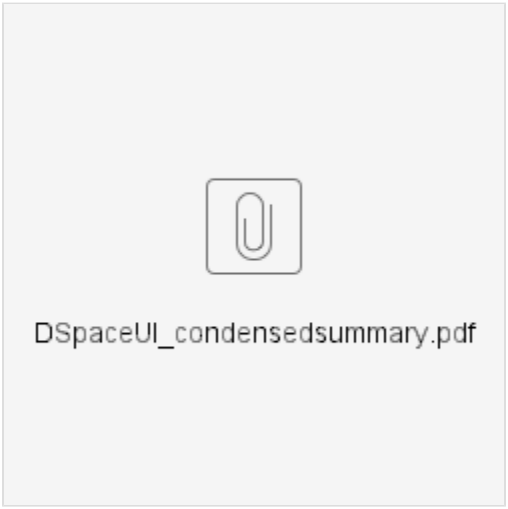
DSpace 7 UI Project Plain Language Summary



DSpace 7 UI project – plain language summary of the update webinar (slides/recording here: http://duraspace.org/node/3103)

Background to the DSpace 7 project

- Starting in 2015, a DSpace 2015-18 Strategic Plan was created/adopted, along with a Technical RoadMap. For more details see: 2015 Strategic Planning Activities and the OR2015 presentation "DSpace Technology Roadmap"
  - The #1 priority on that RoadMap was to adopt a new, single User Interface (to replace the aging JSPUI and XMLUI).
- In late 2015, a DSpace UI Prototype Working Group was established to develop/create a DSpace UI Prototype Challenge. The goal of this challenge was to hold a "friendly competition" to prototype several different technologies for a new, single User Interface. The Prototype Challenge ended in Dec 2015.
- In early 2016, the DSpace UI Prototype Working Group began analyzing the 9 submitted UI Prototypes, and put out several calls for feedback from the broad community. See also 2016 Strategic Planning Activities.
  - The two options were narrowed to either a Java UI or a Client Side Javascript UI.
  - During this activity/analysis, we discovered that the Angular 2 platform had been released (in beta) in Dec 2015. While one Prototype featured Angular v1, we had several concerns (namely Search Engine Optimization and accessibility) with Angular 1. The Angular 2 release promised to support both SEO and accessibility
- In March 2016, at the DuraSpace Summit, the DSpace Steering Group and DSpace Leadership Group were presented with pros/cons of a Java UI versus a Client Side Javascript UI. We also detailed our (very early) findings on Angular 2 regarding its very promising claims to support SEO / accessibility.
  - Steering and Leadership saw great promise in a Client Side UI built on Angular 2. But, no one was comfortable with finalizing that decision without a proof of concept.
  - Immediately after the Summit, four institutions (DuraSpace, Atmire, Texas A&M and Cineca) decided to collaboratively build a rapid proof of concept / prototype UI on Angular 2.
- From March to June, the four institutions publicly collaborated to build a basic, proof-of-concept Angular 2 UI against the DSpace 5 REST API
  - During this prototyping, it was discovered that our REST API was lacking in features, and was not currently capable of supporting a full-fledged client side UI.
- By late May / early June, the group presented findings on Angular 2 to the DSpace Steering/Leadership Groups. Both groups approved of the direction.
- In June, at OR2016, the proof-of-concept Angular UI was presented/demoed. See the OR2016 presentation "Introducing the New DSpace User Interface"
- (From June until November 2016, developer concentration moved over to getting DSpace 6.0 released. DSpace 6 is the final release to include the JSPUI and XMLUI.)
- In late 2016 / early 2017, a DSpace 7 Working Group (2016-2023) was established to formalize the process of building a new Angular2 UI and enhanced REST API for the DSpace 7.0 release.
  - Early in 2017, this UI Working Group has concentrated on planning out the architecture (especially for the enhanced REST API) and building the "framework" for the Angular UI.
  - In parallel, a DSpace 7 Marketing Working Group (2016-2020) was established to help with outreach, and update/capture use cases needed to be met by the new UI. See: Use Cases

Breaking down the technology

What is Angular 2? Why choose Angular2?

Angular 2 is the second version of the Angular Javascript framework built by Google: https://angular.io/

There are several reasons Angular 2 caught our immediate attention when it was beta released in late 2015 (For more on some of these see: https://angular.io/features.html)

1. Angular 2 supports Search Engine Optimization (SEO). In other words, applications built with Angular 2 can be easily indexed/searched by Google or Google Scholar (NOTE: we verified this by running a series of tests with Google Scholar team).
2. Angular 2 supports Accessibility guidelines. Screenreaders (and similar) can have issues with Client Side Javascript applications. However, again, we verified Angular 2's accessibility promise with the help of accessibility experts at U of Kansas
3. Angular 2 supports web archiving (e.g. Internet Archive harvesting and similar). This also can be problematic for some Client Side Javascript applications. However, we verified this with the help of RCAAP (Portugal).
4. An Angular 2 application works even when Javascript is turned OFF. While this may sound strange, this is the feature that supports #1-3 above. Your users don't need to even have Javascript running to use an Angular 2 application. To support this concept, Angular 2 pre-compiles Javascript into "static" HTML on the serverside. So, when Javascript is turned OFF, your users simply request new pre-compiled, static HTML pages each time they click links or buttons. This is done by a module called "Angular Universal" (that is bundled with Angular 2)
5. Angular 2 applications are written in a language called TypeScript (built by Microsoft). TypeScript is an enhanced version of Javascript (and while it's new, it is widely supported by Microsoft and Google). Typescript also supports many Java-like concepts. So, we feel developers familiar with Javascript or Java will be able to pick up this new language quickly. It is also easier to debug (with many debuggers/editors) than traditional Javascript.
6. Finally, the Angular framework is the most popular / widely used client side Javascript framework. So, there are many opportunities for support, and many third-party plugins available for building/enhancing Angular 2 applications.

Finally, Angular also provides all the benefits of a client-side (Javascript) application. Those benefits include:

- *More dynamic and modern user experience.* While this can also be achievable in server-side technologies (Java, Ruby, etc), those technologies would require utilization of Javascript frameworks (jQuery or similar) to provide such an experience.
- *Better separation of concerns between UI and backend.* Building the user interface on a client side technology forces a distinct separation between server-side (backend) code and the user interface. While this separation is not required, it is a best practice. It also has the advantage of allowing the UI to potentially run on a separate server from the backend (REST API).
- *Better / enhanced REST API (for future integrations).* While DSpace has had a REST API since DSpace 4.0, this REST API is very limited in functionality and practical use cases. Building a client-side application *requires* a stable, well-documented REST API that provides all necessary UI features/functionality. A fully featured REST API also has a secondary benefit of providing an easier path towards future integrations with DSpace (by other third-party platforms or plugins).
- *Innovative / exciting.* No software platform can last forever without continual modernization / enhancements based on the latest technologies. DSpace is no different. Both of the existing UIs are outdated. (While it has received recent code refactoring, the JSPUI was initially released in 2002. The XMLUI was released in 2008, but relies on an unmaintained, nearly obsolete, Apache Cocoon framework.) Updating DSpace to use modern technologies will not only improve the user experience and UI development processes, it'll also help us get newer developers involved and excited about DSpace.

Why are we re-building the REST API?

While DSpace has had a REST API since DSpace 4.0, this REST API is very limited in functionality and practical use cases:

- 4.x REST API is read-only.
- 5.x - 6.x REST API allows for basic read/write functionality. However, many other UI features are still lacking, including:
  - All administrative functionality (i.e. anything in Admin UI screens)
  - Search/Browse via Solr (including browse by facets)
  - Submission Process (You can manually create new Items and add metadata via the REST API, but it is not a staged process and requires a large number of requests to complete)

In addition, the existing REST API has the following disadvantages:

- While it is a decent, first-try at a REST API, it does not follow any modern best practices for REST APIs (e.g. HATEOAS (Hypertext As The Engine Of Application State), ALPS (Application Level Profile Semantics), etc)
- Similar to the first point, the REST API is mostly "handcrafted", custom code (i.e. it defines its own response syntaxes, not based on any widely adopted standards). While it does provide basic documentation on those responses, there is no formal REST "contract" (a contract is detailed documentation on how a REST API "promises" to interact with any external system)
- It uses a third-party library (Jersey) that is not widely used in the DSpace codebase, and does not provide the modern best practices mentioned above.

Therefore, we have decided to rebuild the REST API for the following benefits:

1. **We can update it to follow modern best practices** such as HATEOAS (Hypertext As The Engine Of Application State), ALPS (Application Level Profile Semantics), and using the HAL (Hypertext Application Language) response format.
   a. Updating to use these best practices would make our REST API easier to understand, and allow widely-used third-party tools to immediately interact with it (e.g. a HAL Browser exists which can parse/respond to **any** REST API that uses the HAL response format). Since our current REST API is completely custom, no third-party tools exist that can interact with it immediately.
2. **We can update it to use modern technologies** (such as those provided by the Spring Framework)
   a. As of DSpace 6, the DSpace Java API now uses Spring technologies throughout. There are newer Spring technologies that allow us to more easily implement modern REST API best practices (e.g. Spring HATEOAS). So, rather than building our REST API in a custom manner, we can rely on third-party, Spring libraries to implement these REST API best practices.
3. **We can more easily enhance and maintain it.**
   a. Updating our technologies and using third-party libraries (like Spring Framework libraries) to achieve best practices also will make our REST API codebase easier to manage and maintain in the long run. Less of our code will be custom, and we'll be able to receive enhancements/improvements from the widely used Spring libraries.

Keep in mind, if you have local, custom tools or applications that already make use of the (existing) DSpace REST API, they will need updating once this new REST API is released (as we get further along, documentation will be provided). However, if you do not currently make use of the REST API, these REST API changes will have no effect on you (other than to provide the backend that the new Angular UI).

What have we done so far on this?

As of March 2017, both the new REST API and the Angular UI are still in their "infancy". Much of early 2017 was spent hashing out how the new REST API would be built, and beginning to build mockups of that REST API, which the Angular team is basing early development on. However, both projects will begin taking larger steps in the very near future as we work towards a goal of a basic alpha-level demo of search/browse functionality by OR2017 in June.

During early 2017, much of the REST API work was around analyzing REST API technologies and best practices to determine whether we truly needed to rebuild our REST API, or whether we could "morph" the existing REST API to meet the needs of the Angular UI. In the end, for the reasons detailed above, we've decided a new REST API will be the easier route for the long term. This team has begun laying the groundwork/foundation of the new REST API. This work is progressing in the following areas:

- The REST API codebase is in the main codebase at: https://github.com/DSpace/DSpace/tree/rest7
- Updates / ongoing discussions occur on Slack (#rest-api channel) and weekly meetings. See notes at DSpace 7 Working Group (2016-2023)

During early 2017, much of the Angular UI work revolved around building out the groundwork (in Angular) to support the new REST API, and beginning to create interactions with a "mock" (i.e. faked) REST API. As the new REST API is still in very active development, the Angular team has had to develop against faked versions (until the real one is far enough along to interact with). This work is progressing in the following areas:

- The Angular UI codebase is at https://github.com/DSpace/dspace-angular/
- Updates / ongoing discussions occur on Slack (#angular-ui channel) and weekly meetings. See notes at DSpace 7 Working Group (2016-2023)

Other Questions:

– what improvements will I see as an admin? Will my end users see from these changes?

How are we using these technologies to build the UI - a description, in plain English, of the potential that is provided by using the combination of Angular2 + improved REST API

Project steps (doesn't have to have dates, but something to indicate where in the process you feel we are up to) – would be good to provide something visual for this, rather than text as the page will be text heavy

Tim used the phrase 'building the frameworks' – what does this mean in terms of the development roadmap?