# DSpace 7 - Angular UI Development



## Quick Links

- The source code can be found on GitHub: https://github.com/DSpace/dspace-angular
- There's a project board on GitHub: https://github.com/orgs/DSpace/projects

## Developer Resources

- Overview of the new Technology
- DSpace-Angular Workshop : this workshop was held at OR2017, but all materials are available freely online
    - **Video Tutorial**: A shorter version of this workshop was held as a tutorial at the North American User Group meeting at Georgetown University on Aug 23, 2017. This shorter Angular tutorial (which doesn't include hands-on activities) was recorded and made available at https://youtu.be/c4AJ8HeZzcw
- Official Angular for TypeScript Style Guide
- RxJS - creating new sequences or using existing sequences
- DSpace Wiki: TypeScript-Guideline

## How to install locally

### Fast approach - Angular UI front end only

If you are primarily interested in the Angular UI development, without having to deal with the DSpace back-end, you can choose to only setup the UI components, and have them talk to the REST API of the public, demo rest api

These installation steps are outlined in the OR2018 Workshop or in README.md of the Angular UI project.

### Full Approach - DSpace 7 backend + Angular UI Frontend

Please be aware that this all is work in progress and will change often. As we did not release any version (neither an alpha or beta version) yet, there is no stable state to which we can refer to. Nevertheless let us give you some short hints on what you can do to install your own local version:

- Currently the DSpace 7 UI is read-only (submit/edit/admin tools will be coming). Therefore, if you want test content in place, you'll need to do one of the following:
    - Option #1: Start with an install of  DSpace 6, create some communities, collections and archive some items. In this situation, you'll perform an "ant update" to DSpace 7.
    - Option #2: Create some AIPs from a DSpace 5 or 6 instance to load into DSpace 7 for testing.
- Compile/Install/Deploy the current main (https://github.com/DSpace/DSpace/tree/main) using the normal DSpace install process ("mvn package" and "ant update" or "ant fresh_install").
    - This codebase is very similar to DSpace 6, but it includes no XMLUI or JSPUI, and has a new "dspace-spring-rest" project (which is the DSpace 7 REST API)
- Deploy the webapp dspace-server-webapp as /server (if you use another path, you may want to change the index.html file within the webapp)
- Start your servlet container and take a look into its logfiles
    - If you have problems deploying the webapp in cause for slf4j and log4j, it may be necessary to delete [dspace]/webapps/spring-rest /WEB-INF/lib/slf4j-log4j12-1.7.22.jar
- Test if dspace-spring-rest was loaded successfully by opening it in a browser. You can compare it to the demo site: https://api7.dspace.org/server/
- Install dspace-angular. You can try it as described in the OR2017 DSpace Angular Workshop or take a look into the Readme.md in the dspace-angular git repository.

# How to contribute

## Prerequisites

- You'll need to be in the dspace-angular github team to get the required access. Send your github username to Art Lowel (Atmire) to get an invite, or ask on Slack (https://dspace-org.slack.com/)
- Fork the repository on Github
- Refer to DSpace 7 UI Technology Stack for learning resources about the technologies used.
- We highly recommend joining DSpace's Slack as it's a great place to ask questions and get feedback
    - Slack invites are available at https://goo.gl/forms/s70dh26zY2cSqn2K3

## Workflow

- Have a look at the project board
- Take an issue that's in the ready section and has nobody assigned to it
- Assign yourself
- When you start working on it, move the issue to the "in progress" section
- Work on a separate branch for the issue on your fork
- When you're ready, fire a pull request
- in the comments of the pull request, write something akin to "this PR connect's to #{the ID of the issue}". That way the issue will be moved automatically to the review column.
- When at least two people have reviewed and approved your PR, it can be merged in master.
- You can also help out by reviewing the pull requests of other people
- Please keep an eye on your pull request afterwards, the reviewers may have questions or comments about it, or ask you to tackle things in a different way, before they can approve it
- Most discussions about the task or the pull request can happen through the github & project board comments.
- If it's more complex you can bring it up in one of these meetings.
- After your Pull Request has been merged, drag the issue to the done column on the project board. (this can also be automated by adding "this merge closes #{the id of the issue}" in the merge comment.
- If you've claimed an issue, but can't work on it for some reason, please remember to unassign yourself and put it back in the "ready" column so someone else can take over.

## Guidelines

### General

- Before firing a PR, always ensure your code works on the server (disable javascript in your browser and see if it still works) as well as the client, and that it works with the AoT build (`yarn run start`) as well as the webpack build (`yarn run watch`)
- Keep adaptability in mind. An institution installing DSpace will often want to modify a few things about the UI. The easier we can make that, the better. Therefore keep your components small (divide them in to sub-components), make sub-modules for coherent functionality, use SASS variables, etc.
- ~~We agreed to remove the concept of Communities from the UI. They should be called Collections as well~~. The decision to remove the concept of Communities from the UI was reversed in the meeting of April 13th

### Code Style

- Follow the official Angular style guide. It contains guidelines for naming files, directory structure, etc.
- TSLint can help you with that. It will run automatically whenever the code is rebuilt (even during a watch task) or you can run it manually with `npm run lint`

### Documentation

- Document your code in the TypeDoc format

### Testing

- Write tests for your code. We use jasmine for unit testing and protractor for end-to-end.
- Ensure your code doesn't break any existing tests.

### Creating a new component

- A new component should consist of at least 3 files, in the same folder
    - an HTML file, containing the template e.g. `sidebar.component.html`
    - an SCSS file, containing the style. Create this even if it's to remain empty. e.g. `sidebar.component.scss`
    - a TypeScript file, containing the component definition e.g. `sidebar.component.ts`
        - Its `templateUrl` attribute should contain a reference to the HTML file: e.g. `templateUrl: './sidebar.component.html'`
        - Its `styleUrls` attribute (this is an array) should contain a reference to the style file: e.g. `styleUrls: ['./sidebar.component.scss']`

- it's selector attribute should start with ds- This ensures the component name contains a dash, and indicates it's a dspace component. e.g. selector: 'ds-sidebar'
- Example:

```
@Component({
    selector: 'ds-sidebar',
    styleUrls: ['./sidebar.component.scss'],
    templateUrl: './sidebar.component.html'
})
export class SidebarComponent {
}
```

## (S)CSS

- General
  - Avoid writing (S)CSS if you can. Instead use Bootstrap 4 CSS classes and components where possible. That way it will be much easier to keep the style consistent across many different contributors.
  - If you want to use a bootstrap component, take a look at the ng-bootstrap docs first. Chances are they've already turned it into an angular component for you.
  - If you need to add margins, padding, change font-sizes or colors etc, make use of the bootstrap variables rather than entering numerical values. (e.g. write `margin-bottom: $spacer-y;` instead of `margin-bottom: 15px;` and `color: $brand-primary;` instead of `color: #0275d8;`)
  - If there's no other way but to add a numerical value or color code yourself, consider turning it in to a variable, so it can be reused by others. Also calculations that are used more than once can benefit from being stored as variables: e.g. `$ds-table-margin-bottom: ($font-size-base * 2) - ($gutter-width / 2);`
- Component Styles
  - Angular 2 makes use of component styles. By default, these can't *leak out* and affect the style of other components. Not even child components
  - The style for a component should go in a file called `name.component.scss`, in the same directory as the component. e.g. for `src/app/home/home.component.ts` the component style file is `src/app/home/home.component.scss`.

## i18n

- Don't hard code user-facing text in components. We're using ngx-translate for i18n. You use it by adding your key to resources/i18n/en.json, and then using that key in the component with the `translate` pipe. e.g.:
  - in en.json put: `"my-route.my-component.my-label-descriptor": "My User Facing Text";`
  - in your component put: `"<h1>{{ 'my-route.my-component.my-label-descriptor' | translate }}</h1>"`
- For more info see the ngx-translate docs

## State changes

- The state in this application is managed by ngrx.
- Using it properly requires some discipline from developers and reviewers.
- If you're new to redux or ngrx, take a look at the resources on the technology page
- Anything that's dynamic about the application is called a state change, and should be saved in the store.
- That means a boolean that saves if the navigation is open or closed, or a list of DSpace Objects from the backend, anything that can change as the application runs.
- Every state change should be defined as an action. in a file called `${feature-name}-actions.ts`. e.g. `host-window.actions.ts`
  An action definition consists of
  - a type, in the format `dspace/feature-name/ACTION-NAME`, added to an ActionTypes object.
    - e.g. `dspace/host-window/RESIZE`
  - a class to represent the action, that takes the components of the payload as the parameters of its constructor
    - e.g. HostWindowResizeAction
- The effect of each action on the state should be defined in the reducer, in a file called `${feature-name}-reducer.ts`. e.g. `host-window.reducer.ts`
  - keep in mind that a reducer shouldn't modify the previous state in any way. You can test for this using deep-freeze.
- If an action is be the source of another action, that relation is described in an *effect*.
  - e.g. submitting the login form dispatches a `LOGIN_REQUEST` action, with the user's credentials as its payload.
  - an AuthorizationEffect class listens for that action, and when it occurs it calls the rest api with those login credentials
  - if the REST API answers positively, the AuthorizationEffect dispatches a new `LOGIN_SUCESS` action, with the token that was returned.
  - if the REST API returns an error, the AuthorizationEffect dispatches a new `LOGIN_ERROR` action, with the error message that was returned.
- All reducers need to be added to a single reducer aggregator file per module before they can be used
- The same needs to happen for effect files

## DataServices and RemoteData objects

- There is a DataService for each type of DSpaceObject.
- These data services ensure resources get fetched from the rest api (the *mock* rest api at this point), and stored in the ngrx store
- The find methods in the DataServices return an Observable stream of RemoteData objects
- RemoteData objects contain the status of the request (loading/failed/succeeded) and the data (or error message)

This gist shows how you would use it in practice:

- In the component I specify `collection$`. The dollar is a convention to indicate an observable
- In the OnInit: `this.collection$ = this.cds.findAll();`
- And then in the template, handle the different cases: dedicated messages for `isLoading` and `hasFailed` and the actual data for `hasSucceeded`
    - of course in a more realistic scenario you might send the error to a notification service instead of handling it yourself
- In the template I use an `ng-container` and the `async` pipe to unwrap the observable: every time the observable changes the contents of this block will be re-rendered, and `collectionRD` will have the latest value
- The question marks check for null or undefined: the rest of the statement won't be executed unless that variable before the question mark has a value.