AngularJS 1.x Component-based Design

by Jeremy Zerr

Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr

What is a Component anyways?

- ☐ Interface + Template + Logic = Component
- ☐ The Interface includes inputs and outputs to allow it to be reused and be self-documenting.
- ☐ The template is a partial piece of HTML/CSS with some templating language. Allows the component to render itself.
- ☐ There is some Logic included, in the form of a Javascript object or class. It's own dependencies are taken care of.
- ☐ It's all bundled together in as simple of a feature to allow an application to be composed of lots of these components.

Here is a Component in Angular 2

```
import {Component} from 'angular2/angular2'

@Component({
  selector: 'my-component',
  template: '<div>Hello my name is {{name}}. <button (click)="sayMyName()">Say my nar
})
export class MyComponent {
  constructor() {
    this.name = 'Max'
  }
  sayMyName() {
    console.log('My name is', this.name)
  }
}
```

- Taken from learnangular2.com

Here is a Component in React

```jsx
export class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {count: props.initialCount};
  }
  tick() {
    this.setState({count: this.state.count + 1});
  }
  render() {
    return (
      <div onClick={this.tick.bind(this)}>
        Clicks: {this.state.count}
      </div>
    );
  }
}
Counter.propTypes = { initialCount: React.PropTypes.number };
Counter.defaultProps = { initialCount: 0 };
```

- Taken from Facebook React docs

Why do I want a Component-based architecture in Angular 1.x?

- ☐ One step closer to an upgrade path to Angular 2 (ngUpgrade)
- ☐ We should always strive for designing simple, reusable components.
- ☐ Design using the smart/dumb component philosophy (container/presentation-only)
- ☐ Encourage using one-way data binding
- ☐ We already have directives as our starting place. It's all about using directives in a better way.

How do we achieve a Component-based architecture in Angular 1.x?

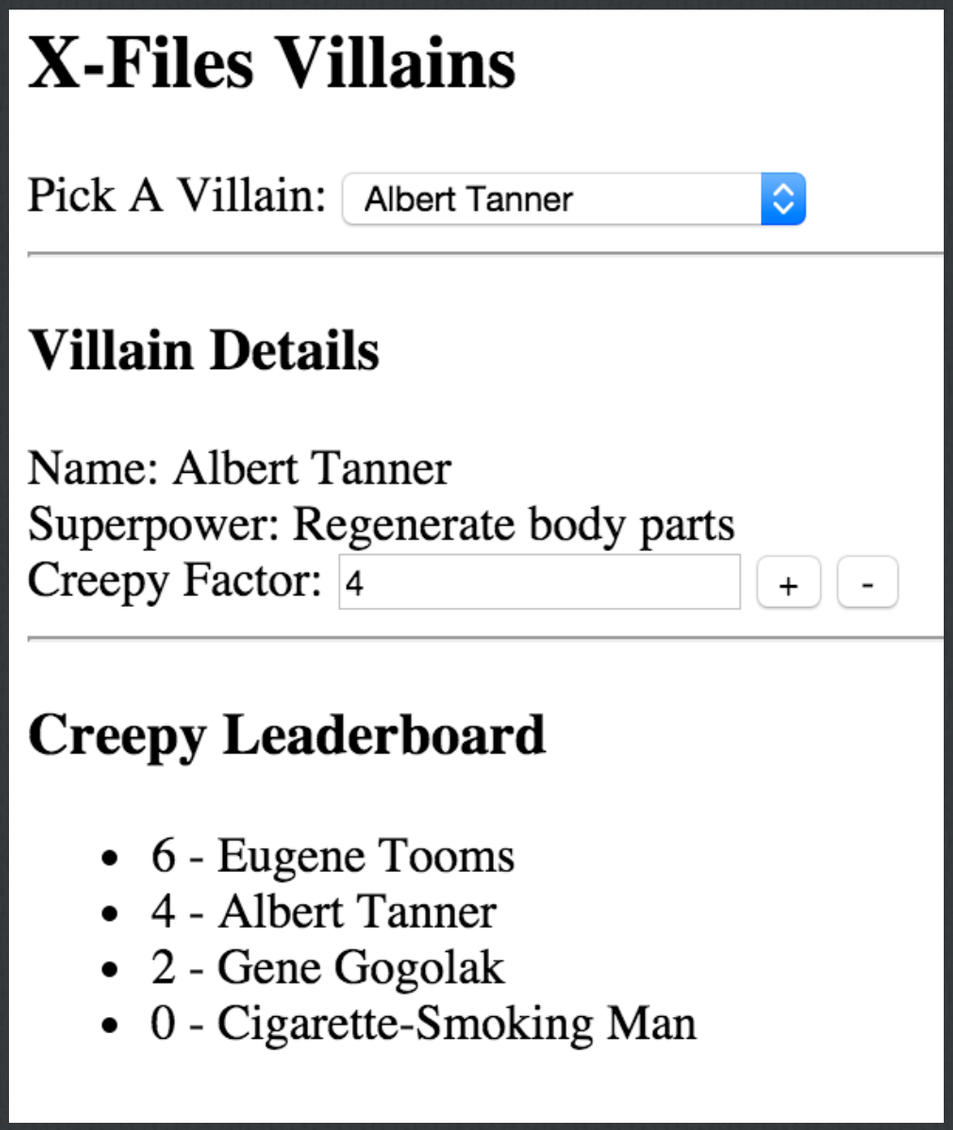
- [ ] No standalone controllers or templates, only directives that contain both. In Angular 1.4 directives are the closest thing we have to Components. Angular 1.5 adds a component that we can use.
- [ ] Use directive bindToController and controllerAs syntax so the directive controller is like a class.
- [ ] Our directive interface should use a convention to denote inputs and outputs, and not modify any of the inputs so we have only one way data binding between components.
- [ ] Create dumb components that use inputs, send outputs, and don't have dependencies
- [ ] No $scope inheritance, all isolate scope.
- [ ] This post on Teropa is a great read.

AngularJS Examples

X-Files Villains app in 4 parts

X-Files Villains App - Demo

- [ ] X-Files is Baaaaaacck!!!!
- [ ] Vote on how creepy each of the different X-Files villains are.



X-Files Villains App - Part 1

- [ ] Simple, one controller, one template, no directives
- [ ] Template renders and causes modifications to the data that is all stored in $scope in the controller.
- [ ] Plunker Link

X-Files Villains App - Part 2

- [ ] Refactor to directives
- [ ] One controller, three directives, one for each section of the app
- [ ] Data is all stored on $scope in controller, and passed via directive scope into the directives, where it is modified via 2-way binding.
- [ ] Controller <-> Directive 1
- [ ] Plunker Link

X-Files Villains App - Part 3

- ☐ Refactor to component-like directives
- ☐ One container directive, three dumb directives called in the container directives template
- ☐ Any data changes in the dumb directives are done via callback functions that modify data in the container directive controller.
- ☐ Container Directive -> Dumb Directive 1 -> Container Directive
- ☐ Plunker Link

Leaderboard Directive Diff

```
(function(){
  var module = angular.module('villains');
  module.directive('villainLeaderboard', function() {
    return {
      scope: {
        villains: '='
      },
      templateUrl: 'villains/villainLeaderboard/villainLeaderboard.html'
    };
  });
}());
```

```
(function(){
  var module = angular.module('villains');
  module.directive('villainLeaderboard', function() {
    return {
      scope: {}, // isolate
      bindToController: { // bind right to this in controller
        villains: '='
      },
      templateUrl: 'villains/villainLeaderboard/villainLeaderboard.html',
      controller: function() {},
      controllerAs: 'vm'
    };
  });
}());
```

List Directive Diff

```
(function(){
  var module = angular.module('villains');
  module.directive('villainList', function() {
    return {
      scope: {
        villains: '=',
        selectedVillain: '='
      },
      templateUrl: 'villains/villainList/villainList.html'
    };
  });
}());
```

```
(function(){
  var module = angular.module('villains');
  module.directive('villainList', function() {
    return {
      scope: {}, // isolate
      bindToController: { // bind right to this in controller
        villains: '=',
        onSelect: '&', // function takes single argument of villainId
        isSelected: '&' // function takes single argument of villain
      },
      templateUrl: 'villains/villainList/villainList.html',
      controller: function() {},
      controllerAs: 'vm'
    };
  });
}());
```

Detail Directive Diff

```
(function(){
  var module = angular.module('villains');
  module.directive('villainDetail', function() {
    return {
      scope: {
        villain: '='
      },
      templateUrl: 'villains/villainDetail/villainDetail.html',
      controller: function($scope) {
        // these functions were moved in from the app controller
        // to this directive controller
        $scope.moreCreepy = function(villain) {
          villain.creepy++;
        };
        $scope.lessCreepy = function(villain) {
          villain.creepy--;
        };
      }
    };
  });
}());
```

```
(function(){
  var module = angular.module('villains');
  module.directive('villainDetail', function() {
    return {
      scope: {}, // isolate
      bindToController: { // bind right to this in controller
        villain: '=',
        onMoreCreepy: '&', // takes single argument of villainId
        onLessCreepy: '&',  // take single argument of villainId
        onChangeCreepy: '&' // takes two arguments, villainId and creepy
      },
      templateUrl: 'villains/villainDetail/villainDetail.html',
      controller: function($scope) {},
      controllerAs: 'vm'
    };
  });
}());
```

Enforcing One Way Data Binding

- [ ] We want to ensure one-way data flow.
- [ ] A convention does not guarantee anything. Must create something formal.
- [ ] Let's create a buffer between our directive and the outside world by creating a copy when object changes.

```
bindToController: { // bind right to this in controller
  externalVillain: '=villain', // let's buffer this param
  onMoreCreepy: '&', // takes single argument of villainId
  onLessCreepy: '&',  // take single argument of villainId
  onChangeCreepy: '&' // takes two arguments, villainId and creepy
},
```

```
// we copy off the villain to prevent this directive
// from modifying the villain object passed in
$scope.$watch('vm.externalVillain', function(newVal, oldVal) {
  vm.villain = angular.copy(newVal);
}, true);
```

Container Directive

- [ ] Has data concerns. All data changes, fetching, all initiated from the Container Directive controller.

AngularJS 1.5 Adds Components - Part 4

- [ ] Compare the syntax changes from 1.4 (left) to 1.5 (right) to refactor to a component building on the changes in Part 3

```
(function(){
  var module = angular.module('villains');
  module.directive('villainDetail', function() {
    return {
      scope: {}, // isolate
      bindToController: { // bind right to this in controller
        externalVillain: '=villain', // let's buffer this param
        onMoreCreepy: '&', // takes single argument of villainId
        onLessCreepy: '&',  // take single argument of villainId
        onChangeCreepy: '&' // takes two arguments, villainId and creepy
      },
      templateUrl: 'villains/villainDetail/villainDetail.html',
      controller: function($scope) {
        var vm = this;
        // we copy off the villain to prevent this directive
        // from modifying the villain object passed in
        vm.villain = angular.copy(vm.externalVillain);
        $scope.$watch('vm.externalVillain', function(newVal, oldVal) {
          vm.villain = angular.copy(newVal);
        }, true);
      },
      controllerAs: 'vm'
    };
  });
}());
```

```
(function(){
  var module = angular.module('villains');
  module.component('villainDetail', {
    bindings: {
      externalVillain: '<villain', // let's make this one way data binding
      onMoreCreepy: '&', // takes single argument of villainId
      onLessCreepy: '&',  // take single argument of villainId
      onChangeCreepy: '&' // takes two arguments, villainId and creepy
    },
    templateUrl: 'villains/villainDetail/villainDetail.html',
    controller: function($scope) {
      var vm = this;
      // You still have to do this to have one way databinding on objects
      vm.villain = angular.copy(vm.externalVillain);
      $scope.$watch('vm.externalVillain', function(newVal, oldVal) {
        vm.villain = angular.copy(newVal);
      });
    },
    controllerAs: 'vm' // if we skip, will default to $ctrl
  });
}());
```

Plunker Link - Part 4

AngularJS 1.5 Adds One Way Databinding

- Sadly, it only really works for primitives (Number, String, Boolean, etc)

```
bindings: {
  //externalVillain: '<villain', // let's make this one way data binding
  name: '<', // test one way databinding on a primitive, the villain name
  villain: '<', // test one way databinding on an object
  onMoreCreepy: '&', // takes single argument of villainId
  onLessCreepy: '&',  // take single argument of villainId
  onChangeCreepy: '&' // takes two arguments, villainId and creepy
},
```

- Objects set up as one-way binding still have changes affect the outside.
- Still need to copy object to make one-way data binding with an object possible

```
Name: <input type="text" ng-model="vm.villain.name"/>
One Way Name: <input type="text" ng-model="vm.name"/>
```

Name: Eugene Tooms
One Way Name: Eugene Tooms

More about Components in Angular 1.5

- ☐ Only use restrict ‘E’ for Element
- ☐ Can use controller: ‘myController as $ctrl’ shortcut syntax for controller and controllerAs
- ☐ Inputs as ‘<‘ or ‘@‘, Outputs as ‘&’.
- ☐ No link or compile function possible
- ☐ AngularJS Component Doc

Thanks!

Jeremy Zerr

Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr