# 2016-11-15 DSpace 7 UI Working Group Meeting notes



| | |
|---|---|
| **Date** | 15 Nov 2016 |
| **Goals** | • Make a number of decisions about the structure, build tools, ways of working, etc. |

## Detailed note for the Angular 2 UI meeting (1st hour)

### Seed Project & build tools

#### Description

- There is no seed project ideally suited for our purpose at the moment.
- Angular-cli is the recommendation from the angular team, and is assumed to be used in most guides and posts
- But the current stable angular-cli branch isn't compatible with angular universal
- there is a fork that adds compatibility released as a separate npm package called universal-cli
- That has also been merged in to angular-cli a few days ago: PR
- But isn't part of a release yet
- We could
  - start from the universal seed project
  - start from universal-cli
  - start from the angular-cli master branch
- Something else to keep in mind is the ability to keep it modular and extensible. If this becomes the DSpace UI, how do customisations to it work?
  - Do people fork the UI github and work on the code?
  - Does the UI become an NPM package that people include in their own projects?
  - Should there be some sort of overlay system, like in the the current XMLUI?

#### Conclusions

- None of us have enough experience with these projects. They need to be tried out and compared. Art Lowel (Atmire) will do this by next meeting
- We want to use tslint and codelyzer in our build, for code analysis
- We also want to include an editorconfig file to help with code consistency across different developers

### Testing

#### Description

- Do we want to unit tests for every part of the UI?
- Do we want end to end testing
- Jasmine for unit tests, protractor for end-to-end, karma as a runner?
- Do we want to encourage BDD?

#### Conclusions

- We want to work as test driven as possible.

- However we don't want to make unit tests a hard requirement to be able to commit anything. But there have to be tests before it can be merged in to master
- We can always relax that policy if it proves to be too restrictive
- End to end test shouldn't be required but should be encouraged
- Jasmine, protractor and karma are fine

## i18n

### Description

- Angular 2.0.1 added a first version an i18n system, however it's still tagged as experimental and that shows.
- It's not easy to work with
    - your translations files have to be generated from the templates using a tool. If you add fields afterwards, you have to generate it again, and merge the already translated messages in
- It doesn't seem to support variables
- and doesn't support language switching at runtime.
    - It creates a version of the entire app in every language and you need to go to the one in the language you'd like.

### Conclusions

- we'll stick to ng2-translate for now, which is the third party module we used in the prototype, as it doesn't have all these downsides.

## CSS

### Description

- Framework
    - Angular material still has multiple downsides
        - lacks a number of components
        - lacks customisability
        - lacks basics we need, like a grid system
    - Bootstrap 4 while in alpha seems like a better choice
- Do we want to use component styles or not?
    - while that has the advantages
        - that your style is highly modular
        - that component styles can't influence or break each other
    - How will that work in respect to theming?
        - will that require someone writing a theme to modify the component CSS for every component?

### Conclusions

- We'll use bootstrap 4, with ng-bootstrap instead of ng2-bootstrap
- Component styles shouldn't be a problem for themablity as global, overriding styles can be added using the view encapsulation property of the app component
- So we'll use component styles

## Storing state & Data Abstraction Layer

### Description

- In order to end up with a robust system, with good performance, that's easily testable and minimizes the risk of unexpected side effects, we want to avoid leaving the implementation of calls to the backend, and the storage of state up to the individual developer of a component or service.
- To manage state in the application we could look towards Redux, and ngrx
- For a structured data abstraction layer JS Data is worth investigating
- Both of these could be used separately or together.
- While they have many advantages, they'll also make it more difficult for new developers to get involved. Is the extra structure worth it?

### Conclusions

- There is also universal storage, part of angular universal that has some overlap
- Not enough people have experience with redux, js data or universal storage.
- We will research this by the next meeting, in order to come to an informed decision.

## UI Style Guide

### Description

- Should we create a document that describes how the UI should work, what it's components are, how they should be designed, what is customizable in the theme, etc.
- Do we create one single theme, or a version that is a blank slate, and an opinionated version that showcases what can be achieved
- Do we make (part of) this before development can start?

### Conclusions

- In order to achieve a cohesive design it's important to start formalizing these things early on
- We need a first draft during the initial stages of development, and adjust it later on if necessary
- We should focus on a "blank slate" theme first, as it will be easier to create an opinionated version started from that as the other way around
- Afterwards create the opinionated theme

## Misc

### Description

- Are we going to require that code be documented using TypeDoc as in the prototype?
- Will we use a code style guide?
- How do we decide and communicate about the use of a new 3d party library or component?
- Do we put resources on the Duraspace or github wiki?
- We could remove the concept of communities from the UI, and use the term collection for everything, even if nothing changes to the backend. Is dat something we want to do?
    - it would bring DSpace in line with PCDM

### Conclusions

- We'll ask everybody to use TypeDoc
- We'll use the standard angular code style guide as a starting point, deviations from it can be added later on.
- If you want to introduce a new 3d party dependency you should bring it up in these meetings
- We'll put everything on the Duraspace wiki, but will add a summary to the README.md file on github
- We'll remove the concept of communities from the UI. However we should take care not to introduce too many changes in DSpace 7 to keep the workload under control.

# Detailed note for the new REST API meeting (2nd hour)

## REST API Contract

We agree about the opportunity to base our work on a clear contract document to enable the Angular 2 team to work on mockup services and provide early feedback without the need to wait for a full implementation on our side. We have discussed about the best way to share the documentation and receive feedback and we have agreed to stay as much as possible close to the official DSpace channels to keep all the community involved. Concretely this mean that:

- discussion will happen on the dspace-devel mailing list
- source code will be hosted in the official DSpace repository on the master branch (DSpace 7)
- a new maven webapp will be introduced to build the new REST API
- the contract will be maintained if possible on the dspace wiki, the decision about which framework to use to build the REST API could lead to a difference prefer as a static website build on a github doc repository (see point Spring Documentation REST project)

As good example of well documented REST API contract we will look to

- Fedora 4 REST API Documentation: https://wiki.duraspace.org/display/FEDORA4x/RESTful+HTTP+API
- Atlassian JIRA REST API: https://docs.atlassian.com/jira/REST/cloud/

At minimum the REST API contract needs:

- list all the available endpoints
- provide the list of accepted methods and their meaning for all the endpoints
- provide the details about the request format expect as input for all the endpoints with samples
- provide the details about the response format for all the endpoints with samples
- provide details about response code, and error codes for all the endpoints

## Backward compatibility

We discussed about the need to guarantee compatibility across new version of the REST API both in regards to DSpace 6 vs DSpace 7 than DSpace 7 vs DSpace 7+. The general agreement was to keep in place support for the DSpace 6 REST API for at least 1 version (so in some way deprecate the current REST API in DSpace 7) as a separate webapp build out-of-box in DSpace 7.

The new REST API should be structured in a way that new version are additive in information and functionalities so that the impact on the client is minimal and, when the client is properly programmed ignoring unknown information, new version of the new REST API are fully backward compatible.

## Design decision

We agree to implement support for the HateOAS paradigm. We need to take a decision about the specific hypermedia format to use to keep the new API consistent and easy to use by client developers. Candidates are as follow

- HAL - http://stateless.co/hal_specification.html

- JSON+LD - http://json-ld.org/
- JSONAPI - jsonapi.org/format/

HAL is supported out-of-box by the Spring Data REST project that looks as one of the most interesting framework to use to build the new REST API. JSONAPI looks as the most advanced standard and it is originated by the ember community.

## Framework to evaluate

We want to keep the development of DSpace as effective and sustainable as possible. To do that we need to pick technologies and framework that play nicely together with no or minimal effort from the DSpace developers (focus on developing DSpace not the framework) in this regards we are looking to the solution provided in the Spring family

- Spring Data Rest - http://projects.spring.io/spring-data-rest/
  It allows to build almost automatically REST API from Spring Data Repository. It provides configuration options to customize the resulting REST endpoints, parameters and provide support for API discovery (Application Level Description) and profile (how much details and information include in a specific response with client negotiation - profile)
- Spring REST Documentation - https://projects.spring.io/spring-restdocs/
  It allows to build the REST contract using test. This will guarantee a good test coverage other than accurate and easy to maintein documentation as it will be close to the source code
- Katharsis a thirdy part framework for JSONAPI REST API integrated with Spring MVC https://github.com/katharsis-project

## Other potential goals

We have discussed about

- the possibility to merge community and collection: we agree to keep a look to this option, the Angular 2 UI team will start to design an UI where there is not difference between Community and Collection. We need to check how much complex is to really remove such difference in the data model or at least hide such difference behind the REST API (i.e. have a single endpoint that will talk with both communities & collections). We agree that **this is not a primary goal** for the project and it will be delayed if it cannot be accomplished with minimal effort
- to simplify the packaging of DSpace ending up in a single webapp including all the supported modules SWORD, SWORDv2, RDF, OAI. Servlet 3 and web-fragment could be a good way to achieve that and also allow extension of the new REST API / single webapp in a pluggable way. We agree that **this is not a primary goal** for the project and it will be delayed if it cannot be accomplished with minimal effort
- authentication & authorization. **We want to explore the use of Spring Security** to simplify both aspects of the DSpace architecture

## Next steps

Make experiments with the proposed frameworks

Some PRs should be prepared to

- add a new (almost empty) web application to hold the new REST API code
- remove the JSPUI and XMLUI maven project from the DSpace github repository moving them to dedicated backup repositories under the DSpace organisation

Prepare an exhaustive list of functionalities that need to be exposed over the REST API staring from the need for the next Angular 2 UI milestone (Browse)

List current known limitation of the DSpace REST API