Cashier Module User Interface

The user interface is based on Backbone.js and interacts with OpenMRS mainly through the Webservices.rest module. It also makes heavy use of the backbone-forms extension to Backbone.

Backbone was chosen for the Cashier module partially out of developer familiarity, but also out of the desire to create a very responsive screen for bill entry. The work on the Cashier UI will hopefully provide the basis for future responsive hospital management UIs.

Screen Loading

Each screen in the user interface has a controller in web.controller and a template in src/main/webapp, but these are mostly place holders for routing servlet requests. Most of the templates include */localHeader.jsp* which includes the following JavaScript:

1. init.js: sets up a global openhmis object and populates it with some basic configuration---mostly HTTP paths.
2. curl.js: the AMD loading utility.

After this, most templates load a single JavaScript file from the resources/js/screens folder. These *screens* JS modules are responsible for loading a screen's dependencies and rendering views.

Generic Models and Views

The Cashier UI makes use of customized Backbone models, collections and views to support the manipulation and display of server resources in as generic a way as possible.

GenericModel

The openhmis.GenericModel (model/generic.js module) extends Backbone's Model class to provide support for OpenMRS-specific operations (e. g., (un)retire/(un)void, purge) as well for the REST module.

url

The GenericModel supports several ways of setting the URL for the server resource:

urlRoot: Manually set the full URL for the resource, overriding any settings in a collection or openhmis.config. This is an option that can be specified when instantiating a model.; for example:

```
var model = new openhmis.GenericModel({}, { urlRoot: "/path/to/model" });
```

restUrl and openhmis.config: Alternatively, the REST resource URL can be specified using a combination of the model's *meta* attribute and the openhmis namespace configuration (specified in init.js). See below for more information on the *meta* attribute. For example, if **openhmis.config** is set to_ /openmrs/rest/ *and* ***restUrl*** *is _patient*, the model's **url** property will evaluate to */openmrs/rest/patient/(uuid).*

meta

The meta property of a GenericModel is used to specify certain metadata about the model. Here is an example from the Bill model:

```
openhmis.Bill = openhmis.GenericModel.extend({
    ...
    meta: {
        name:        "Bill",
        namePlural:  "Bills",
        openmrsType: "data",
        restUrl:     "bill"
    }
    ...
}
```

name and **namePlural:** The name of the model, and the pluralized name, respectively. Used by generic views for display purposes. (**TODO:** this could be done automatically in many cases and only be specified in exceptional cases)

openmrsType: "data" or "metadata"; certain behaviour (e.g. retire/void) changes based on this. If not given, GenericModel tries to guess.

restUrl: The name of the REST resource, which will be concatenated to **openhmis.config**.

schema

The GenericModel makes use of the backbone-forms concept of *schema*, which is an object that describes the attributes of the model. In the future it would be ideal to get the schema for models directly from the server; for now they are hard coded in the JavaScript versions of the models.

The schema is used in the serialization of the model. If a schema is specified, only the attributes defined in the schema will be serialized. If no schema is specified, the model will be serialized by the default Backbone method.

Here is an example of a model schema:

```
ExampleModel = openhmis.GenericModel.extend({
    ...
    schema: {
        name:    { type: 'Text' },
        patient: { type: 'Object', objRef: true },
        status:  { type: 'Text', readOnly: true }
    }
    ...
}
```

This schema could also be used by backbone-forms to automatically generate a form for this model, but two attributes of the schema will affect how the model is serialized when it is saved to the server:

readOnly: Attributes marked readOnly will not be added to the serialized object that is saved to the server. This allows non-updatable attributes to be displayed in a form but not serialized during a save.

objRef: Attributes marked as objRef will be treated as object references. This means that they may be accessible in JavaScript as a full object, but when they are serialized, they should simply be sent as the UUID of the object that the model attribute references. The REST module supports this convention and will attempt to retrieve the appropriate object by its UUID if given a string value for a REST resource attribute.

GenericCollection

The openhmis.GenericModel (model/generic.js module) extends Backbone's Collection class to provide extra support related to the REST interface, including paging and search support.

url

A GenericCollection can help supply the resource URL when not available from the model.

baseUrl: Set the URL for models in this collection, overriding **openhmis.config.restUrlRoot**. Given as an option when instantiating a GenericCollection.

url: Set the resource name portion of the URL. This will either be concatenated to **openhmis.config.restUrlRoot** or **baseUrl** (if specified).

If no URL options are specified, the collection will attempt to discover a URL from its **model** attribute.

GenericListView

The openhmis.GenericListView (view/generic.js module) is a Backbone View that will display a GenericCollection in a list.

options

model: A GenericCollection to display

itemView: The view type to use to display each item in the list. Defaults to GenericListItemView.

schema: Can be specified to override the schema of the models in the collection.

listTitle: Title to be displayed for the list.

itemActions: A list of actions to enable for the items in the list. GenericListItemView supports "remove" and "inlineEdit".

includeFields: A list of attributes in the model's schema to display as columns in the list. Defaults to all the attributes in the model's schema.

excludeFields: A list of attributes in the model's schema to exclude from the list's columns.

showPaging: Whether to display pagination controls. Defaults to true.

pageSize: Set the initial number of results to show in the list.

showRetiredOption: Whether to display the option of showing/hiding retired/voided items.

hideIfEmpty: If true and the view's collection is empty, the entire list view will not be displayed. Defaults to false.