REST API

Overview

The REST API for DSpace is provided as part of the "server" webapp (`[dspace-source]/dspace-server-webapp/`). It is available on the `/api/` subpath of that webapp (i.e. `${dspace.server.url}/api/`), though a human browseable/searchable interface (using the HAL Browser) is also available at the root path (i.e. `${dspace.server.url}`).

The REST API only responds in JSON at this time.

REST Contract / Documentation

The REST Contract is maintained in GitHub at https://github.com/DSpace/RestContract/blob/dspace-7_x/README.md

This contract provides detailed information on how to interact with the API, what endpoints are available, etc. All features/capabilities of the DSpace UI are available in this API.

Finding which REST API Endpoint to use

When first trying the use the DSpace REST API, it can be difficult to determine where to begin. This brief guide provides a few hints on where to start.

First, it's important to be aware that **every single action in the User Interface can be done in the REST API.** So, if you can achieve something in the User Interface, then it's also possible to do via the REST API.

A few key endpoints to be aware of:

- Authentication: https://github.com/DSpace/RestContract/blob/dspace-7_x/authentication.md
- CSRF Tokens (required for all non-GET requests): https://github.com/DSpace/RestContract/blob/dspace-7_x/csrf-tokens.md
- Submission via REST API: https://github.com/DSpace/RestContract/blob/dspace-7_x/submission.md
- Search via REST API (across all object types): https://github.com/DSpace/RestContract/blob/dspace-7_x/search-endpoint.md
    - Some endpoints also provide a "/search" subpath: https://github.com/DSpace/RestContract/blob/dspace-7_x/search-rels.md

How to find which endpoint(s) to use for any feature or action:

1. Open the DSpace User Interface in your browser window. You can even use our Demo Site (https://demo.dspace.org/) if you don't have the User Interface installed or running locally.
2. In your Browser, open the "Developer Tools"
    a. In Chrome, go to "More Tools Developer Tools"
    b. In Firefox, go to "Web Developer Web Developer Tools".
    c. In Microsoft Edge, go to "More Tools Developer Tools".
3. Once in "Developer Tools", open the "Network" tab. This tab will provide information about *every single call that the User Interface makes to the REST API.*
4. Now, perform an action or use a feature in the User Interface in your browser window.
5. Analyze what calls were just sent to the REST API in your "Network" tab. Those are the exact REST API endpoints that were used to perform that action.
    a. NOTE: Some actions may use *multiple endpoints.*
6. Finally, lookup the documentation for those endpoint(s) in the REST Contract / Documentation (see link above)

REST Configuration

The following REST API configurations are provided in `[dspace]/config/rest.cfg` and may be overridden in your local.cfg

Property:	`rest.cors.allowed-origins`
Example Value:	`rest.cors.allowed-origins = ${dspace.ui.url}`

Informational Note:	Allowed Cross-Origin-Resource-Sharing (CORS) origins (in "Access-Control-Allow-Origin" header). Only these origins (client URLs) can successfully authenticate with your REST API *via a web browser*. Defaults to `${dspace.ui.url}` if unspecified (as the UI must have access to the REST API). If you customize that setting, MAKE SURE TO include `${dspace.ui.url}` in that setting if you wish to continue trusting the UI. Multiple allowed origin URLs may be comma separated (or this configuration can be defined multiple times). Wildcard value (*) is NOT SUPPORTED. Keep in mind any URLs added to this setting must be *an exact match with the origin*: mode (http vs https), domain, port, and subpath(s) all must match.. So, for example, these URLs are all considered different origins: "http://mydspace.edu", "http://mydspace.edu:4000" (different port), "https://mydspace.edu" (http vs https), "https://myapp.mydspace.edu" (different domain), and "https://mydspace.edu/myapp" (different subpath). *NOTE #1:* Development or command-line tools may not use CORS and may therefore bypass this configuration. CORS does not provide protection to the REST API / server webapp. Instead, its role is to protect browser-based clients from cookie stealing or other Javascript-based attacks. All modern web browsers use CORS to protect their users from such attacks. Therefore DSpace's CORS support is used to protect users who access the REST API via a web browser application, such as the DSpace UI or custom built Javascript tools/scripts. *NOTE #2:* If you modify this value to allow additional UIs (or Javascript tools) to access your REST API, then you may also need to modify `proxies.trusted.ipranges` to trust the IP address of each UI. Modifying trusted proxies is only necessary if the `X-FORWARDED-FOR` header must be trusted from each additional UIs. (The DSpace UI currently requires the `X-FORWARDED-FOR` header to be trusted). By default, `proxies.trusted.ipranges` will only trust the IP address of the `${dspace.ui.url}` configuration. (Requires reboot of servlet container, e.g. Tomcat, to reload)
Property:	`rest.cors.allow-credentials`
Example Value:	`rest.cors.allow-credentials = true`
Informational Note:	Whether or not to allow credentials (e.g. cookies) sent by the client/browser in CORS requests (in "Access-Control-Allow-Credentials" header). For DSpace, this MUST be set to "true" to support CSRF checks (which use Cookies) and external authentication via Shibboleth (and similar). Defaults to "true" if unspecified. (Requires reboot of servlet container, e.g. Tomcat, to reload)
Property:	`rest.projections.full.max`
Example Value:	`rest.projections.full.max = 2`
Informational Note:	This property determines the max embeddepth for a FullProjection. This is also used by the SpecificLevelProjection as a fallback in case the property is defined on the bean. Usually, this should be kept as-is for best performance.
Property:	`rest.projection.specificLevel.maxEmbed`
Example Value:	`rest.projection.specificLevel.maxEmbed = 5`
Informational Note:	This property determines the max embed depth for a SpecificLevelProjection. Usually, this should be kept as-is for best performance.

Property:	`rest.properties.exposed`

Example Value:	`rest.properties.exposed = plugin.named.org.dspace.curate.CurationTask`
	`rest.properties.exposed = google.analytics.key`

Informational Note:	Define which configuration properties are exposed through the `http://<dspace.server.url>/api/config/properties/` REST API endpoint. If a rest request is made for a property which exists, but isn't listed here, the server will respond that the property wasn't found. This property can be defined multiple times to allow access to multiple configuration properties. Generally, speaking, it is ONLY recommended to expose configuration settings where they are necessary for the UI or client, as exposing too many configurations could be a security issue. This is why we only expose the two above settings by default.

REST Spring Boot Configuration

Because the REST API is a Spring Boot web application, you can also configure or override *any* Spring Boot settings in your local.cfg. DSpace's default Spring Boot configuration can be found in `[src]/dspace-server-webapp/src/main/resources/application.properties`. A few common settings from Spring Boot which you may wish to override in your local.cfg include:

Property:	`spring.servlet.multipart.max-file-size`
Example Value:	`spring.servlet.multipart.max-file-size = 512MB`
Informational Note:	Per Spring Boot docs, this setting specifies the maximum size of file that can be uploaded via Spring Boot (and therefore via the DSpace REST API). A value of "-1" removes any limit. DSpace sets this to 512MB by default.
Property:	`spring.servlet.multipart.max-request-size`
Example Value:	`spring.servlet.multipart.max-request-size = 512MB`
Informational Note:	Per Spring Boot docs, this setting specifies the maximum size of a single request via Spring Boot (and therefore via the DSpace REST API). That means if multiple files are uploaded at once, this is the maximum total size of all files. A value of "-1" removes any limit. DSpace sets this to 512MB by default.

Technical Design

The REST API & Server Webapp are built on Spring Boot and Spring HATEOAS, using Spring Security. It also aligns with Spring Data REST (though at this time it doesn't use it directly because of incompatibility with the DSpace data model).

The REST API is stateless, aligns with HATEOAS (Hypertext as the Engine of Application State) principles, returning HAL formatted JSON. This allows the REST API to be easily browsable/interactable via third-party tools that understand HAL & HATEOAS, such as the HAL Browser. JSON Web Tokens (JWT) are used to store state/session information between requests.

For better security, the REST API requires usage of CSRF tokens for all modifying requests.

More information can be found in the REST Contract at https://github.com/DSpace/RestContract/blob/dspace-7_x/README.md#rest-design-principles

DSpace Demo REST-API HAL Browser

- https://demo.dspace.org/server/

DSpace Python REST Client Library

The Library Code is a DSpace Platinum certified Service Provider. They created a python library to use the REST-API of DSpace 7 and upwards out of python: https://github.com/the-library-code/dspace-rest-python.