Technology Overview

DSpace open source software is free to use, and community supported.

DSpace consists of both a frontend (User Interface) and a backend (REST API & other machine interfaces). A brief overview of the technologies used for each is provided below.

DSpace Frontend (UI) Technologies

The DSpace Frontend provides the User Interface which allows people to interact with DSpace. It requires a DSpace backend, and cannot be run standalone.

The DSpace Frontend is built on the Angular platform, written in the Typescript language. It uses Bootstrap & HTML5 for theming/styling & strives for WCAG 2.1 AA alignment. The frontend also uses Angular Universal for "server-side rendering", which allows it to function even when Javascript is unavailable in the browser. For more information on Angular Universal, see the the Angular University guide.

More information on installing the DSpace Frontend can be found in the Installing DSpace guide.

DSpace Backend (REST API) Technologies

The DSpace Backend provides the REST API, which is required by the DSpace Frontend. It also provides additional machine interfaces for interacting with data in DSpace, such as OAI-PMH, SWORDv2 Server, SWORDv1 Server and various command-line (CLI) tools. The DSpace backend can be run standalone, but it doesn't provide a user friendly web interface (which is why the DSpace frontend is recommended).

The DSpace Backend is built on Spring Boot, written in Java. The REST API portion of the backend is built on Spring Technologies, including Spring REST, Spring HATEOAS, and aligns with Spring Data REST. The REST API uses the Spring Data REST Hal Browser as a basic web interface for exploring the REST API. All REST API responses are returned in JSON format.

The DSpace Backend requires a relational database (usually PostgreSQL), used to store all the metadata and relationships between objects. All files uploaded into DSpace are stored on the filesystem (any operating system is supported). Apache Solr is also required, and is used to index all objects for searching/browsing.

More information on installing the DSpace Backend can be found in the Installing DSpace guide. More information on the REST API specifically can be found in our REST Contract.

How the Frontend and Backend interact

Here is a high level overview of what happens when a user interacts with DSpace when the user interface is running in *production* mode:

1. *Initial static page via server-side rendering (SSR):* When a user initially visits any page in the DSpace user interface (UI), this triggers server-side rendering (SSR) via Angular Universal. This means that the UI (Javascript) application is run *on the server* by Node.js. The result is that a *static* HTML page is generated, which will be sent back to the user.
    a. This process of rendering the static HTML page will result in Node.js making requests to REST API to gather all the data necessary to build the *static* HTML page.
2. *Static page is dynamically replaced by UI application:* The user briefly sees the generated static HTML page *while the UI (Javascript) application is downloading to their browser* . This allows the user to immediately see the DSpace User Interface even before it becomes interactive. As soon as the UI application finishes downloading, it dynamically replaces that static HTML page, making the User Interface interactive to the user. (The time between the UI page appearing and becoming interactive is usually unnoticeable to a user.) This entire process is handled by Angular Universal.
3. *Interactions with the UI application send requests to the REST API (client-side rendering):* As soon as the UI becomes interactive, it runs entirely in the user's browser (as any other Javascript application). This means that when the user interacts with the application (by clicking links/buttons or typing in fields, etc), this will send requests from the user's browser to the REST API (backend). This is called client-side rendering (CSR) as all HTML is generated within the user's browser.
    a. At this point, every action in the User Interface will generate one or more requests to the REST API to gather necessary data. These requests are all visible in the user's browser (in the "Network" tab of the browser's "Developer tools").

Keep in mind, SSR can be potentially taxing for very large pages with a lot of objects or data display. This is because Node.js has to make requests to the REST API to gather all the data for the page before rendering the static HTML. Because of this, we do also document some Performance Tuning suggestions for the User Interface (e.g. there is an option to cache these SSR generated static pages in order to generate them less frequently).

Some bots and clients may use server-side rendering at all times

For bots or clients *without the ability to run Javascript*, every page request will trigger SSR (server-side rendering). This is because the static HTML page can never be dynamically replaced by the User Interface application (in step 2 above). However, this behavior is necessary to support Search Engine Optimization. Some search engine bots cannot run Javascript & therefore cannot index sites which do not generate static HTML pages.
Running the user interface in development mode disables SSR and may impact SEO

Running the user interface (frontend) in *development* mode will only utilize client-side rendering (CSR) (as described in step 3 above). This means server-side rendering (SSR) will never occur, and all HTML will be generated in the user's browser. The result is that bots or clients *without the ability to run Javascript* will be unable to interact with the site (which can negatively impact Search Engine Optimization)