Brainstorms on a Future UI

[June 2015] These brainstorms have now moved into a "Single UI Project"

While this "Brainstorms" page has been kept for historical reference, the development of a Single UI has been established as the top priority of the DSpace Technology RoadMap. Therefore, the brainstorms and discussions on this page are now slightly outdated.

The plans for developing a "future UI" have moved to the following wiki pages:

- Design - Single UI Project - Describes the schedule/plan for prototyping, designing and building a "future UI" based on the RoadMap priorities
- Prototyping a Future UI - Describes in more detail the prototyping phase of the "future UI" (which is the first phase of the overall "Single UI Project").

Discussion at 2015 DuraSpace Membership Summit

These brainstorms were discussed at a higher level during a Day 2 breakout of the 2015 DuraSpace Summit entitled "*The DSpace UI(s) – Can We Converge on a Single One?*". The general consensus during those discussions seemed to be that we should consider consolidation into a single UI.

The following slidedeck was presented during the discussions, detailing some of the breakdown of the current (as of March 2015) DSpace user base: http://www.slideshare.net/tdonohue/discussion-on-dspaces-two-uis-duraspace-summit-2015

Background Info: Why are we brainstorming this (again) now?

Establishment of DSpace Governance

- In 2014, DuraSpace helped the DSpace project establish it's first DSpace Steering Group. While initially "appointed", going forward this Steering Group will be elected. They now control the allocation of funds donated to DSpace (and the DSpace Tech Lead reports to them).
- In 2015, DuraSpace helped the DSpace project establish it's first DSpace Leadership Group. This Leadership Group is a larger group of community key stakeholders (primarily representing institutions who are also DuraSpace Members who have given money to the DSpace project). The Leadership Group will elect future Steering Group members, and they also represent the broad DSpace community and can vote to accept /reject any proposals from the Steering Group, Committers or DCAT. (NOTE: This group is still in the process of being formed)
- The Steering Group's role is to "ask the right questions" and make general suggestions for how the DSpace product may wish to move forward. They will work directly with Committers and DCAT to actually help **answer** those questions (Committers are still the primary DSpace technology decision makers, and DCAT is still the primary DSpace "use case" decision makers).
- One of the first questions that the Steering Group has asked is essentially: "Why are we shipping DSpace with two User Interfaces again? Doesn't that split up our resources significantly and make it harder to develop for DSpace? We should consider whether it is worth consolidating to **one,** out-of-the-box UI."

Questions this Brainstorm seeks to help answer

So, the question(s) this page is trying to brainstorm include:

1. Why are we shipping DSpace with two UIs (JSPUI & XMLUI)? Are there any advantages to doing so?
2. Should we consolidate into a single UI?
3. If the answer to consolidation is "yes", what UI should we consolidate under? Should we just ship with the JSPUI? Should we just ship with the XMLUI? Or should we build a new, modern replacement UI and ship with that?

Resources & Timeline

- *Assuming we did decide to rebuild/rewrite one of our existing UIs, or even build a new UI, how would we get enough resources (i.e. developers) to do this in a timely manner?*
  - If we decided to revamp or build a new UI, the Committers can recommend that to DSpace Steering. Assuming Steering approves, they would ask the Leadership group to vote on the idea.
  - If the Leadership group votes to approve the idea, then the Steering & Leadership would seek out the necessary resources to make this happen.
  - As some of the institutions represented on Steering & Leadership have DSpace developers (or even Committers) on staff, the hope would be that they would donate some developer time to help achieve our goals in a timely manner.
- *When would this happen? What is the timeline?*

    - There are NO set timelines for this decision as of yet. It's merely a brainstorm to get a sense of what the developers and Committers feel may be the best direction forward.
    - Tim Donohue will be updating the Steering Group on this discussion as it progresses, and if any timelines are set, the entire community will be informed.

Other Questions?

If you have other questions which are not answered here, please feel free to ask them (either paste them in this section, or email Tim Donohue)

Multiple UIs vs One UI

Why are we shipping DSpace with two UIs (JSPUI & XMLUI)? Are there any advantages to doing so?

Before deciding on a future direction for the DSpace UI(s), we have to face up to the "elephant in the room". **We currently are building, maintaining and supporting two UIs (JSPUI & XMLUI) under a single Committers group.**

Therefore, in order to move forward, we must make a decision on whether this direction is the best one for DSpace. As such, here's some pros/cons to multiple vs single UIs...(feel free to add your own)

Possible Benefits of Multiple UIs

- **Choice:** Having multiple UIs provides users & developers with a choice. They can choose which UI better fits their needs or their local technology expertise.
- **Competition:** Having multiple UIs provides friendly competition between UI developers. As one UI makes improvements / enhancements, it encourages the other to do the same (or risk losing users to the "better" UI).

Possible Disadvantages of Multiple UIs

- **Developer Resources:** Building, supporting & maintaining two UIs essentially requires twice as many developer resources. If the community is large enough (which arguably DSpace is), there may be enough developers to support this. However, this becomes less maintainable when a single Committers group is expected to be knowledgeable enough on **both** UIs to support/build/maintain both simultaneously. Two UIs really requires two committers teams (one specifically devoted to each UI).

Possible Benefits of a Single UI

- **Developer Resources:** Obviously, one UI requires less developer resources to build, support and maintain.
- **Easier to "Roadmap":** It is much easier to plan out a long term roadmap/plan for DSpace if we have a single UI which all features **must** integrate into. It becomes harder to plan out features that must be supported in multiple UI frameworks / infrastructures

Possible Disadvantages of a Single UI

- **One UI technology must rule them all** : Can we all come together to decide on a common technology framework that actually will meet all our needs? Or are there actually separate needs / use cases that warrant the building of distinct UIs (similar to Hydra project)

Should we consolidate into a single, out-of-the-box UI? (Please VOTE!)

Given the benefits and disadvantages above, one thing seems abundantly obvious: **We cannot reasonably expect to continue supporting two UIs with a single Committers team.** Or to restate that, it is unreasonable to expect any Committer (who are all volunteers, working at their own jobs) to be well versed enough to support, maintain, develop and review fixes for multiple UIs simultaneously. This is an obvious misuse of the volunteer resources provided. Each institution has already made their own personal decision on which UI they wish to use, yet we are essentially forcing some institution's developers (e.g. Committers) to also be knowledgable on the **other** UI (which they never use on a day to day basis).

Given this, it only seems reasonable to also conclude:

- **Conclusion: Our DSpace Committers group can only reasonably build/support/maintain a single, out-of-the-box DSpace UI.**
    - Please note this does NOT state there should only be **ONE UI** (as noted above there are some advantages to multiple UIs). It simply states that there will only be one **out-of-the-box** UI.
    - If there are enough developers/institutions who see an ongoing need for a secondary UI, they are welcome to build, support and maintain a secondary, optional UI with their **own, separate group of developers/committers.**
        - A sidenote of sorts: If a secondary "committers group" were to form around a secondary UI, it may someday make sense to "split" the "DSpace Committers Group" into several "sub-teams": One team in charge of the underlying API / REST API, one team in charge of the primary, out-of-the-box UI, one team in charge of the secondary UI (if any). These teams would likely have some overlapping members, but they'd each be self-sufficient and more tailored to the needs of each individual sub-modules.
- **Opinions? Please feel free to add +1 / 0 / -1 to this conclusion, and any comments you may have**
    - I AGREE: We only should maintain a single, out-of-the-box DSpace UI. If a secondary UI is built (or continues to be maintained), it should be maintained by a separate team of committers / developers (and therefore become a separate project or organization in GitHub).
        - +1 Tim Donohue
        - +1 Bram Luyten (Atmire)

        - +1 Mark Diggory (however, with the caveat that UI application logic be integrated into DSpace core such that additional UI may be easily authored and maintained externally)
        - (*add your name here, if you agree with the above conclusion. Feel free to also add additional thoughts/comments*)
        - +1 Claire Knowles
        - +1 Emilio Lorenzo (arvo)
    - I DISAGREE: We should continue to support/maintain multiple out-of-the-box DSpace UIs with our existing DSpace Committers Team
        - (*add your name here, if you disagree with the above conclusion. Feel free to also add additional thoughts/comments)*

What makes a good UI (framework)? What common "use cases" do we need to keep in mind?

The following is a list of features/needs/use cases which we feel would make a good User Interface / User Interface framework. Since not all of these features/needs would have the same importance, we've categorized some as "required", "recommended" or "optional". (*Please feel free to add more ideas /thoughts, if we are missing anything*)

1. **Open Source (*required*):** Obviously. Also we need to avoid GPL and similar licenses which are incompatible with BSD.
2. **Easy to run "out-of-the-box" (*required*):** in keeping with DSpace Vision, any UI or UI framework must be easy to get running "out-of-the-box".
    a. **DCAT feedback 2015-03-10:** We're not sure what easy to run out of the box means for a UI or UI framework. Does that mean that the framework in itself can't have too many dependencies? How would one framework qualify as easier to run compared to another one?
3. **Ease of Branding/Theming (*required*):** A User Interface should be easy for institutions of all sizes to brand or theme. This means even smaller institutions (without a full time DSpace developer) should be able to theme or brand DSpace with some amount of ease. At a minimum, things like the header/footer/color scheme and basic layout should be simple to modify or customize. Ideally, the UI would support third-party themes (e.g. Bootstrap themes from http://bootswatch.com/ or similar) which can be easily applied to the UI to change its entire look and feel.
    a. **DCAT feedback 2015-03-10:** We see it as a substantial feature/requirement to be able to apply **different** themes to different sections of DSpace (collections, communities). It would be great if "some amount of ease" would be more tightly defined as: Configurable within the user interface itself and does not require a rebuild or restart of the system, especially when we're talking about basic theme config changes.
    b. in addition to general "look and feel" theming, it should be possible to easily configure basic functionality, such as the order and selection of metadata fields to display on simple and full item pages. Such customisation belongs at a configuration level, and does not need to be intermingled with design concerns.
4. **Responsive Web Design (*required*) :** a UI should be responsive and mobile-friendly, adapting to the size of various devices.
    a. Bootstrap support (*recommended*): Ideally, the UI would support Bootstrap, since it is one of the most widely used and supported responsive frameworks
5. **HTML5 Support (*required*):** a UI should be able to support HTML5. Ideally, it is built primarily with HTML5 in mind, rather than only supporting some aspects of HTML5.
6. **REST API friendly (*highly recommended*):** a UI should be built with the idea of "separation of concerns". For example, the UI framework should include NO business logic or Database query logic, etc. It should also have no knowledge of the underlying storage framework (e.g. Database schemas, file storage locations, etc). Instead, ideally it would communicate with DSpace primarily through the REST API (and other similar layers, e.g. Solr or Elastic Search). It would NEVER communicate directly with the database or other underlying storage layers.
7. **Faceted/Filtered Search/Browse friendly (*highly recommended*):** a UI should easily integrate with a faceted/filtered search engine/server (such as Solr pr Elastic Search) or a generic API which can communicate with said faceted/filtered search engine (e.g. Discovery, Blacklight)
8. **Rapid Development support / Developer friendly (*highly recommended*):** a UI should be easy to develop against and improve upon. Ideally in a popular technology or language. Local developers should not need to go through extensive training to work with the UI. The framework and technology ideally should be widely used, so that newer developers can also quickly come up to speed. (Some examples: Ruby on Rails is a popular widely used technology/language. As is, seemingly, the Java Play! framework. Both are obviously much more widely used and easier to develop with than say Apache Cocoon)
    a. **DCAT feedback 2015-03-10:** This requirement could benefit from being split into two: on one hand there is the availability of learning resources, examples and a large community that results in developer friendliness. The other part would be the long term longevity /sustainability. One particular framework could be very well documented with nice examples, but if it is controlled by a smaller number of organizations it might score bad on a criterium for long term sustainability.
9. **Active, third party plugin ecosystem (*highly recommended*):** a UI framework should ideally come with an active plugin/module/tool ecosystem. This is not only the sign of a strong community around the UI framework, but also eases the development burden on DSpace developers, as we no longer need to build all features specific to DSpace. (For example, a UI framework that came with its own, third-party Authentication plugins would allow us to utilize that rather than building our own plugins for Shib/LDAP, etc)
    a. **DCAT feedback 2015-03-10:** Plugins like Authentication or other elements related to business logic might be out of scope for many frameworks that only deal with UI. It would be interesting to see how this requirement conflicts with the "separation of concerns" in 6. REST-API friendly.
10. **Standard way of dealing with internationalization (i18n) or translations (*required*)**: DSpace has multiple international language communities who each manage their own set of translations for the interfaces. Migration from the current way of managing these translations to the new framework should be possible. Contribution of new translations should not be more difficult than it is today.
11. **Java-friendly (*recommended*):** DSpace's underlying framework & API is Java, and likely will remain Java. There are no plans to completely rewrite DSpace. However, this does NOT mean the UI needs to also be written in Java, but it may be best that the UI technology is Java-friendly or even in a language that is similar to or based off Java (e.g. Javascript, Groovy, even Ruby is similar enough).
12. **Flexible URL Structure**: It might be too limiting to work with a UI framework that imposes a very specific, limiting URL structure.
    a. **DCAT feedback 2015-03-10:** Even though not all DSpace urls can be exactly the same in the new framework, DCAT still sees it as essential that ~~handle based~~ dspace item URLS should still be preserved. The use case here is that when someone has linked directly to a DSpace item url (and if the institution was not using handle.net), the links should ideally not break. As an alternative, it would be nice to have a ghost app or redirection service to keep the old dspace urls alive in case they are replaced by new ones. It is also very likely that URL namespace and structure should not be a UI Framework concern, but business logic/API design.
    b. Opinions:
        i. Mark Diggory : I just want to comment that **with the handle system, DSpace does not need to have "handle based address locations"** , they are an unnecessary redundancy and IMO, a "Red Herring" that appears to confuse those interested in citing DSpace content (http://some-dspace-host/handle/NNN.N/NNNN != http://hdl.handle.net/NNN.N/NNNN). In fact, removing this redundancy would eliminate such confusion and clarify what URI are handles and what URI are not handles. **The entire point of the handle system and handle URI is to create a Persistent Identifier for the resource that is dereference able and different than its actual location** . The reason for doing this is so that the PID cited for a resource can have its mapping adjusted in the handle resolver and still be resolvable should that resource need to be moved to a new platform. Please note that the inclusion of "/handle/NNN.N/NNN" as the URL for a Community, Collection or Item was an early developer decision in the initial creation of the system and that the options and impacts may not have been fully analyzed. It may be better

for the overall architecture and future of DSpace to focus on defining and better improving the integration and mapping capabilities of the CNRI Handle plugin and resolver rather than than to force these ambiguous resource addressing conventions on a future system and its users.

UI Framework Analysis (Please add more!)

Here's a few possible UI frameworks which we may wish to analyze for a single future UI. A much larger listing of various web application frameworks appears on Wikipedia: https://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

Please feel free to add more that you feel would be worth analyzing for DSpace!

UI Framework	Languages / Technologies	Widely Adopted?	Ease of Customization	Responsive web design support	HTML5 support	REST-friendly	Faceted /Filtered Search /Browse friendly	Rapid Development friendly	Third-party plugin ecosystem	Notes
Existing DSpace XMLUI	Java, Apache Cocoon,XSLT, **also leverages Spring WebMVC**	No	Not really (except maybe at Bootstrap level with Mirage2)	Mirage 2 theme = Yes Other themes = No	No	No	Yes	No	No	Apache Cocoon has very little adoption and support these days, and hasn't had a new release in many years. Apache Cocoon could be considered forked locally by most of the third party projects that utilize it.
	Personal opinions on DSpace XMLUI: • Tim Donohue: My personal opinion is that, as it currently exists, the XMLUI should not be the choice going forward as it is based on an outdated, nearly obsolete framework (Apache Cocoon). In my opinion, it would require abandoning Apache Cocoon to be in consideration. • Graham Triggs Drawbacks are size of the framework, complexity of the framework, lack of adoption and support for Apache Cocoon. • Mark Diggory: Note, since DSpace 1.8 XMLUI also provides Spring WebMVC context and control (whose viewing technology is still Cocoon). I consider the points *REST-friendly* and *Rapid Dev friendly* to be subjective, we to rapid dev in Cocoon often, but it certainly is not a Rails, Grails, Play experience). Also note, Considerable parts of JSPUI were copied to XMLUI and placed into Cocoon Action, Matcher and Transformer classes, this UI logic could be compartmentalized separate from both UI and used across all web-applications including dspace-rest. Examples include authentication session management, request management, context management, resource resolution, even parts of Submission and Workflow (jspui , xmlui) . An ideal path forward would migrate these features out of xmlui/jspui, where possible, make them UI agnostic, and place them into dspace-api.									
Existing DSpace JSPUI	Java, JSPs	No	Not really (again, except maybe at Bootstrap level with Mirage2)	Yes	A few areas (e.g .HTML5 upload), but not overall	No	Yes	No	No	The JSPUI codebase is approximately 13+ years old, despite some recent work to update it to use Bootstrap.
	Personal opinions on DSpace JSPUI: • Tim Donohue : My personal opinion is that, as it currently exists, the JSPUI should not be the choice going forward, as its codebase is extremely dated and not easy to work with (despite the recent UI redesign). In my opinion, it would require a major overhaul to be in consideration. To be clear, this doesn't mean JSPUI is "dead", just that it would need a lot of cleanup work / redesign if we want to go this route. • Graham Triggs : A rewrite would be essential - preferably moving away from JSP to a templating engine, even if not using a recognized MVC framework. However, the benefits of being based on a widely known technology and having a small footprint are apparent. • Mark Diggory : Any move to repurpose or evolve of JSPUI should include a rewrite of certain features: the DSpace JSP tag lib should not include html, beans and JSTL should be leveraged instead. See comments regarding XMLUI/JSPUI consolidation above.									
Spring WebMVC	Java, Many View Technologies (JSP,FreeMarker, Groovy, Thymeleaf , etc)	Yes	Yes	Dependent on View technology	Dependent on View technology	Dependent on View technology	Dependent on View technology	Dependent on Framework choices	No	Many java based frameworks utilize Spring MVC under the hood,
	Personal opinions on Spring MVC Framework: • Mark Diggory : As a core technology of many of the frameworks below, Spring MVC has a strong uptake. As with many of the frameworks below, we are not necessarily locked into the view technology choices for all our user base. We may set the stage for a migration to frameworks below by first adopting a practice of using Spring MVC in both the XMLUI and JSPUI. See comments above in XMLUI • Chris Wilper : Having used Spring MVC on several projects, I've seen cases where it has driven a pure REST/HATEOAS API, as well as HTML-producing endpoints (backed by XSLT transformation, velocity/freemarker templates, etc.). It has been around for quite a while and has a huge community behind it. A related project is Spring Web Flow, which looks a possible alternative to Cocoon's webflow for orchestrating certain user tasks, currently used by XMLUI. • Bram Luyten (Atmire): On the view end, Thymeleaf is a popular XML/XHTML/HTML5 template engine. See also this article Spring MVC: from JSP and Tiles to Thymeleaf • Graham Triggs : On Thymeleaf, it's probably one of the nicest "designer friendly" templating engines. However, the price to be paid for that appears to be 2x slower performance compared to Mustache or Freemarker. http://www.slideshare.net/jreijn/comparing-templateenginesjvm - it's worth considering who we really see as editing those templates as to whether they should be more designer or developer friendly (with CSS for actual styling, I would lean towards the latter).									
Play! Framework	Java, Scala	Yes, some major sites use it according to Wikipedia		Yes, can be used with Bootstrap				Yes	Yes, has a modules repository	
	Personal opinions on Play Framework: • Graham Triggs I had a brief play with it a while ago. It's a neat technology, but has drawbacks in being more tailored to Scala than Java, and lacking documentation. It's also very dependent on using the Play toolset, even though in the background it can use Maven to manage dependencies, there would be a lack of synergy between front end and back end development, which might be an issue.									
Spring Boot	Java	Not yet. It's still very new (1.0.0 released in 2014). However, the Spring IO platform itself is very widely used, and Spring Boot seems to have a lot of activity on GitHub, Stackoverflow, etc. Note, Grails is part of the Spring I/O application						Yes, it's built as a rapid development friendly version of Spring	Built on Spring, so you can use other Spring projects	

		stack. Appears to run directly on Boot in this case.								
	Personal opinions on Spring Boot: • Graham Triggs Initial tests are quite positive. Obviously integrates very well with the Spring ecosystem, yet you can easily create a 'standard' Maven project for the application. Uses many templating engines (my preference is for Freemarker). • Mark Diggory : Mostly a Spring runtime container, will support other views, does not directly addresses the two UI problem. Introduces a larger scale re-architecture project than just UI consolidation. • Tim Donohue : In about one day's time, I've built a basic "experiment" of Spring Boot + Thymeleaf + Bootstrap on DSpace (master). This was mostly just to "learn" Spring Boot, but I found it so easy to work with that I was able to do a lot rapidly. The experiment itself is extremely basic (not a full fledged prototype). But it "works", and Spring Boot integrates "automatically" with our existing Spring beans (no major re-architecture necessary, though we do it later on if we wanted). Code at https://github.com/tdonohue/DSpace/tree/spring-boot-ui/dspace-ui									
Ruby on Rails	Ruby	Yes		Yes, has a Rails Bootstrap app, plus many gems				Yes	Yes, in form of Rails plugins & Ruby gems	
	Personal opinions on Ruby on Rails: • Graham Triggs Clean separation of front end and back end (via REST calls), will add latency and may cause problems for switching between front and back end development. Also, hosting concerns (JRuby vs separate Ruby and Java containers) • Mark Diggory : Much more than a UI framework, opens the door to large questions concerning rewriting DSpace, takes us away from a single java technology stack									
Hydra Framework	Ruby on Rails, Fedora, Blacklight	Not worldwide, but has a growing following in libraries, etc. The base technology, Ruby on Rails is widely adopted		Yes (well, Sufia uses Bootstrap)		Yes (uses REST to communicate with Fedora)	Yes (Blacklight)	Yes	Yes, because it's Ruby on Rails, you often can use Rails plugins and/or Ruby gems	Hydra doesn't currently "work" with DSpace. It would likely be a major endeavor to either migrate DSpace into a "Hydra Head" web application or "port" Hydra as a UI on top of DSpace's underlying architecture. However, if we decided on the former (create a DSpace-like Hydra Head), there are members of the Hydra Community who are currently striving to do that same thing.
	Personal opinions on Hydra: • Mark Diggory : Much more than a UI framework, in either case its a larger migration path and endeavor than consolidating DSpace onto a single UI technology. In either of the proposed cases it does not solve the immediate need for a single UI.									
Grails	Groovy (based on Java), **Also based on Spring WebMVC**	Yes, large number of sites using Grails listed on website		Yes, has several Bootstrap plugins		Yes		Yes	Yes, has a plugins repository	
	Personal opinions on Grails: • Graham Triggs Pivotal (the main commercial developer) has recently announced that it is no longer sponsoring Groovy / Grails. This is apparently because of an increasing amount of community support, however, it may point to concerns in the future. • Mark Diggory : Grails has its own DispatchServlet (much like XMLUI did with Cocoon prior our modifications) I • To grok grails as quickly as possible its Ruby on Rails for Java: This means its more than a UI technology, we would be rewriting much of DSpace in the process. Example of Grails application in three pages, you generate the entire application stack, from Hibernate to UI in several commands. Again, this leverages WebMVC and the UI technology shown in the example is Spring WebMVC+JSP, but it supports other view technologies like Grails Server Pages. Seems to be REST/AJAX friendly. However, IMO, we are back to rewriting core DSpace to implement DSpace as a Grails App.									
JQuery UI	Javascript	Yes		Yes, e.g. there is a JQuery UI theme for Bootstrap		Yes			Yes, has a plugin repository	Client side JavaScript based user interfaces ("single page web applications"), often have problems with accessibility. It might be good to investigate how to handle this prior to selection.
	Personal opinions on JQuery UI: • *Mark Diggory : Does not address the Server side functionality that is needed to maintain use of this technology, I would exclude from list as a supporting library of functionality found in UI technology stacks in other technologies in this list.*									
Backbone.js (Javascript with RESTful JSON interface & Model-View-Presenter)	Javascript	Yes, large number of major sites listed on Wikipedia & their homepage		Yes, or at least you can use it in conjunction with Bootstrap.		Yes		Yes	Yes, has plugins and extensions	Designed for developing "single page web applications". It could prove difficult to use with DSpace because of the complexity of a repository system. Client side JavaScript based user interfaces ("single page web applications"), often have problems with accessibility. It might be good to investigate how to handle this prior to selection.
	Personal opinions on Backbone.js:									

	• *Art Lowel (Atmire) Backbone is too low level imo. You still have to write a lot of boilerplate code yourself. We should probably replace it with one of the more modern JS MV* frameworks as alternatives to ember js, like angular, knockout or react* • *Mark Diggory : Does not address the Server side functionality that is needed for persistence, a full suite of REST services would need to be present, questions would need to be answered regarding if workflows are managed as client side activities or server side activities.*									
Ember.js (Client-side Javascript web application using MVC)	Javascript	Yes, see their list of users on website		Yes, can use in conjunction with Bootstrap, e.g. https://indexiatech.github.io/ember-components/#/overview		Yes		Yes	Yes, there's an "addon" repository	Uses Grunt, Bower, NPM (all of which are also in use by Mirage 2 theme) Client side JavaScript based user interfaces ("single page web applications"), often have problems with accessibility. It might be good to investigate how to handle this prior to selection.
	Personal opinions on Ember.js: • *Art Lowel (Atmire) Ember is very "opinionated" which is great to guide you in to using best practices to solve common problems. But it can get tricky if you need to solve an uncommon problem and you have to fight the system to make it work. However I'd like to add a +1 for ember.* • *Mark Diggory : Does not address the Server side functionality that is needed for persistence, a full suite of REST services would need to be present, questions would need to be answered regarding if workflows are managed as client side activities or server side activities.*									
Vaadin	Java	Unsure, their Community page has a tagline which exhorts you to "join 150,000 devs"	Yes, they seem to prioritize working with plugins/addins, and have a large component repository	Yes	Yes	Yes	Yes	Yes	Yes, see their component repository	Seems to have a large community, and many freely-available learning resources. Seems a good fit for existing DSpace development practices (empahsis on working with Maven, plugins for working in the major IDEs), it has a free book.
	Personal opinions on Vaadin:									