(/)

AngularJS Best Practices: Directory Structure

Adnan Kukic ((https://scotch.io/author/kukicadnan)@kukicadnan (https://twitter.com/kukicadnan)) October 9, 2014 63 Tutorials (https://scotch.io/category/tutorials) angularJS (https://scotch.io/tag/angular-js)

We spend a lot of time writing code. In the early phases of a project, the directory structure doesn't matter too much and many people tend to ignore best practices. In the short term, this allows the developer to code rapidly, but in the long term will affect code maintainability. AngularJS is still relatively new and developers are still figuring out what works and doesn't. There are many great ways to structure an app and we'll borrow some principles from existing mature frameworks but also do some things that are specific to Angular.

(https://

(https://scotch.io
your-first-nod
Subscribe and get ou
Build Your First N
(https://scotch.io/
your-first-nod

Email Address

Subscrib

POPULAR ON

In this article, I will cover best practices regarding directory structures for both small and large AngularJS apps. This may be a hot button issue with some developers and while there is no “perfect” way to structure an app, I will be writing from experience and lessons learned from projects I’ve worked on.

Standard Structure

First of all, let’s go over what not to do. Many AngularJS tutorials show an app structure that resembles the code below:

Angular M
Material D
(https://sc
talk/angular-material-vs
design-lite)
Build an A
A Lightwe
to Angula
(https://scotch.io/tutori
app-with-vue-js-a-lightv
alternative-to-angularjs
5 Things Y
Know Abc
Positionin
(https://sc
talk/5-things-you-might
about-the-css-positioni
Automatic
Laravel Da
with Acce
Mutators
(https://scotch.io/tutori
format-laravel-databas
accessors-and-mutator
Build A W
with Back
Listening
(https://scotch.io/tutori
application-with-backbo
for-events)
Getting St
(plus a Fre
Starter Th
(https://scotch.io/tutori
started-with-jekyll-plus-

```
app/
----- controllers/
---------- mainController.js
---------- otherController.js
----- directives/
---------- mainDirective.js
---------- otherDirective.js
----- services/
---------- userService.js
---------- itemService.js
----- js/
---------- bootstrap.js
---------- jquery.js
----- app.js
views/
----- mainView.html
----- otherView.html
----- index.html
```

273 241

This is a very typical app structure that I see. On the surface, it seems to make and is very similar to a lot of MVC frameworks. We have a **separation of concerns**, **controllers have their own folder**, **views have their own folder**, **external libraries have their own folder**, etc.

OUTLINE

A Better Structure and Fc ▾

bootstrap-3-starter-the

Creating a
Discovery
(part 1)

(https://scotch.io/tutori

photo-discovery-app-w

**LEARN NODE ANI
WITH OUR E**

(http://bit.ly/

**A VAGRANT LA
THAT JUST W**

The main problem with this directory structure is not apparent when you are working with only a handful of views and controllers. In fact, it is preferable to follow this approach when writing a tutorial for example or for smaller application. This structure makes it very easy for the reader to visualize and conceptualize the concepts you are covering.

(http://bit.ly/1

DEAD SIM
OFF-CANVAS

(http://bit.ly/1

This approach falls apart, however, when you start adding additional functionality to the app. **Once you have more than 10 controllers, views and directives, you are going to have to do a lot of scrolling in your directory tree to find the required files.**

SCOTCH.IO ST

For example, say you are building a blog with Angular. You decide that you would like to add the author information to the bottom of each article. Well now, you have to find the blog directive, controller, potentially the service and finally the view before you can even look at the whole picture and start making edits.

(http://shop.s

Say a few months down the line, you are adding additional features to your blog and want to rename a particular feature, again it's a hunt throughout the directory structure to find the affected files, edit them, make sure they are all in sync, and then make the changes.

A Better Structure and Foundation

Let’s get to best practices and what you should be doing to build scalable and maintainable AngularJS apps that your coworkers will love you for. **An ideal AngularJS app structure should be modularized into very specific functions.** We also want to take advantage of the wonderful AngularJS directives to further compartmentalize our apps. Take a look at a sample directory structure below:

```
app/
----- shared/   // acts as reusable c
---------- sidebar/
--------------- sidebarDirective.js
--------------- sidebarView.html
---------- article/
--------------- articleDirective.js
--------------- articleView.html
----- components/   // each component
---------- home/
--------------- homeController.js
--------------- homeService.js
--------------- homeView.html
---------- blog/
--------------- blogController.js
--------------- blogService.js
--------------- blogView.html
----- app.module.js
----- app.routes.js
assets/
----- img/      // Images and icons f
----- css/      // All styles and sty
----- js/       // JavaScript files w
----- libs/     // Third-party librar
index.html
```

This directory structure is much harder to read and understand from the get go. A newcomer to Angular may be completely turned off by this complex approach, and that is why

you see tutorials and examples in Angular following the simpler directory structure found in examples earlier. Let's dive into the directory structure above and see what's going on here.

index.html

The index.html lives at the root of front-end structure. The `index.html` file will primarily handle loading in all the libraries and Angular elements.

Assets Folder

The assets folder is also pretty standard. It will contain all the assets needed for your app that are not related your AngularJS code. There are many great ways to organize this directory but

they are out of scope for this article. The example above is good enough for most apps.

App Folder

This is where the meat of your AngularJS app will live. We have two subfolders in here and a couple JavaScript files at the root of the folder. The `app.module.js` file will handle the setup of your app, load in AngularJS dependencies and so on. The `app.route.js` file will handle all the routes and the route configuration. After that we have two subfolders – **components** and **shared**. Let's dive into those next.

Components Folder

The `components` folder will contain the actual sections for your Angular app. These will be the static views ,directives and services for that specific section of the site (think an admin users section, gallery creation section, etc). Each page should have it's own subfolder with it's own controller, services, and HTML files.

Each component here will resemble a mini-MVC application by having a view, controller and potentially services file(s). If the component has multiple related views, it may be a good idea to further separate these files into 'views', 'controllers', 'services' subfolders.

This can be seen as the simpler folder structure shown earlier in this article, just broken down into sections. **So you could essentially think of this as multiple mini Angular applications inside of your giant Angular application.**

Shared Folder

The `shared` folder will contain the individual features that your app will have. These features will ideally be directives that you will want to reuse on multiple pages.

Features such as article posts, user comments, sliders, and others should be crafted as AngularJS Directives. Each component here should have it's own subfolder

that contains the directive JavaScript file and the template HTML file.

In some instances, a directive may have it's own services JavaScript file, and in the case that it does it should also go into this subfolder.

This allows us to have definitive components for our site so that a slider will be a slider across the site. You would probably want to build it so that you could pass in options to extend it. For example, you could have:

```
<!-- user a slider directive to loop
<slider id="article-slider" ng-repeat
</slider>
```

Now this slider is accessible from any part of our site so we're not reinventing the wheel. We also just have to change it in one place, the `shared` folder and it will update sitewide.

Best Practices (For Huuuuge Apps)

If you are developing a really large application in AngularJS, you will want to go even further and modularize your app. Here are some additional tips on how to accomplish this.

Modularize the Header and Footer

A good practice here would be to create a `Core` subfolder under components, and then a

subfolder for the Header and Footer and any additional components that will be shared across many pages.

Modularize the Routes

In the structure above we didn't do this, but another good practice for very large apps is to separate the routes into separate files. For example you might add a `blogRoutes.js` file in the `/views/blog/` subfolder and there include only the routes relevant to the blog such as `/blog/:slug`, `/blog/:slug/edit`, `blog/tags:/tags`, etc.

Don't Forget to Minify

If you do decide to opt in and build your AngularJS apps in a modularized fashion, be sure to concatenate and minify your code before going into production. There are many great extensions for both Grunt and Gulp that will help with this – so don't be afraid to split code up as much as you need.

You may not want to necessarily have just one giant `.js` file for your entire app, but concatenating your app into a few logical files like:

- `app.js` (for app initialization, config and routing)
- `services.js` (for all the services)

This will be greatly beneficial for reducing initial load times of your app.

If you need some more tips on minifying, check out our guide: Declaring AngularJS Modules For Minification (https://scotch.io/tutorials/javascript/declaring-angularjs-modules-for-minification)

Keep the Names Consistent

This is more of a general tip, but this will save you a headache in the future, when writing components and you need multiple files for the component, try to name them in a consistent pattern. For example,

`blogView.html`, `blogServices.js`, `blogController.js`.

Benefits of the Modularized Approach

The example above shows a modularized approach to building AngularJS. The benefits of this approach include:

Code Maintainability

Follow the approach above will logically compartmentalize your apps and you will easily be able to locate and edit code.

Scalable

Your code will be much easier to scale. Adding new directives and pages

will not add bloat to existing folders. Onboarding new developers should also be much easier once the structure is explained. Additionally, with this approach, you will be able to drop features in and out of your app with relative ease so testing new functionality or removing it should be a breeze.

Debugging

Debugging your code will be much easier with this modularized approach to app development. It will be easier to find the offending pieces of code and fix them.

Testing

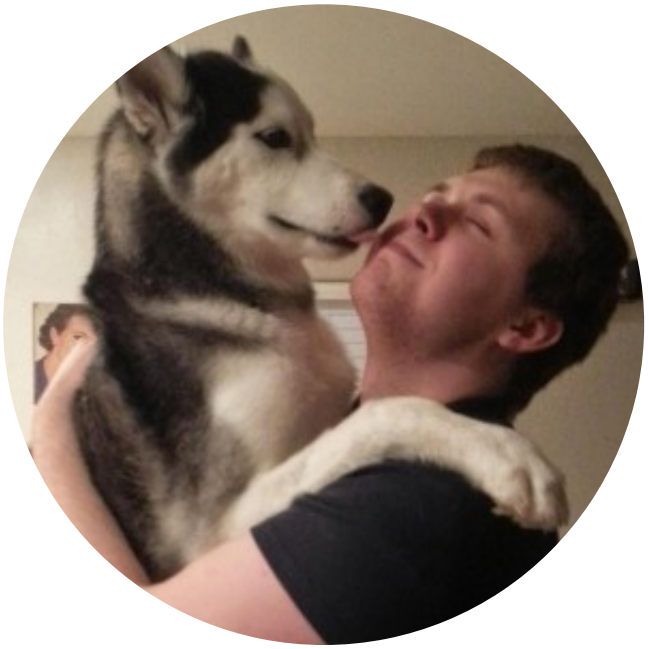
Writing test scripts and testing modernized apps is a whole lot easier then non-modularized ones.

In Conclusion

To conclude, this article covered some of the best practices in regards to structuring an AngularJS app. It is easy to ignore good practices in order to save time upfront. We all have a tendency to just want to start writing code. Sometimes this passion can hurt us in the long run when our awesome apps grow and become popular and then we're stuck rewriting or even worse maintaining badly thought out code. I hope this article had some helpful tips.

I plan on building a barebones AngularJS application structure that should follow the best practices outlined in this article that will help you get started building Angular apps quickly and efficiently. Keep a lookout for that in the coming weeks. Stay tuned for **Part 2** where we put these concepts into practice!

In the meantime, be sure to check out John Papa's AngularJS Style Guide (https://github.com/johnpapa/angularjs-styleguide) for additional tips on AngularJS best practices and while you're add it give Todd Motto's AngularJS Guide (https://github.com/toddmotto/angularjs-styleguide) a look too.



(https://scotch.io/author/kukicadnan)

ADNAN KUKIC (HTTPS://SCOTCH.IO/AUTHOR/KUKICADNAN) (@KUKICADNAN (HTTPS://TWITTER.COM/KUKICADNAN))

MEAN stack is the funnest stack.

View My Articles (https://scotch.io/author/kukicadnan)

READ NEXT

(https://scotch and-angular- to-do-app- application- organization-

(https://scotch routing- using-ui- router)

NODE AND ANGULAR TO-DO APP: APPLICATION ORGANIZATION AND STRUCTURE (HTTPS://SCOTCH.IO/TUTORIALS/NODE-AND-ANGULAR-TO-DO-APP-APPLICATION-ORGANIZATION-AND-STRUCTURE)

ANGULARJS ROUTING USING UI-ROUTER (HTTPS://SCOTCH.IO/TUTORIALS/ANGULAR-ROUTING-USING-UI-ROUTER)

(https://scotch page-apps- with- angularjs- routing-and-

(https://scotch of-angularjs- applications)

SINGLE PAGE APPS WITH ANGULARJS ROUTING AND TEMPLATING (HTTPS://SCOTCH.IO/TUTORIALS/SINGLE-PAGE-APPS-WITH-ANGULARJS-ROUTING-AND-

INTERNATIONALIZATION OF ANGULARJS APPLICATIONS (HTTPS://SCOTCH.IO/TUTORIALS/INTERNATIONALIZATION-OF-ANGULARJS-APPLICATIONS)

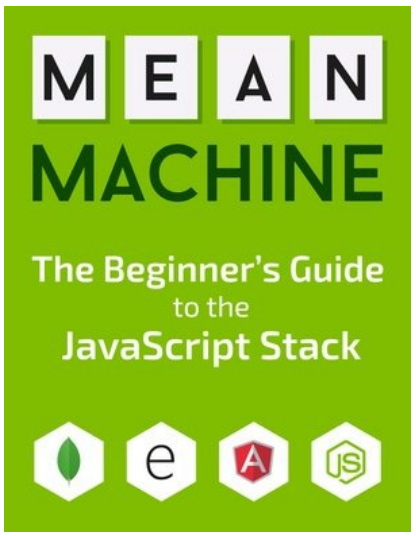
TEMPLATING)

(https://scotch
angularjs-
modules-for-
minification)

DECLARING ANGULARJS MODULES FOR MINIFICATION (HTTPS://SCOTCH.IO/TUTORIALS/ANGULARJS-MODULES-FOR-MINIFICATION)

(https://scotch
to-correctly-
use-
bootstrapjs-
and-

HOW TO CORRECTLY USE BOOTSTRAPJS AND ANGULARJS TOGETHER (HTTPS://SCOTCH.IO/TUTORIALS/HOW-TO-CORRECTLY-USE-BOOTSTRAPJS-AND-ANGULARJS-TOGETHER)



(https://leanpub.com/mean-machine)

SUBSCRIBE FOLLOW LIKE +1

(HTTPS://SCOTCH.IO/FEED) (HTTPS://TWITTER.COM/SCOTCHIO) (HTTPS://WWW.FACEBOOK.COM/SCOTCHIO) (HTTPS://PLUS.GOOGLE.COM/+SCOTCHIO/POSTS)

Get valuable tips, articles, and resources straight to your inbox. Every Tuesday.

Email Address

Subscribe

Comments Community 1 Login

Recommend 1 Sort by Best

Join the discussion…

Pedro C. • a year ago

There are any project seed out there with this kind of structure ? I already saw some with a similar structure, but I like to know which one do you recomend ?
thank you, great article...

12 • Reply • Share ›

Jeremy Morgan Pedro C. • 4 months ago

I have created one here: https://github.com/JeremyMorga...
It is similar and based on advice from this article.

1 • Reply • Share ›

Alexander Savvopulo Pedro C. • 6 months ago

angular-seed is doing something similar. check on github.

• Reply • Share ›

Adam Buczynski • a year ago

A problem I encountered when using this kind of modular approach, is when using Grunt to concatenate and minify files, the order in which files are loaded can break the modules because files are loaded in alphabetical order.

Suppose we have a "cart" component and the corresponding Angular module is called "myApp.cart". Let's say the module itself (along with configuration, routes, etc.) is defined in it's own dedicated file, e.g. cartModule.is:

```
angular.module('myApp.cart', [
//dependencies
])
```

Then, if the controller is defined in cartController.js and a service in cartService.js, both of these files need to state which angular module they belong to:

```
angular.module('myApp.cart')
```

Now, if you concatenate these three files, the controller file will be loaded first, and Angular will complain that the module myApp.cart does not exist/hasn't been defined previously.

Other then resorting to Require.js, have you found any (elegant) solution to prevent this problem and make sure that the file containing the module definition is always loaded first?

5 • Reply • Share ›

markgdyr Adam Buczynski
• a year ago

The way I currently do it is set up my build task to concat all module files first, then everything else. An example in gulp: https://gist.github.com/markgo... — Grunt should be pretty similar.

4 • Reply • Share ›

Adam Buczynski
markgdyr • a year ago

Thanks, yes, I've resorted to this solution as well, but the drawback is that I had to name all module files with a .module.js suffix, which causes unwanted clutter.

Hopefully Angular 2 will address this and make file loading in arbitrary order possible.

• Reply • Share ›

Pieter ➚ Adam Buczynski
• 7 months ago

I would really recommend you start using something like browserify in this case. It will also allow you to easily split out dependencies into multiple single files if necessary (though I mostly just use one or two large files). In that case the order of the files really don't matter, as dependencies are properly resolved. By just manually concatenating you're essentially doing exactly what browserify, require et. al are built for.

• Reply • Share ›

Adam Buczynski ➚
Pieter
• 7 months ago

I've actually changed my approach since I posted last comment. Now, every separate file just is it's own angular module. That way, the order in which they are loaded doesn't matter, since dependencies are properly indicated. Nor do I need a .module suffix, or other hacks. So now I might have

someModule.js,
someModule.ctrl.js,
someRelated.service.j
etc. each in their own
angular modules.

• Reply •
Share ›

Sebastian Kuligowski Adam Buczynski • a year ago

My approach is to use asynchronous loading with $script loader and angular-loader.js module. Each AngularJs module of your AnuglarJS application is registered with random order by angular-loader. So the order is not important at this moment. When all files are ready you can run manual bootstrap of your application (angular.bootstrap(id, modules). It works for me for development and for production as well. It works also with all js files concatenated together with random order.

Check https://github.com/angular/ang... to see what I'm talking about.

• Reply • Share ›

Adam Buczynski Sebastian Kuligowski • a year ago

Sorry if I misunderstood you, but the order in which modules are loaded is not the issue here.

The problem is that each module might be spread out over multiple files, and that you cannot load a file which uses a particular module, before that module has been defined. I don't see how asynchronous loading of files would solve this problem.

• Reply • Share ›

Sebastian Kuligowski
Adam Buczynski
• a year ago

I see your point. I usually have one module per file. In my projects all the modules are really small and are defined in the same file. So in each file I have:

```
angular.module('contro
[

'dateFormatter', ...
]).

directive('alertDialog',
function() {...}).

directive('openDialog,
function() { ...});
```

This apporach allows you to move files easly between projects/parts of the project. The order is also not important because the definition of the module is there.

I don't use AngularJS modules to group filters or similar directives. For example in my approach each filter has its own module:

```
angular.module('filter.v
[]).
.filter(function() {
return function(arr) {}
));
```

• Reply •
Share ›

Adam Buczynski
Sebastian Kuligowski
• a year ago

Yes, that's what I thought. I have quite large projects, and for clarity I put module configuration/routes and controllers in their own files. This does make the load order important.

• Reply •
Share ›

Sebastian Kuligowski
Adam Buczynski
• a year ago

I also have all things in separate files. The difference is that you create large modules containing several files. My module is always in a single file, I treat module as the kind of namespace. In the past I was using the same approach as you. Increasing granularity of modules was actually the best thing we've done to increase flexibility of our projects.

• Reply •
Share ›

Adam Buczynski
Sebastian Kuligowski
• a year ago

Well, regardless of module size, they all go in multiple files just

to maintain a division of controller and config/routes/run logic and to keep things consistent. Do you put all config/routes/run logic always in the same file as your controllers, regardless of size? Or do you put your controllers in a separate module (or namespace as you called it)?

Edit: I think I know what you mean. Suppose there is a cart module, with a cart controller and a cart service. What you would do, is define the config/routes/run logic in one module, the controller in another, and the service in yet another, like so?

angular.module('Cart', []).config(...);

angular.module('Cart.C []).controller('CartCtrl', ...);

angular.module('Cart.S []).provider('Cart', function() { ... });

• Reply • Share ›

Sebastian Kuligowski
Adam Buczynski • a year ago

Thats right. This is exactly that what I mean

mean.

Let's consider following template:

<div ng-controller="myFeature

Then I create file my-feature.js:

angular.module('featur

see more

• Reply • Share ›

t1t0 Sebastian Kuligowski • a year ago

Sorry if my question sounds a little stupid, but i'm not a pro as you guys, i'm a beginner on development and specialy on angular. You say that is better make a file for single module, filter, controller, service by components of the app. But how can i do to load this files every dinamically depending the route. for example, i have main page and now i'm gonna go to the route /mainapp/#/something this route load the view something.html but html has some controllers and

controllers and services that i nedd to load dinamically. until now i have to load from the main app but i know it's not the right way to do it. So, please can you explain to me how can i do that (in simple words please). PD: sorry for my english.

• Reply • Share ›

Sebastian Kuligowski
t1t0 • a year ago

I'm against lazy loading of js files (depending on a route). The navigation through my application should be as smooth as possible - it means no waiting for the content after click/touch/swipe event.

If you have small/medium application then load all javascripts at the begining using asynchronous loading - it speeds up starting of your application. It's the best solution and I'm using this approach for almost every project. Even more - load your all html templates and put them to the $templatesCache before showing the first view

first view.

(https://) (https://) (https://) (http://) (https://scotch.io)

Web design and development tutorials for the masses.