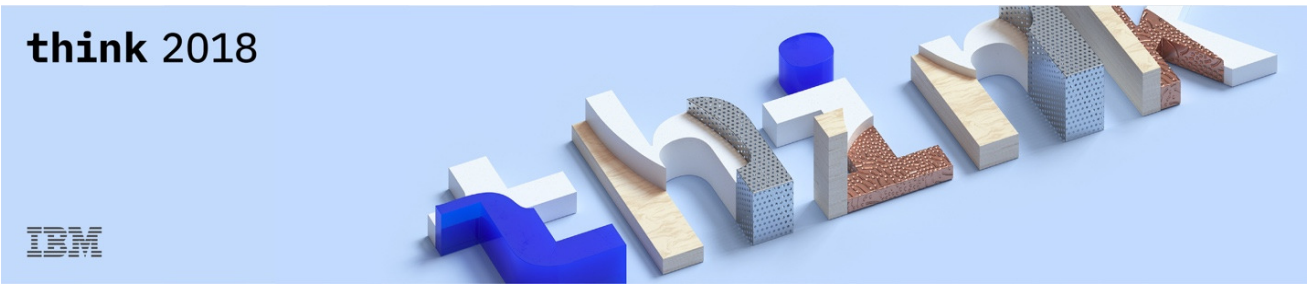


Lab Center – Hands-On Lab

Session 7721

IBM Planning Analytics 2.0 SDK

How TM1Web API Can Be Applied to Create a Custom Dashboard

Hubert Heijkers, IBM, hubert.heijkers@nl.ibm.com

Table of Contents

Getting ready

To be able to give you the best experience possible, and to allow us, authors, to be able to make last minute changes to the setup, samples and instructions for this Hands-On Lab, and because in our experience there is always something that we want to change last minute😉, we've build in a 'get out of jail free card'.

As such there are a couple of steps that need to be executed before your machine is ready.

1 – Grabbing the latest files for the update

The latest versions of the files needed on this box, and the sources you'll be working with in this lab, are all kept together in one Git repository on github.com.

Open a command box and execute the following command to grab the content of this repository:

```
git clone https://github.com/hubert-heijkers/think2018-7721
```

Now let's go to the folder holding the actual update:

```
cd think2018-7721\vmupdate
```

2 – Updating the Virtual Machine

Next, we'll execute a little batch file that updates a bunch of files and does some set up needed for the lab later. This update can be executed by typing the following command in the command box:

```
vmupdate.bat
```

Your VM is now up to date. You can now find the latest version of this document here:

```
C:\HOL-TM1SDK\Documents
```

Having an electronic copy of the instructions, most notably the Word document, might come in handy later when you'll be 'writing' some code;-).

That's all, enjoy the lab!

Creating a Custom Dashboard using TM1Web API

This section describes how the **TM1Web Script Library** can be used to embed TM1Web components (Websheets and CubeViews) into a custom web application. Also, you will be able to try TM1Web's **URL API**.

The application will demonstrate how context can be synchronized between TM1Web API components and how a Websheet can be effectively used to enhance user interaction.

Configuration

TM1Web's API is enabled by default and already running on your lab system. No configuration is required.

TM1 is installed in the default location:

```
C:\Program Files\IBM\Cognos\tm1_64
```

The folder containing TM1Web's 'application', as well as other services, named '`webapps`', is located within that installation and the TM1Web application can be found here:

```
C:\Program Files\IBM\Cognos\tm1_64\webapps\tm1web
```

This is the root folder in which we'll add our work in this lab.

You have the choice to follow along these lab instructions and build the dashboard we are going to create from scratch or you can borrow and steal, or even nick the whole end result, from files stored in the following location:

```
C:\HOL-TM1SDK\tm1web
```

Simply copying the content of this folder into tm1web folder containing the TM1Web application would get you from A to Z at once.

If you are not, which is the whole purpose of this lab obviously, follow along and make the instructor aware of any issues and/or difficulties you might run into.

Note for TM1 10.2.2 FP1: There is a bug in the shipping version of TM1Web, which prevents cross domain scripting calls to be allowed by TM1Web. To enable this by default, the tm1web_config.xml, located in <tm1installdir>\webapps\tm1web\WEB-INF\configuration, needs to be modified.

The new value for CrossDomainAccessList needs to be changed to "*":

```
<add key="CrossDomainAccessList" value="*" />
```

In Versions TM1 10.2.2 FP2 and later this issue has been addressed.

Preparation

This sample uses the CarSales database, which is already started, as a service, on your machine.

Note that some required files for the sample web application have already been placed in the appropriate directories and will be referenced in the code later during this exercise. Most notably:

C:\Program Files\IBM\Cognos\tm1_64\webapps\tm1web\scripts\dashboard\WidgetContainer.js

Which provides a container control, allowing easier styling of widgets.

Building your own Dashboard

You'll be creating a dashboard, and extend it, in a couple of steps.

Step 1: The Dashboard and a Login Screen

First, we'll create a file containing the code behind the login dialog that we'll need to show to allow a user to login to the system. Create a new java-script (.js) file name `LoginDialog.js` in the `C:\Program Files\IBM\Cognos\tm1_64\webapps\tm1web\scripts\dashboard\` folder.

Add the following code to the `LoginDialog.js` file:

```
// A login dialog based off of the TM1 Web JavaScript Library login dialog.
define([
    "dojo/_base/declare",
    "dojo/_base/lang",
    "dojo/_base/xhr",
    "dojo/dom-class",
    "tm1web/api/session/LoginDialog",
    "dijit/form/TextBox",
    "dijit/form/RadioButton",
    "dijit/form/CheckBox",
    "dijit/form/Select"
], function(declare, lang, xhr, domClass, LoginDialogBase) {

    var LoginDialog = declare(LoginDialogBase, {

        // We don't want the admin host or TM1 server to be visible,
        // so we disable that here.
        adminHostVisible: false,
        tm1ServersVisible: false,

        // Use CarSales as the TM1 server to authenticate against.
        tm1Server: "CarSales",

        // Shows the login dialog and sets the page's background.
        show: function() {
            this.inherited(arguments);

            domClass.add(document.body, "loginBackground");
        },

        // Hides the login dialog and removes the page's background.
        hide: function() {
            this.inherited(arguments);
```

```
            domClass.remove(document.body, "loginBackground");
        },

        // Attempts to login to TM1 Web.
        login: function() {
            // If the specified information is not valid or the login button
            // is disabled, abort the login process.
            if(!this._isValid() || this._loginButton.get("disabled")) {
                return;
            }

            this.set("loginDisabled", true);

            // REST request to log in to TM1 Web and execute the _onLogin
            // method when login completes.
            xhr.post({
                url: "http://localhost:9510/tm1web/api/TM1Service/login",
                handleAs: "json",
                content: {
                    param0: this.get("adminHost"), // param0 is admin host
                    param1: this.get("tm1Server"), // param1 is TM1 server
                    param2: this.get("username"), // param2 is username
                    param3: this.get("password") // param3 is password
                },
                load: lang.hitch(this, "_onLogin")
            });
        },

        // Handles the response from the TM1 Web login call.
        _onLogin: function(response) {
            // If the response has a reply (indicating login succeeded)
            if(response.reply) {
                this._onSubmit();

                // Get the session token from the response
                var sessionId = response.reply.sessionToken;

                // Notifiy listeners that a login has occurred and pass along
                // the session token/id.
                this.onLogin(sessionId);
            }
            else { // If no result, login failed
                this.set("loginDisabled", false);
                alert("Login failed...");
            }
        }
    });

    return LoginDialog;
});
```

This code calls a TM1Service to login with the credentials provided by the user. TM1 Adminhost and TM1 Server selection are disabled in this dialog.

A session token, which is provided from the response in case of a successful login, will be used subsequently to identify the user on the TM1 Server.

Next, we'll create another file, names `main.js`, in the same `webapps\tm1web\scripts\dashboard\` folder, in which we'll put the code responsible for creating the login dialog as well as the creation of the dijit container, which will contain the TM1Web API components.

Add the following code to the `main.js` file:

```
// Entry point for the application.
define([
    "dijit/layout/BorderContainer",
    "./LoginDialog",
    "dojo/domReady!"
], function(BorderContainer, LoginDialog) {

    // Callback after login which loads the dashboard using the TM1 Web
    // session token from the login step.
    function loadDashboard(sessionToken) {
        // Create a container for the menu bar and dashboard and place it
        // on the document body.
        var borderContainer = new BorderContainer({
            style: {
                width: "100%",
                height: "100%",
                overflow: 'auto'
            }
        }).placeAt(document.body);
    };

    // Create a login dialog that loads the dashboard once login completes.
    var loginDialog = new LoginDialog({
        onLogin: loadDashboard,
        onHide: function() {
            loginDialog.destroyRecursive();
            loginDialog = null;
        }
    });

    loginDialog.show();
});
```

Last, but not least, we'll need the dashboard itself. For that we'll create a HTML file, named `dashboard.html`, in the `webapps\tm1web\` folder itself!

dashboard.html is the entry point for the application and the only HTML file that will be used. It references dojo as well as what is in main.js.

Add the following code to the dashboard.html file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
	<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

	<!-- Load all of the stylesheets required for TM1 Web and Dojo -->
	<link href="scripts/dojo/resources/dojo.css" rel="stylesheet" type="text/css"
/>
	<link href="scripts/dijit/themes/claro/claro.css" rel="stylesheet"
type="text/css/>
	<link type="text/css" rel="stylesheet"
href="scripts/tm1web/themes/flat/flat.css">
	<link href="scripts/tm1web/themes/flat/workbook/Worksheet.css"
rel="stylesheet" type="text/css" />
	<link href="scripts/tm1web/themes/flat/cubeview/CubeViewer.css"
rel="stylesheet" type="text/css" />

	<!-- Load all of the stylesheets required for this application -->
	<link href="css/style.css" rel="stylesheet" type="text/css" />
	<!-- Load the dojo configuration file. -->
	<!--<script type="text/javascript"
src="scripts/tm1web/common/DjConfig.js"></script>-->
	<script>
	var dojoConfig = {
		paths: {
			"dashboard": "../dashboard"
		}
	};
	</script>

	<!-- Load Dojo. -->
	<script src="scripts/dojo/dojo.js" data-dojo-config="locale:'en-us'"></script>
	<!--<script type="text/javascript" src="scripts/dashboard/main.js"></script>--
>

	<!-- Load our application -->
	<script type="text/javascript" language="javascript">
		require(["dashboard/main"]);
	</script>
</head>
<body class="claro tm1web">
</body>
</html>
```

To ensure that the login page is working correctly, open a browser and navigate to:

`http://localhost:9510/tm1web/dashboard.html`

The credentials for the CarSales database are, our infamous 'admin' and 'apple' pair!

After logging in, the page should remain blank.

Step 2: Displaying a Websheet

In this section a TM1Web API Websheet component will get added to the dashboard.

Create another file, names `Dashboard.js`, in the `webapps\tm1web\scripts\dashboard\` folder again, in which we'll put the code responsible for displaying the TM1Web API Websheet components and synchronize the subsets.

Add the following code to the `Dashboard.js` file:

```
define([
    "dojo/_base/declare",
    "dojo/_base/lang",
    "dojo/_base/array",
    "dijit/layout/ContentPane",
    "tm1web/websheet/Workbook",
    "./WidgetContainer"
], function(declare, lang, array, ContentPane, Workbook, WidgetContainer) {

    var Dashboard = declare(ContentPane, {

        // A session token corresponding to a TM1 Web session.
        sessionToken: null,

        // Creates the DOM required for this widget. A Dojo-ism.
        buildRendering: function() {
            // Execute the base class method.
            this.inherited(arguments);

            this.domNode.style.padding = "20px";
        },

        // Actions to take after the widget has been created. A Dojo-ism.
        postCreate: function() {
            // Execute the base class method.
            this.inherited(arguments);
            this._createWorkbooks();
        },

        _createWorkbook: function(args) {
            // Instantiate a container with a workbook inside
            // and place it on the dashboard.
            var widget = new WidgetContainer({
                widget: new Workbook({
```

```
                    region: args.region,
                    style: args.style,
                    sessionToken: args.sessionToken,
                    path: args.path,
                    tm1Server: args.tm1Server
                }),
                style: args.containerStyle
            }).placeAt(this.containerNode);

            widget.startup();

            return widget;
        },

        _createWorkbooks: function() {

            var containers = this._widgetContainers = [];

            var widget = this._createWorkbook({
                containerStyle: "width:1050px; height: 550px;",
                region: "center",
                style: "height: 100%; width: 100%;",
                sessionToken: this.sessionToken,
                path: "Applications/Applications/dynamicReport.xlsx",
                tm1Server: "CarSales"
            });

            containers.push(widget);

        },

    });

    return Dashboard;
});
```

For `Dashboard.js` to get utilized, `main.js` needs to be modified. Make the changes, shown in red below, to the code in the `main.js` file:

```
// Entry point for the application.
define([
    "dijit/layout/BorderContainer",
    "./LoginDialog",
    "./Dashboard",
    "dojo/domReady!"
], function(BorderContainer, LoginDialog, Dashboard) {

    // Callback after login which loads the dashboard using the TM1 Web
    // session token from the login step.
```

```
    function loadDashboard(sessionToken) {
        // Create a container for the menu bar and dashboard and place it
        // on the document body.
        var borderContainer = new BorderContainer({
            style: {
                width: "100%",
                height: "100%",
                overflow: 'auto'
            }
        }).placeAt(document.body);

        var dashboard = new Dashboard({
            region: "center",
            style: "height: 100%; width: 100%",
            sessionToken: sessionToken
        });

        borderContainer.addChild(dashboard);
    };

    // Create a login dialog that loads the dashboard once login completes.
    var loginDialog = new LoginDialog({
        onLogin: loadDashboard,
        onHide: function() {
            loginDialog.destroyRecursive();
            loginDialog = null;
        }
    });

    loginDialog.show();
});
```

After completing this part of the exercise, we can once again test the results of our work so far. Open a browser and once again navigate to:

`http://localhost:9510/tm1web/dashboard.html`

Remember, the credentials for the CarSales database are, our infamous ‘admin’ and ‘apple’ pair!

After logging in, the page should now show a single Websheet.

Step 3: Synchronize subsets for multiple Websheets

In this section two more Websheets are added to the dashboard and subset element selections changes will end up being synchronized.

Make the changes, shown in `red` below, to the code in the `Dashboard.js` file:

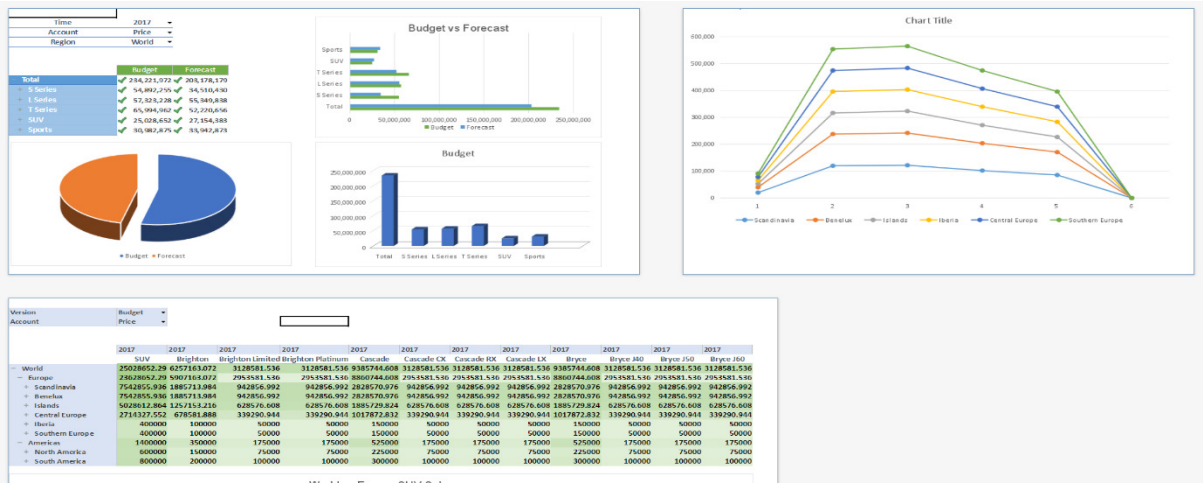
```
define([
    "dojo/_base/declare",
    "dojo/_base/lang",
```

```
    "dojo/_base/array",
    "dijit/layout/ContentPane",
    "tm1web/websheet/Workbook",
    "./WidgetContainer"
], function(declare, lang, array, ContentPane, Workbook, WidgetContainer) {

    var Dashboard = declare(ContentPane, {

        // A session token corresponding to a TM1 Web session.
        sessionToken: null,

        // Creates the DOM required for this widget. A Dojo-ism.
        buildRendering: function() {
            // Execute the base class method.
            this.inherited(arguments);

            this.domNode.style.padding = "20px";
        },

        // Actions to take after the widget has been created. A Dojo-ism.
        postCreate: function() {
            // Execute the base class method.
            this.inherited(arguments);
            this._createWorkbooks();
        },

        _createWorkbook: function(args) {
            var onSubnmChange = lang.hitch(this, "_onSubnmChange");
            // Instantiate a container with a workbook inside
            // and place it on the dashboard.
            var widget = new WidgetContainer({
                widget: new Workbook({
                    region: args.region,
                    style: args.style,
                    sessionToken: args.sessionToken,
                    path: args.path,
                    tm1Server: args.tm1Server,
                    onTitleDimensionElementChange: function(subnmInfo) {
                        // When a title is changed, apply the same information to
                        // other widgets in the dashboard.
                        onSubnmChange(this, subnmInfo);
                    }
                }),
                style: args.containerStyle
            }).placeAt(this.containerNode);

            widget.startup();

            return widget;
        },
```

```
_createWorkbooks: function() {

    var containers = this._widgetContainers = [];

    var widget = this._createWorkbook({
        containerStyle: "width:1050px; height: 550px;",
        region: "center",
        style: "height: 100%; width: 100%;",
        sessionToken: this.sessionToken,
        path: "Applications/Applications/dynamicReport.xlsx",
        tm1Server: "CarSales"
    });

    containers.push(widget);

    widget = this._createWorkbook({
        containerStyle: "width:1280px; height: 640px; top: 620px;",
        region: "center",
        style: "height: 100%; width: 100%;",
        sessionToken: this.sessionToken,
        path: "Applications/Applications/SUVWorldSales.xlsx",
        tm1Server: "CarSales"
    });

    containers.push(widget);

    widget = this._createWorkbook({
        containerStyle: "width:850px; height: 550px; right: 30px;",
        region: "center",
        style: "height: 100%; width: 100%;",
        sessionToken: this.sessionToken,
        path: "Applications/Applications/Chart_only.xlsx",
        tm1Server: "CarSales"
    });

    containers.push(widget);

},

// Returns a collection of all widgets by aggregating the
// widgets inside of all widget containers on the dashboard.
getWidgets: function() {
    // array.map() iterates all the elements in an array, passing them
    // to the callback function and then returning a new array with any
    // of the modified results.
    return array.map(this._widgetContainers, function(widgetContainer) {
        return widgetContainer.widget;
    });
},

_onSubnmChange: function(source, subnmInfo) {
```

```
            subnmInfo.rawIndex = null;
            subnmInfo.columnIndex = null;
            array.forEach(this.getWidgets(), function(widget) {
                if(widget !== source) {
                    widget.set("subset", subnmInfo);
                }
            });
        }

    });

    return Dashboard;
});
```

After these changes are saved, the dashboard can be tested again. This time three Websheets should appear. A change of a title element in one of the Websheets will trigger the other two Websheets to reflect that change.



Step 4: Adding a menu

Finally, we'll add some menu options to the dashboard allowing the user to add products to the Products dimension.

Create a file named `MenuBar.js`, in the `webapps\tm1web\scripts\dashboard\` folder and add the following code to it:

```
// Menu bar widget containing various actions such as recalculate, add a product, and
logout.
define([
    "dojo/_base/declare",
    "dojo/_base/lang",
    "dojo/_base/array",
    "dojo/_base/xhr",
```

```
    "dijit/MenuBar",
    "dijit/DropDownMenu",
    "dijit/PopupMenuBarItem",
    "dijit/MenuItem",
    "dijit/Dialog",
    "tm1web/websheet/Workbook"
], function(declare, lang, array, xhr, MenuBarBase, DropDownMenu, PopupMenuBarItem,
        MenuItem, Dialog, Workbook) {

    var MenuBar = declare(MenuBarBase, {

        // A session token corresponding to a TM1 Web session.
        sessionToken: null,

        // Creates the DOM required for this widget. A Dojo-ism.
        buildRendering: function() {
            // Execute the base class method.
            this.inherited(arguments);

            this.domNode.className += " vision";
        },

        // Actions to take after the widget has been created. A Dojo-ism.
        postCreate: function() {
            // Execute the base class method.
            this.inherited(arguments);
            this._createActionsDropDown();
        },

        _createActionsDropDown: function() {
            // Create a drop down menu with various actions.
            var actionMenu = DropDownMenu();
            actionMenu.domNode.className += " vision";

            // Add a 'Recalculate' action.
            actionMenu.addChild(new MenuItem({
                label: "Recalculate",
                onClick: lang.hitch(this, "_recalculate")
            }));

            // Add an 'Add Product' action.
            actionMenu.addChild(new MenuItem({
                label: "Add Product",
                onClick: lang.hitch(this, "_addProduct")
            }));

            // Add a 'Logout' action.
            actionMenu.addChild(new MenuItem({
                label: "Logout",
                onClick: lang.hitch(this, "_logout")
            }));
```

```
            // Create the top level menu button and attach the
            // drop down menu as the popup when clicked.
            this.addChild(new PopupMenuBarItem({
                label: "Actions",
                popup: actionMenu
            }));
        },

        // Recalculate all of the widgets on the dashboard.
        _recalculate: function() {
            array.forEach(this.dashboard.getWidgets(), function(widget) {
                widget.recalculate();
            });
        },

        // Opens a modal dialog containing a websheet which allows users
        // to add a new product.
        _addProduct: function() {
            var dialog = null;

            var addProduct = new Workbook({
                region: "center",
                style: "height: 100%; width: 100%;",
                sessionToken: this.sessionToken,
                path: "Applications/Applications/AddProduct.xlsm",
                tm1Server: "CarSales",
                style: "width: 500px; height: 200px; overflow: hidden;"
            });

            // Create a modal dialog which will contain the websheet.
            var self = this;
            dialog = new Dialog({
                onHide: function() {
                    // When the dialog is hidden, destroy the websheet so it can be
                    // garbage collected and recalculate all widgets on the
dashboard.
                    addProduct.destroy();
                    dialog = null;
                    self._recalculate();
                }
            });

            // Add the websheet to the dialog.
            dialog.containerNode.style.padding = "0";
            dialog.containerNode.appendChild(addProduct.domNode);

            // Show the dialog and initiate the websheet lifecyle.
            dialog.show();
            addProduct.startup();
        },
```

```
        // Issues a logout command to TM1 Web for the current session.
        _logout: function() {
            xhr.post({
                async: false,
                url: "http://localhost:9510/tm1web/api/TM1Service/logout",
                handleAs: "json",
                content: {
                    param0: this.get("sessionToken")
                }
            });
            document.location.reload();
        }
    });

    return MenuBar;
});
```

This code makes use of a dijit component called MenuBar. A menu item for adding a product is added. Clicking this menu item will display a floating Websheet, which allows the user to enter a new product and with an Action Button to run a TI process, which will add the new product to the Products dimension.

The TM1 components used for this action are part of the database already.

Websheet:	Applications/Products/Add Product
TI Process:	Add_Product

`MenuBar.js` will be used by `main.js` and therefore needs to be modified. Make the changes, shown in red below, to the code in the `main.js` file:

```
// Entry point for the application.
define([
    "dijit/layout/BorderContainer",
    "./LoginDialog",
    "./MenuBar",
    "./Dashboard",
    "dojo/domReady!"
], function(BorderContainer, LoginDialog, MenuBar, Dashboard) {

    // Callback after login which loads the dashboard using the TM1 Web
    // session token from the login step.
    function loadDashboard(sessionToken) {
        // Create a container for the menu bar and dashboard and place it
        // on the document body.
        var borderContainer = new BorderContainer({
            style: {
                width: "100%",
                height: "100%",
```

```
                overflow: 'auto'
            }
        }).placeAt(document.body);

        var dashboard = new Dashboard({
            region: "center",
            style: "height: 100%; width: 100%",
            sessionToken: sessionToken
        });

        var menuBar = new MenuBar({
            region: "top",
            dashboard: dashboard,
            sessionToken: sessionToken
        });

        borderContainer.addChild(menuBar);
        borderContainer.addChild(dashboard);
    };

    // Create a login dialog that loads the dashboard once login completes.
    var loginDialog = new LoginDialog({
        onLogin: loadDashboard,
        onHide: function() {
            loginDialog.destroyRecursive();
            loginDialog = null;
        }
    });

    loginDialog.show();
});
```

Now that we have all the components in place we can test the finished dashboard. After logging in, the menu on the top left should contain three items:

- Recalculate
- Add Product
- Logout

When selecting “Add Product”, a smaller popup window appears, allowing input for a new product name and a pick list for the consolidated element, to which the new product will roll up to.

After entering one or more new products and closing the dialog, the new products will appear in the Websheet to the left top of the dashboard.

Logout will close the session and bring the user back to the login screen.

Challenge

Haven't had enough yet? 😉

There are two more items in the TM1 Database, which can be used to add functionality for deleting a product.

Websheet:	Applications/Products/Delete Product
TI Process:	Remove_Product

After completing the exercises, you should be able to add a menu item for deleting a product to the menu and, when selected, have it subsequently remove that product from the database.

PS The complete version of the dashboard provided in the C:\HOL-TM1SDK\tm1web directory includes the solution to this challenge as well!

Enjoy!

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

We Value Your Feedback!

- Don't forget to submit your Think 2018 session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
- Access the Think 2018 agenda to quickly submit your surveys from your smartphone, laptop or conference kiosk.