DRS2 APIs Harvard Library Lab Project – 3/1/13 Phase 1 Progress Report (rev2)

Spencer McEwen, Randy Stern

The goal of this phase of the project has been to scope the work and additional funding required to implement APIs for DRS2.

Summary: We have done the work necessary to scope the DRS2 APIs.

- Randy Stern and Spencer McEwen met weekly to define the project goals and spec a strawman API for a "Public API" for publicly accessible content and metadata in DRS2. See appendix 1.
- A wiki site was constructed (https://wiki.harvard.edu/confluence/display/libdrsapi/DRS+API+Project) to document the vision for the project, collect user stories and requirements for the APIs, and outline the evolving API specification
- 3 stakeholder meetings were held to gather input – one with Library Technology Services staff, and 2 with a range of potential consumers of DRS APIS – Library Imaging Services, Berkman Center, Harvard Law Innovation Lab, HUIT Academic Technology Services, Schlesinger Library, Medical Informatics, and DataVerse Network developers. Feedback influenced the initial proposed design:
    - The initial proposal was for a public-only API. There were substantial Harvard staff requirements to be able to search and access entire collections, including "restricted to Harvard" data. So, we plan to add optional HTTP authentication to the API, as well as an API access key. See appendix 2.
    - There were requests to be able to access master content files, such as TIFF or JP2 images, so that these could be delivered to end users for certain collections or so that special delivery systems could be created. This will be enabled for authenticated and authorized API users.
    - There were requests for the ability to bulk download DRS metadata and content. This could come in the form of an OAI data provider. Since DRS2 content and metadata are very large, and a program that iterates over a list of content could effectively produce a bulk download, we decided to defer this requirement.
    - There were requests for a IIIF API for image content delivery. We have included this in the proposal as a separately fundable phase.

Scoping of remaining work

For scoping, we have divided the scoped implementation work into 3 additional phases. Phase 1 and 2 are a pair. Phase 3 can be done in parallel or sequentially.

Phase	Resources	Work time estimate	Start date	End date
1. Technical design, including authentication/authorization for access to restricted DRS resources	Spencer	105 hours	5/1/13	8/1/13
2. Implementation –Implementation of search, metadata read, and content read APIs, including authentication /authorization for access to restricted DRS resources. Development work will utilize existing LTS development server. *NOTE: This estimate assumes Spencer does the API development work and we use the contractor to backfill on his DRS2 work.*	Spencer (supervision) Java contractor	32 hrs* 455*	8/1/13	12/1/13
3. Implementation of content delivery API that conforms to the International Image Interoperability Framework for the DRS Image Delivery Service.	Java contractor or LTS developer	280 hrs	6/1/13	9/1/13

*This is our best estimate at present, but we will reassess after Phase 2 is complete.

Planned Work

PHASE 2 – Technical design, including authentication/authorization for access to restricted DRS resources. In this phase, Spencer will work to specify the details of the API by writing the API documentation. Review the API again with stakeholders. He will also write a technical design document that can be used for supervised software development.

PHASE 3. Implementation of search, metadata read, and content read APIs, including authentication /authorization for access to restricted DRS resources. Develop the software in java as an extension of the DRS2 web services layer that is already in place. Utilize RESTEasy, J2EE, and existing DRS2 search web service and metadata access protocols.

PHASE 4. Implementation of content delivery API that conforms to the International Image Interoperability Framework for the DRS Image Delivery Service. The existing DRS2 Image Delivery Service (IDS) implements the Luratech Image Content Server, and a proprietary API for delivering JPEG images from JPEG or JPEG2000 master images. This project will augment the IDS by adding a second API layer that is compatible with the open access IIIF (http://lib.stanford.edu/iiif). This will allow Harvard digital manuscript page images to be displayed in third party image viewer and web applications that are IIIF compatible, including the Mellon funded Digital Medieval Manuscript Interoperability project (http://lib.stanford.edu/dmm). This work is not dependent on Phase 3 work, and can be started at any time resources are available.

Appendix 1 – Strawman API

The proposed API will be RESTful. It will support content negotiation: API calls may return alternate MediaTypes as requested, such as "*application/xml*" and "*application/json*".

All APIs listed here are subject to change during the detailed design phase/

/drs/apiV10/searchMetadata/<query>

Item types	objects, files, batches, events
Input	expose solr queries directly (with some field renaming)
results schema	solr document (with some field renaming and redaction)
results formats	XML, JSON, CSV

/drs/apiV10/searchContent/<query>

Item types	full text OCR data for page turned objects
Input	keyword strings, much more (see FTS API Spec)
results schema	Full Text search service XML format, including links to specific pages in PDS documents that match the query text (see FTS API Spec)
results formats	XML, JSON, CSV

/drs/apiV10/getMetadata/<URI>|<drsID>|<delivery URN>

Item types	objects, files
Input	URI, DRSID, or delivery URN
results schema	DRS specific, (with some field renaming and redaction)
results formats	XML, JSON

Issues

1. May add helper APIs as well
    1. getMetadata/<object or file URI> (essentially the entire METS object for objects)
    2. getAdminMetadata/<object or file URI>
    3. getDescriptiveMetadata/<object or file URI>
    4. getRightsMetadata/<object or file URI>
    5. getEventsMetadata/<object or file URI>
    6. getTechnicalMetadata/<file URI>
    7. getStructure<object URI>
    8. getFileList<object URI>

getFileContent/<URN>|<file URI>

Item types	files (Access Flag=P only)
Input	URN, file URI
results schema	file format dependent – IIIF for images
results formats	file format dependent, deliverable formats only (e.g. no JP2, JPEG only at max image size)

Appendix 2 – API Security

Authentication

The DRS2 API will make a distinction between anonymous public users and authenticated users. Authenticated users have full access to their content, including the metadata and original quality images. Anonymous public users would be restricted to certain metadata fields and content with a P access flag. Max image sizes and audio stream/download rules would also be enforced.

Several options exist for authentication

1. Basic HTTP authentication. LTS will create and store api_key = app_key; api_secret_key = app_key_pass for each registered application. These can be passed in via HTTP headers or as query string parameters
2. Query authentication. Sign query string parameters - LTS creates app keys and passwords. Query string must be ordered, include a timestamps, encrypted with the password, then sent to the API.
3. Client certificate authentication. We would need to create and distribute certificates. Jboss validates the client, the API code reads the api_key (or equivalent) from the certificate.
4. Use an API key only?

HTTPS will be required regardless. LTS will need to store the api key and password secrets. Example) https://www.stormpath.com/docs/api/authentication

Ideally the DRS2 Web Admin would allow a user to regenerate their API secret key.

Since all requests will be over HTTPS then option 1, basic HTTP Authentication should provide sufficient security and be easy for any client to use.

Authorization

Applications must be registered and associated with the specific DRS2 owner codes that they should be able to access. The existing LTS Policy service may be able to store the application ID, owner codes, and DRS2 API role.

API Rate Limiting

Several options exist for rate limiting API access:

1. Allow authorized users to have x requests per minute or hour. Unauthorized (public) users would be tracked by IP address and have a lower number of requests.
2. Use a leaky bucket algorithm: http://stackoverflow.com/questions/1375501/how-do-i-throttle-my-sites-api-users Authorized users would be allowed a certain rate; unauthorized (public) users would be tracked by IP and be allowed a lower rate.

To accommodate future load balancing these values need to be either tracked in a database or a distributed memory cache (EHCache, Redis, Infinispan)