**Fall**

16

**nCino**

Angular 2 Summary With Best Practices

Amanda Fouts

A short summary of popular Angular 2 best practices along with short explanations

08

**Fall**

Angular 2 Best Practices

# Angular Architecture

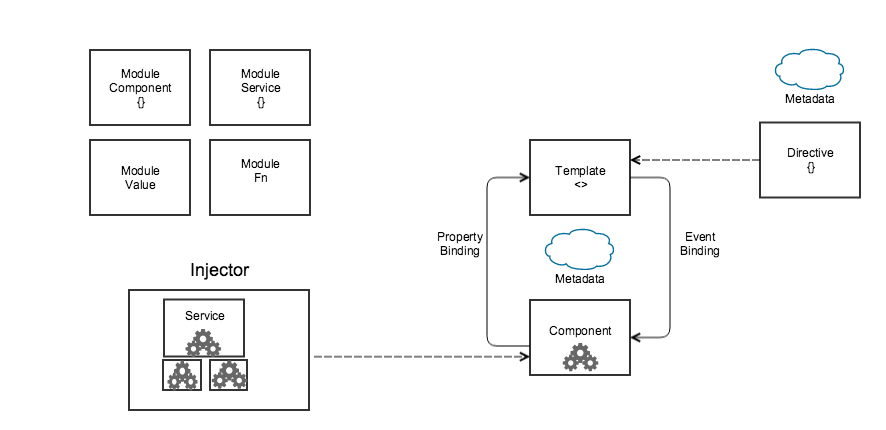


Figure - Angular Architecture (<https://angular.io/docs/ts/latest/guide/architecture.html>)

# Linter

Linter for Angular 2: <https://www.npmjs.com/package/codelyzer>

tslint and ng2lint linters can be used in combination with codelyzer.

# Angular 2 Guidelines

## Small Functions

Small functions using the single responsibility principle are encouraged. Programmers should design the small functions with maximum reuse in mind.

## App Structure and Angular Modules

All of the app code goes into a folder called app. Depicted below is an ideal directory structure. All features should be housed in their own module. Each component, service, or pipe should be in its own file. Third party vendor scripts should be stored in another folder beside app. Lazy loaded files should have their own folder.

( <https://scotch.io/tutorials/angularjs-best-practices-directory-structure>)

## LIFT

Apps should be designed to locate items quickly, identify items at a glance, with the flattest structure possible and using DRY principles.

L 🡪 Locate quickly

I 🡪 Identify at a glance

F 🡪 Flattest structure possible

T 🡪 DRY

## Prefer Reference and State Immutability

Immutable means constant and unchanging, once you instantiate the object, you can’t change its properties. To achieve State Immutability:

* Use const bindings when declaring references.
* Use Immutables (from Facebook) persistent data structures.
* Use Redux and ng2-redux for predictable state management.

State immutability reduces the burden of having to track mutations.

## Modules

app/

----- shared/

--------- sidebar/

------------ sidebarDirective.js

------------ sidebarView.html

--------- article/

------------ articleDirective.js

------------ articleView.html

----- components/ //mini app

-------- home/

----------- homeController.js

----------- homeService.js

----------- homeView.html

------- blog/

---------- blogController.js

---------- blogService.js

---------- blogView.html

----- app.module.js

----- app.routes.js

assets/

----- img/

----- css/

----- js/

----- libs/ //jquery, etc.

index.html

Basically, Angular modules are classes that use an @NgModule decorator. Modules allow you to disperse your application into logical blocks of functionality.

Use ES6 modules. A transpiler with larger browser support should also be utilized. It is preferred to group all dependency imports together at the top of the file and all exports at the bottom of the file.

## Components

Angular components are similar to controllers in other MVC frameworks and related to user interfaces within the application.

Stateless components are preferred. Keep state management in Redux (and out of Angular) as much as possible. Pass properties and event handlers down. Event handlers should feature an on prefix. Using a prefix for component selectors prevents name collisions in templates. Use templateURL when depending on external template files. Using markup over strings eases template management. @Input and @Output decorators are preferred over their @Component properties. When declaring component inputs and outputs, prefer the decorator version.

## Templates

Angular templates are similar to views in other MVC frameworks. They are altered forms of HTML that describe how the application renders the interface.

For templates, one-way data binding should be utilized as it keeps data changes predictable and in-sync. Avoid referencing local template variables and side-effects in template expressions. Use the NgClass directive for managing multiple class names at the same time. Us the NgStyle directive when setting several inline styles at the same time. Property bindings should use ngSwitch. Use an asterisk in front of ngSwitchWhen and ngSwitchDefault.

## Services

A service is a class with well-defined, logical functionality. Some examples of services might included managing data, logging, etc.

Try not to depend on services in stateless view components.

## Directives

Directives in Angular are markers attached to particular DOM elements which apply a specific behavior or transformation to the element.

Use attribute selectors with directives.

## Routing

Angular routers enable navigation from one view to the next as users perform application tasks.

Containers should be used as component routers.

## TypeScript

Type definitions in the form of interfactes to make implicit contracts between parts explicit and allow for tooling to catch integration bugs.

## Dependency Injection

Dependency injectors are utilized to inform a particular component about which services it requires to be functional.

Use @Injectable() on all service definitions. @Injectable maintains consistency and prevents bugs when a dependency is added later.

## GIVE AN EXAMPLE OF DEPENDENCY INJECTION USING ANGULAR 2

<https://angular.io/docs/ts/latest/guide/dependency-injection.html>

# References

1. https://github.com/rangle/angular2-guidelines

2. <https://angular.io/docs/ts/latest/guide/architecture.html>

3. <https://airbrake.io/blog/javascript/angularjs-best-practices>