DSpace 7 REST: Coding DSpace Objects

Code Branch

The code of the DSapce 7 REST API have been merged in the master branch: https://github.com/DSpace/DSpace/tree/master/dspace-spring-rest

Code Representation of a DSpace Object

DSpace API Object (hibernate): org.dspace.content.DSpaceObject

This is the representation of an object from the DSpace database. Since DSpace 6, this object is populated with hibernate.

Rest Object: org.dspace.app.rest.model.DSpaceObject

This is a plain old java object (pojo) representation of a DSpace object.

Coding the REST object

- Add private properties
    - Add a property to the class DSpaceObjectRest
- Add a get/set method
- Add get and set to the class DSpaceObjectRest
- To exclude a property from JSON representation, add @JsonIgnore to the get method or to the property
    - Example of exclusion in the class DSpaceObjectRest
    - Another example of exclusion in the class DSpaceObjectRest
- the REST Object should not includes collections (List, Set, etc.) to maintain relations with other REST Objects unless it makes sense to load in memory all the linked objects, instead it should contains information (probably in form of annotation) to discovery linked entities. When a collection could be already available in the corresponding Hibernate object depending on the performed operation the collections should be annotated to provide a more efficient strategy to load the (paginated) information than the native (unpaginated) hibernate lazy-loading

Hateoas Object: org.dspace.app.rest.model.hateoas.DSpaceResource

This representation of an object allows for

1. the embedding of other DSpace objects within the object. Embedded objects are always linked.
    a. property returns a RestModel object
2. linking to other external DSpace objects
    a. property returns a RestModel. Property is marked with @JsonIgnore annotation.

The base class in this package uses reflection to identify attributes that are actual links to other REST resources.

- DSpaceResource

If an attribute is of type RestModel, then the code will

1. wrap the linked REST resource inside a DSpaceResource (so to have the identifier, self link, and links to other resources). The wrapper is actually created by the Repository responsible of the specific resource (ItemRepository, BitstreamRepository, etc.)
    a. https://github.com/DSpace/DSpace/blob/master/dspace-server-webapp/src/main/java/org/dspace/app/rest/model/hateoas/DSpaceResource.java#L58
    This give a chance to add custom logic for extra links in specific resource
2. put the wrapper in the embedded section
3. clean the attribute (not sure if useful/required/right):
    a. https://github.com/DSpace/DSpace/blob/master/dspace-server-webapp/src/main/java/org/dspace/app/rest/model/hateoas/DSpaceResource.java#L64

Converter Object: org.dspace.app.rest.converter.DSpaceObjectConverter

Convert between the REST representation of an object and the Hibernate representation of an object.

Coding a converter object

- In the fromModel() function, map all object attributes from the persistence/Hibernate model to the REST model.
  - https://github.com/DSpace/DSpace/blob/master/dspace-server-webapp/src/main/java/org/dspace/app/rest/converter/DSpaceObjectConverter.java#L33-L50
- In the toModel() method, map all attributes from the REST model to the persistence model.

Repository Object: org.dspace.app.rest.repository.DSpaceRestRepository

Provide repository interface functions that return and manage REST representations of DSpace objects. It should provide methods to

- get a specific instance of the object (findOne)
- get all the instances, paginated (findAll)
- save an instance (save)
- delete an instance (delete)

Additional methods should be added following the conventions defined in the subsequent paragraphs.

Repository Object: search methods

Any additional methods that return a subset of the collection exposed by the repository should be annotated with @SearchRestMethod annotation so to be discovered as "search" capabilities of the repository and automatically exposed over the /search endpoint sub-path of the resource type (i.e. /community/search/top).

The java method in the repository class can be named in any way, the sub-path used to build the rest endpoint is by default equals to the method name but can be forced to a specific value using the name parameter in the annotation

The java method must return a Page of rest resources or a single resource but can accept any kind of arguments. If the method has a Pageable argument it is automatically bind, the other argument are bind from HTTP parameters using the Spring Converter Framework but they need to be annotated with the @Param annotation where the name attribute define the name of the HTTP parameter (see an example here)

Repository Object: massive conditional update/deletion

TBD. Probably it could be useful to introduce an approach similar to the one adopted for find methods with a common /bulk sub-path

Linked Repository Object (TODO - proposal)

Resources are typically linked with other resources of the same of different type. For example a collection is linked with the items that belong to the collection. It is not effective to include a list of items in the collection object because it doesn't scale. Also if the Hateoas object can wrap the list adding pagination on the server side we will have hit the full list. This mean that the **linked objects need to be retrieved using optimized and paginated methods**. This could be done essentially in two ways:

1. the Hateoas object adds a link, say *items* in the CollectionResource that refers to a search methods in the item repository (i.e. /items/search/findByCollection). The limit of this approach is that update of association couldn't be addressed on the same endpoint breking one of the REST principle related to the URL structures and HTTP verbs. It doesn't make sense to send a POST to the endpoint /items/search/findByCollection?id=xxxx to map a new item under the collection xxx. The same issue arise if we work on the other side of the relation, i.e. if we put the focus on the single item we will end with the endpoint /collections/search/findByItem?id=xxx.
   This mean that the search endpoint should be used for read-only relations or "custom views" over a relation.
2. the link is managed directly at the resource level. In our example this mean that the *items* link will refers to an endpoint like /collections/:uuid/items
   This bring the issue to decide where put the implementation of the retrieval logic. It could be placed in the Repository Object assigned to the specific type (for example CollectionRestRepository) or in a separate class. According to the Repository Pattern (https://martinfowler.com/eaaCatalog/repository.html) a repository is dedicated to a single Business Entity so this should be avoid. Thinking to the relation in terms of a separate Business Entity, it mal

Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Unable to locate Jira server for this macro. It may be due to Application Link configuration.

In we face with the issue to list all the items av ad-only relation) implementing it as a search methods in the itemRestRepository will mean couple the item, core data model, with the browse system that is instead something additional maybe a plugin or extra feature. Using a separate repository for the relation will automatically add an uniform support for update and deletion operations over the relation and also will help to lazy load additional information during the wrapping of the object in its HAL Resource representational. Indeed the HAL wrapper could inspect the object class and, if a relation is requested by the projection to be included in the serialization, the HAL wrapper can invoke the same repository methods invoked when the specific relation endpoint is hit. This will assure consistency and avoid code duplication

Managing Object Initialization (TODO)

- When referencing a linked object, how do we control the initialization/load of reference object?
  Proposal: as manage partial object is a common scenario in REST implementation regardless to the representation format used (HAL in our case). It makes sense to add support for this capability directly on the RestModel base class. It should be able to discriminate which properties are really null and which are uninitialized allowing the HAL Wrapper to lazy load relation if needed (see Linked Repository Object). The Converter object should set only properties that are already initialized in the Hibernate object avoiding to hit hibernate lazy loading. If the property is required to be included in the requested projection Hibernate Lazy load should be used only if there are not a defined Linked Repository Object able to manage the requested relation

Build the REST Controller(s)

The goal is to have a pluggable infrastructure so that new endpoint can be added implementing the previous relevant classes without the explicitly need to add new Spring MVC controllers. This assure easy and maintenable uniform implementation of the behavior described in the REST contract in terms of HTTP Verbs meaning, ETAG, URL structure, etc.

Up to now we have two REST controller

- org.dspace.app.rest.RootRestResourceController
  It is responsible to produce the root HAL document listing all the defined endpoints of the REST API
- ALPSController (TODO).
  upport for the ALPS protocol, see Unable to locate Jira server for this macro. It may be due to Application Link configuration.
- org.dspace.app.rest.RestResourceController
  It is the single point of entry of all the REST requests related to resources. It delegates the retrieval and save logic to the Repositories, HAL wrappers and Converters previously described.

For specific functionalities it will be probably easier to create a separate controller instead to force the use of the Repository model. This is true for features that don't deal with a specific resource like the ROOT HAL Document, ALPS, the global search (Discovery)