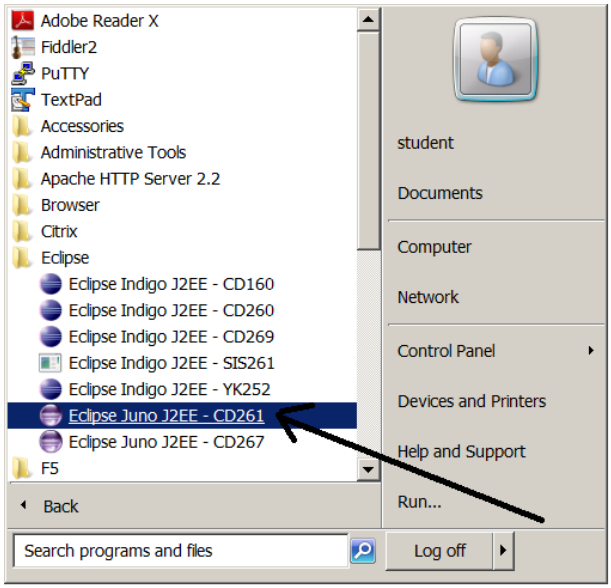
Exercise 1

In this first exercise we will start the eclipse IDE, verify the NW Cloud settings and finally import a simple web application that will serve both as our base application and the starting point UI. We will use the same IDE used for NetWeaver Cloud hands-on session CD261.

Verify eclipse settings

1. Start Eclipse

   a. Click on ***Start→All Programs→Eclipse→Eclipse Juno J2EE – CD261***.
   If a security warning message appears, just click on ***Run***.



2. Verify Proxy Settings

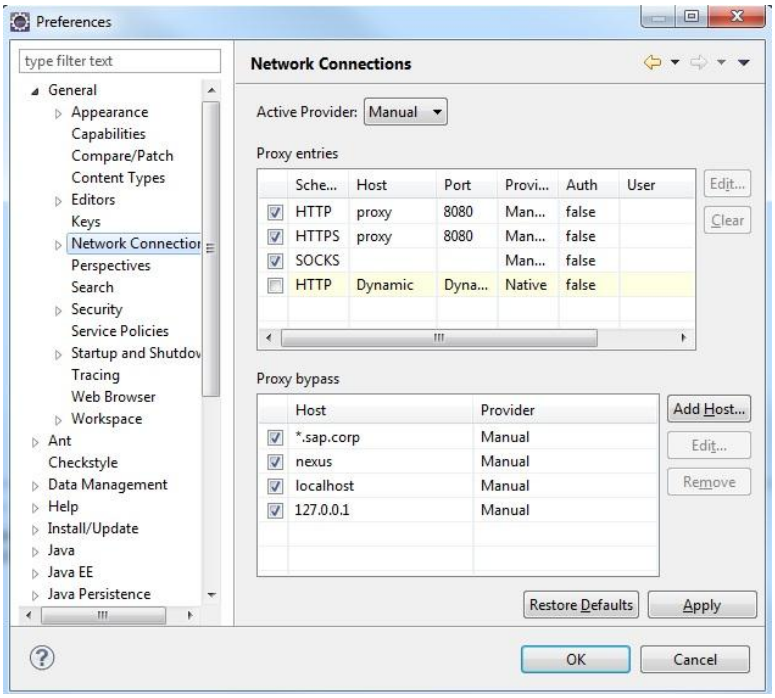
   a. In Eclipse, click on ***Window→Preferences***.

   b. Open the ***General*** tab and click on ***Network Connections***.

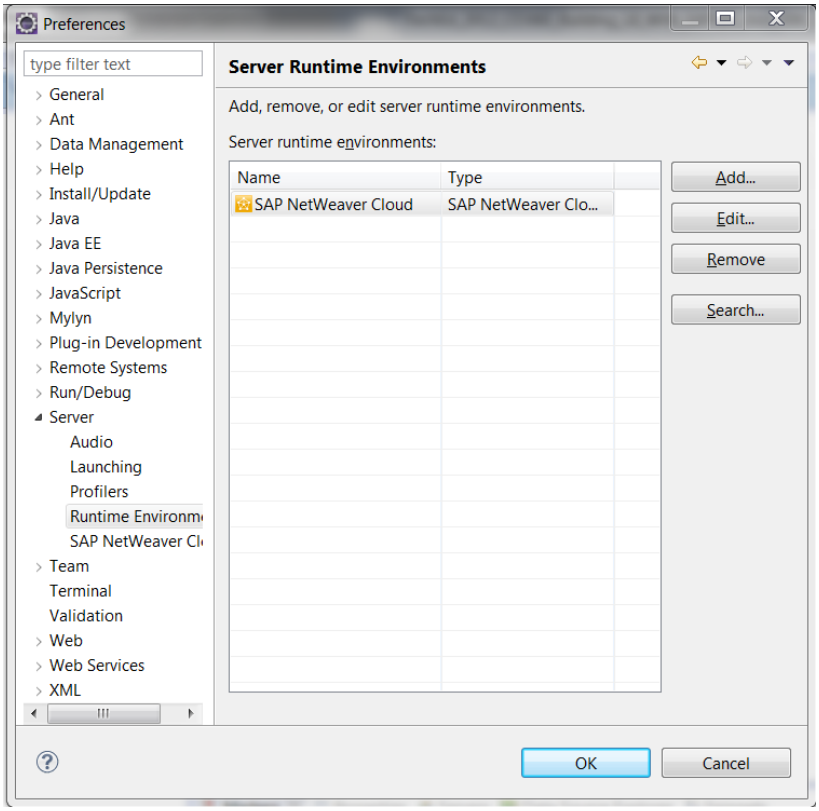
   c. Verify that at least the entries listed below are checked. If not, then update accordingly by switching to Active Provider to Manual first.

   Proxy entries:
   HTTP, proxy, 8080, False
   HTTPS, proxy, 8080, False

   Proxy bypass:
   **.sap.corp*
   nexus
   localhost
   10..*.**

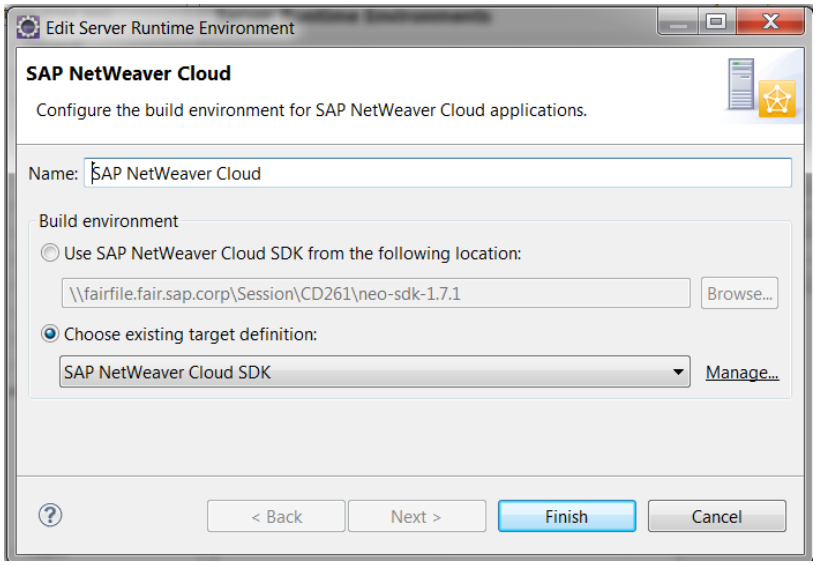


3. Verify Target Runtime Settings
    a. In the ***Preferences*** window, open the ***Server*** tab.
    b. Click on ***Runtime Environments***.
    c. Verify that there is a ***SAP NetWeaver Cloud*** entry. Click on it and then click on the ***Edit*** button.

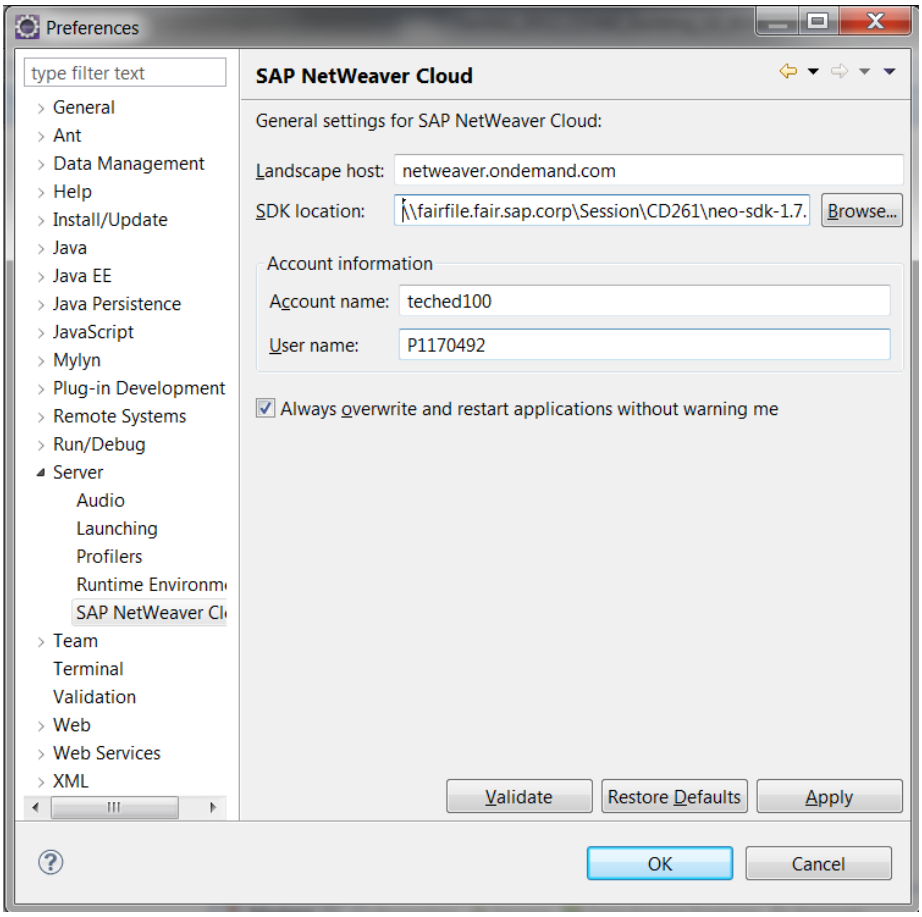


d. Verify that the popup window contains the following information:

- *Name:* ***SAP NetWeaver Cloud***
- *Build environment: Choose existing target definition:* ***SAP NetWeaver Cloud SDK***.
- Click on the ***Cancel*** button.

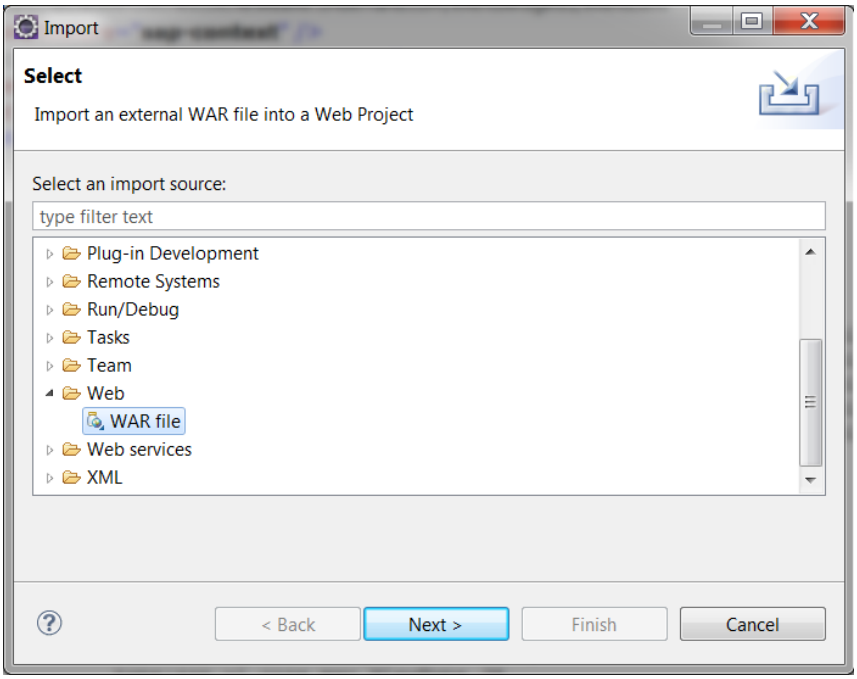


4. Verify NW Cloud Server Settings
   a. In the Preferences window, still in the Server tab, click on SAP NetWeaver Cloud.
   b. Verify the following fields:
      - *Landscape host:* ***netweaver.ondemand.com***
      - *SDK location:* ***\\fairfile.fair.sap.corp\Session\CD261\neo-sdk-1.7.1***
   c. Update the following fields:
      - *Account name:* ***teched<your assigned number> (e.g. teched100)***
      - *User name:* ***<your assigned username>***
   d. Click on the ***Validate*** button and verify a positive validation result in a popup window. Click on ***OK***.
   e. Click on the ***Apply*** button.



Import the initial events management web project

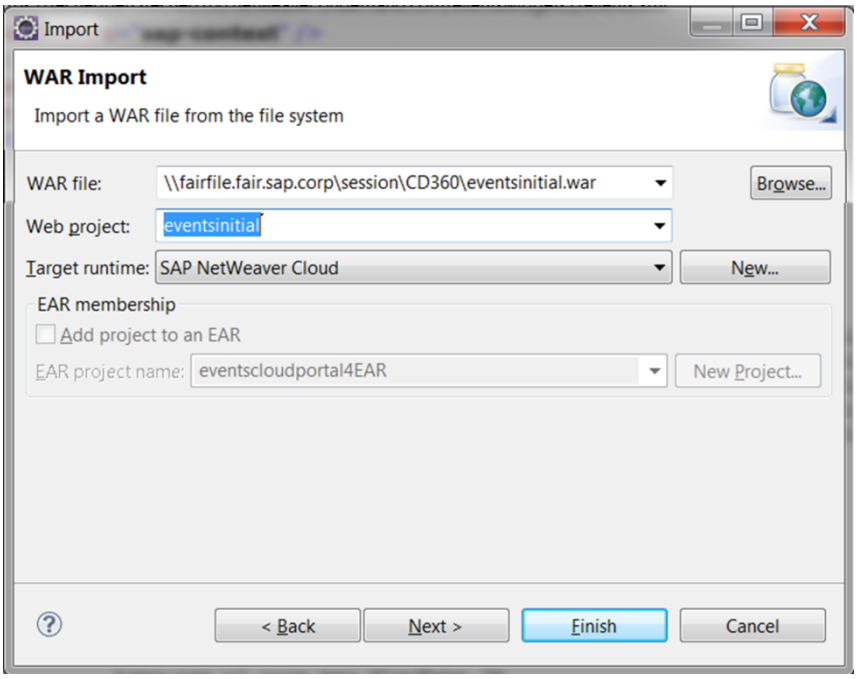
1. Import the war file into a new web project
    a. In Eclipse, click on File→Import…
    b. Select import source: Web→WAR file



c. Click Next

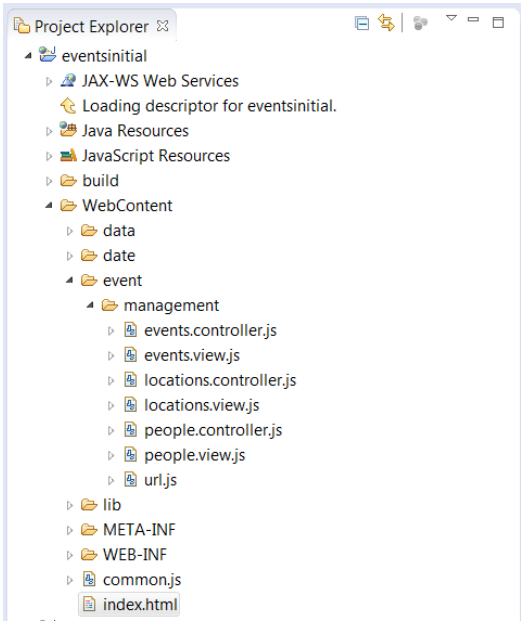
d. Browse to war file \\fairfile.fair.sap.corp\session\CD360\eventsinitial.war war file

e. Name the Web project *eventsinitial*



f. Click Finish

g. Optional: browse through the new project's JavaScript and HTML source code



Get to know the application functionality

1. Run the application

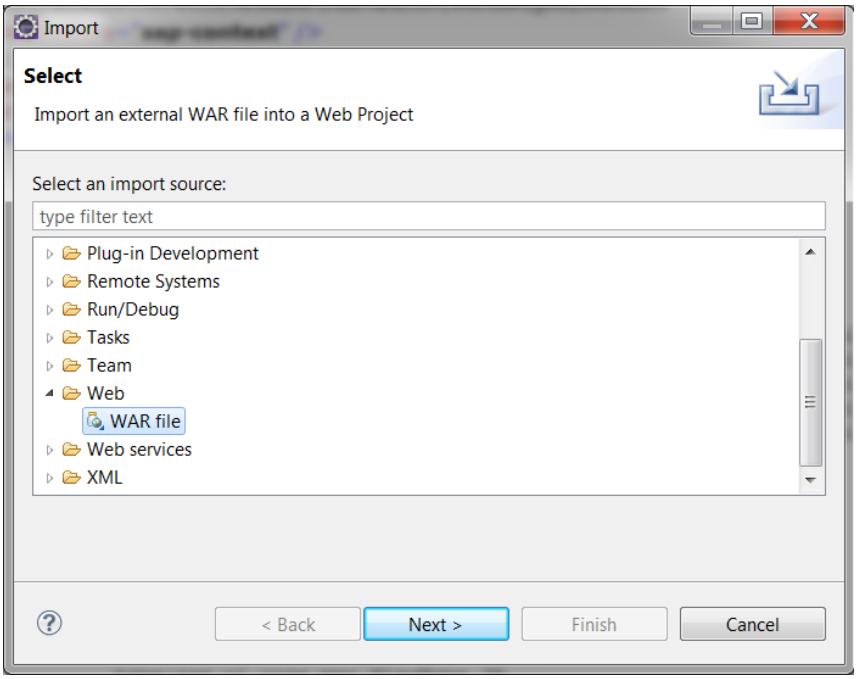
    a. In Firefox or Chrome open URL https://eventstechedd2012.netweaver.ondemand.com/eventsinitial/index.html; if prompted, login with your SCN credentials

Import the widgetized events management web project

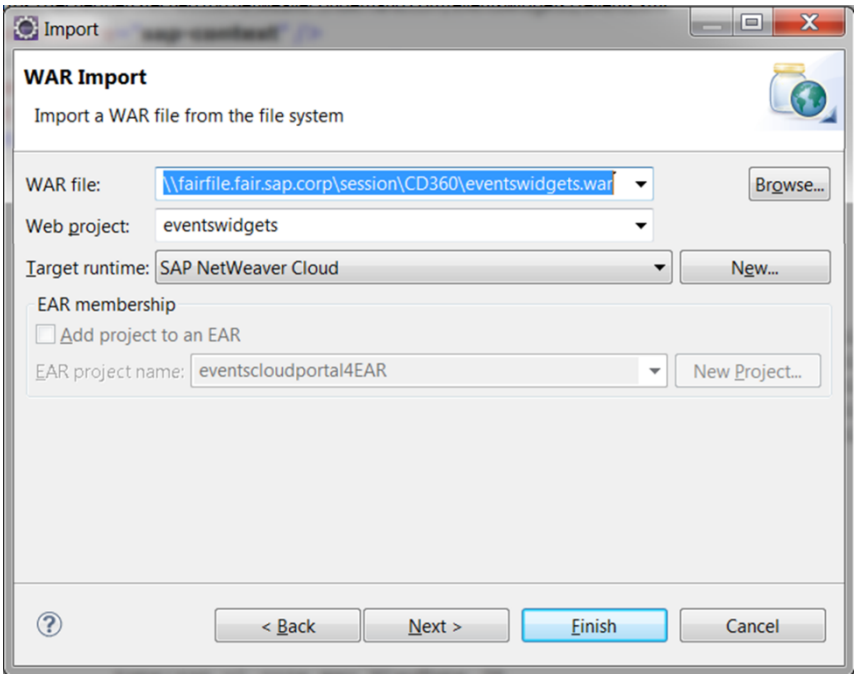
As a preparation to the next exercises, we will now import another web project, this time containing the source code for the widgetized version of the events management application.

a. In Eclipse, click on File→Import…

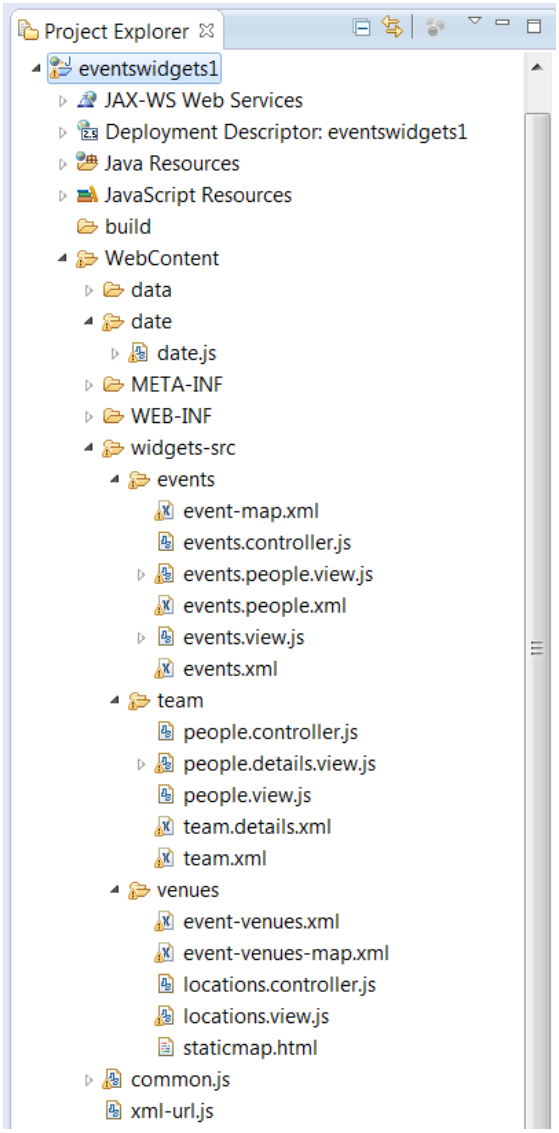
b. Select import source: Web→WAR file



c. Click Next
d. Browse to war file \\fairfile.fair.sap.corp\session\CD360\eventswidgets.war
e. Name the Web project *eventswidgets*



f. Click Finish
g. This should be the result:

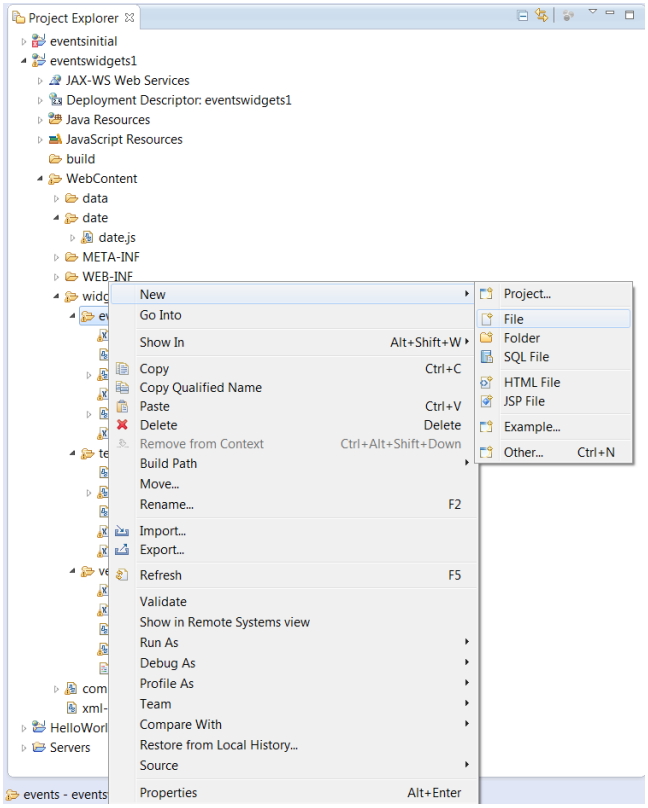


Exercise 2

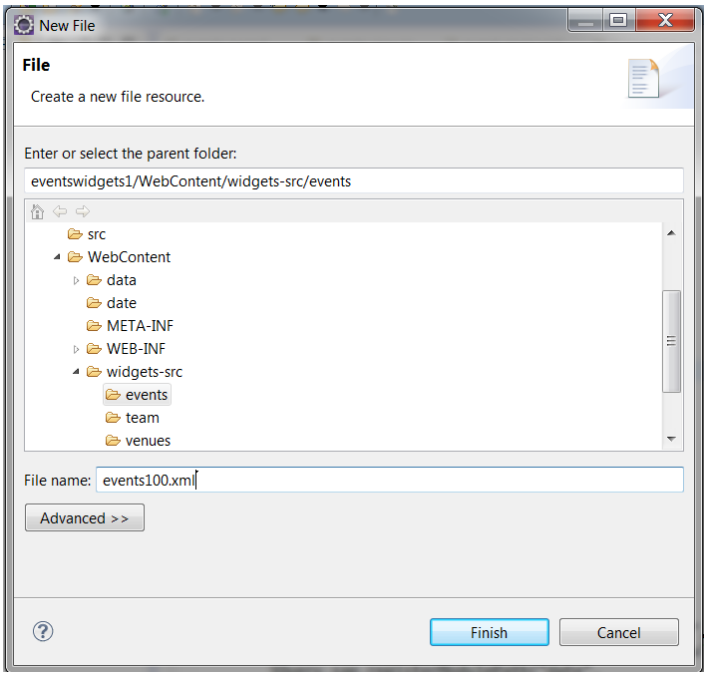
The next four exercises deal with the creation of the widgets. Generally, every widget is defined by an Open Social specification XML file (or spec XML); the JavaScript and HTML source code used by the widget can either be included directly in the spec XML or referenced using a bootstrap mechanism used in the Cloud Portal. We will see both types in the following exercises. Due to time constraints we will not do most of the actual coding during the session.
In Exercise 2 we will wigetize the *Events* section creating three widgets: a table widget holding the events data, a participant detail widget that shows information about the people attending a selected event and a map widget showing the venue location of a selected event in Google maps.

Event Table Widget

1. Create a new xml file for the events table widget
    a. In the eventswidgets web project imported in the former step browse to folder *events*
    b. Right click on the folder and choose New→File

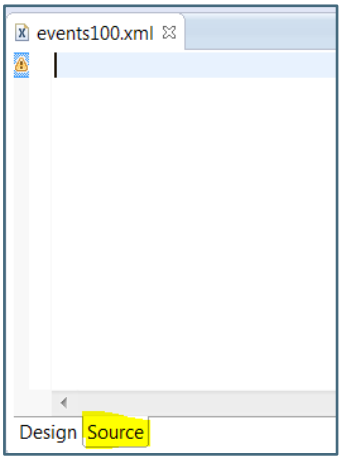


c. Name the file *events<your account suffix>.xml* (e.g. events100.xml)



d. Double click on the new file to edit

e. Click on the *source* tab:



f. Enter the following snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>

  <ModulePrefs title="Events">

  </ModulePrefs>

  <Content type="html">
    <![CDATA[

    ]]>
  </Content>
</Module>
```

This is the most basic form of a spec XML; it composes of one main Module section that in turn holds two sections: ModulePrefs and Content. The ModulePrefs is the header where widget properties are defined (e.g. *title*) and Open Social features are declared; the content is the actual source code of the widget

g. Add the Open Social feature declarations to ModulePrefs section:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>

  <ModulePrefs title="Events">

    <!--START: SAPUI5 feature-->
    <Require feature="sap-ui5">
      <Param name="sap-ui-params"></Param>
    </Require>
    <!--END: SAPUI5 feature-->

    <!--START: Publish/Subscribe feature-->
    <Require feature="pubsub-2"/>
    <Require feature="sap-context"/>
    <!--END: Publish/Subscribe feature-->

  </ModulePrefs>

  <Content type="html">
    <![CDATA[

    ]]>
  </Content>
</Module>
```

In this widget we will use the SAPUI5 feature enabling the usage of the SAPUI5 libraries within the widget (with no additional imports) and the Publish/Subscribe feature allowing widgets to communicate with each other.

h. Now add the content section:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>

  <ModulePrefs title="Events">

    <!--START: SAPUI5 feature-->
    <Require feature="sap-ui5">
      <Param name="sap-ui-params"></Param>
    </Require>
    <!--END: SAPUI5 feature-->

    <!--START: Publish/Subscribe feature-->
    <Require feature="pubsub-2"/>
    <Require feature="sap-context"/>
    <!--END: Publish/Subscribe feature-->

  </ModulePrefs>


  <Content type="html">
    <![CDATA[
        <body class="sapUiBody" role="application">
                        <div id="content"></div>
        </body>

        <script>
          var onLoad = function(){
                var url = gadgets.util.getUrlParameters().url,
                        path = url.substring(0,url.lastIndexOf('/'));

        jQuery.sap.registerModulePath("common",           gadgets.io.getProxyUrl(path +
"/../../common").replace('?', '?nocache=1&')) ;
        jQuery.sap.registerModulePath("eventController",  gadgets.io.getProxyUrl(path +
"/events.controller"));
        jQuery.sap.registerModulePath("eventView",        gadgets.io.getProxyUrl(path +
"/events.view"));
        jQuery.sap.registerModulePath("date",             gadgets.io.getProxyUrl(path +
"/../../date/date"));
        jQuery.sap.registerModulePath("url",              gadgets.io.getProxyUrl(path +
"/../../xml-url").replace('?', '?nocache=1&'));

        jQuery.sap.require("common");
        jQuery.sap.require("eventController");
        jQuery.sap.require("eventView");
        jQuery.sap.require("date");
        jQuery.sap.require("url");

        var view = sap.ui.view({
            viewName:"event.management.events",
            type:sap.ui.core.mvc.ViewType.JS
        });
        view.placeAt("content");

      }
      gadgets.util.registerOnLoadHandler(onLoad);
</script>

    ]]>
  </Content>
</Module>
```

The Content is composed of standard HTML and JavaScript code within CDATA. It has a very simple *body* section defining a div and a script that is used to bootstrap 5 additional JavaScript files using jQuery APIs and to define and place a SAPUI5 view. The referenced files are already in the same *events* folder within the *eventswidgets* project, notice they are almost identical to the ones used in the *eventsinitial* project. Whenever a row is selected in the table, an event is published with the row context, enabling the other widgets to respond (implemented in the events.view.js file).

2. Save the XML file

Map Widget

1. Create a new xml file for the events map widget
    a. Follow the same steps as before, name the new file *event-map<your account suffix>.xml* (e.g. event-map100.xml)
    b. Edit the new file and enter the following snippet to the *source* tab:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="Events Google Maps Subscriber" height="250">
    <!--START: Publish/Subscribe feature-->
    <Require feature="pubsub-2"/>
    <Require feature="sap-context"/>
    <!--END: Publish/Subscribe feature-->
  </ModulePrefs>
  <Content type="html">
    <![CDATA[
         <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%; width:100%">
<head>
         <script language="JavaScript">
         var onLoad = function(){

      /* START: Subscriber functionality */

      var subId;
      var googleMapUrlPrefix = "http://maps.google.com/maps?output=embed&t=m&num=1&q=";

      function updateMap (topic, context) {

          if (topic === "init-sub") {
              return;
          }

          var location = context.getPropertyByKey("eventLocation")[0];
          if (location !== "") {

              document.getElementById('mapIframe').src = googleMapUrlPrefix + (location);
          }

      }

      /*
       * Subscribe to Site Context using sap-context feature.
       */
      var subscribe = function subscribe() {
          subId = gadgets.sapcontext.subscribe(updateMap);
      }

      setTimeout(subscribe,500);

      /* END: Subscriber functionality */

    }
    gadgets.util.registerOnLoadHandler(onLoad);
</script>
</head>

<body  bgcolor="white" style="font-family:Arial;height:100%; width:100%; margin: 0;">

                  <div id="googleMap" style="height:100%; width:100%">
                      <iframe id="mapIframe" frameBorder="0"
src="http://maps.google.com/maps?output=embed&t=m&num=1&q=usa" style="height:100%; width:100%"></iframe>
        </div>

         </body>
</html>
]]>
  </Content>
</Module>
```

As you can see, this spec XML is different than the events.xml in that the source code is entirely embedded in the content CDATA section. Using the Publish-Subscribe feature, it responds to a row

selection event in the event table widget by subscribing to the corresponding event and changing the query URL sent to the Google map API to match the selected row's venue location.

2. Save the XML file

Participants Details Widget

1. Create a new xml file for the participants details widget
   a. Follow the same steps as before, name the new file *events.people<your account suffix>.xml* (e.g. events.people100.xml)
   b. Edit the new file and enter the following snippet to the *source* tab:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>

  <ModulePrefs title="Events Attending People">

    <!--START: SAPUI5 feature-->
    <Require feature="sap-ui5">
      <Param name="sap-ui-params"></Param>
    </Require>
    <!--END: SAPUI5 feature-->

    <!--START: Publish/Subscribe feature-->
    <Require feature="pubsub-2" />
    <Require feature="sap-context" />
    <!--START: Publish/Subscribe feature-->

  </ModulePrefs>

  <Content type="html">
    <![CDATA[
         <body class="sapUiBody" role="application">
                        <div id="people"></div>
         </body>

         <script>
           var onLoad = function(){

                  var url = gadgets.util.getUrlParameters().url,
                           path = url.substring(0,url.lastIndexOf('/'));

        jQuery.sap.registerModulePath("common",           gadgets.io.getProxyUrl(path +
"/../../common").replace('?', '?nocache=1&')) ;
        jQuery.sap.registerModulePath("eventController",  gadgets.io.getProxyUrl(path +
"/events.controller"));
        jQuery.sap.registerModulePath("eventPeopleView",  gadgets.io.getProxyUrl(path +
"/events.people.view"));
        jQuery.sap.registerModulePath("date",             gadgets.io.getProxyUrl(path +
"/../../date/date"));
        jQuery.sap.registerModulePath("url",              gadgets.io.getProxyUrl(path + "/../../xml-
url").replace('?', '?nocache=1&'));

        jQuery.sap.require("common");
        jQuery.sap.require("eventController");
        jQuery.sap.require("eventPeopleView");
        jQuery.sap.require("date");
        jQuery.sap.require("url");

        var view = sap.ui.view({
            viewName:"event.management.events.people",
            type:sap.ui.core.mvc.ViewType.JS
        });
        view.placeAt("people");
      }
      gadgets.util.registerOnLoadHandler(onLoad);
</script>

    ]]>
  </Content>
</Module>
```

Although this spec XML seems quite similar to the events.xml, the content is entirely different; the main part is implemented in the referenced events.people.view.js file. Similar to the map widget, also this widget responds to the selection of a row in the event table widget by displaying all the participants of the selected event.

2. Save the XML file

Exercise 3

In the next exercise we will widgetize the *Events Venues* section. This page will be composed of two widgets: a table widget listing the different available venues that enabled adding, deleting and editing these venues, and a map widget that shows the selected venue location on Google maps (identical to the map widget in the events page). This page will have has an alternative display mode: a custom menu in the venue table widget enables toggling between the Table + Map widget and a single expanded map view showing all the venue locations together in the same map.

Venue Table Widget

1. Create a new xml file for the participants details widget
    a. Follow the same steps as before this time starting from the *venues* folder; name the new file *event-venues<your account suffix>.xml* (e.g. event-venues100.xml)
    b. Edit the new file and enter the following snippet to the *source* tab:

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>

  <ModulePrefs title="Event Venues">

    <!--START: SAPUI5 feature-->
    <Require feature="sap-ui5">
      <Param name="sap-ui-params"></Param>
    </Require>
    <!--END: SAPUI5 feature-->

    <!--START: Publish/Subscribe feature-->
    <Require feature="pubsub-2"/>
    <Require feature="sap-context"/>
    <!--END: Publish/Subscribe feature-->

    <!--START: Maximize Feature-->
    <Require feature="sap-widget"/>
    <!--END: Maximize Feature-->

    <!--START: Menu Feature-->
    <Require feature="sap-menu"/>
    <!--END: Menu Feature-->

    <!--START: Gadget Preferences Feature-->
    <Require feature="gadgetprefs"/>
    <!--END: Gadget Preferences Feature-->

  </ModulePrefs>

  <Content type="html">
    <![CDATA[
         <body class="sapUiBody" role="application">
                        <div id="content"></div>
         </body>

         <script>
           var onLoad = function(){

                  var url = gadgets.util.getUrlParameters().url,
                           path = url.substring(0,url.lastIndexOf('/'));

        jQuery.sap.registerModulePath("common",              gadgets.io.getProxyUrl(path +
"/../../common").replace('?', '?nocache=1&')) ;
        jQuery.sap.registerModulePath("locationController",  gadgets.io.getProxyUrl(path +
"/locations.controller"));
        jQuery.sap.registerModulePath("locationView",        gadgets.io.getProxyUrl(path +
"/locations.view"));
        jQuery.sap.registerModulePath("date",                gadgets.io.getProxyUrl(path +
"/../../date/date"));
        jQuery.sap.registerModulePath("url",                 gadgets.io.getProxyUrl(path + "/../../xml-
url").replace('?', '?nocache=1&'));

        jQuery.sap.require("common");
        jQuery.sap.require("locationController");
        jQuery.sap.require("locationView");
        jQuery.sap.require("date");
        jQuery.sap.require("url");

        var view = sap.ui.view({
            viewName:"event.management.locations",
            type:sap.ui.core.mvc.ViewType.JS
        });
        view.placeAt("content");
      }

      gadgets.util.registerOnLoadHandler(onLoad);
</script>

    ]]>
  </Content>
</Module>
```

As you can see in the ModulePrefs section, this widgets uses some new Open Social features: in addition to SAPUI5 and Publish/Subscribe we already saw, also the Maximize, Menu and Gadget Preferences features are used here. We will take a closer look at these features in exercise 5.

2. Save the XML file

Venue Map Widget

1. By now we know how to create the spec XMLs, therefore in the next widgets we will only browse through the code of the XMLs already created in the eventswidgets project.
    a. From the *venues* folder, open file *event-venues-map.xml*
    b. Browse through the code in the *source* tab; as you can see it's almost identical to the map widget in the *events* page.

Exercise 4

In this short exercise we will browse through the code of the two widgets composing the *My Team* section: Team Table widget and Team Details widget. Both widgets include code similar to what we have already seen in the previous widgets.

Team Table Widget

1. Browse through the code of file team.xml
    a. From the *team* folder, open file *team.xml*
    b. Browse through the code in the *source* tab; as you can see it's very similar to the other table widgets.

Team Details Widget

1. Browse through the code of file team.details.xml
    a. From the *team* folder, open file *team.details.xml*
    b. Browse through the code in the *source* tab; as you can see it's almost identical to the Participants Details widget in the events page.

Exercise 5

In this exercise we will take a deeper look at some of the Cloud Portal Open Social features. Specifically, we'll see the use of features Publish/Subscribe, Menu and Gadget Preferences. Declaring these (or any other) features in the widget spec XML will load the required code at client runtime, so developers can just use the APIs for these features without worrying about the implementation.

Publish/Subscribe Feature

This feature allows different widgets to communicate with each other; this is achieved by an eventing mechanism where widgets can publish key value pairs (usually based on runtime events) while other widgets can subscribe to these keys. When a subscribed key is published, value is passed to the subscribed widget that can then react. A simple example is the communication between the Events Table widget and the Event Map widget. We'd like to show an event venue location in map widget whenever an event is selected in the events table.

1. Publish - browse through the code of the events table widget

    a. Open file events.xml from the folder events in the eventswidget project

    b. Notice that the feature is declared in the ModulePrefs section:

```
<Require feature="pubsub-2"/>
<Require feature="sap-context"/>
```

    c. Browse through the following code in file *events.view.*js:

```
…
        var onSelect = function(oEvent) {
            // get the binding context of the first selected row
            var oContext = oEvent.getParameter("rowContext");

            var oModel = oEvent.getSource().getModel();

            var people = oModel.getProperty("person",
                oContext);
            var venue = oModel.getProperty("eventLocation",
                oContext);

            /* START: Publish Events*/

            publish("eventPeople", people);
            publish("eventLocation", venue.name);

            /* END: Publish Events*/
        };

        /* START: Define Publish Function*/

        var publish = function publish(key, value) {
            gadgets.sapcontext.publish(key, value);
        };

        /* END: Define Publish Function*/
…
```

    As you can see, when a row is selected in the table, the following key value pairs are published: (“eventPeople”, people) and (“eventLocation”, venue.name). The function implementation simple consists of a call to the API: gadgets.sapcontext.publish(key, value).

2. Subscribe - browse through the code of the events map widget

    a. Open file *event-map.xml* from the folder events in the eventswidget project

    b. Notice that also here the same feature is declared in the ModulePrefs section:

```
<Require feature="pubsub-2"/>
<Require feature="sap-context"/>
```

    c. Now browse through the following code in *event-map.xml*:

```
…
      /* START: Subscriber functionality */

      var subId;
      var googleMapUrlPrefix = "http://maps.google.com/maps?output=embed&t=m&num=1&q=";

      function updateMap (topic, context) {

          if (topic === "init-sub") {
              return;
          }

          var location = context.getPropertyByKey("eventLocation")[0];
          if (location !== "") {

              document.getElementById('mapIframe').src = googleMapUrlPrefix + (location);
          }

      }

      /*
       * Subscribe to Site Context using sap-context feature.
       */
      var subscribe = function subscribe() {
          subId = gadgets.sapcontext.subscribe(updateMap);
      }

      setTimeout(subscribe,500);

      /* END: Subscriber functionality */
…
```

As you can see, the updateMap function reacts to a publish event by updating the Google maps URL with the value of the key eventLocation.

Menu Feature

This feature enables the development of a custom menu to a widget. It is used in the venue table widget to switch between a table view and an expanded map view of all venue locations

1. Browse through the code of the venue table widget
    a. Open file event-venues.xml from the folder venues in the eventswidget project
    b. Notice that the feature is declared in the ModulePrefs section:

```
<Require feature="sap-menu"/>
```

    c. Browse through the following code in file events.view.js:

```
…
            /* START: MENU Feature*/
            gadgets.sap.menu.remove({
                ID: Constants.MENU_ID
            });

            //Add View Change Menu action
            gadgets.sap.menu.addMenu({
                ID: Constants.MENU_ID,
                tooltip: 'Change View',
                icon: Constants.MENU_ICON_REGULAR,
                hoverIcon: Constants.MENU_ICON_HOVER
            });

            //Table View Menu Item Creation
            gadgets.sap.menu.addMenuItem({
                ID: Constants.VIEWS.TABLE.ID,
                menuID: Constants.MENU_ID,
                text: 'Table View',
                icon: Constants.VIEWS.TABLE.MENU_ITEM_ICON,
                callback: function switchToTableView() {
                    prefs.setPreference("viewMode", "table");
                    //Restore
                    if (parent.App.base.isDefined(widget)) {
                        widget.callbacks.restore.reduce();
                    }

                }.bind(this)
            });

…

            //Map View Menu Item Creation
            gadgets.sap.menu.addMenuItem({
                ID: Constants.VIEWS.MAP.ID,
                menuID: Constants.MENU_ID,
                text: 'Map View',
                icon: Constants.VIEWS.MAP.MENU_ITEM_ICON,
                callback: switchToMapView.bind(this)
            });

…
            /* END: MENU Feature*/
```

Using this feature a custom menu and two menu items are added to the menu: Table View and Map View. Selection of these menu items will call functions switching to the selected view.

Gadget Preferences Feature

This feature enables persistence of user preferences with regards to widget attributes. It is used here also in the venue table widget to save the user's view preference (table or map).

1. Browse again through the code of the venue table widget
    a. Open file event-venues.xml from the folder venues in the eventswidget project
    b. Notice that the feature is declared in the ModulePrefs section:

```
<Require feature="gadgetprefs"/>
```

    c. Browse through the following code in file events.view.js:

```
…
var prefs = new gadgets.GadgetPrefs(),
    moduleId = prefs.getWidgetId(),
    widget = parent.App.workspace.controller.getWidget(moduleId);
…
function switchToTableView() {
                    prefs.setPreference("viewMode", "table");
                    //Restore
                    if (parent.App.base.isDefined(widget)) {
                        widget.callbacks.restore.reduce(); }
…
function switchToMapView() {
                …
                prefs.setPreference("viewMode", "maximize");
                gadgets.sap.widget.openView({
                    mode: "maximize",
                    url: locationString,
                    contentType: "url",
                    stateful: true,
                    listeners: {
                    }
                });
            };
…
```

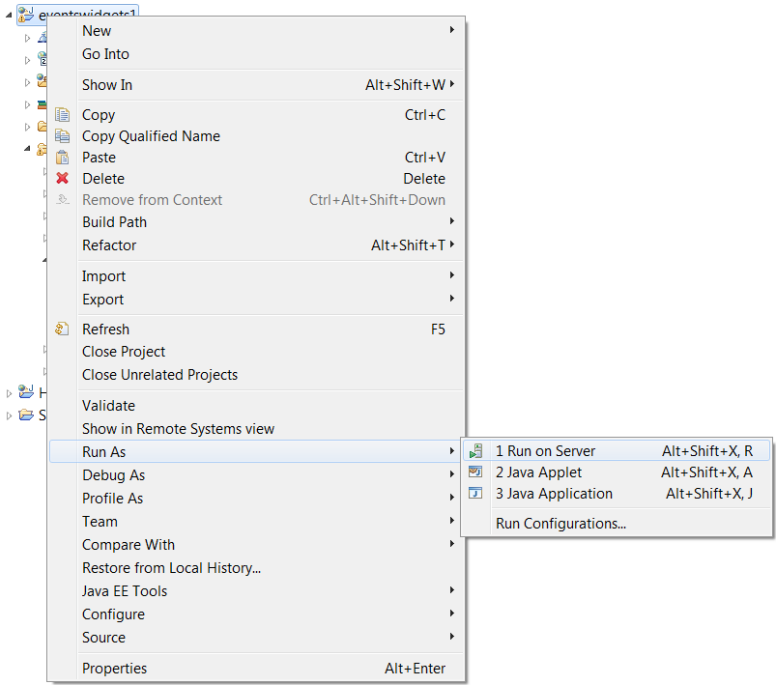
Every time a view mode changes, this selection is saved to the persisted prefs variable, so that the next time the user enters this page, the previous display mode will be active.

Exercise 6

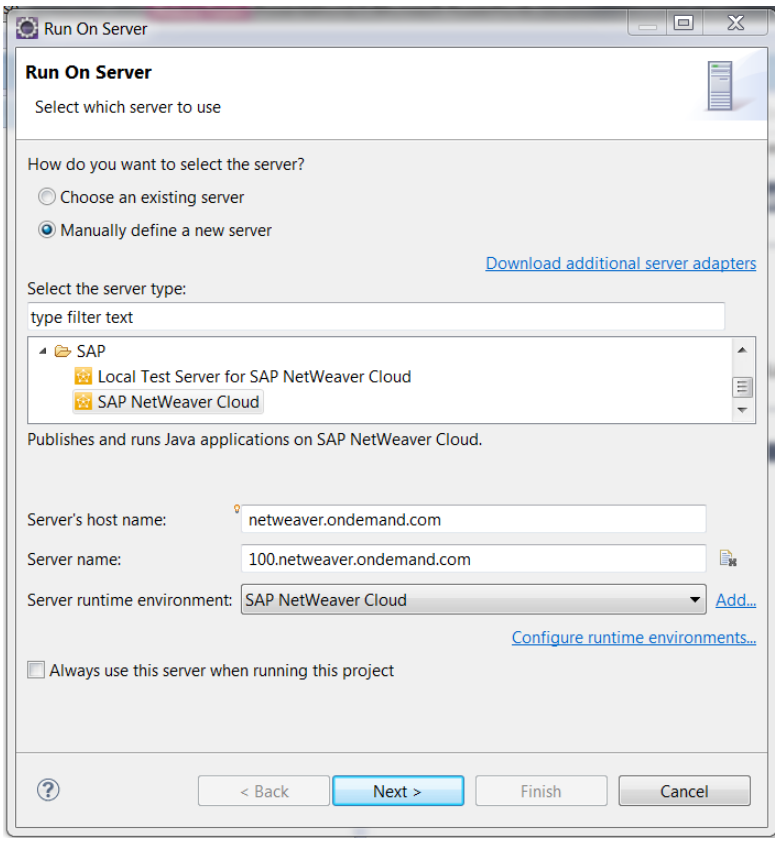
In this exercise we will deploy the widgets to NetWeaver Cloud and add them as Open Social widgets to the Cloud Portal content catalog.

Deploy widgets to NetWeaver Cloud

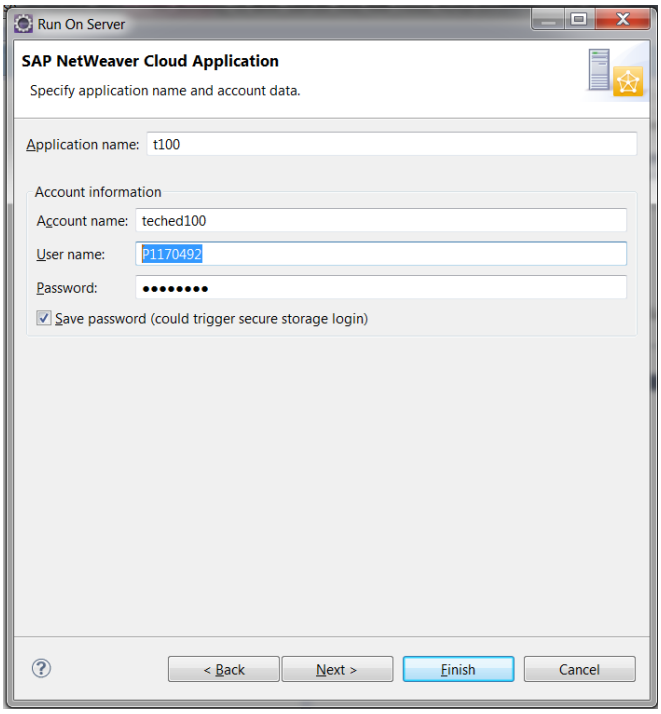
1. In eclipse, right click on the eventswidget project and choose Run As → Run on Server:



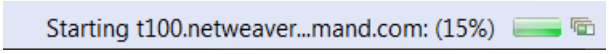
2. Create a new server
    a. In the Run On Server wizard, choose the *Manually define a new server*
    b. In Server Type choose SAP NetWeaver Cloud
    c. Server host name should be netweaver.ondemand.com (this is the default)
    d. Change the Server name to **t**<your account suffix>.netweaver.ondemand.com, e.g. **t100.netweaver.ondemand.com**
    e. Server runtime environment should be SAP NetWeaver Cloud (this is the default)
    f. Click Next



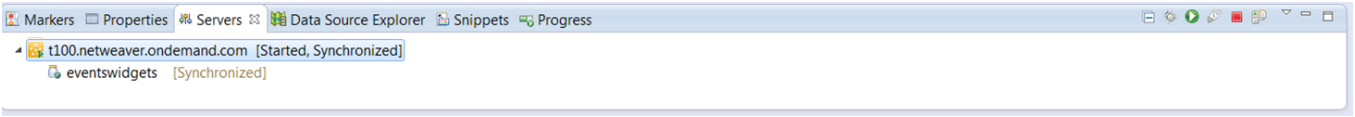
    g. In the next screen, in application name type **t**<your account suffix>, e.g**. t100**
    h. In Account name type teched<your account suffix>, e.g. teched100
    i. Enter your assigned username and password, e.g. P1170492/Abcd1234
    j. Click Finish



3. In the bottom right corner of eclipse you'll notice that the application is being deployed, this might take a few minutes



4. When this is done, you should see the following in the Servers tab in the lower part of eclipse:

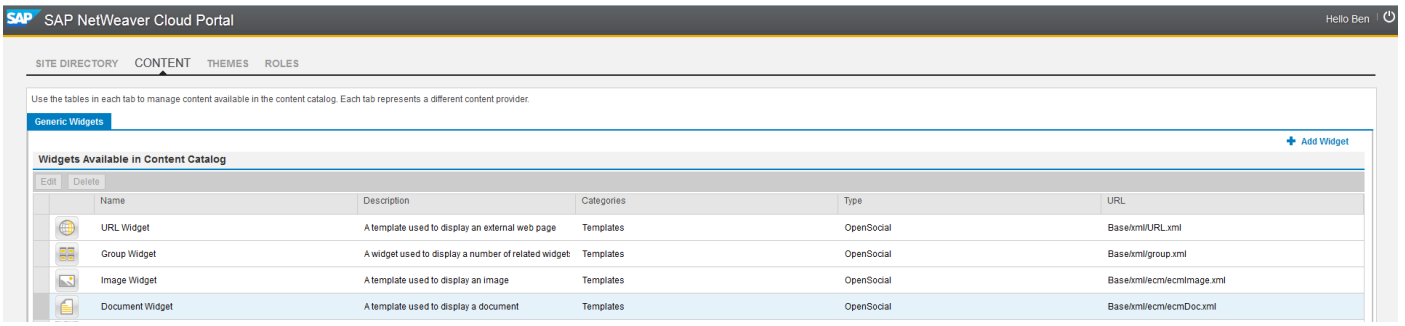


5. Next step is testing the URLs to the deployed spec XMLs. Generally, the path is https://<your application name><your account name>.netweaver.ondemand.com/eventswidgets/<path to XML as in the eventswidgets project>;
e.g. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/venues/event-venues.xml
    a. Make sure entering the following URLs in a browser show the expected XML files
    Note that these are just examples for application t100 and account teched100:
        i. https://t100teched100.netweaver.ondemand.com/eventswidgets1/widgets-src/events/events.xml
        ii. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/events/event-map.xml
        iii. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/events/events.people.xml
        iv. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/venues/event-venues.xml
        v. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/venues/event-venues-map.xml
        vi. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/team/team.xml
        vii. https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/team/team.details.xml

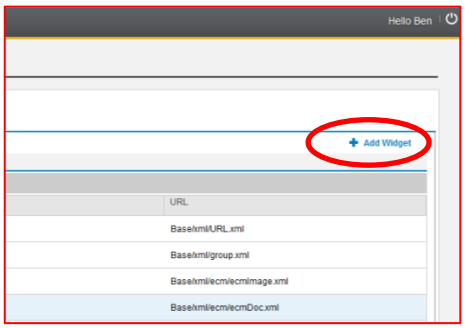
Create Open Social widgets in Cloud Portal

We will now import the widgets to Cloud Portal and add them to the content catalog

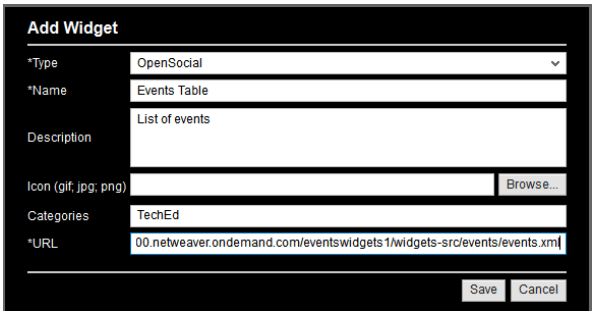
1. Logon to Cloud Portal
    a. Open the browser. And go to https://cloudsessionbteched2012-<your account name>.netweaver.ondemand.com/portal/index.html, e.g.: https://cloudsessionbteched2012-teched100.netweaver.ondemand.com/portal/index.html
    b. On the SDN login screen sign on with the given credentials
2. Navigate to Content tab:



3. Add OpenSocial widgets
    a. Click on +Add widget button



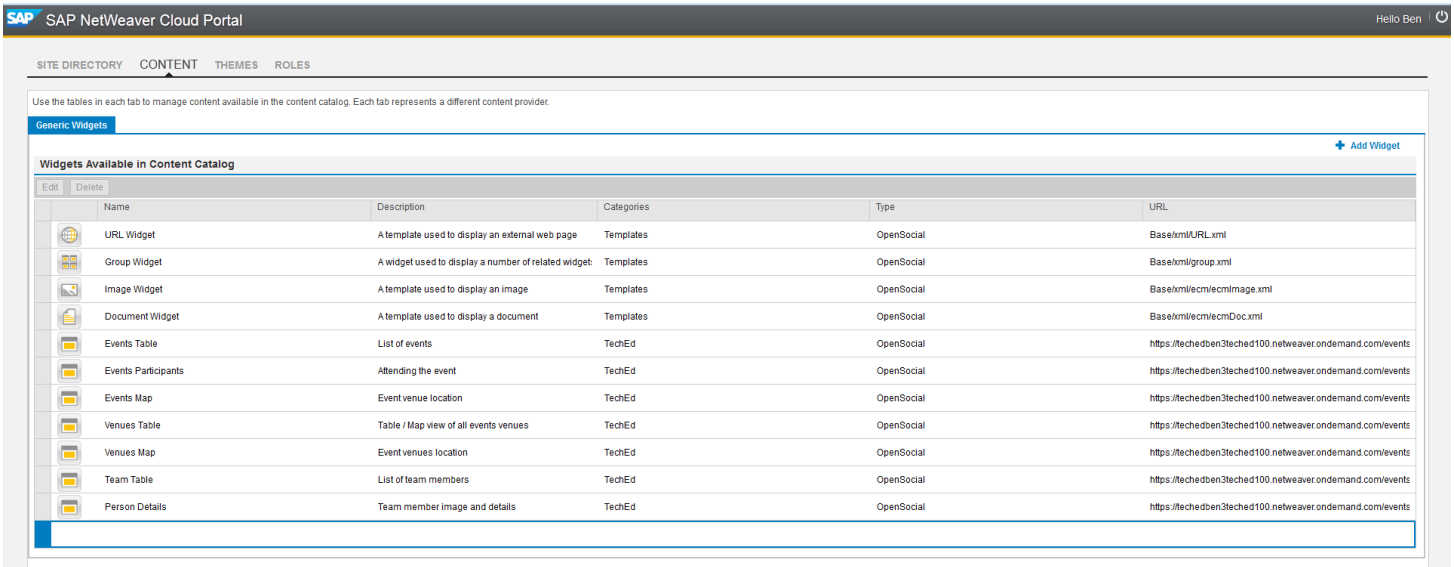
    b. In the Add dialog enter the following:
        - Type: choose OpenSocial
        - Name: <see table below>
        - Description: <free text> (optional)
        - Icon: <any image you see fit> (optional)
        - Categories: TechEd (optional)
        - URL: <see table below>
    c. Click Save



d. Repeat this step for all 7 widgets using values from this table:

Widget Type	Widget Name	Categories	URL
Open Social	Events Table	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets1/widgets-src/events/events.xml
Open Social	Events Map	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/events/event-map.xml
Open Social	Events Participants	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/events/events.people.xml
Open Social	Venues Table	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/venues/event-venues.xml
Open Social	Venues Map	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/venues/event-venues-map.xml
Open Social	Team Table	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/team/team.xml
Open Social	Person Details	TechEd	https://t100teched100.netweaver.ondemand.com/eventswidgets/widgets-src/team/team.details.xml

e. The Content Catalog should now look like this:

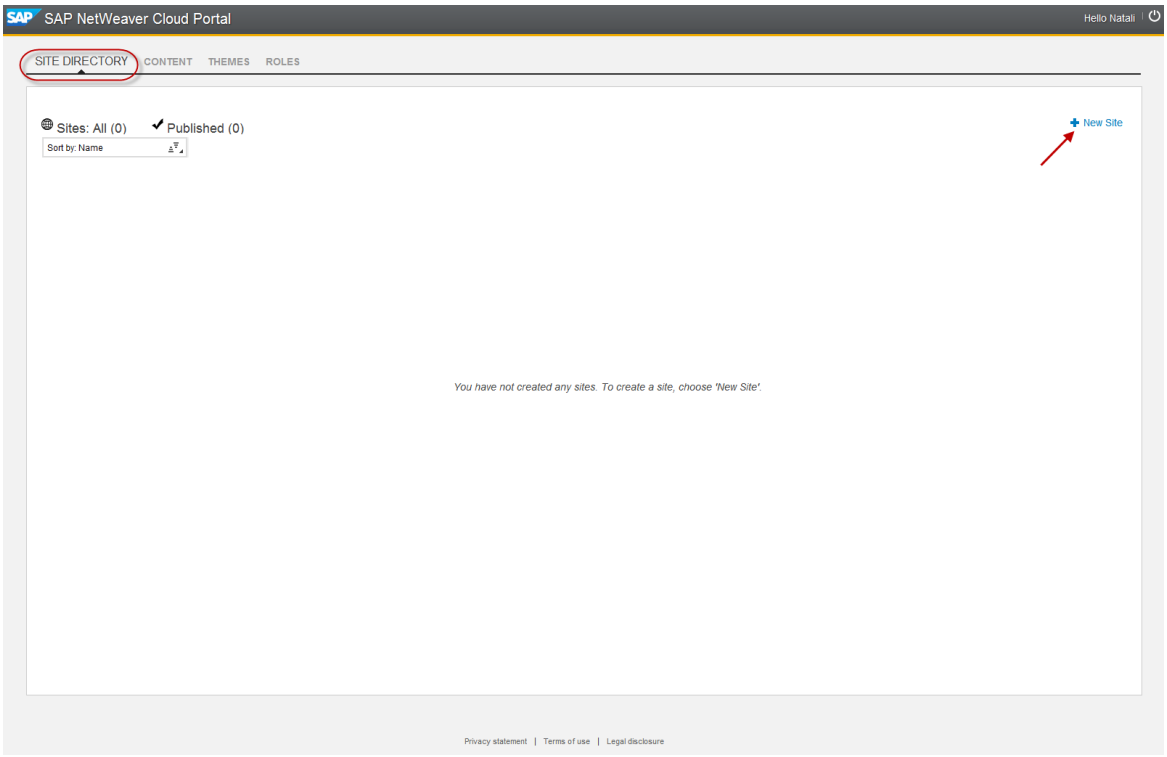


Exercise 7

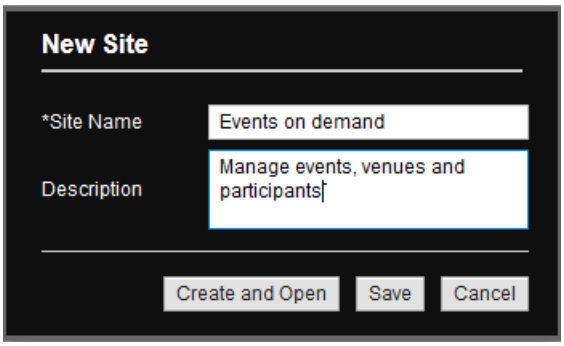
In this exercise we will create a site for our application and design its layout.

Create a Site

1. Navigate to Site Directory Tab
2. Click on +New Site button

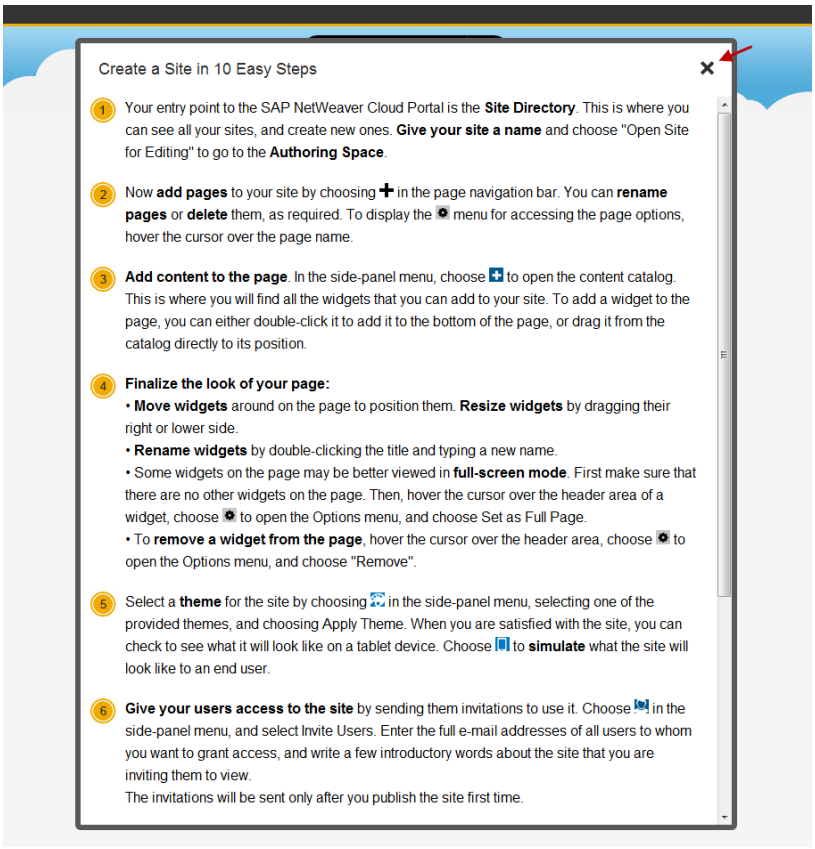


3. In the New Site dialog enter the following:
   - Site name: Events On Demand
   - Description: <free text>
   - Click “Create and Open” button

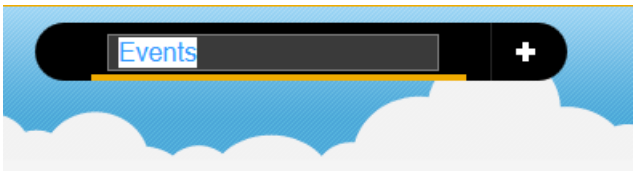


Add widgets and design the site layout

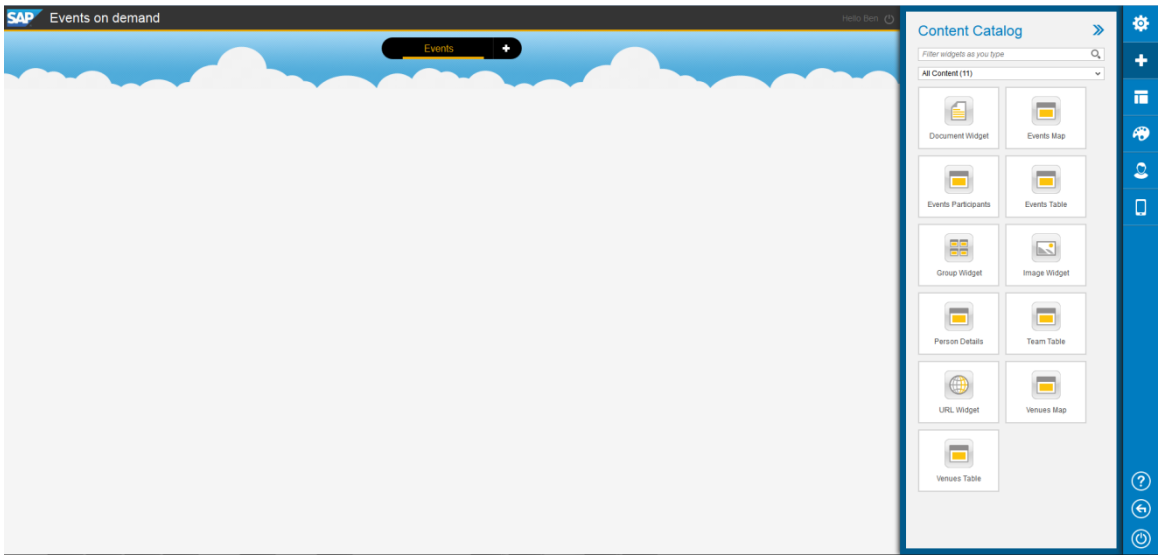
4. Create a Site in 10 Easy Steps
   a. In the new created site, read the Tip popup to guide you how to create a Site
   b. When finishing close it by clicking the X button – top right corner



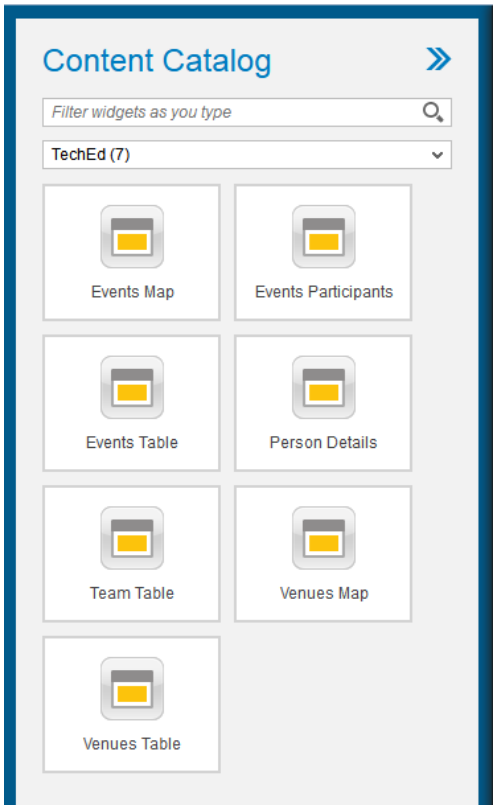
5. Rename first page
    a. In the pages bar double click the default page name
    b. The name is now editable, change it to: “Events”



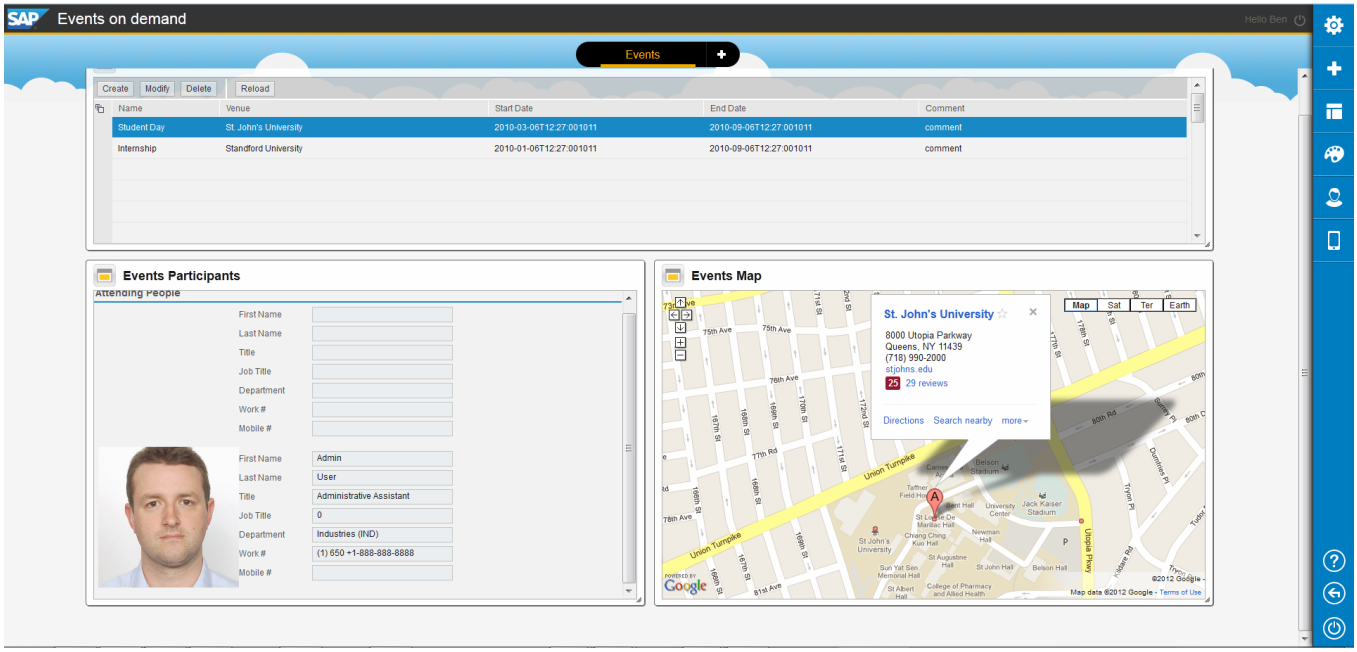
    c. Click Enter to apply changes
6. Open Content catalog by clicking the + icon in the site Authoring panel- blue strip on the right



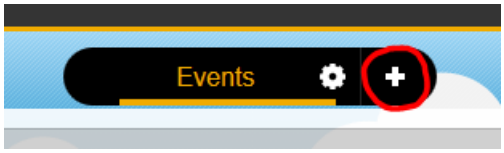
7. Optionally filter to show only the TechEd category



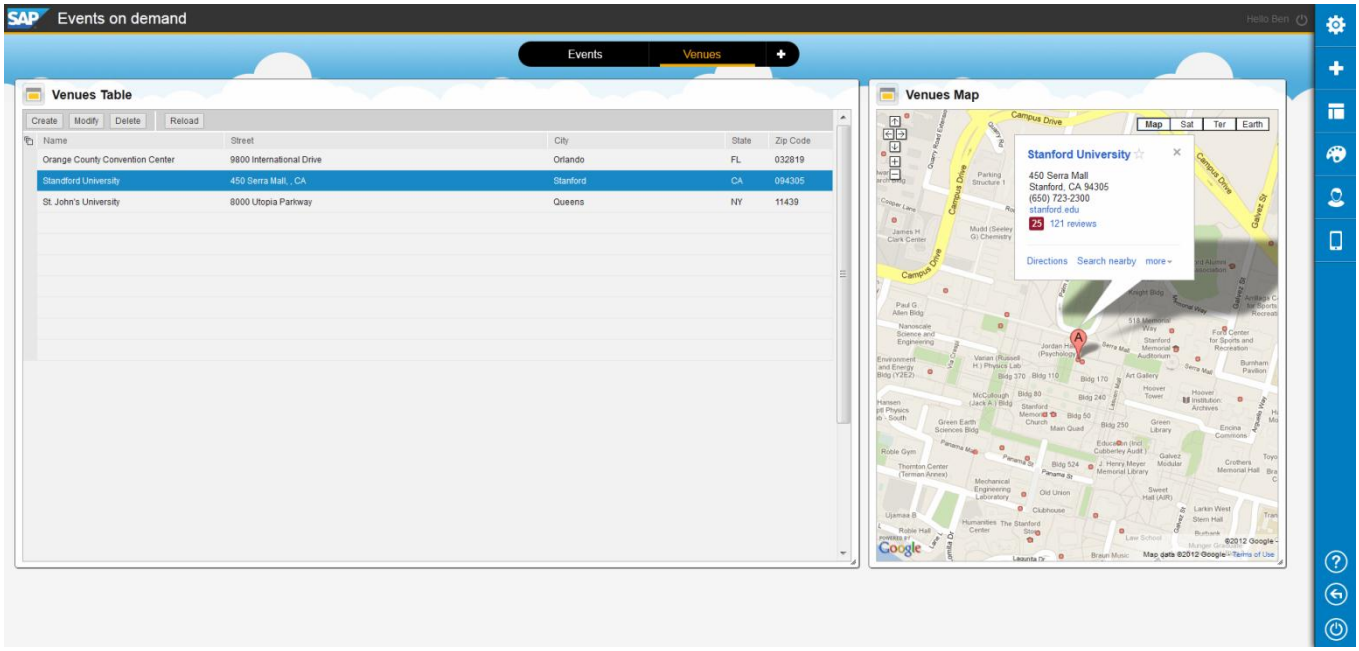
8. Add widgets Events Table, Events Map and Events Participants by double clicking or drag and dropping the widget to the Events page
9. Change the position and size of the widgets



10. Add a new Page by pressing the + icon next to the Events page

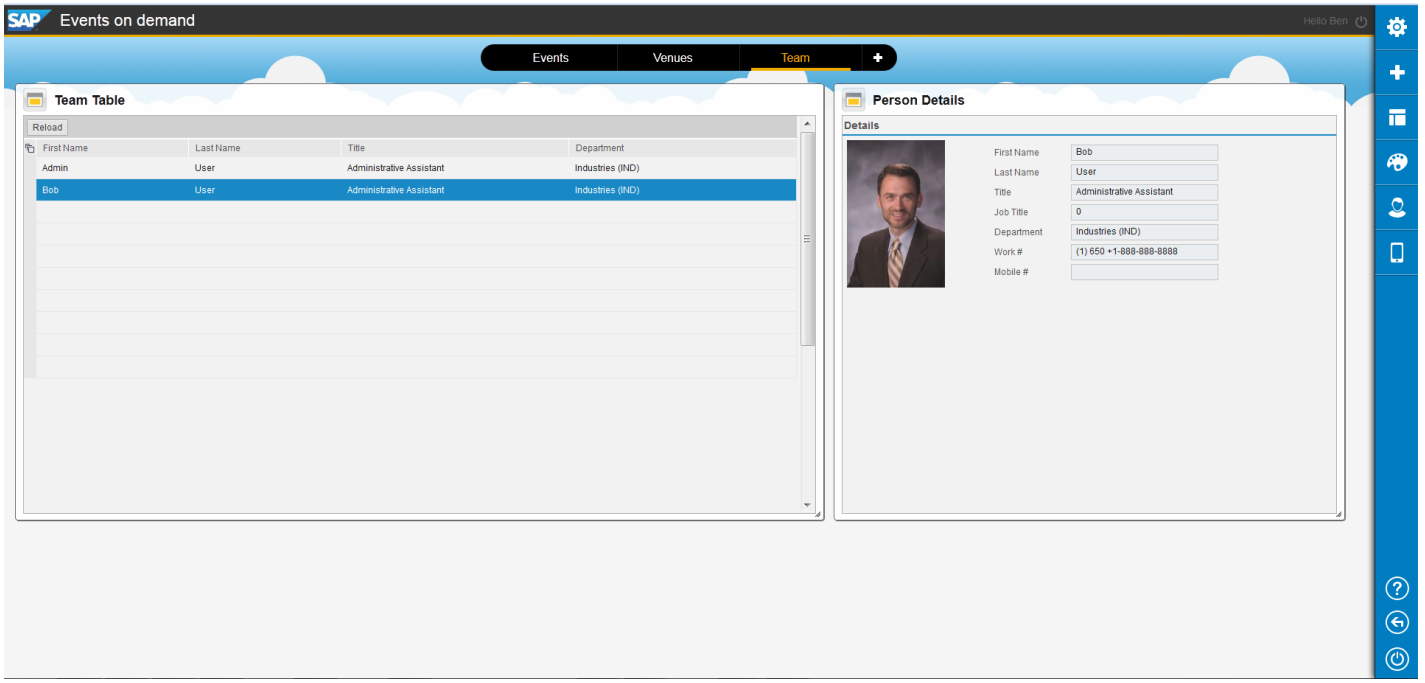


11. Rename this page to Venues
12. Add widgets Venues Table and Events Map by double clicking or drag and dropping the widget to the Venues page
13. Change the position and size of the widgets



14. Add a third page by pressing the + icon next to the Venues page

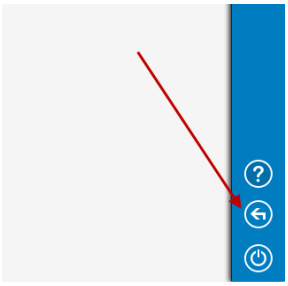
15. Rename this page to Team
16. Add widgets Team Table and Person Details by double clicking or drag and dropping the widget to the Team page
17. Change the position and size of the widgets



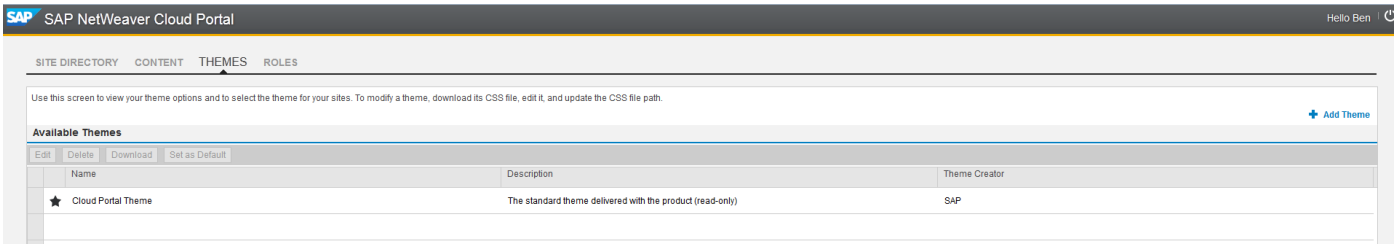
Exercise 8

Change the site theme

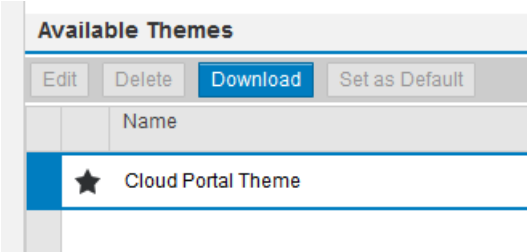
1. Go Back to site directory by clicking on the arrow icon on the bottom right



2. Navigate to Themes tab



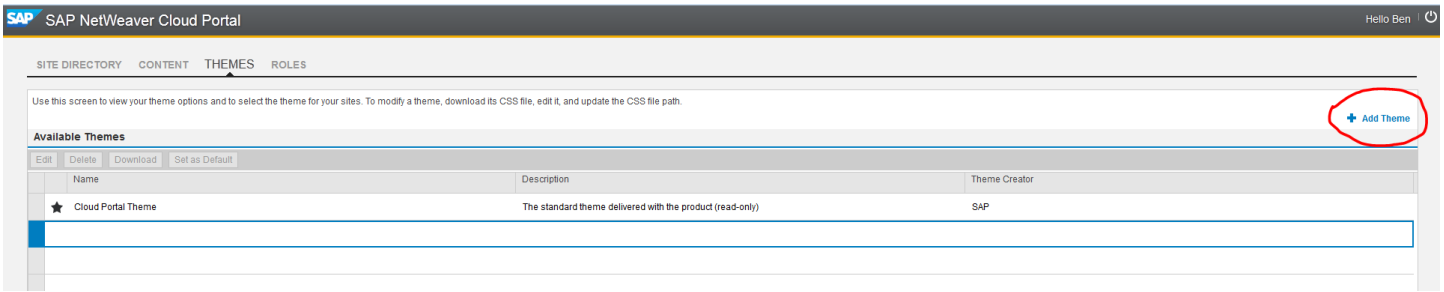
3. Download the default theme



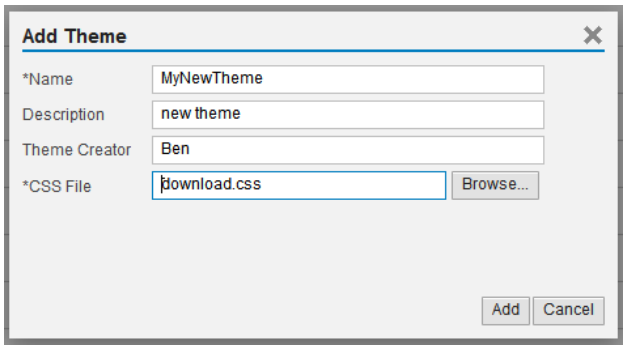
4. Edit CSS file
    a. Open the saved CSS file with any text editor (notepad/notepad++ etc'.).
    b. Search for "site-background" string (ctrl+f).
    c. Replace the data Uri (in the url property) with the following URL: http://t3.gstatic.com/images?q=tbn:ANd9GcS4idLfnJgEOxoe4buoTmYc-EDjUhCpD8Zh-k8If_WD8kg-G7Zs3g

```
/* Site background properties */
.site-background {
    background-image:
url(http://t3.gstatic.com/images?q=tbn:ANd9GcS4idLfnJgEOxoe4buoTmYc-EDjUhCpD8Zh-
k8If_WD8kg-G7Zs3g);
    background-size: auto;
    background-color: whiteSmoke;
    background-repeat: repeat-x;
    background-position: 0 37px;
}
```

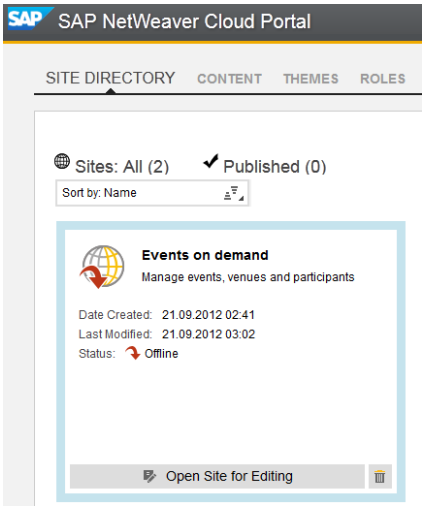
    d. Note: the default data URI is a very long string
    e. Save your changes
5. Create new Theme
    a. Back to Cloud portal, in the Themes tab click +Add Theme button



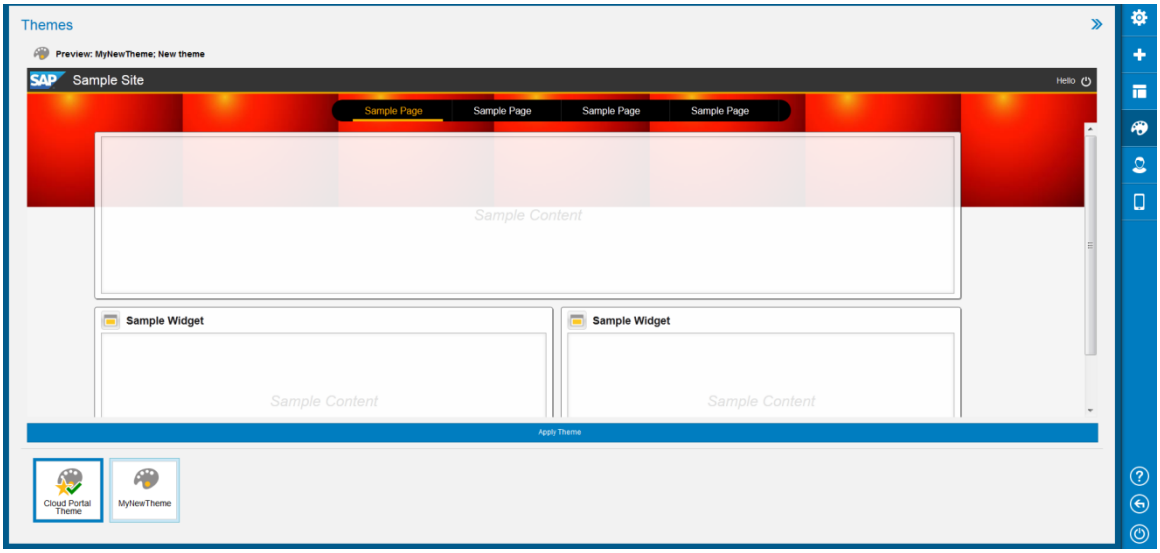
    b. In the Add dialog enter the following:
        i. Name: My new theme
        ii. Description: <free text>
        iii. Theme Creator: <your name>
        iv. CSS file: browse the file you edited in the step before



6. Enter site “Events on demand”
    a. Navigate to site directory tab
    b. Hover “Events on demand” site and click “Open site for editing” button



7. Preview and apply theme
    a. Click on Themes icon in the site Authoring panel
    b. In the Theme catalog at the bottom, preview “My new Theme” by clicking it once
    c. Apply new Theme by pressing the long button above the theme tiles



Exercise 9

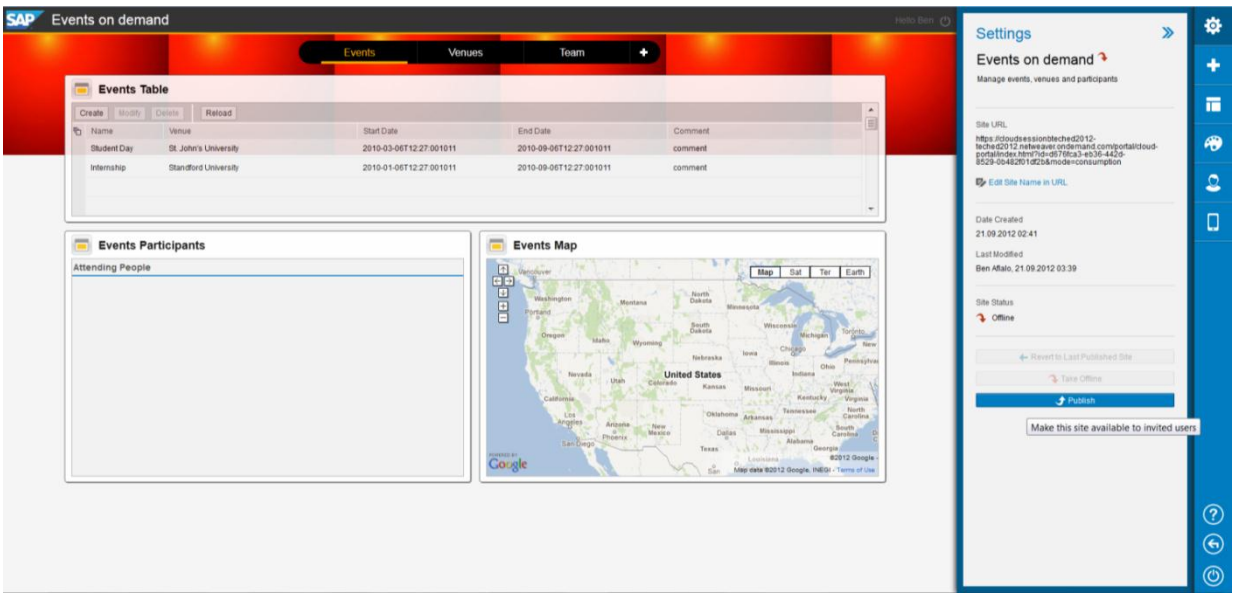
In this exercise we will preview our site in tablet, publish the site and view it in consumption mode.

Preview site in tablet

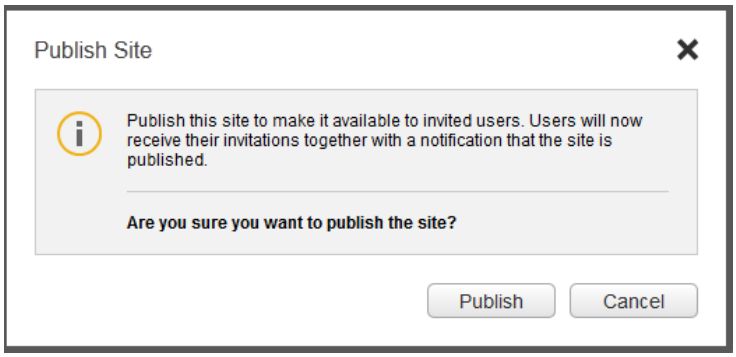
1. In order to see what your site will look like on mobile, click on the site simulation icon at the site Authoring panel
2. Choose landscape/ portrait view by clicking the mobile icons at the top right corner
3. Close the site simulation by clicking the blue arrow located at the top right corner of the panel

Publish the site

1. Click on the Settings icon in the site Authoring panel – blue strip at the right
2. Click the Publish button at the bottom of the panel



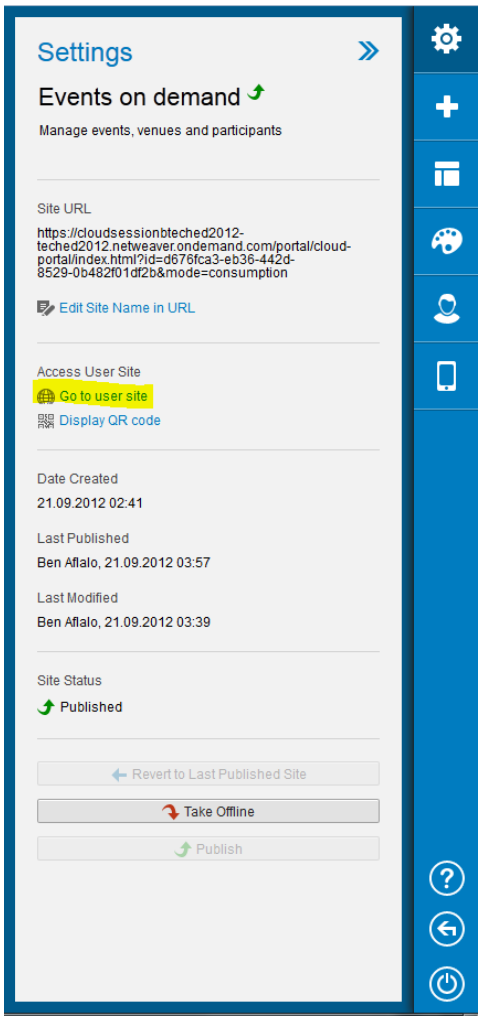
3. Approve publishing the site in the confirmation dialog



4. Close the dialog says the site is now published

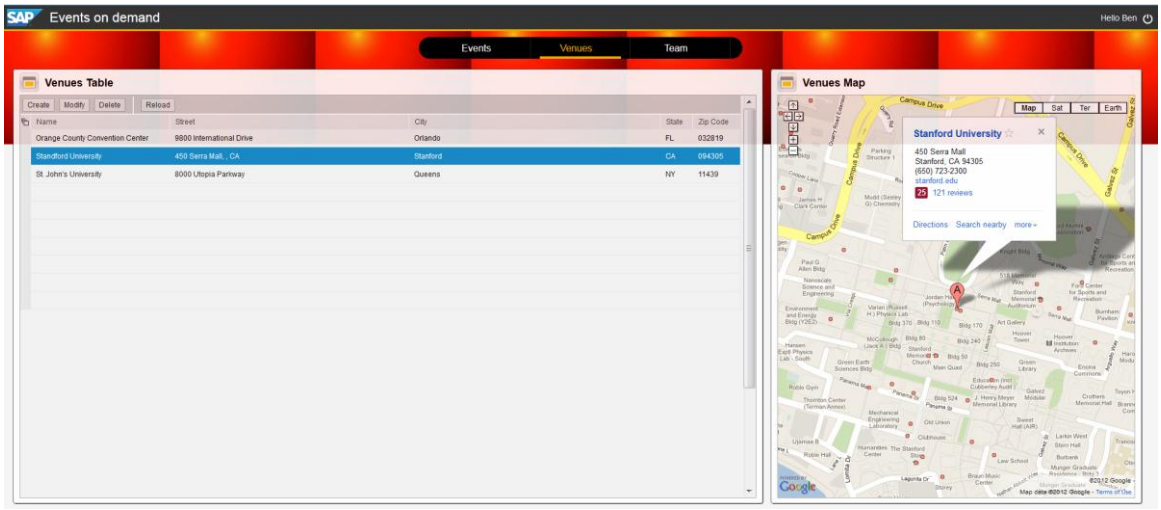
View site in consumption mode

5. Still in the Settings panel, right click on “Go to user site” and choose to open the link in new tab in order to see the site in consumption mode



Note: only published sites have this ability.

6. See the created site in consumption (navigate between pages, maximize widgets, select table entries, etc.)



This concludes today's workshop. Thank you and enjoy creating beautiful site using SAPUI5 and Cloud Portal!