Filter by Content Type

- Symfony Controller and routing configuration to accept a type parameter
- JavaScript application routing to accept a type parameter
- Request the server with the given type
- Content Type drop down list in the server response
- Filter by Content Type by using the drop down list

This step is very similar to the previous one on adding pagination. We are going to add a new `type` parameter in the whole component *stack*; the main difference with the `offset` case is that the user will pick a Content Type in a drop-down list instead of just using the Next/Previous links in the page.

Symfony Controller and routing configuration to accept a `type` parameter

In addition to the `offset` parameter, the Symfony action has to accept a `type` parameter. Like in the previous step, this involves changing the `routing.yml` configuration file and changing the logic in the Controller to do a search request based on this parameter if it is provided. This is done in the related commit.

JavaScript application routing to accept a `type` parameter

Like in the `offset` case, we have to add a new route to support the `type` parameter. Again, this is done in the application plugin with the following code:

ezconf-listappplugin.js

```
app.route({
    name: "eZConfListOffsetTypeIdentifier",
    path: "/ezconf/list/:offset/:typeIdentifier",
    view: "ezconfListView",
    service: Y.eZConf.ListViewService,
    sideViews: {'navigationHub': true, 'discoveryBar': false},
    callbacks: ['open', 'checkUser', 'handleSideViews', 'handleMainView'],
});
```

Request the server with the given type

Again, this is very similar to the `offset` case, now that we have a Content Type identifier in the matched route parameters, we can request the server to get a filtered list by changing the `_load` implementation in our view service:

_load method in ezconf-listviewservice.js

```
_load: function (callback) {
    // the request allows to retrieve the matched parameters
    var offset = this.get('request').params.offset,
        typeIdentifier = this.get('request').params.typeIdentifier,
        uri;

    if ( !offset ) {
        offset = 0;
    }
    uri = this.get('app').get('apiRoot') + 'list/' + offset;
    if ( typeIdentifier ) {
        uri += '/' + typeIdentifier;
    }

    Y.io(uri, {
        method: 'GET',
        on: {
            success: function (tId, response) {
                this._parseResponse(response);
                callback(this);
            },
            failure: this._handleLoadFailure,
        },
        context: this,
    });
},
```

After doing hat, you can manually tweak the URI to get a filtered list. For instance, if you go to `/ez#/ezconf/list/10/folder` you should only see Locations of Folders in the repository.

Content Type drop down list in the server response

Of course, the end user is not supposed to tweak the URI to filter by Content Type. To provide a usable interface, we will add a drop-down list where they can choose a Content Type to only get the corresponding list. Again, this is done server side by modifying the Controller to fetch the list of Content Types and Content Type Groups with the Content Type Service. Once this is done, the types and the groups can be passed to the Twig template to build a drop-down list in the following way:

Drop down list building

```
{% block content %}
<div class="ezconf-list-toolbar">
    <label for="types-list">Filter by content type</label>
    <select id="types-list" class="ezconf-list-types">
        <option value="">None</option>
    {% for group in groups %}
        <optgroup label="{{ group.identifier }}">
        {% for type in typesByGroup[group.identifier] %}
            <option value="{{ type.identifier }}"{% if type.identifier ==
typeIdentifier %}selected{% endif %}>{{ type.names['eng-GB'] }}</option>
        {% endfor %}
        </optgroup>
    {% endfor %}
    </select>
</div>

<!-- ... the rest of the content goes here -->

{% endblock %}
```

The complete change for that can be seen in the corresponding commit.

Filter by Content Type by using the drop down list

The only remaining feature is now to take into account the user interaction on the added drop-down list. The view is responsible for that and it will basically subscribe to the `change` DOM event on the list and will fire the navigateTo event like we did in the `offset` case:

Drop down list change handling

```
YUI.add('ezconf-listview', function (Y) {
    Y.namespace('eZConf');

    Y.eZConf.ListView = Y.Base.create('ezconfListView', Y.eZ.ServerSideView, [], {
        events: {
            '.ezconf-list-location': {
                // tap is 'fast click' (touch friendly)
                'tap': '_navigateToLocation'
            },
            '.ezconf-list-page-link': {
                'tap': '_navigateToOffset'
            },
            '.ezconf-list-types': {
                'change': '_filterByType'
            }
        },

        _filterByType: function (e) {
            var select = e.target;

            this.fire('navigateTo', {
                routeName: 'eZConfListOffsetTypeIdentifier',
                routeParams: {
                    offset: "0", // offset: 0 does not work, because of a bug in
PlatformUI
                    typeIdentifier: select.get('value'),
                }
            });
        },

        // ... the rest of the view code should still be there
    });
});
```

Basically, when the user selects a value in the drop-down list, the view fires the `navigateTo` event to navigate to the `eZConfListOffsetTypeIdentifier` route with the corresponding Content Type identifier and the offset set to 0. The view service is still handling this event and takes the user to the corresponding page in the application.

Results
And that's it! We now have a working interface to navigate through Content with a flat list that can be filtered by Content Type. The resulting code can be seen in the `8_filter` tag on GitHub, this step result can also be viewed as a diff between tags `7_pagination` and `8_filter`. The conclusion page lists the potential improvement and some bugs in our current implementation. It also gives a list of documentation pages to read to go into more detail.