Structuring AngularJS Projects

- In **my opinion** the goal of a folder structure for any application is to make development both:-
  - (A) **Efficient** for an existing developer to get access to the file where they want to develop.
  - (B) **Obvious** for a new developer coming onto an existing project where they might find a file and *place new files*.
- If your project is small, say <500 lines of code. Then it's fine to just keep everything in one file since that does meet the criteria of being **Efficient** and **Obvious**.
- However as you start adding more and more code to one file, although the structure is still **Obvious** it becomes harder to find the piece of code you are looking for so it stops becoming **Efficient**. Thats when we start to break out code into different files and folders.
- The challange is how to break files appart into a structure that is still **Obvious**.
- There is not official guidance from the AngularJS team as to how to structure your code, and there are quite a few voices on the internet who talk about differing methods. I believe this is because whats **Obvious** to one person isn't **Obvious** to another.
- This doc will explain the two most popular methods, but what's most important is that you think about this problem yourself. The goal isn't to stick blindly to some method, the goal is to find a structure that works for you that is both **Effient** and **Obvious**.

By Convention

This method is the simplest and in my opinion the most **Obvious**. It involves breaking out the app.js into multiple folders, one for each **type** of component you can find in AngularJS.

```
controllers/
|- controller1.js
|- controller2.js
services/
|- service1.js
|- service2.js
filters/
|- filter1.js
|- filter2.js
directives/
|- directive1.js
|- directive2.js
app.js
1.
```

So in the structure above, we would place all our controllers in a folder called controllers, and the same goes for the services, directives and filters.

Each file would have it's own module.

If the main module is called "myapp" then we might name the module in controller1.js something like.

```
angular.module("myapp.controllers.controller1");
```

and then in app.js we would include all those modules as dependancies of our main application.

```
angular.module("myapp", [
  "myapp.controllers.controller1",
  "myapp.controllers.controller2",
  "myapp.services.service1",
  "myapp.services.service2",
  "myapp.filters.filter1",
  "myapp.filters.filter2",
  "myapp.directives.directive1",
  "myapp.directives.directive2",
]);
```

Pros

- The advantage of this approach is that it's super **Obvious** where each file goes,

when looking for a controller you will find it in controllers folder. It's also very clear where to place the file for a new controller.

Cons

- The arguments against this are that when the project gets larger, e.g. 20 controllers, then this approach becomes less **Efficient**.

By Feature

This method is the most **Effient** however in my opinion suffers from not being very **Obvious**.

It involves breaking out the app.js into multiple folders, one for each **Feature** of your application.

The idea is that a developer will tend to work in blocks of time only within a single feature, therefore all the files they need will be in the same folder they are already in.

```
common/
|- controller.js
|- service.js
|- directive.js
|- filter.js
shop/
|- controller.js
|- service.js
|- directive.js
blog/
|- controller.js
|- service.js
|- directive.js
|- filter.js
app.js
```

- So in the above example, the app has been broken down into two features and a common folder.
- The common folder is for components that are shared amongst the other features, such as a shared directive.
- All the components for a feature should be in the features folder.

Pros

The advantage of this approach is that if you are working on the shop feature, you know where all the shop files are, i.e. it's **Efficient**.

Cons

- Esp. when working with other developers at times which feaure folder a file should belong in isn't **Obvious**, e.g. should a blog like page that's part of the shop be in the blog feature folder or the shop feature folder?
- In my experience *unless strictly managed* the feature method devolves into all developers sticking code in the common folder.