REST API Contract

This page was an initial attempt to start a conversation over the new REST API contract. We have now agree to have this conversation over a dedicated GitHub repository using pull requests and issues

see https://github.com/DSpace-Labs/RestContract

Other useful resources about the necessary endpoints and functionalities of the REST API are:

- Terrence W Brady's brainstorming on GitHub https://github.com/terrywbrady/restBrainstorm/blob/master/README.md
- John Francis Mukulu's brainstorming on Google Docs https://goo.gl/tPND8g

Old notes not yet moved to the above github repository

Browse milestone - Feature requests

Browse milestone

The repository has a homepage. The community / collection / item structure can be browsed. There are community and collection home pages and item pages. Bitstreams can be downloaded. The repository can be indexed by google scholar.

Bitstream

- name
- description
- format
- size
- download URL
- relationships
    - Bundle
    - Metadata
    - What with derived bitstreams?
        - thumbnails
            - would be best to keep the logic that determines bitstream X is the thumbnail of bitstream Y out of the UI.
            - andrea: rest is indeed better, but should we improve this in the datamodel?
            - tim: do it in rest, keep it based on the filename in dspace 7 fix it later
            - andrea: it is easy, using metadata for all
        - extracted texts
        - …

Bundle

- Name
- relationships
    - Bitstreams
    - PrimaryBitstream
        - tim: should primarybitstream be on bundle or item?
        - andrea: let's just keep using the current implementation and keep it there
    - Item
    - Metadata

Item

- Name
- Handle
- lastModified
- isArchived
- isWithdrawn
- relationships
    - Bundle
    - Metadata
    - Collection(s)

Collection/Community

- Name

- Handle
- copyrightText
- introductoryText
- shortDescription
- sidebarText
- license
- number of items
- relationships
    - Parent(s)
    - Collections
    - Items
    - Logo

What are the “browse” endpoints?

- `/collections/65/items` or `/items?collection=65` (or `items?q=collection:65`, …)
- maybe both?
- andrea: items can be in multiple collections
- andrea: start with `/items` if we find a good use case to use `/collections/54/items` we can implement it later

General questions

Handle

- returns a reference to the proper endpoint?
- andrea: redirect, that way it can be the same for other persistent identifiers

Is there a use for a separate metadata endpoint in rest?

- where you could retrieve or query for metadata rather than items/collections
- as opposed to discovery, where the unit is the item.
- tim: wait until we identify a real use case
- andrea: don’t think so
- terry: the DSpace 6 rest query tools query on metadata from the perspective of an item. It is possible that a metadata service could be useful as a query starting point. Example: find all metadat with a URL to a DOI

JSON API or HAL

- Some references
    - HAL: Hypertext Application Language
    - Hypermedia as the Engine of Application State
- andrea: json api is more complete, but spring-data-rest works with HAL
- art: katharsis works with json api, but I have no experience with it
- art: and spring-data-rest looks like the bigger project so I would be ok with partial json api support due to that

How does pagination work?

- useful info
    - # results per page
    - total # of elements/pages
    - HATEOAS link to first/previous/next/last page
    - both agree
- Sorting?
    - direction
    - sort field(s)

Limiting the number of fields in the response

- profiles
    - minimal
        - all DSO’s have this
        - for e.g. in the trail

        - fields
            - DSO name
            - id
    - list-summary
        - all
            - DSO name
            - id
        - Item specific
            - authors
            - date issued
        - for collections/communities it is identical to minimal
    - list-detailed
        - all
            - DSO name
            - id
        - collection/community specific
            - description
        - Item specific
            - authors
            - date issued
            - abstract
    - view
        - item
            - author
            - date issued
            - abstract
            - identifier.uri
        - collection/community
            - returns everything
    - omitting the profile returns everything
    - e.g. GET `/items?field-profile=list-summary`
    - andrea & tim: ok
    - andrea: should be configurable on the backend
- also retain the ability to specify a custom set of fields?
    - e.g. GET `/items?fields=handle,dc.contributor.creator,dc.identifier.doi`
    - maybe that should be: GET `/items?fields=handle&fields[metadata]=dc.contributor.creator,dc.identifier.doi` because metadata is a relationship of item, not an attribute
    - andrea: not sure if necessary, if you have a rest endpoint that allows the creation a new profile
    - tim: let’s go with a limited set of profiles (with configurable fields), and leave configuring new profiles via the rest api until later
    - terry: I like the idea of a configurable profiles appropriate to an instance

How do we handle relationships for objects that have a lot of them?

- e.g. a collection can contain several thousand items
- but an item can also have more bitstreams that it may make sense to display at once.
- That problem is separate from regular pagination as the relation section should only return HATEOAS links, nothing else, unless you ask for it.
- However maybe we can use the same params, and first/previous/next/last links for relationships. e.g. see this forum post
    - andrea: we may avoid this problem by leaving out some relationships, for example, don’t specify all items in a collection objects
    - andrea: we can also mitigate this problem by using profiles that work on relationships as well as attributes, so you could specify a profile that leaves out certain relationships
    - tim: if all else fails we may need to look at pagination

How does inclusion of related resources work?

- often it will make sense to request related resources in the same request
- e.g. request an item, and get not only the bundles, but the bitstreams inside those bundles as well
- `include` param? e.g. GET `/items/15?include=bundles.bitstreams`
    - would return the item, it’s bundles and their bitstreams in a single response
    - multiple includes can be comma separated: GET `/collections/23?include=parents,logo`
- we’ll that the JSON API way

Every DSpace Object needs to show its location in the trail

- When showing an item page you don’t want to trigger a sequence of GETs to simply get the names of all its parent objects.
- Should we include the entire path to the root in the relationships section?

    - Maybe an includes option with a profile for the trail that only returns their name and an identifier (to be able to link to them)
        - That way we can cache the parent objects already, and just add extra information when we need it.
- Andrea: we could also solve this by showing in the trail how you arrived there instead of where it is in the tree, but there will most likely be a use case where we need to show the item in the repository tree

Use UUIDs?

- Not easy to type or to memorize
- What if two items merge, or versioning is used. We need a persistent URI for an object.
- Maybe we should also have a shorter ID that's only unique within one type of object?
- We could also use handles
    - In the UI I'd prefer to use `items/5123` instead of `handle/123456789/4561` handle should redirect to the item url
    - But we could use `items/123456789/4561` or some kind of permutation or hash of the handle
    - Problem with versioned items perhaps?
- tim: we have to use UUIDs or handles. Not calling them handles may not be a bad idea