Best Practice Recommendations for Angular App Structure

Proposal:

Change angular-seed, yeoman/generator-angular, (the Google-internal example go/nghellostyle), and other demo apps to model the following directory structure. Eventually, develop tooling to make development more efficient using assumptions based on these conventions.

Motivation:

Our toy and demo apps currently group files functionally (all views together, all css together, etc...) rather than structurally (all elements of a view together, all common elements together, etc...)
When used in a real larger scale app (e.g. Doubleclick For Advertisers), readability is improved by making the directory and app structure consistent with the architectural design of the app. We also want to be able to develop tools to make app creation and management easier, and this is simplified with a common structure.
This proposal outlines a more functional structure based on a repeating hierarchy and some use-case examples.

For more on motivation and the need for a recommendation of some kind, see this lengthy discussion https://github.com/yeoman/generator-angular/issues/109

Why a fractal hierarchy?

We wanted to be able to create a common set of rules that can be applied to a logical unit within an application. The application developer can then decide how many logical units are necessary, and apply the rules to all of them in the same way. This makes it easier for large applications to develop a structure that reflects the application itself, while applying the same rules at each level of the hierarchy.

This structure will make it possible for us to develop common tooling for Angular apps, and also support future plans for Angular where routing better supports greater modularity of views.

Why propose naming conventions?

We want to address a couple of issues that we see now in Angular code:

- It's fairly common now to have the same name used for different kinds of .js files -- a Service and a Filter both using the same name, for example. When editing files in an IDE it can be hard at a glance to tell which is which. Adding a naming convention for types of files helps to resolve this.
- There are currently multiple ways of organizing by extension or naming convention. Unit tests, for example, are named *_test.js, *.spec.js, or test/*.js, depending on who wrote the code. Having a single recommendation makes it easier to develop tools.

Got an edge case we haven't anticipated?

Tell us about it! We're interested in hearing what we might have missed.

Proposed Example Directory Structure

Recursive hierarchical structure

We propose a recursive structure, the base unit of which contains a module definition file (`app.js`), a controller definition file (`app-controller.js`), a unit test file (`app-controller_test.js`), an HTML view (`index.html` or `app.html`) and some styling (`app.css`) at the top level, along with directives, filters, services, protos, e2e tests, in their own subdirectories.

Components and Sub-sections

We group elements of an application either under a "components" directory (for common elements reused elsewhere in the app), or under meaningfully-named directories for recursively-nested sub-sections that represent structural "view" elements or routes within the app:

Components

- A components directory contains directives, services, filters, and related files.
- Common data (images, models, utility files, etc.) might also live under components (e.g. components/lib/), or it can be stored externally.
- Components can have subcomponent directories, including appended naming where possible.
- Components may contain module definitions, if appropriate. (See "Module definitions" below.)

Sub-sections

- These top-level (or nested) directories contains ***only*** templates (.html, .css), controllers, and module definitions.
- We stamp out sub-level child sub-sections using the same unit template (i.e., a section is made up of templates, controllers, and module definitions), going deeper and deeper as needed to reflect the inheritance of elements in the UI.
- For a very simple app, with just one controller, sub-section directories might not be needed, since everything is defined in the top-level files.
- Sub-sections may or may not have their own modules, depending on how complex the code is. (See "Module definitions" below.)

Module definitions

- In general, 'angular.module('foo')' should be called only once. Other modules and files can depend on it, but they should never modify it.
- Module definition can happen in the main module file, or in subdirectories for sections or components, depending on the application's needs.

Naming conventions

We lean heavily on the Google JavaScript Style Guide naming conventions and propose a few additions:

- Each filename should describe the file's purpose by including the component or view sub-section that it's in, and the type of object that it is as part of the name. For example, a datepicker directive would be in components/datepicker/datepicker-directive.js.
- Controllers, Directives, Services, and Filters, should include `controller, directive, service,` and `filter` in their name.
- File names should be lowercase, following the existing JS Style recommendations. HTML and css files should also be lowercase.
- Unit tests should be named ending in `_test.js`, hence "`foo-controller_test.js`" and "`bar-directive_test.js`"
- We prefer, but don't require, that partial templates be named *something*.html rather than *something*.ng. (This is because, in practice, most of the templates in an Angular app tend to be partials.)

Examples

A Very Simple App Example

Consider a very simple app with one directive and one service:

```
sampleapp/          the client app and its unit tests live under this directory
   app.css
   app.js
   app-controller.js
   app-controller_test.js
   components/ module def'n, services, directives, filters, and related
      foo/          files live here. "foo" describes what the module does.
         foo.js     The 'foo' module is defined here.
         foo-directive.js
         foo-directive_test.js
         foo-service.js
         foo-service_test.js
   index.html
e2e-tests/                        e2etests are outside the scope of this
sampleapp-backend/                proposal, but could be at the same level as
                                  backend/ and sampleapp/... we leave this
                                  up to the developer.
```

Or, in the case where the directive and the service are unrelated, we'd have:

```
sampleapp/
  app.css
  app.js
  app-controller.js
  app-controller_test.js
  components/
    bar/                          "bar" describes what the service does
       bar.js
```

```
              bar-service.js
              bar-service_test.js
           foo/                          "foo" describes what the directive does
              foo.js
              foo-directive.js
              foo-directive_test.js
         index.html
```

This structure is recommended, not prescriptive. However by using some other format in some cases (or, for example, by moving the directives and services upwards in the hierarchy to live under `sampleapp/`) instead of grouping them under `components/` we would need to develop separate tooling for large and small apps. We are aiming to define a single spec that can be implemented and automated against. Pragmatically, the recommendation as a best practice here means "we'll build tools that are primarily compatible with this proposed structure".

A More Complex App Example

For a more complex app -- for example, a ticketing application that has a user and an admin login, where some sections of the UI and routing are common to both users while other subsections are only available to one or the other, we want to isolate files belonging to each component, thus:

```
---
sampleapp/
       app.css
       app.js                      top-level configuration, route def'ns for the app
       app-controller.js
       app-controller_test.js
       components/
              adminlogin/
                     adminlogin.css        styles only used by this component
                     adminlogin.js              optional file for module definition
                     adminlogin-directive.js
                     adminlogin-directive_test.js
              private-export-filter/
                     private-export-filter.js
                     private-export-filter_test.js
              userlogin/
                     somefilter.js
                     somefilter_test.js
                     userlogin.js
                     userlogin.css
                     userlogin.html
                     userlogin-directive.js
                     userlogin-directive_test.js
                     userlogin-service.js
                     userlogin-service_test.js
       index.html
```

```
        subsection1/
                subsection1.js
                subsection1-controller.js
                subsection1-controller_test.js
                subsection1_test.js
                subsection1-1/
                        subsection1-1.css
                        subsection1-1.html
                        subsection1-1.js
                        subsection1-1-controller.js
                        subsection1-1-controller_test.js
                subsection1-2/
        subsection2/
                subsection2.css
                subsection2.html
                subsection2.js
                subsection2-controller.js
                subsection2-controller_test.js
        subsection3/
                subsection3-1/
                        etc...
```

As Applied to the Public Phonecat App

Applying this to our real demo example, here is the current structure for the angular-phonecat app (some detail omitted), organized by type

```
---
   app/
        css/
           app.css
           bootstrap.css
        img/
           phones/
               nexus-s.0.jpg
               nexus-s.1.jpg
               nexus-s.2.jpg
               nexus-s.3.jpg
           glyphicons-halflings-white.png
           glyphicons-halflings.png
        js/
           app.js
           controllers.js
           directives.js             this file is empty, artifact from angular-seed
           filters.js
           services.js
        partials/
           phone-detail.html
```

```
            phone-list.html
        phones/
            nexus-s.json
            phones.json
        index-async.html
        index.html
    test/
        e2e/
            runner.html
            scenarios.js
        unit/
            controllersSpec.js
            directivesSpec.js
            filtersSpec.js
            servicesSpec.js
---
```

And this is what the angular-phonecat app would look like with the new proposed structure.

```
---
app/
    app.css                             was app/css/app.css
    app.js                              was app/js/app.js
    bootstrap.css                       was app/css/bootstrap.css
    components/
        checkmark-filter/
            checkmark-filter.js         was app/js/filters.js
            checkmark-filter_test.js    was test/unit/filtersSpec.js
        phonecat/
            phonecat.js         new file to contain the module routing
            phonecat-service.js         was app/js/services.js
            phonecat-service_test.js    was test/unit/servicesSpec.js
    detail/
        phone-detail.html               was app/partials/phone-detail.html
        phone-detail-controller.js was app/js/controllers.js
        phone-detail-controller_test.js   was test/unit/controllersSpec.js
    index.html                          was app/index.html
    index-async.html                    was app/index-async.html
    list/
        phone-list.html                 was app/partials/phone-list.html
        phone-list-controller.js        was app/js/controllers.js
        phone-list-controller_test.js   was test/unit/controllersSpec.js


app-phonedata/                  outside the scope of the app, just data.
    img/                                might also be located inside components/
        phones/
            nexus-s.0.jpg
```

```
          nexus-s.1.jpg
          nexus-s.2.jpg
          nexus-s.3.jpg
      glyphicons-halflings-white.png
      glyphicons-halflings.png
    phones/
      nexus-s.json
      phones.json
e2e/
    runner.html
    scenarios.js
---
```

A Google-specific Demo Example

Note: this section is only relevant to developers working at Google. If there's a lot of external interest in this use case, we'll post a more general version in the future.

See the Google-internal link go/nghellostyle for an example with the new structure (and full details including BUILD file definitions). Here we have no inheritance between `mainpage` and `about`, so these are just top level subsections.

```
app/                                was "client/app". this is the project name
    about/
          about-controller.js       was "client/app/js/controllers/about.js"
          about-controller_test.js  was "test/unit/controllers/about_test.js"
          about.html                was "client/app/html/about.html"
          about.js                  was "client/app/js/services/module.js"
    app.css                         was "client/app/css/app.css"
    app.js                          was "js/app.js"
    app-controller.js       was "js/controllers/app.js"
    compiledindex.html      was "client/app/compiledindex.html"
    components/
       ransom-filter/
            ransomnote-filter.js            was "client/app/js/filters/ransomnote.js"
            ransomnote-filter_test.js       was "test/unit/filters/ransomnote.js"
        pane/
            pane.js                         new file to contain the module routing
            pane-directive.js       was "client/app/js/directives/pane.js"
            pane-directive_test.js          was "test/unit/directives/pane.js"
        request/
               request-service.js           was "client/app/js/services/request.js"
               request-service_test.js      new file with tests for the request service
        versions/
               versions-service.js          was "client/app/js/services/versions.js"
               versions-service_test.js     new file with tests for the versions service
    genjsdeps.sh
    index.html                      was "client/app/index.html"
    mainpage/
          mainpage.html                     was "client/app/html/home.html"
          mainpage.js               was "client/app/js/directives/module.js"
          mainpage-controller.js            was "client/app/js/controllers/home.js"
          mainpage-controller_test.js       was "test/unit/controllers/home_test.js"
    README
    servecompiled.sh                was "client/app/servecompiled.sh"
e2e/
     runner.html
     scenarios.js

proto/
```