Using the JavaScript REST API Client

The JavaScript REST API Client is a JavaScript library meant to ease the use of the eZ Platform REST API. For now, it can only be used in a web browser.

Installation

In the PlatformUIAssetsBundle

Since the JavaScript REST Client is one of the foundations of the Platform Backend Interface, the client is provided by the PlatformUIAssetsBundle which is installed by default. As a result, the client is directly available and can be embedded in any Platform-generated page with the following Twig code:

Embedding the JavaScript REST Client

```
<script src="{{
asset('bundles/ezplatformuiassets/vendors/ez-js-rest-client/dist/CAPI.js')
}}"></script>
<!-- or the minified version -->
<!-- <script src="{{
asset('bundles/ezplatformuiassets/vendors/ez-js-rest-client/dist/CAPI-min.js')
}}"></script> -->
```

With Bower

Alternatively, the JavaScript REST Client can be installed directly in any project with Bower:

Installing with bower

```
$ bower install --save ezsystems/ez-js-rest-client
```

After using this command, `dist/CAPI.js` or `dist/CAPI-min.js` are available in `bower\_components/ez-js-rest-client/`.

Manual install

It is also possible to directly retrieve either `dist/CAPI.js` or `dist/CAPI-min.js` in the Github repository of the project.

Usage examples

Once included, `CAPI.js` exports the `eZ` namespace which contains `eZ.CAPI`, the constructor function of the client. This constructor must receive the API end point and an authentication agent responsible for handling the authentication (session or basic auth). This is detailed in the Instantiation and authentication section below.

The auto-generated API documentation of the JavaScript REST API client is available online. Like in the Public API, the code is organized around 3 main services:

- the Content Service
- the Content Type Service
- the User Service

In essence, the operations available through those services are asynchronous, so all the corresponding methods accept a callback function as its last argument. This callback function will be called when the operation has been done and it will receive two arguments:

1. `error`: depending on the success of the operation, this parameter is either `false` or a `CAPIError` instance representing the error
2. `response`: it's always of a `Response` instance allowing you to retrieve any information from the REST API response

Instantiation and authentication

The `eZ.CAPI` constructor function expects two parameters:

1. the API end point URI
2. an authentication agent instance to configure the client for the authentication mechanism configuration in eZ Platform.

The JavaScript REST Client comes with two authentication agents for the Session and Basic Auth authentication mechanism.

Session auth

The Session Auth Agent expects an object describing the existing Session or containing the credentials for the user to create the corresponding session. So if the user is not yet authenticated, the client can be instantiated with:

Session Authentication (new session)

```
var capi,
    credentials = {
        login: 'admin',
        password: 'publish',
    };

capi = new eZ.CAPI(
    'http://example.com',
    new eZ.SessionAuthAgent(credentials)
);
capi.logIn(function (error, response) {
    if ( error ) {
        console.log('Error!');
        return;
    }
    console.log("I'm logged in");
});
```

> Instead of passing the `credentials` to the `eZ.SessionAuthAgent` constructor, it is also possible to pass this object as the first parameter of the `logIn` method.

If the user already has a session, the agent expects an object describing the session, something like:

Session Authentication (existing session)

```
var capi,
    sessionInfo = {
        name: 'eZSESSID', // name of the session, might also be something like
eZSESSID98defd6ee70dfb1dea416cecdf391f58
        identifier: '6pp87ah63m44jdf53b35qlt2i7', // session id
        href: '/api/ezp/v2/user/sessions/6pp87ah63m44jdf53b35qlt2i7',
        csrfToken: 'memEneT7WvX9WmSlG2wDqUj0eeLRC7hXG--pLUx4dFE', // can be retrieved
with @security.csrf.token_manager Symfony service
    };

capi = new eZ.CAPI(
    'http://example.com',
    new eZ.SessionAuthAgent(sessionInfo)
);
capi.isLoggedIn(function (error, response) {
    if ( error ) {
        console.log('Error!');
        return;
    }
    console.log("I'm logged in");
});
```

Basic auth

When configured in the Basic Authentication, the basic auth agent just expects the user's credentials:

Basic Authentication

```
var capi,
    credentials = {
        login: 'admin',
        password: 'publish',
    };

capi = new eZ.CAPI(
    'http://example.com',
    new eZ.HttpBasicAuthAgent(credentials)
);
capi.logIn(function (error, response) {
    if ( error ) {
        console.log('Error!');
        return;
    }
    console.log("The credentials are valid");
});
```

Loading a ContentInfo or a Content

To load a ContentInfo, you need the Content Service, it is returned by the `getContentService` method on the client instance:

Loading a ContentInfo

```
var capi, contentService,
    contentRestId = '/api/ezp/v2/content/objects/1',
    credentials = {
        login: 'admin',
        password: 'publish',
    };

capi = new eZ.CAPI(
    'http://example.com',
    new eZ.SessionAuthAgent(credentials)
);
contentService = capi.getContentService();
contentService.loadContentInfo(contentRestId, function (error, response) {
    if ( error ) {
        console.log('Error!');
        return;
    }
    // response.document contains the parsed JSON structure
    console.log('Content name: ' + response.document.Content.Name);
})
```

If you run this example, you should see in the browser network panel a GET HTTP request to http://example.com/api/ezp/v2/content/objects/1 wit h the necessary headers to get a JSON representation of the ContentInfo. If you want to load the Content instead, you can use the `loadContent` method.

Moving a Location

To move a Location, the Content Service is also needed, this operation will generate a MOVE HTTP request. If configured for the session authentication mechanism, the client will automatically add the CSRF Token.

Moving a Location

```
var capi, contentService,
    locationRestId = '/api/ezp/v2/content/locations/1/43/53', // Media/Multimedia in a
default install
    newParentLocationRestId = '/api/ezp/v2/content/locations/1/43/52', // Media/Files
in a default install
    credentials = {
        login: 'admin',
        password: 'publish',
    };

capi = new eZ.CAPI(
    'http://example.com',
    new eZ.SessionAuthAgent(credentials)
);
contentService = capi.getContentService();
contentService.moveSubtree(locationRestId, newParentLocationRestId, function (error,
response) {
    if ( error ) {
        console.log('Error!');
        return;
    }
    console.log('Media/Multimedia is now Media/Files/Multimedia');
})
```

Searching for Content or Location

Searching for Content or Location can be done with REST views. REST views can be configured with the search engine criteria to match some Content items or Locations:

REST views

```
var capi, contentService, query,
    credentials = {
        login: 'admin',
        password: 'publish',
    };

capi = new eZ.CAPI(
    'http://example.com',
    new eZ.SessionAuthAgent(credentials)
);
contentService = capi.getContentService();
query = contentService.newViewCreateStruct('test-rest-view', 'LocationQuery'); // use
'ContentQuery' to retrieve a list of Content items
query.body.ViewInput.LocationQuery.Criteria = { // use 'ContentQuery' here as well
    FullTextCriterion: "ez",
};
query.body.ViewInput.LocationQuery.limit = 10;
// query.body.ViewInput.LocationQuery.offset = 0;
contentService.createView(query, function (error, response) {
    if ( error ) {
        console.log('Error!');
        return;
    }
    console.log("Search results", response.document.View.Result.searchHits.searchHit);
})
```

REST views
REST views are designed to be persisted but this feature is not yet implemented. As a result, when calling `createView`, the POST request does not create the view but directly returns the results.