App structure

What should be achieved

- **Modularity**: grouping code into related -as much independant as possible- bundles.
- **Predicatability** and **consistentcy**: finding elements easily, same struture followed by other applications.
- **Flexibility**: modules, components, views, layouts, apps, directive with or without a partial... They are many "things" which can compose an AngularJS project and the app directory structure needs to account for them.
- **Transferability** and **re-usability**: moving elements across modules or applications easily, related elements should be siblings (ex: directive + partial + test + styles).
- **Extensibility** and **scalibility**: structure can grow whithout causing code bloat, there a is defined a place for any new element composing an application.

Anatomy of an AngularJS app

An application is:

- A module itself
- Depending on external modules
- Broken down into several sub-modules

A module can contain serveral elements: **views** (controllers and partials), **components** (directives with or without partials, services and filters), **tests** and **sub-modules** (sharing the same namespace).

However, one should expect a module not to contain all of them depending on their role in an application:

- An application module can contain all of them, but not necessarily components.
- An application sub-module will primarily contain views.
- An external module won't contain views.

Views and layouts

- Views can be nested, introducing the need for layouts. The directory structure needs to be able to accomodate layouts, which don't always come with a controller.
- Views contain business logic and are the glue between injectable components and UI directives (which should be unit tested). Views can only be unit tested partially and end to end tests are able to bridge the gap between unit tested REST end points and unit tested AngularJS components.

Proposed structure

- One file per component (controllers, directives, services, filters, tests...) to limit lines of codes per file and to have cleaner changesets in Git. Files should reflect with their name what they contain (controller, directive, factory, service, provider, filter, ...).
- Folder structure should reflect module structure.
- One file per module definition (including config and run functions)
- If a component has unit tests, it should be a file alongside with a `.spec.js` suffix.
- An app should contain 1 or more modules which contain each 1 or more views. If a sub-module doesn't contain any view, questions should be ask:
  - Should it be an external dependency?
  - Should it be part of another already existing module?
- If a module is a component and contains several elements and therefore a large number of files, they can be organised in folders `directives`, `services` and `filters`. If services and directives need to be added to an app module, the same folder structure can also be used.

Example with an application an user profile page and an admin section for managing users.

```
src/
  index.html
  less/
  app/
    myApp/                              -- myApp module
      myApp.js
      services/
         user-factory.js
      views/
        profile/
          profile-controller.js
          profile.less
          profile.html
      admin/                            -- myApp.admin module
        myApp.admin.js
        admin-layout.html
        admin-layout.less
        users/                          -- myApp.admin.users module
          myApp.admin.users.js
          views/
            list/
              list-controller.js
              list.html
              list.less
            create-edit/
            view/
    myDep/                             -- myDep module
      myDep.js
      component1/                      -- myDep.component1 module
        myDep.component1.js
        component1-directive.js
        component1-directive.spec.js
        component1.html
        component1.less
      component2/                      -- myDep.component2 module
        myDep.component2.js
        directives/
          component2-directive.js
          component2-directive.spec.js
          component2-child-directive.js
          component2.less
          component2.html
        services/
          component2-factory.js
          component2-factory.spec.js
        filters/
          do-this-filter.js
```