Lists App - Behind the Scenes

WOLFcon23

2023-08-24

Vince Bareau, Matt Weaver, PK Jacob

The Lists app aims to provide actionable lists in FOLIO at the point of need. The Lists app will enable users to create ad-hoc lists (lists of records) and will contain a set of pre-generated lists. This session will describe the technical underpinnings for the Lists App and how it opens up solutions in other areas of Folio.

The Folio Lists Application

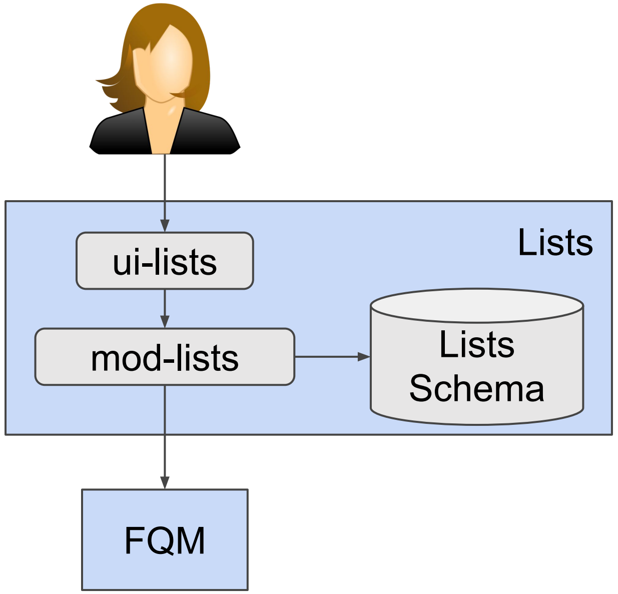
Folio's Lists application aims to provide actionable ad hoc lists from Folio, at the point of need.

- Lists are persisted
- Lists are updatable
- Lists enable various workflows
- Lists can pull data from multiple other Folio applications

Question: But how does it work?

Answer: Very well!

No really, how does it work?



The Lists application is like most others, consisting of:

- UI module: ui-lists
- Backend module: mod-lists
  - Includes Lists storage

However the heavy lifting is done by another component that provides it the data: **FQM**

FQM

What is FQM?

FQM is the **Folio Query Machine**

- Through a simple API, it allows other modules to access Folio data
- FQM is internal to Folio
- FQM does not require the movement of large sets of data
- FQM directly accesses data in situ
- FQM works across modules

Some Key FQM Concepts

FQM delivers **Record Sets** where each **Record** is made up of a number of **Data Fields**.

An **EntityType** is a queryable relation in Folio. A specific set of related data fields identified through an **ID field**.

A **Query** is used to request the specific records which are to be retrieved.

FQM uses **FQL** for its query syntax.

What is FQL?

- **FQL** is the FQM Query Language
- MongoDB-like syntax
  - Simple
  - Unambiguous
  - Powerful, but not too powerful
  - Trivial to parse
  - Designed for use with JSON-structured data
  - Simple structure ➡ Easy to build a UI
- Aligns better to UI rendering of queries

Why not CQL?

CQL is designed for relational datasets and doesn’t work well when data is mostly structured JSON data

Why not CQL?

JSONB may require accessing nested fields:

```
[
{"type": "Type-A", "some_object": { "text": "ping" }},
{"type": "Type-B", "some_object": { "text": "pong" }},
{"type": "Type-A", "some_object": { "text": "foo" }},
{"type": "Type-B", "some_object": { "text": "bar" }},
]
```

If a query is required for records having
`type = "Type-A"` and `"some-object.text" == "ping"`.

This requires a JSON path style query
`field[type == "Type-A"].some-object.text == "ping"`,

which is not supported by CQL.

A Sample FQL Query

```
{
  "item_status": {
    "$in": [
      "Missing",
      "aged to lost",
      "claimed returned",
      "declared lost",
      "long missing"
    ]
  }
}
```

Another Sample FQL Query

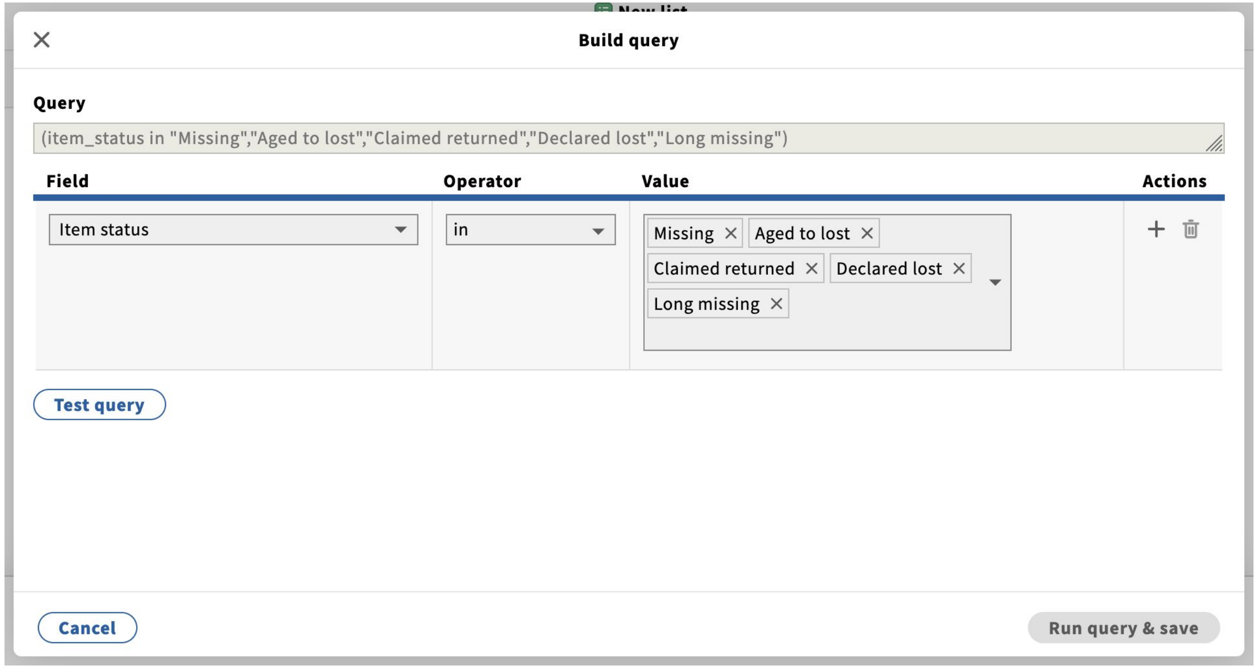
```
{"$and":[{"active":{"$eq":"
true"}},{"user_patron_group
_id":{"$nin":["0003a0cc-46e
5-4ebe-8545-c917d3d8a673","
06f2d60e-0b07-49ca-b8c7-e1d
49808e0b7"]}}]}
```

But users don't want to write FQL queries any more than they want to write SQL queries!

FQM has a Query Builder !

The Query Builder is a Folio UI Plugin and potentially usable by other applications

```
{
  "item_status": {
    "$in": [
      "Missing",
      "aged to lost",
      "claimed returned",
      "declared lost",
      "long missing"
    ]
  }
}
```


Build query

Query

(item_status in "Missing","Aged to lost","Claimed returned","Declared lost","Long missing")

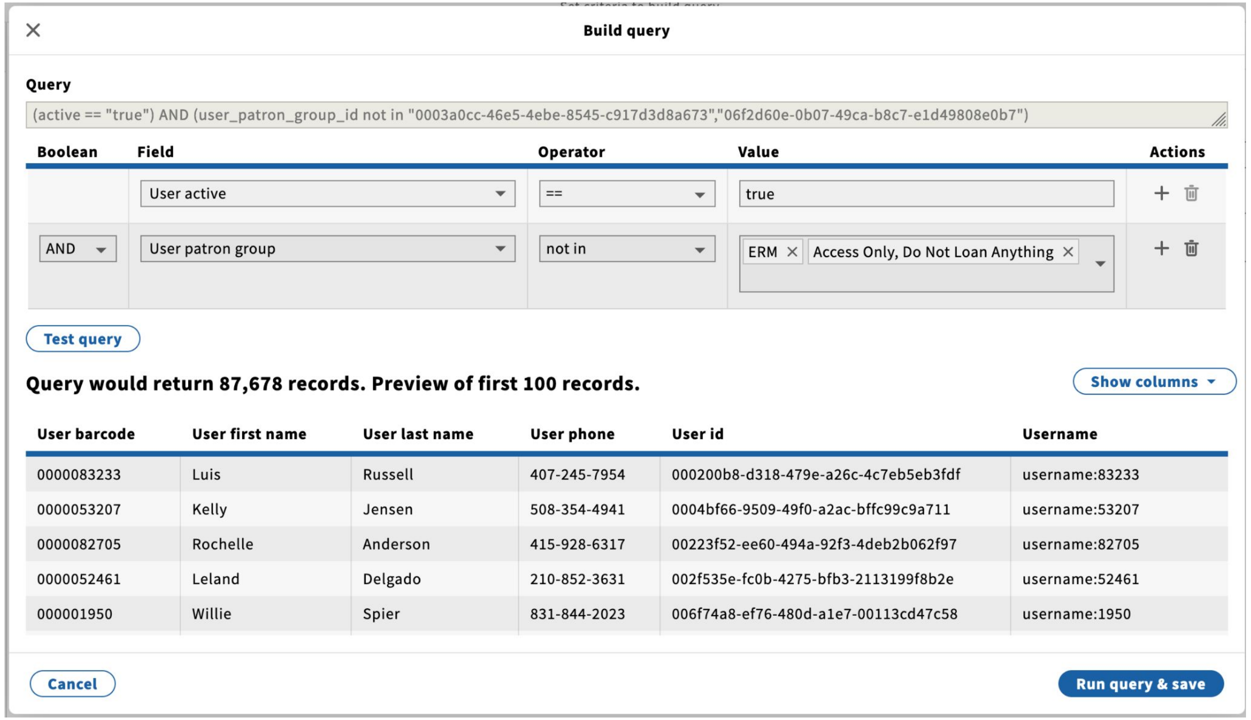
Field	Operator	Value	Actions
Item status	in	Missing, Aged to lost, Claimed returned, Declared lost, Long missing	+ 🗑

FQM has a Query Builder !

The Query Builder is a Folio UI Plugin and potentially usable by other applications

```
{"$and":[{"active":{"$eq":"
true"}},{"user_patron_group
_id":{"$nin":["0003a0cc-46e
5-4ebe-8545-c917d3d8a673","
06f2d60e-0b07-49ca-b8c7-e1d
49808e0b7"]}}]}
```

AKA
"(active == true) AND (user_patron_group not in [ERM, Access Only, Do Not Loan Anything])"



Build query

Query

(active == "true") AND (user_patron_group_id not in "0003a0cc-46e5-4ebe-8545-c917d3d8a673","06f2d60e-0b07-49ca-b8c7-e1d49808e0b7")

Boolean	Field	Operator	Value	Actions
	User active	==	true	+ 🗑
AND	User patron group	not in	ERM ×, Access Only, Do Not Loan Anything ×	+ 🗑

Test query

Query would return 87,678 records. Preview of first 100 records.

Show columns

User barcode	User first name	User last name	User phone	User id	Username
0000083233	Luis	Russell	407-245-7954	000200b8-d318-479e-a26c-4c7eb5eb3fdf	username:83233
0000053207	Kelly	Jensen	508-354-4941	0004bf66-9509-49f0-a2ac-bffc99c9a711	username:53207
0000082705	Rochelle	Anderson	415-928-6317	00223f52-ee60-494a-92f3-4deb2b062f97	username:82705
0000052461	Leland	Delgado	210-852-3631	002f535e-fc0b-4275-bfb3-2113199f8b2e	username:52461
000001950	Willie	Spier	831-844-2023	006f74a8-ef76-480d-a1e7-00113cd47c58	username:1950

Cancel | Run query & save

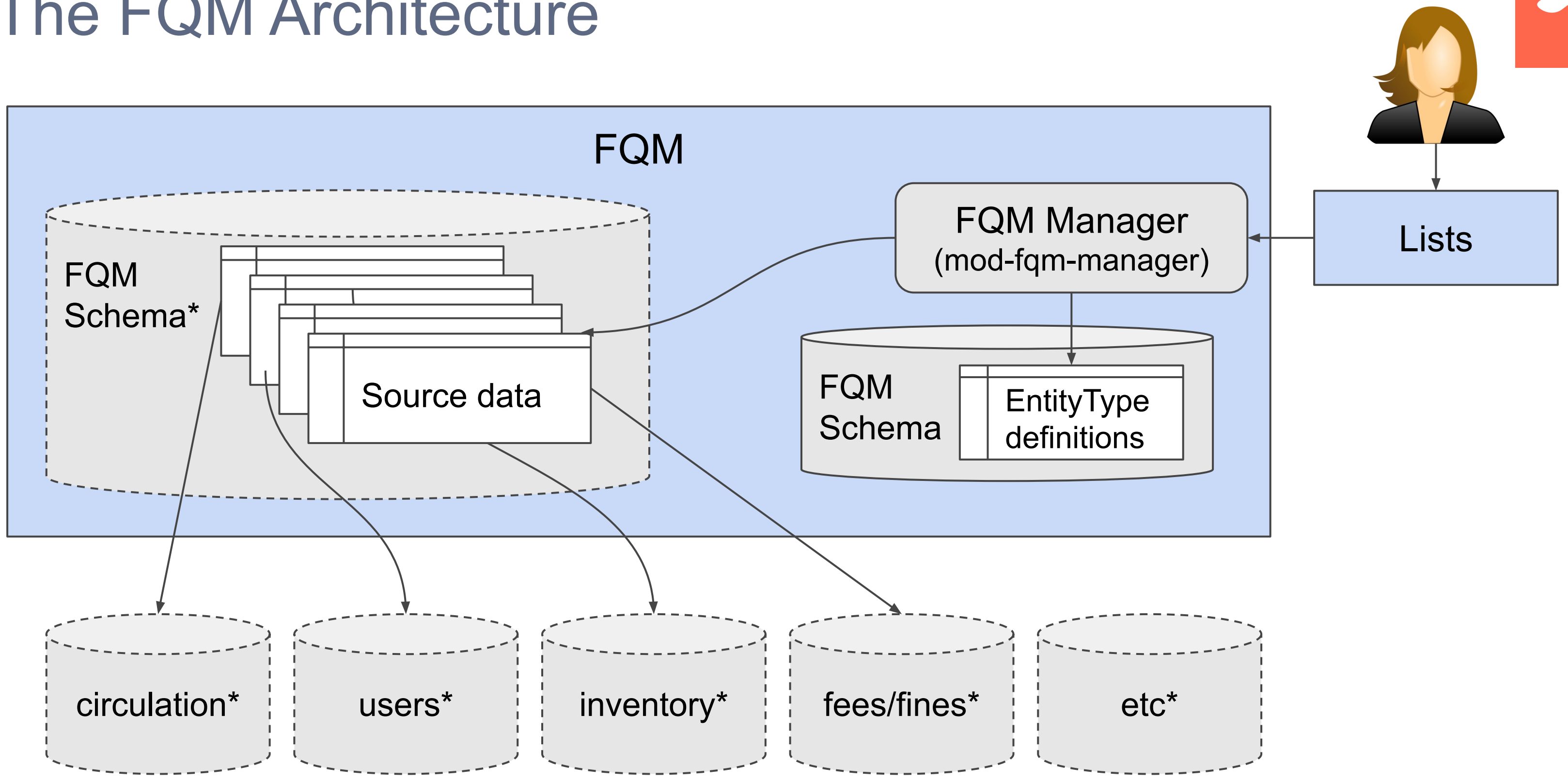
How does FQM access data from other Applications?

- FQM exposes data from other modules via the FQM DB schema
- As with all microservices, the FQM schema is fully controlled by FQM itself
- FQM itself is agnostic as to how FQM schema is populated with data
  - Data could be harvested from Kafka events
  - Data could be replicated from a data warehouse in Folio
  - Data could be provided by DB views pointing at other schemas
  - FQM doesn’t care how the data get there, as long as it’s there
- By default, FQM uses Views

Security

- FQM Manager uses standard Folio API interface module permissions
- EntityTypes are secured at a more granular level.
    - FQM Manager maintains and assigns access control “permissions” for each EntityType
        - Thus a given user may be granted access to query a particular EntityType
        - In a future release data would restricted to the level of source data fields
        - For this FQM Manager could benefit from the Roles and Capabilities model

The FQM Architecture



* it is recommended that FQM be installed on a Folio Read Replica

Other uses of FQM

FQM is an enabler for other areas of Folio.

It is currently being considered to empower several Applications other than the Lists app.

- Bulk Edit:
  - Generate the select lists of IDs on which to operate through cross-module queries
- Delete Functionality
  - Provide cross-module metadata for each Rule to determine eligibility to delete a particular record
- Data Export
  - Generate the data sets required for export
- Data Import
  - Optimize matching rules for record import
- Other Applications
  - Such as native Folio operational reporting

FQM and Bulk Edit

The Bulk Edit application provides simultaneous field editing across a select group of records of a single record type.

It is a two part process:

1. Establish the desired subset of records of a particular
    a. Typically with conditions relying on other record types,
    b. For example: update all inventory.item records that were loaned in the last 29 days.
2. Apply the changes to the subset of records.

The first item (1) could be ideally fulfilled by FQM

FQM and Delete Functionality

By example: before deleting an Inventory.Item record, we must first determine that there are:

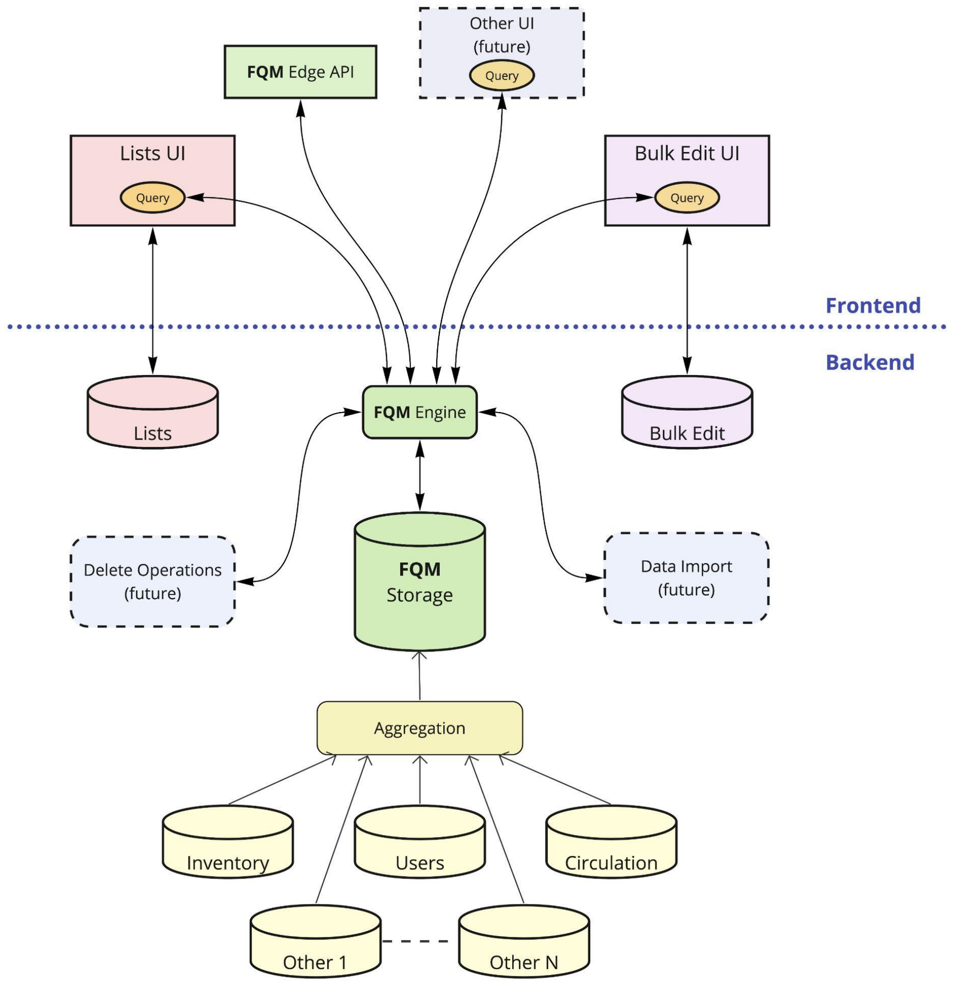
- no open loans
- no pending requests
- no pending orders
- no fes/fines
- no course reserves
- etc…

More generally this would be formalized as a Rule for Deleting a particular entity. Evaluating the Rule requires examining particular fields in different modules. Only when all have returned “clear” is it safe to delete the entity

Rather than (re)implement the functionality in the delete algorithms, it makes sense to use FQM for this purpose. For each Rule, FQM could maintain a single table which delete functionality could consult

FQM: System View

- Supports Multiple Applications
    - Lists App
    - Bulk Edit
    - Delete Functionality
    - Data Export
    - Data Import
    - Other (e.g. Reporting)
- Reusable Query Builder plugin
- Provides an Edge Api for external integration (pending)



Thank you