Qlik Sense SaaS demo tool documentation of source code

Friday, August 12, 2016

10:36 AM

Source code

Please always have a look at [youTube](https://www.youtube.com/playlist?list=PLqJfqgR62cVAZxS34WGnByjASKrGf0Fpk) for an explanation of the concepts of the demo site (screenshot below) and deep dive technical videos.

Machine generated alternative text:
Multi-tenant Publisher 
29-06-2016 
Your SaaS platorm 
Insert Customers 
1 
QMC & API 
Videos 
Documentation 
Select the template 
Stream 
Everyone 
Templates 
Show 5 
0 
3 
Insert customers and users for your SaaS platform 
Customers 
You already selected a template, but please re-insert 
Generate 
Create a stream, copy app, insert script, 
reload and publish the app for each customer 
4 
Filter 
Completed 
Now test Single Sign On 
including dynamic groups 
your customers. 
1: Insert Customers 
O Initialize demo 
No customers 
• This demo simulates your SaaS 
platform. Therefore we first need 
to insert some customers and users 
• Or insert dummy customers (use 
these if you want to test row level 
security) 
Received 2 streams via QRS API from QIik Sense 
Columns 
Guid 
eaf82dOd-e6f1-450d-ab15-0138397f2f7b 
eclaib47-de82-4947-a66b-ddbOb11ecf65 
rows per page 
Received 2 apps via QRS API from QIik Sense 
Columns 
Created date 
07-2016 
Filter 
Delete 
Page 1 
of 1 
Copy app 
selected 
customers 
App 
Row level security 
(do not generate) 
Guid 
40ef70c9-be76-4844- 
ac6a-d53195146c85 
Stream 
Everyone 
Delete 
app 
O 
Template app 
for 
generation? 
O 

This is just an explanation of the source code. You should already understand the concept of multi-tenant SaaS solutions. For an explanation of OEM SaaS integration, see the [integration overview](https://onedrive.live.com/redir?page=view&resid=805405928A75727!1330&authkey=!ANwk5S8KPC__-dw&wd=target%28EMBEDDED%20ANALYTICS.one%7CBF2972BE-48A0-46FA-AF7A-F6D2F80CF06B%2FIntegration%20overview%20-%20Embedded%20analytics%7C407276F9-01A5-4CBF-AD54-7AEFE86739D8%2F%29onenote:https://d.docs.live.net/0805405928a75727/OneNote/Publications/QlikSense/EMBEDDED%20ANALYTICS.one#Integration%20overview%20-%20Embedded%20analytics&section-id={BF2972BE-48A0-46FA-AF7A-F6D2F80CF06B}&page-id={407276F9-01A5-4CBF-AD54-7AEFE86739D8}&end) page, this page also links to the [security](https://onedrive.live.com/redir?page=view&resid=805405928A75727!1330&authkey=!ANwk5S8KPC__-dw&wd=target%28EMBEDDED%20ANALYTICS.one%7CBF2972BE-48A0-46FA-AF7A-F6D2F80CF06B%2FSecurity%20integration%3A%20Provide%20Single%20Sign%20On%20and%20share%20access%20rights%7C51692548-CA14-46D7-BCE5-69C1473E44BD%2F%29onenote:https://d.docs.live.net/0805405928a75727/OneNote/Publications/QlikSense/EMBEDDED%20ANALYTICS.one#Security%20integration%20Provide%20Single%20Sign%20On%20and%20share%20access%20rights&section-id={BF2972BE-48A0-46FA-AF7A-F6D2F80CF06B}&page-id={51692548-CA14-46D7-BCE5-69C1473E44BD}&end) and [API](https://onedrive.live.com/view.aspx?cid=0805405928a75727&id=documents&resid=805405928A75727%211330&app=OneNote&authkey=!ANwk5S8KPC__-dw&&wd=target%28%2F%2FEMBEDDED%20ANALYTICS.one%7Cbf2972be-48a0-46fa-af7a-f6d2f80cf06b%2FProcess%20integration%20and%20automation%20using%20API%27s%7Ca49986de-9191-4321-88f3-c286fdff0b83%2F%29) automation whitepapers.

Sequence diagrams of the flow in the application

We present 2 diagrams

|  |  |
| --- | --- |
| * 1. A [diagram](http://knsv.github.io/mermaid/live_editor/#--QXBwRnVuY3Rpb25zOiBhIGZpbGUgcGVyIFNlbnNlIHR5cGUKTm90ZSByaWdodCBvZiBTZXJ2ZXI6IGEgZmlsZSBmb3IgUVBTLCBhcHBzLCBzdHJlYW1zLCBzeXN0ZW1ydWxlcwpOb3RlIHJpZ2h0IG9mIEFwcEZ1bmN0aW9uczogTWV0ZW9yIHNlcnZlciBzaWRlIGNvbW11bmljYXRlcyB3aXRoIFFsaWsgU2Vuc2UgdXNpbmcgUVJTIFJFU1QgYW5kIGVuZ2luZSAocXNvY2tzKSBBUEkgZm9yIHJlbG9hZHMuIEFsbCBjYWxscyBiZXR3ZWVuIE1ldGVvciBhbmQgU2Vuc2UgZ28gdmlhIGpzIGluIGltcG9ydHNcYXBpXHNlcnZlci4gVGhlIGFwcEZ1bmN0aW9ucyBhbHNvIGNvbnRhaW4gdGhlIGFwcCBnZW5lcmF0aW9uIGNoYWluIG9mIGV2ZW50cy4gKENvcHksIGRlbG) that shows how we get the list of apps in the Meteor.js based demo platform   2. A [diagram](https://is.gd/6ics9x) that shows how the user can generate apps. | Machine generated alternative text: Browser  Meteor  Generation template  OEMPartner template  Server  AppFunctions  Qiik Sense QRS API  QSocks engine API wrapper  Engine API  request introduction page  click go to demo which opens \generation page  op n generation.html and.js in \imp  Generation template consists of multiple  sub templates to keep things organized.  Key templates are OEMpartner.html,  which consist of multiple smaller ones.  The 2 app and streams tables on the right  are populated via the  Template. generation.helpers in  generation.js (appSettings,  streamSettings).  includes OEMPartner.html  the OEM partner template is the left side  of the screen if you open the \generation  page. This page has a lot of if then  statements to react to the current step of  the process. For example, if the user  inserts a customer, the screen should  update to reflect step 2: "select  template". The logic is calculated in the  helper in LJlHelpers.js, the template uses  the "spacebars/handlebars syntax" to show  or not. For example {fif  readyToGenerate}} show a div with texts  and buttons. {V if}}  show generation page |

The starting point for the API communication between Meteor and Qlik Sense is in the

|  |  |
| --- | --- |
| [imports/api/server folder](https://github.com/QHose/QRSMeteor/tree/master/imports/api/server). As you can see I splitted the code based on Sense resource type. QPS, QRS. But do note that the calls to QSOCKS within the app generation occur in [QRSFunctionsApp](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js). | Machine generated alternative text: QPSFunctions.js  QRSFunctionsApp.js  QRSFunctionsStream.js  QRSFunctionsSystemRules.js |
| Client and server split | Meteor uses both client side and server side javascript. See an explanation [here](https://www.discovermeteor.com/blog/what-goes-where/). Also note that both the client as well as the server has (mini) MongoDB running. |

Introduction to Multi-Tenant App publishing with Qlik Sense

Please always have a look at [youTube](https://www.youtube.com/playlist?list=PLqJfqgR62cVAZxS34WGnByjASKrGf0Fpk) for the quick summary, how does SaaS automation work. You will see the picture below. The meteor.js based demo platform tries to simulate the broker in your SaaS platform. This broker should fire the API calls to Qlik Sense so that your customers will get a personalized app which contains their data and in such a way that each customer can only see its own data. (enforced using the Qlik Sense security rules).

SaaS API automation — a deep dive 
Automated activities for each customer 
Create stream 
Copy app 
Replace script 
Reload data 
Publish app 
Template (app) 
Customer 
stream 
Sunny 
Petrol 
Your broker 
(REST) 
API 
Your configuration database 
Customer table 
Connection strings 
User & passwords 
User rights 
Definition of customer specific data model 
including custom fields and tables 
QIik Sense Enterprise 
App 
Sales 
åGZ 
Load the data using 
Full copy of table 
Delta load 
Real-time SQL query 
Connector to (big data) 
sources 
Tables 
Script Example 
Connect to Sunny Petrol 
database 
Select * from sales 
Select * from customer 
Select * 
from product 

The source code

[The generation function](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L21) in the broker in your SaaS platform could look like this

|  |  |
| --- | --- |
| Pseudo code | JavaScript code |
| This function is a loop, in which we iterate:   * For each selected template,   + for each customer     - Call function: Generate the app for this customer/template combination | Machine generated alternative text: for (const customer of customers) {  for (const templateApp of templateApps) {  generateAppForTemp1ate(temp1ateApp, customer) ; |

[The generate app for template function](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L33)

Do not look at the logging code, that is just that you can see all the generated REST Calls in the demo tool. The key part is here:

|  |  |
| --- | --- |
| * + [Create stream](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsStream.js#L48)   + [Copy app](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L278)   + [Replace the script and reload](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L69) (using Engine API via Qsocks)   + [Publish the app](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L376) | Machine generated alternative text: var  var  var  var  streamld  newAppId =  = checkStreamStatus(customer) // create a stream for the customer if it not already exists  copyApp(temp1ateApp. id, customer. name +  - reloadAppAndRep1aceScriptviaEngine(newAppId  result -  publishApp(newAppId, templateApp. name,  publishedAppId =  + templateApp.name);  streamld, customer. name); |

Some other use cases

[Get list of apps](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L332)

[Delete app](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L332)

Engine API connection

We use Qsocks the engine API wrapper to connect to Qlik Sense [here](https://github.com/QHose/QRSMeteor/blob/master/imports/api/server/QRSFunctionsApp.js#L67). In this code we open the app, get the script, replace it with the same script (for demo purposes, in real life you would inject your own database connection of custom script) and reload the app.

A good readable script without the extra logging can be found in the nice example from Loic:

[Elastic, Generate Qlik Sense apps](https://github.com/pouc/qlik-elastic)

Authentication

For demo purposes I tried to use a much authentication options as feasible for OEM integration cases.

|  |  |
| --- | --- |
| End users connect to the meteor proxy. See the [security deep dive video](https://www.youtube.com/playlist?list=PLqJfqgR62cVAZxS34WGnByjASKrGf0Fpk) for more information. Under the hood we use great and simple to use [QlikAuth](https://github.com/braathen/qlik-auth) modules of Rikard Braathen. See the meteor source code [here](https://github.com/QHose/QRSMeteor/blob/master/imports/server/qlikAuthSSO.js#L49). | Machine generated alternative text: AUTHENTICATION  Anonymous access mode  Authentication method  Authentication module redirect URI  No anonymous user  Ticket  http://2008ENT 3000/sso |
| We use header authentication to connect to the Qlik Sense QRS API. Therefore internal meteor to Sense communication goes via the special header proxy. | Machine generated alternative text: Virtual proxy edit  IDENTIFICATION  Description  Pre' X  Session inactivity timeout (minutes)  Session cookie header name  AUTHENTICATION  Anonymous access mode  Authentication meth0d  Header authentication header name  Header authentication dynamic user directory  Header auth  hdr  The prefix must be unique for all virtual proxies u  the IJRL (https://[nodel/[prefixl/)  30  X-Qlik-Session-hdr  The session cookie header name must be uniqu€  No anonymous user  Header authentication dynamic user directory  hdr-usr |
| We use certificates to connect to the QPS API (logout the users), and to connect to the engine API with Qsocks. All the communication function use the config as defined in the [\imports\api\config.js](https://github.com/QHose/QRSMeteor/blob/master/imports/api/config.js#L59) | Machine generated alternative text: if (Meteor.isServer) {  console. log( 'This Sense SaaS demo tool uses this config as defined in the settings-XYZ.json file in the root folder:  Meteor . setting  import crypto from 'crypto' ;  import fs from 'fs';  var senseConfig  "host": Meteor. settings . public. host,  'SenseServerInterna1LanIP": Meteor. settings . private. SenseServerInterna1LanIP,  'port": Meteor. settings . private. port,  'useSSL": Meteor. settings. private. useSSL,  " xrfkey"  'authentication": Meteor. settings . private. authentication,  'virtualProxy": Meteor. settings . private. virtualProxy,  // used to connect via REST to Sense,  'virtualProxyC1ientUsage": Meteor. settings. public. virtualProxyC1ientUsage,  "headerKey": Meteor. settings . private. headerKey,  "headerVa1ue": Meteor. settings . private. headerVa1ue,  'isSecure": Meteor. settings . private.isSecure,  'UDC": Meteor. settings. private. UDC  we authenticate via a http header.  not    Machine generated alternative text: // used for QSocks, the engine API javascript wrapper  var _ engineConfig  host:  senseConfig . SenseServerInterna1LanIP ,  isSecure:  senseConfig. isSecure,  port: Meteor. settings . private. enginePort,  headers: {  'X-Q1ik-User' : Meteor. settings . private. engineHeaders,  key :  certs . key ,  cert:  certs . cert,  passphrase: Meteor. settings . private. passphrase,  rejectUnauthorized: false,  // Don't reject self-signed certs  null  appname : |

Authorization

See the security rules implemented [here](http://saasdemo.qlik.com/securityRules)