Lesson 17: Jasper Reports (continued)

Created by Jacques Marais, last modified on Sep 13, 2018

> *As a System Admin I want to view a report of purchases made.*

Lesson Outcomes

By the end of this lesson you should:

- Know how to include Jasper reports that will be viewed on the front-end
- Understand and know how to use parameters in Jasper reports
- Know how to use custom parameter filters
- Know how to use custom menu items for reports or groups of reports

Modified or Added App Files

`./jasper-reports/AllFarmerPurchases/leaf_banner_gray_0.png`

`./jasper-reports/AllFarmerPurchases/master.jrxml`

`./jasper-reports/AllFarmerPurchases/report.json`

`./jasper-reports/FarmerUserPurchases/leaf_banner_gray_0.png`

`./jasper-reports/FarmerUserPurchases/master.jrxml`

`./jasper-reports/FarmerUserPurchases/report.json`

`./utilities/ReportParameterFilters.mez`

Creating a Jasper Report for the Purpose of Viewing on the Frontend

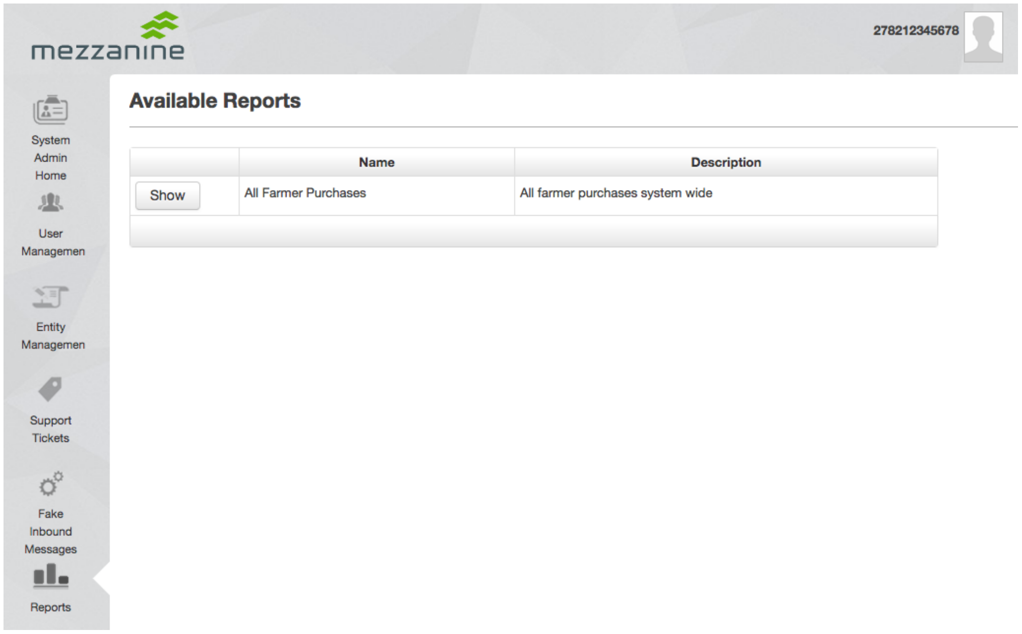
To add a report to a Helium application we must first create a folder called **`jasper-reports`** in the project root directory. Each report will then have its own subfolder with the jrxml files, images and a Helium specific **`report.json`** file. The main report should be named **`master.jrxml`**. There is no convention for naming of subreport files or images.

The **`report.json`** file will contain the configuration information required by Helium for the specific report. This will include which user has access to the report, which parameters are needed by the report and whether the report will be using its own menu item, a report group menu item or the default menu item for reports. For our first example we will consider the report under the **`AllFarmerPurchases`** folder. This report is accessible using the default report menu item. It is accessible for only the `"System Admin"` role, and does not make use of any parameters. The content of the **`report.json`** file is as follows:

```

{
    "name":"All Farmer Purchases",
    "description":"All farmer purchases system wide",
    "rolesAllowed":["System Admin"]
}
```

This results in the following menu item and Helium provided view for the `"System Admin"` role:

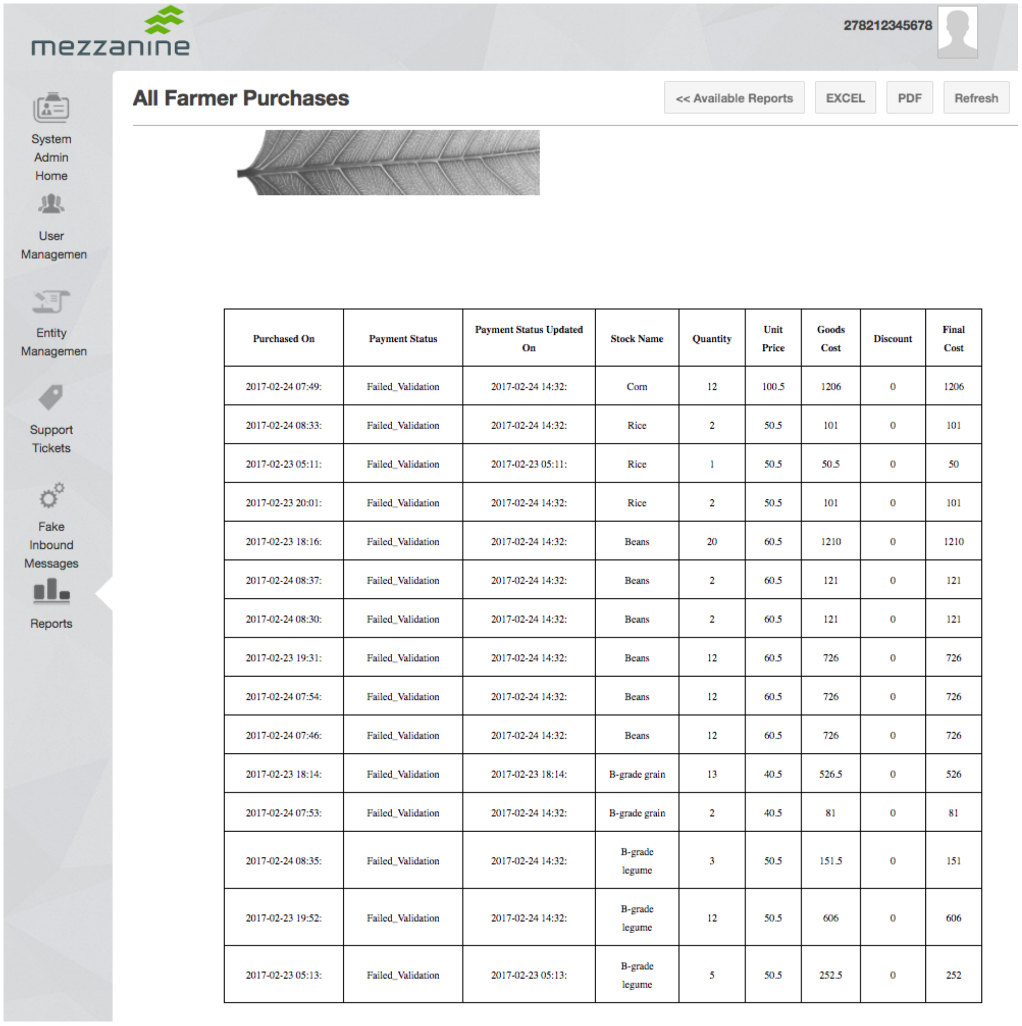


Available Reports

	Name	Description
Show	All Farmer Purchases	All farmer purchases system wide

Any other reports that also use the default menu item will appear in this table. The default menu item for reports is always ordered to be the last menu item in the menu.

Using the Show button, our report will be displayed along with options to download the report in PDF and EXCEL formats.



All Farmer Purchases

Purchased On	Payment Status	Payment Status Updated On	Stock Name	Quantity	Unit Price	Goods Cost	Discount	Final Cost
2017-02-24 07:49:	Failed_Validation	2017-02-24 14:32:	Corn	12	100.5	1206	0	1206
2017-02-24 08:33:	Failed_Validation	2017-02-24 14:32:	Rice	2	50.5	101	0	101
2017-02-23 05:11:	Failed_Validation	2017-02-23 05:11:	Rice	1	50.5	50.5	0	50
2017-02-23 20:01:	Failed_Validation	2017-02-24 14:32:	Rice	2	50.5	101	0	101
2017-02-23 18:16:	Failed_Validation	2017-02-24 14:32:	Beans	20	60.5	1210	0	1210
2017-02-24 08:37:	Failed_Validation	2017-02-24 14:32:	Beans	2	60.5	121	0	121
2017-02-24 08:30:	Failed_Validation	2017-02-24 14:32:	Beans	2	60.5	121	0	121
2017-02-23 19:31:	Failed_Validation	2017-02-24 14:32:	Beans	12	60.5	726	0	726
2017-02-24 07:54:	Failed_Validation	2017-02-24 14:32:	Beans	12	60.5	726	0	726
2017-02-24 07:46:	Failed_Validation	2017-02-24 14:32:	Beans	12	60.5	726	0	726
2017-02-23 18:14:	Failed_Validation	2017-02-23 18:14:	B-grade grain	13	40.5	526.5	0	526
2017-02-24 07:53:	Failed_Validation	2017-02-24 14:32:	B-grade grain	2	40.5	81	0	81
2017-02-24 08:35:	Failed_Validation	2017-02-24 14:32:	B-grade legume	3	50.5	151.5	0	151
2017-02-23 19:52:	Failed_Validation	2017-02-24 14:32:	B-grade legume	12	50.5	606	0	606
2017-02-23 05:13:	Failed_Validation	2017-02-23 05:13:	B-grade legume	5	50.5	252.5	0	252

Specifying and Using Report Parameters

As a next step we will make a copy of the report for viewing by the `"Farmer"` role. We will modify the report query to make use of the built-in `USER_APP_ROLE_ID` parameter provided by Helium. This will limit the purchases shown to those of the current farmer user. Note that the mentioned parameter is provided by Helium and does not need to be mentioned in the **`report.json`** file. The updated report query now contains the following where clause:

```
where farmer._id_ = $P{USER_APP_ROLE_ID}::uuid
```

The **`report.json`** file is as follows:

```
{
    "name":"Your Purchases",
    "description":"Your purchases",
    "rolesAllowed":["Farmer"]
}
```

We can now apply further filtering. Filtering criteria can be applied to show only the following purchases:

- Purchases between a start and end date
- Purchases from a specific shop
- Purchases over a certain cost

We modify our **report.json** file as follows:

```
{
    "name":"Your Purchases",
    "description":"Your purchases",
    "rolesAllowed":["Farmer"],
    "parameters":[
        {
            "parameterName":"startDate",
            "displayName":"Start date",
            "type":"date"
        },
        {
            "parameterName":"endDate",
            "displayName":"End date",
            "type":"date"
        },
        {
            "parameterName":"shopId",
            "displayName":"Shop",
            "type":"uuid",
            "searchType":"Shop.name"
        },
        {
            "parameterName":"minCost",
            "displayName":"Minimum purchase cost",
            "type":"string"
        }
    ]
}
```

Note how we used a **string** type for the `minCost` parameter. Helium supports the following paramater types:

- **boolean**
- **date**
- **datetime**
- **string**
- **uuid**

Any parameter that is of a different type, can be captured as a **string**.

Also note that for each parameter added to the **report.json** file, a parameter should also be added to the Jasper report. Failure to do so will result in the report not compiling:

```
|--------------------------------------------------------------
| The Jasper reports for the project "helium-tut" failed to
|--------------------------------------------------------------
| #     | Message
```

For parameters in the **report.json** file that are of type **uuid**, the matching parameter in the Jasper

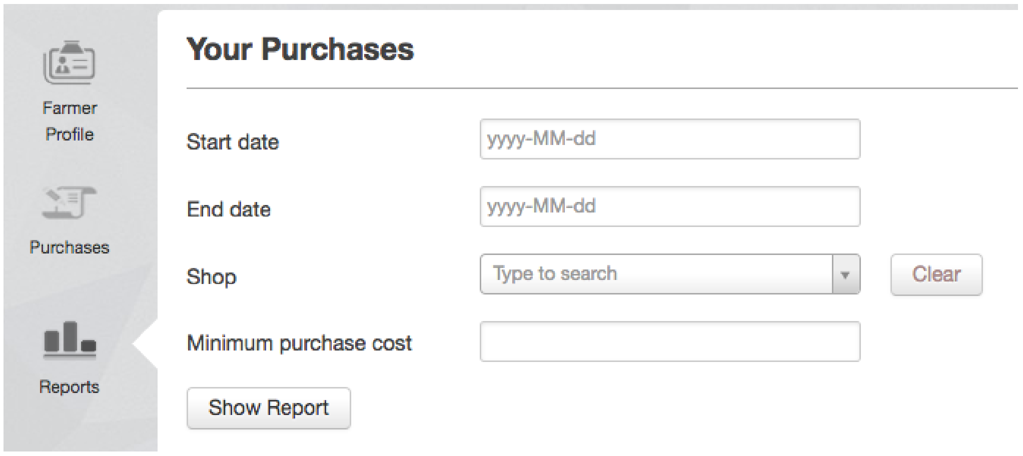
```
|-------------------------------------------------------------
| 1     | [Your Purchases] The master.jrxml file doesn't dec
| 2     | [Your Purchases] The master.jrxml file doesn't dec
| 3     | [Your Purchases] The master.jrxml file doesn't dec
| 4     | [Your Purchases] The master.jrxml file doesn't dec
|-------------------------------------------------------------
The last command completed in 0.22 seconds
```

report itself should be of type **java.lang.String**.

Even though we have specified a **uuid** type for the shopId parameter in the **report.json** file, this is mostly for Helium to provide an appropriate input widget and does not represent the type of the parameter that is received by the Jasper report. In this case the type of the parameter on the Jasper Report side can be specified as **java.lang.String** and cast to a **uuid** in the report query as appropriate.

```
<parameter name="shopId" class="java.lang.String">
    <parameterDescription><![CDATA[]]></parameterDescription
    <defaultValueExpression><![CDATA[""]]></defaultValueExpr
</parameter>
```

With the parameters added to our **report.json** file and the report modified accordingly, we will see the following view when selecting our report on the frontend before displaying the actual report:



Custom Filters For Parameters

In the above example users must select a start and end date and shop. The list of shops to select from is, however, not influenced by the values of the start and end date. In order to filter the shops listed using the start and end dates a special Helium report parameter filtering mechanism must be used. To make use of this we will create a new Unit called ReportParameterFilters under the **utilities** folder of our project. Here we specify what should happen when the start and end dates are selected and which shops should be provided to select from:

```

unit ReportParameterFilters;
date startDate;
date endDate;

// Called when the start date is selected from the rep
void setStartDate(date startDateParam) {
    startDate = startDateParam;
}

// Called when the end date is selected from the repor
void setEndDate(date endDateParam) {
    endDate = endDateParam;
}

// Used as a collection source from which the shop rep
Shop[] shopFilter() {
    FarmerPurchase[] purchases = FarmerPurchase:and(
        lessThanOrEqual(purchasedOn, endDate),
        greaterThan(purchasedOn, startDate)
    );

    return Shop:relationshipIn(purchases, purchases);
}
```

To link the app logic to our report parameters make use of the `searchDestination` and `searchSource` attributes in our **`report.json`** file:

```
{
    "name":"Your Purchases",
    "description":"Your purchases",
    "rolesAllowed":["Farmer"],
    "parameters":[
        {
            "parameterName":"startDate",
            "displayName":"Start date",
            "type":"date",
            "searchDestination":"ReportParameterFilters:setS
        },
        {
            "parameterName":"endDate",
            "displayName":"End date",
            "type":"date",
            "searchDestination":"ReportParameterFilters:setE
        },
        {
            "parameterName":"shopId",
            "displayName":"Shop",
            "type":"uuid",
            "searchType":"Shop.name",
            "searchSource":"ReportParameterFilters:shopFilte
        },
```

Search destination functions on **`uuid`** report parameter should have **`uuid`** parameters and not

```
        {
            "parameterName":"minCost",
            "displayName":"Minimum purchase cost",
            "type":"string"
        }
    ]
}
```

custom object type parameters.

Instead of listing all shops in the system, users will only be presented by shops with purchases between the start and end date parameter values.

Note that when a search destination is specified for a parameter that is of type **`uuid`**, it is not the object instance represented by that **`uuid`** that is sent to the search destination function as an argument but instead the id of the selected object instance. Consider the example below where the `shopId` parameter has an accompanying search destination instead of a search source:

```
{
    "parameterName":"shopId",
    "displayName":"Shop",
    "type":"uuid",
    "searchType":"Shop.name",
    "searchDestination":"ReportParameterFilters:setShopFilte
}
```

```
unit ReportParameterFilters;
uuid selectedShop;
.
.
// Called when the shop is selected from the report pa
void setShopFilter(uuid shopId) {
    selectedShop = Shop:read(shopId);
}
.
.
```

Note the use of the `read` built-in function. This is used instead of an `equals` selector and return a single object instance instead of a collection. Although not relevant in this specific use case, a **`null`** value will be returned if an id that does not represent an existing shop is used as an argument to the `read` built-in function above.

Custom Report and Report Group Menu Items

So far we have demonstrated the default reports menu item provided in Helium. Developers also have the option to make reports available from other report specific menu items using the following attributes, as example, in the **`report.json`** file:

```
```

```
"menuGroupName":"Purchases Reports",
"menuItemOrder":"0",
"menuItemIconName":"report_icon.png"
```

Using the above, reports using the same menu group name will be grouped together under a menu item that leads to a view with a table of the specified reports. Each report can then be accessed using Show button on the table. If instead the following is used, a single report can be directly accessed from a menu item:

```
"menuItemName":"Purchases Report",
"menuItemOrder":"0",
"menuItemIconName":"report_icon.png"
```

Summary of Fields in report.json

For reference see below a summary of the available attributes that can be specified in a **`report.json`** file.

Field	Description
name	The name of the report that will be displayed on the report specific view and the table listing available reports
description	Description of the report that will be displayed on the report specific view and the table listing available reports
rolesAllowed	Roles in the application for which this report should be available
parameters	Parameters that are to be sent to the report. Helium will prompt for these before displaying the report
menuItemName	The name of the custom menu item that represents the report. Specifying this will result in an additional menu item in the application that navigates to this report exclusively. The report will also not be listed under the standard reports view/table when this is specified.
menuItemIconName	The file name of the icon to be used in case the report has a custom menu item. This icon should be bundled with the report source in report's specific folder, and should have the same format as used for normal menu item icons. If this not be specified but the custom "menuItemName" is, the default report icon will be used.
menuItemOrder	Specifies the position of the custom menu item similarly to how the position is specified for normal app menu items. If nothing is specified but a "menuItemName" value is, it will be position at the bottom of the menu.
menuGroupName	Specifies a menu under which one or more reports can be displayed in table format, can be used in conjunction with menuItemIconName & menuItemOrder. All reports that belong to this group needs the attribute to be associated with the resulting menu item. But only one needs to specify menuItemIconName & menuItemOrder for it to be taken into account.

Field	Description
hiddenFormats	Disable rendering of specific formats for the Jasper report by populating the *hiddenFormats* array with any combination of HTML, XLSX and PDF.
searchDestination	The setter methods that will be invoked during the autocomplete phase. This setter method needs to be in the format of the fully qualified DSL name i.e. `UnitName:functionName`. The method when declared on your presenter will have to take one parameter of the same type it's declared parameter type. These setter methods all always be invoked before the active "searchSource" autocomplete result is fetched enabling you to set a unit variable and to use it as part of your result filtering. See below for a combination of setter and getter method calls to do advanced filtering of results for your autocomplete.
searchSource	The getter method that will be used to fetch results from autocompletion on the specified parameter field. The use of this getter method can be as simple as `PersistenceObject:all().`

Lesson Source Code

Lesson 17.zip

No labels

2 Comments

John Vorster

Thank you @ Adriaan Klue and @ Jacques Marais ... I have seen both of you contribute to this page.

It is a great resource and I was able to vastly improve on how jasper reports are implemented on mVacciNation.

I had two main stumbling blocks, which perhaps you could address through examples for people like me who are new to jasper reports:

1) The variable declaration in FarmerUserPurchases > master.jrxml ... I could not actually "use" that variable declaration in the SQL statement, for example by filtering the shop._id_ column by the shopId passed in. It constantly through an error related to the variable class "java.util.UUID". I could resolve that only by passing in everything (including the UUID fields) as java.lang.String and then casting to UUID within the SQL. That worked without a hitch.

Therefore, in my experience, this will not allow the shopId to be used within the SQL select:

```
<parameter name="shopId" class="java.util.UUID">
<parameterDescription><![CDATA[]]></parameterDescription>
<defaultValueExpression><![CDATA[""]]></defaultValueExpression>
</parameter>
```

This will however allow the shopId to be used within the SQL select:

```
<parameter name="shopId" class="java.lang.String">
<parameterDescription><![CDATA[]]></parameterDescription>
<defaultValueExpression><![CDATA[""]]></defaultValueExpression>
</parameter>
```

2) When trying to set a DSL object based on the UUID passed in from the report UI, I struggled to "get" the fact that I had to receive a UUID parameter and convert that UUID to an object within helium. I assumed that by assigning the object to the correctly typed parameter it would be setting the object. This did not throw an error, but the object I was trying to set in this way, remained a null. Looking at this now, this might be obvious... but in the absence of a helium error, it took me a while to figure this one out.

Therefore this did not work... ie uDistrict remained a null after assignment:

```
void setDistrict(District d) {

uDistrict = d

}
```

But this did:

```
void setDistrict(uuid id) {

if(id==null){
uDistrict = null;
}

District[] districts = District:equals(_id,id);
if(districts.length()>0){
uDistrict = districts.get(0);
} else {
uDistrict = null;
}

return;

}
```

Jacques Marais

Thanks @ John Vorster . I'll be sure to add some content to cover the above mentioned points.