Organizing Your App

John Papa
http://johnpapa.net
Twitter: @john_papa

pluralsight
hardcore developer training

Start Small, Start Simple

Leave Room to Grow

Organizing Your App

John Papa
http://johnpapa.net
Twitter: @john_papa

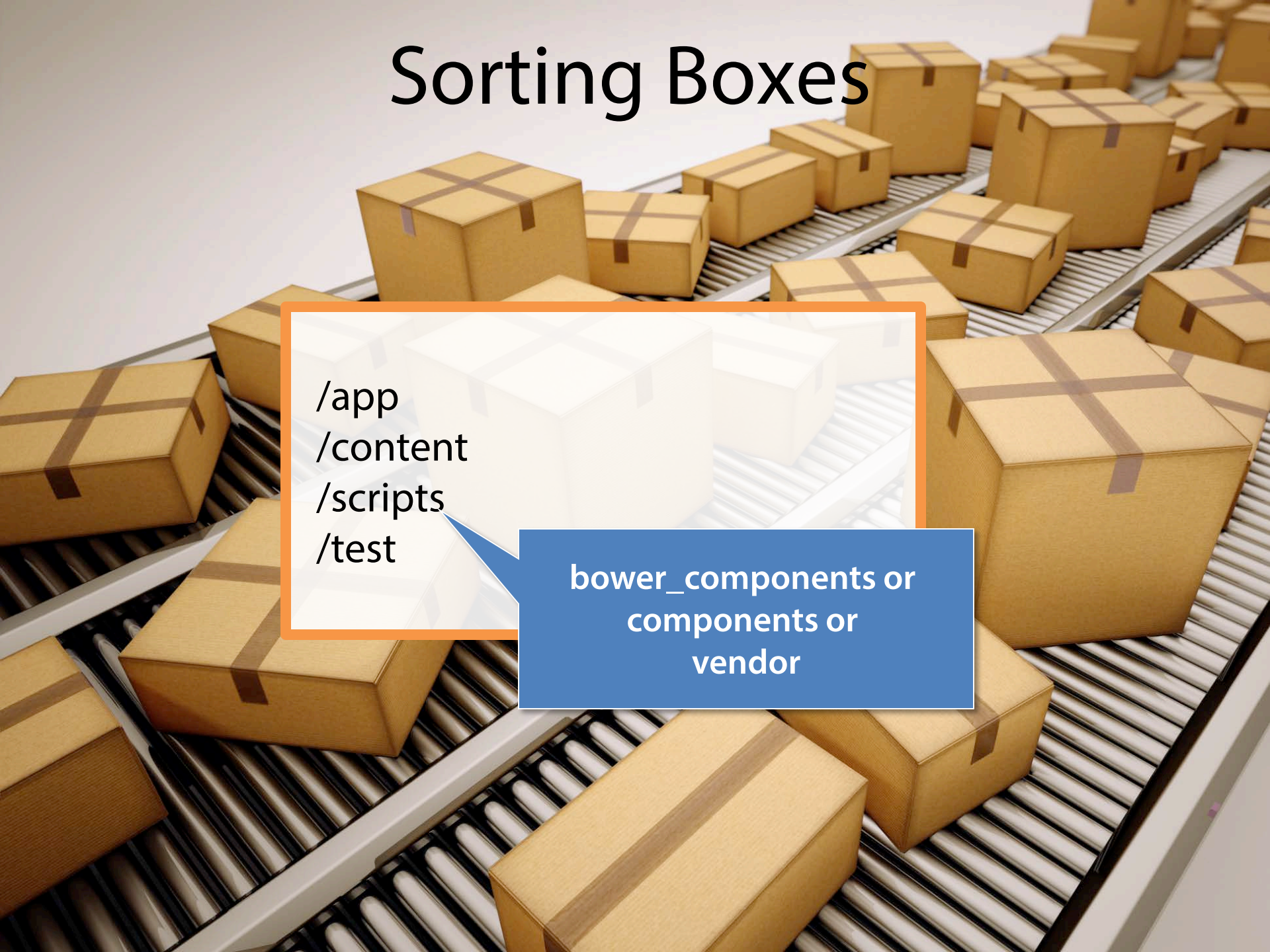
pluralsight
hardcore developer training

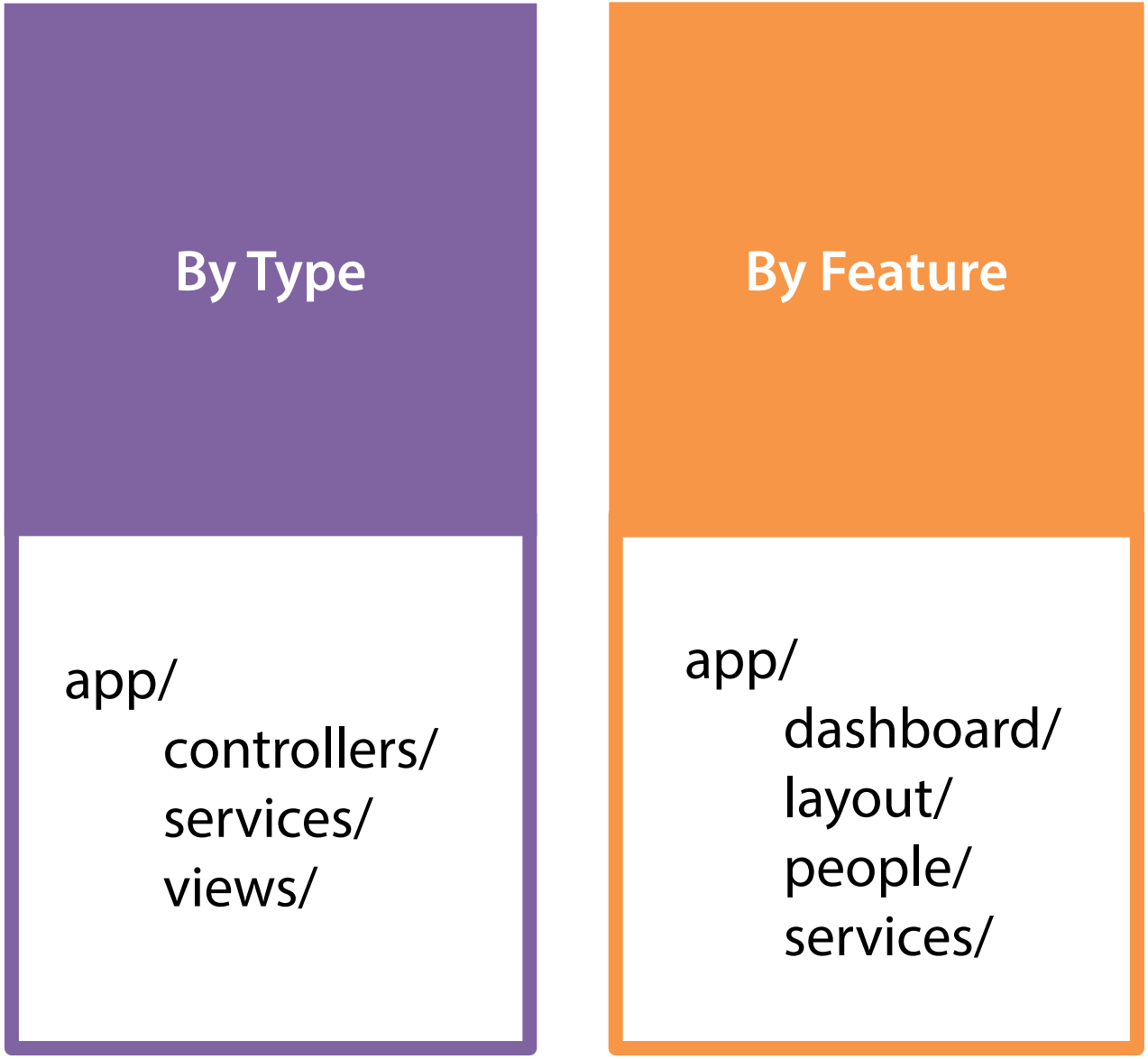
Benefits of a Solid Structure for Your AngularJS App

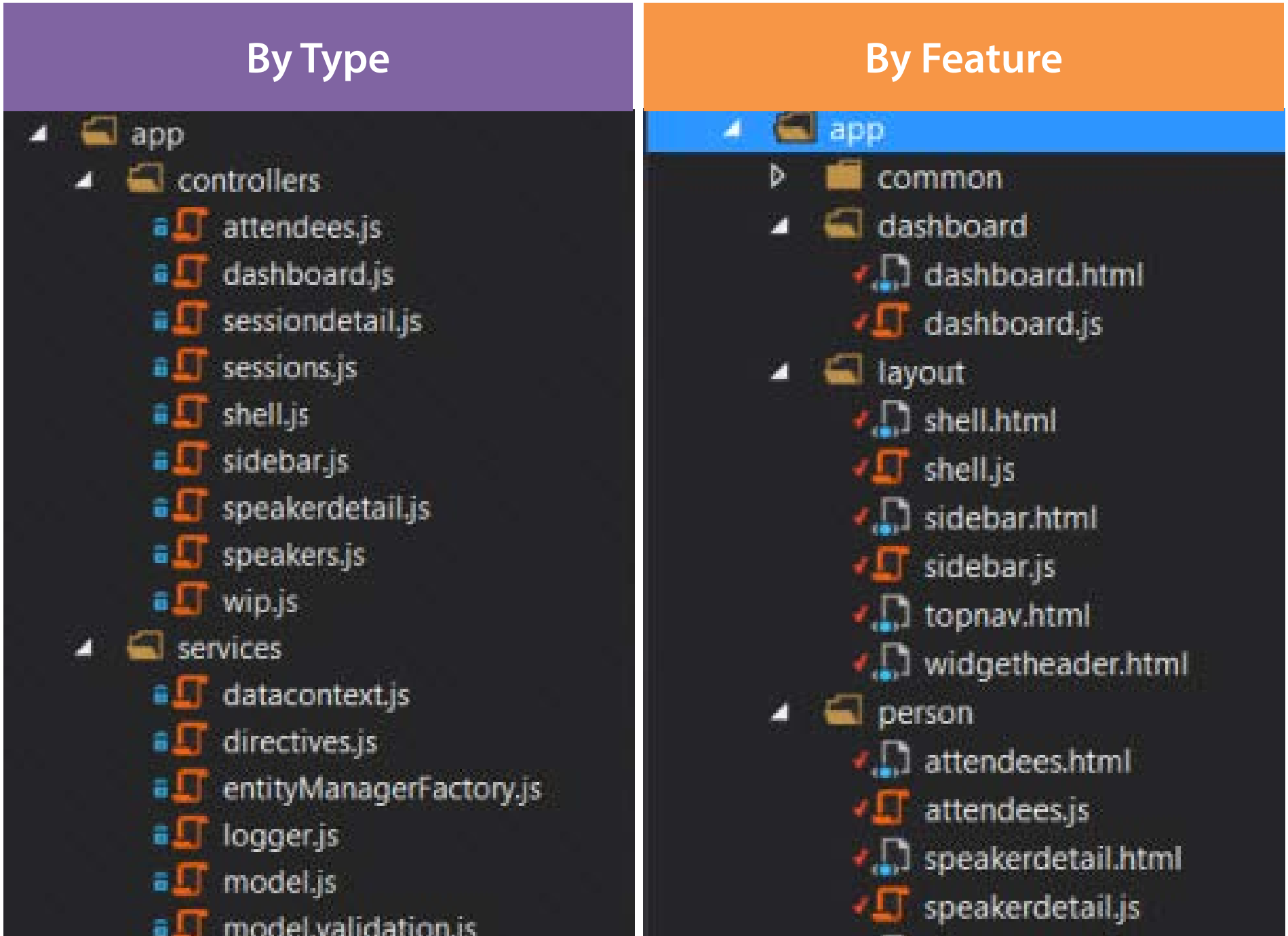
- Faster and easier to develop
- Aligns with SoC
- Easier to find and locate code
- Transitions to modularity when its time to extend your app

Choose A Path

Sorting Boxes




By Type
app/
controllers/
services/
views/
By Feature
app/
dashboard/
layout/
people/
services/


By Type
app
controllers
attendees.js
dashboard.js
sessiondetail.js
sessions.js
shell.js
sidebar.js
speakerdetail.js
speakers.js
wip.js
services
datacontext.js
directives.js
entityManagerFactory.js
logger.js
model.js
By Feature
app
common
dashboard
dashboard.html
dashboard.js
layout
shell.html
shell.js
sidebar.html
sidebar.js
topnav.html
widgetheader.html
person
attendees.html
attendees.js
speakerdetail.html
speakerdetail.js

“If someone says you are doing it wrong, ignore them”

Dan Wahlin

Options for Structuring

Best Advice: Be Consistent!

By Type?

- **Folders for views, controllers, factories, directives**

By Feature?

- **Folders for Speakers, Tracks, Sessions?**

Somewhere in the middle?

- **Folders for feature, then by type?**

The LIFT Principle

The LIFT Principle

L

Locating our code is easy

I

Identify code at a glance

F

Flat structure as long as we can

T

Try to stay DRY

Above the Fold: Controller

```
(function () {
    'use strict';

    angular
        .module('app.avengers')
        .controller('Avengers', ['common', 'dataservice', Avengers]);

    function Avengers(common, dataservice) {
        var vm = this;
        vm.avengers = [];
        vm.title = 'Avengers';

        activate();
        ////////////////
```

Immediately Identifiable

Above the Fold: Factory

```
(function () {
    'use strict';

    angular
        .module('app.core')
        .factory('dataservice', ['$http', 'common', dataservice]);

    function dataservice($http, common) {
        var logger = common.logger;

        var service = {
            getAvengersCast: getAvengersCast,
            getAvengerCount: getAvengerCount,
            getAvengers: getAvengers,
            ready: ready
        };

        return service;
        ////////////////
```

What the factory exposes

Naming Conventions

Naming Conventions

Like app structure, there are multiple “right” ways

Keys

- Be consistent
- Make it immediately identifiable (refer to LIFT)
- Agree on a convention with your team and company

Naming Modules

```
// Main app Module
// app.js
angular.module('app', [
    'avengers'
]);


// Avengers Module
// avengers.module.js
angular.module('avengers', ['ngRoute']);
```

Good starting place

May be helpful for multi module apps

Naming Controllers

```
// avengersController.js
angular
    .module('app.avengers')
    .controller('AvengersController', ['common', 'dataservice', AvengersController]);

function AvengersController(com    dataservice) {
```

"Controller" suffix

```
<section data-ng-controller="AvengersController as vm">
```

Naming Controllers

```
// avengers.js
angular
    .module('app.avengers')
    .controller('Avengers', ['common', 'dataservice', Avengers]);

function Avengers(common, dataservice) {
```

No "Controller" suffix

```
    <section data-ng-controller="Avengers as vm">
```

Naming Factories

```
// logger.js
angular
    .module('app')
    .factory('logger', ['$log', factory ] );

function factory( $log ) {
    var logger = {
        error   : error,
        info    : info,
        success : success,
        warning : warning
    };

    return logger;
    ////////////////////////
```

Same as file name

Naming Directives

```
// ccWidgetHeader.js
angular
    .module('app.widgets')
    .directive('ccWidgetHeader', ['$window', ccWidgetHeader]);

function ccWidgetHeader () {
    //Usage:
    //<div data-cc-widget-header title="vm.map.title"></div>
```

Same as file name

Choose a consistent prefix

Code Naming Conventions

Name controllers using PascalCase

i.e. Avengers

Choose descriptive names for components

Avoid generic names like utility

Use camelCasing for non Controller components such as factories

i.e. localStorage, modalDialog, modelValidator

File Naming Conventions

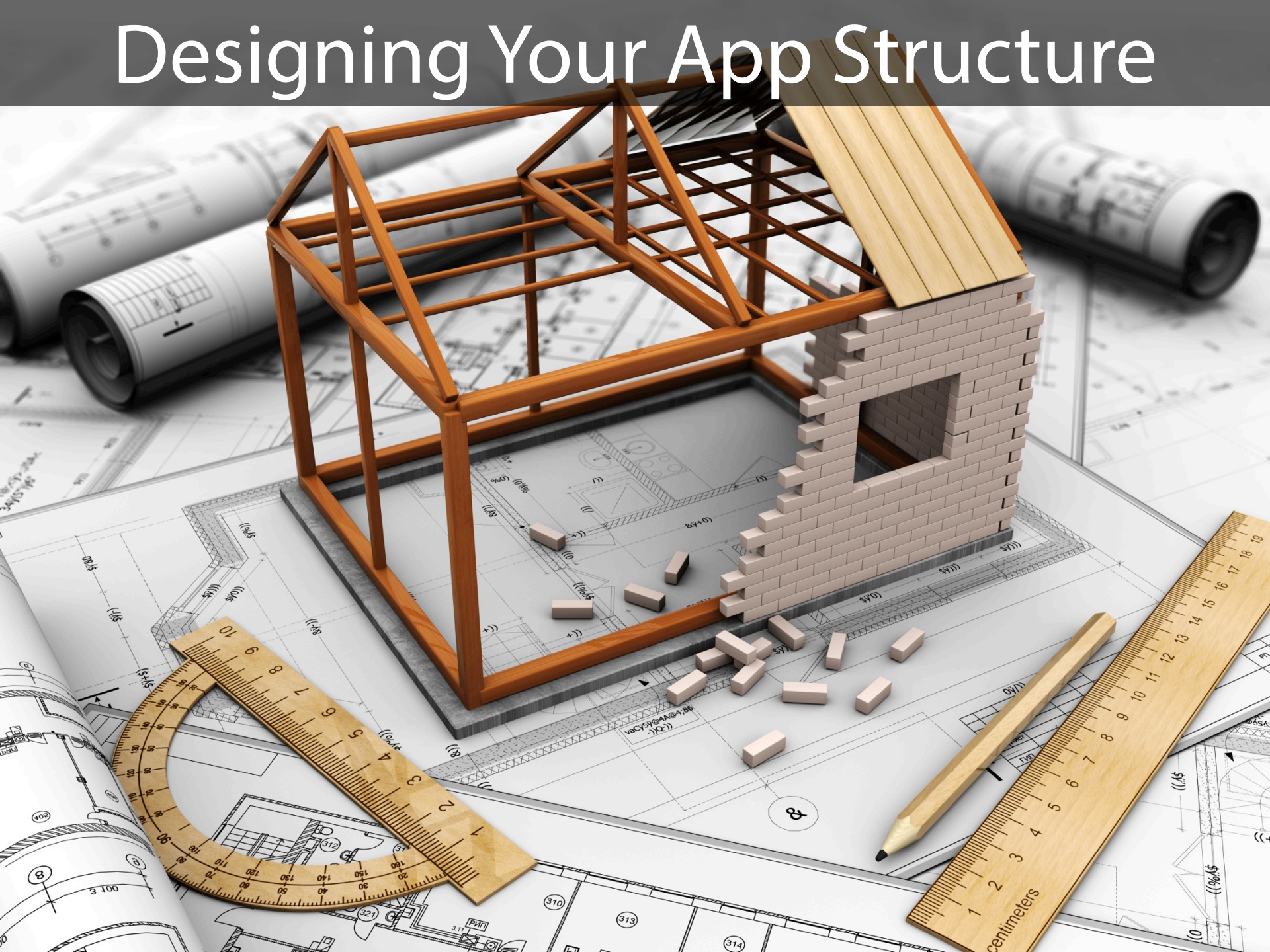
Use dots, dashes or camelCasing for multi-word file names

i.e. localStorage.js, speakerDetail.js speaker-detail.js

Choose descriptive names

Be consistent

Designing Your App Structure

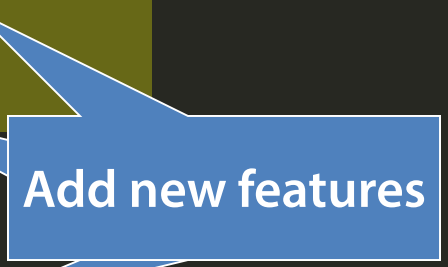


Common Starting Structure

```
app/
    app.js                  // main app module
    config.js               // module configuration
    dataservice.js          // data service
    sessions.html           // view
    sessions.js             // controller
```

Locating Becomes Difficult

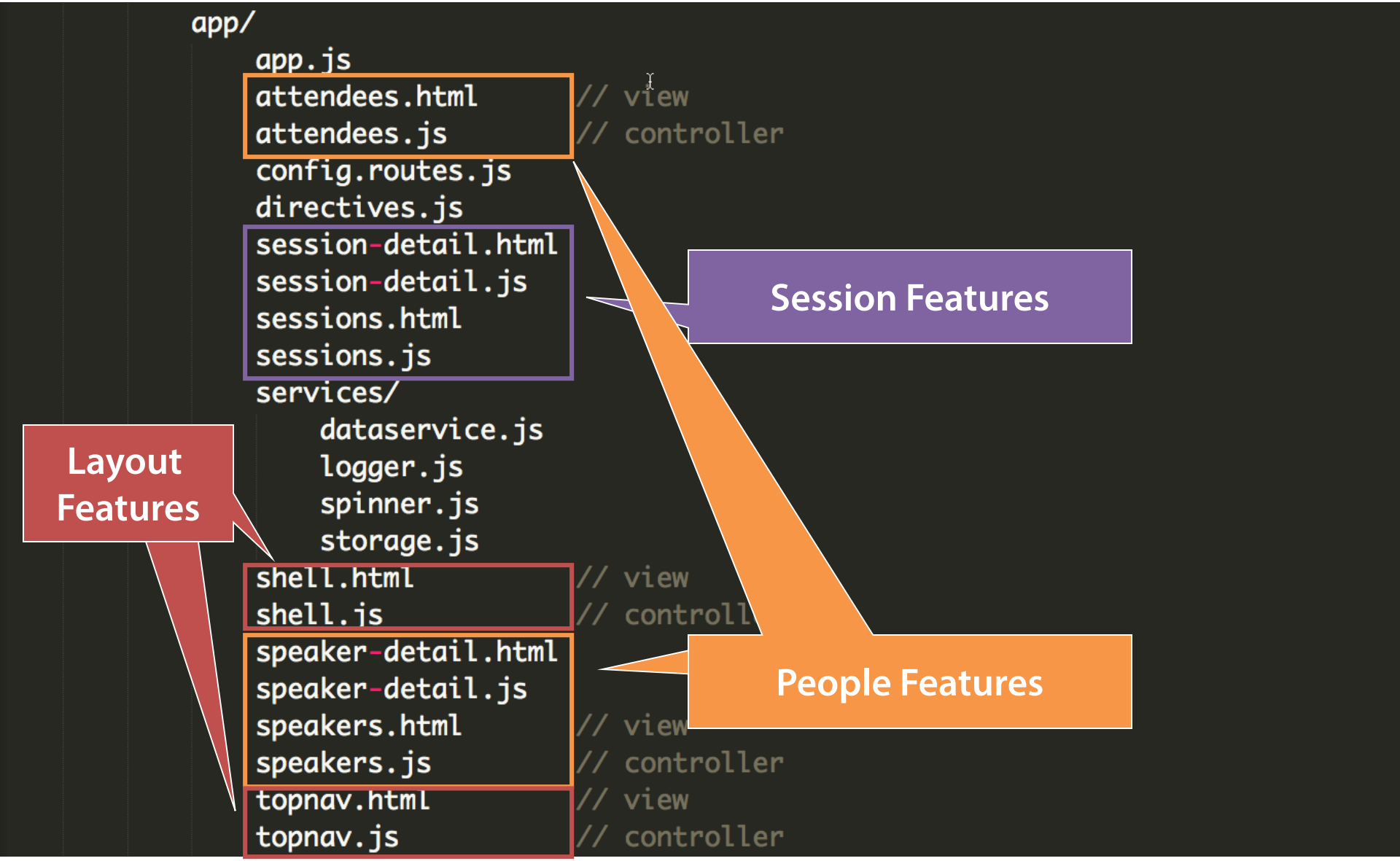
```
app/
    app.js                  // main app module
    config.js               // module configuration
    dataservice.js          // data service
    directives.js           // directives
    logger.js               // logger service
    session-detail.html     // view
    session-detail.js       // controller
    sessions.html           // view
    sessions.js             // controller
    spinner.js              // spinner service
    storage.js              // storage service
```



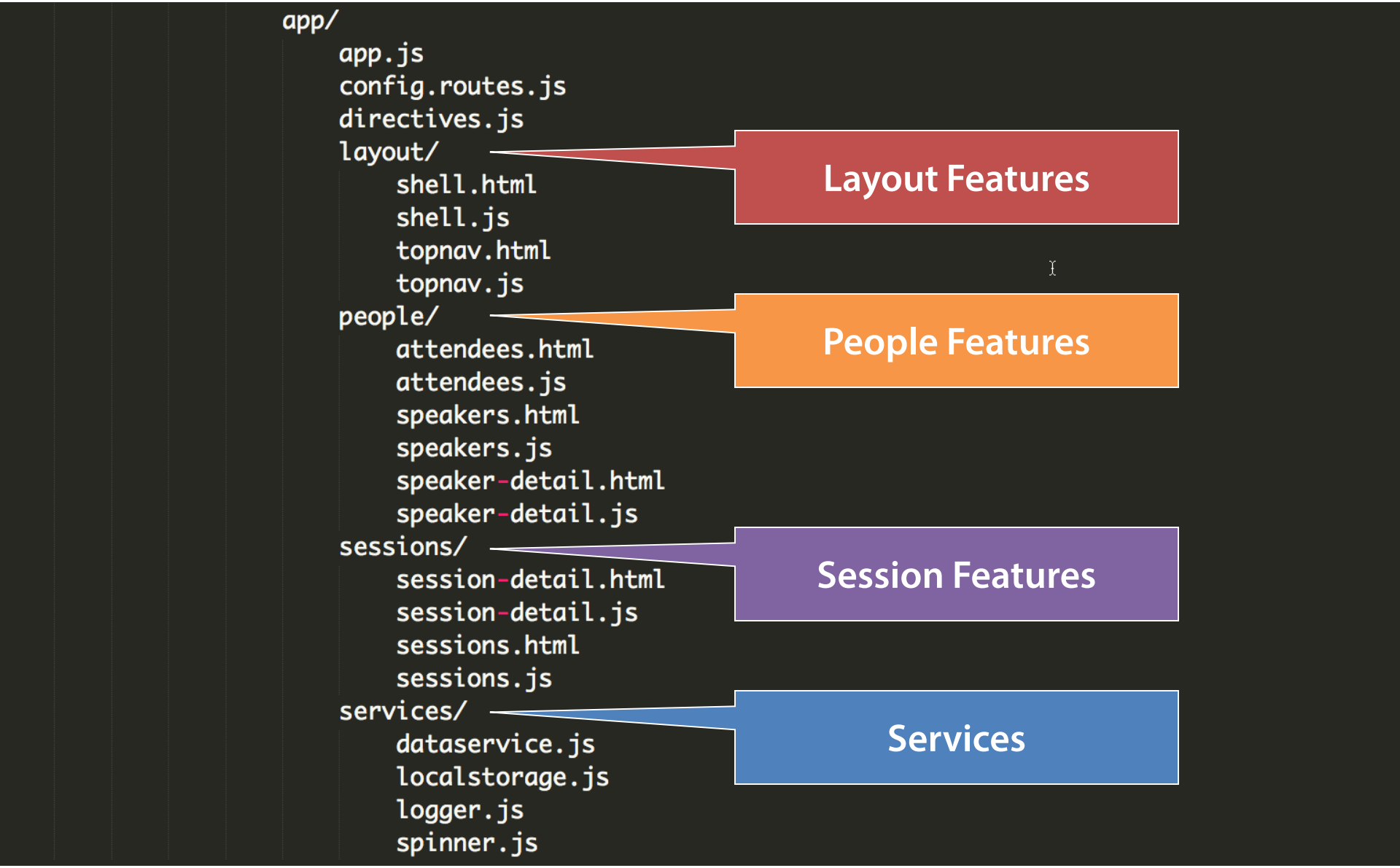
Add a Folder for Services

```
app/
    app.js
    config.js
    directives.js
    session-detail.html
    session-detail.js
    sessions.html
    sessions.js
    services/              // shared services
        dataservice.js
        logger.js
        spinner.js
        storage.js
```

Getting a Little Uncomfortable



Structure Begins to Take Shape



Summary

Follow the LIFT principle

Start with the smallest that makes sense

When in doubt, organize by feature

Be consistent with your team

AngularJS File Templates

WebStorm 7 and 8

http://jpapa.me/ngstormtmpl

Visual Studio 2013

http://sidewaffle.com