Practical Development of Web Applications with JavaScript and AngularJS

Unit 4. Reusable Components. Forms. REST Services.

Reusable Components

Reusable Components

- They use **isolate** scope
- Do not depend on or modify the parent scope
- Expose public API either as HTML element's attributes or as a *controller*.
- AngularJS 2.0 will do it better (Shadow DOM and Web Components)

Demo Price Range Directive

How to create Reusable Components

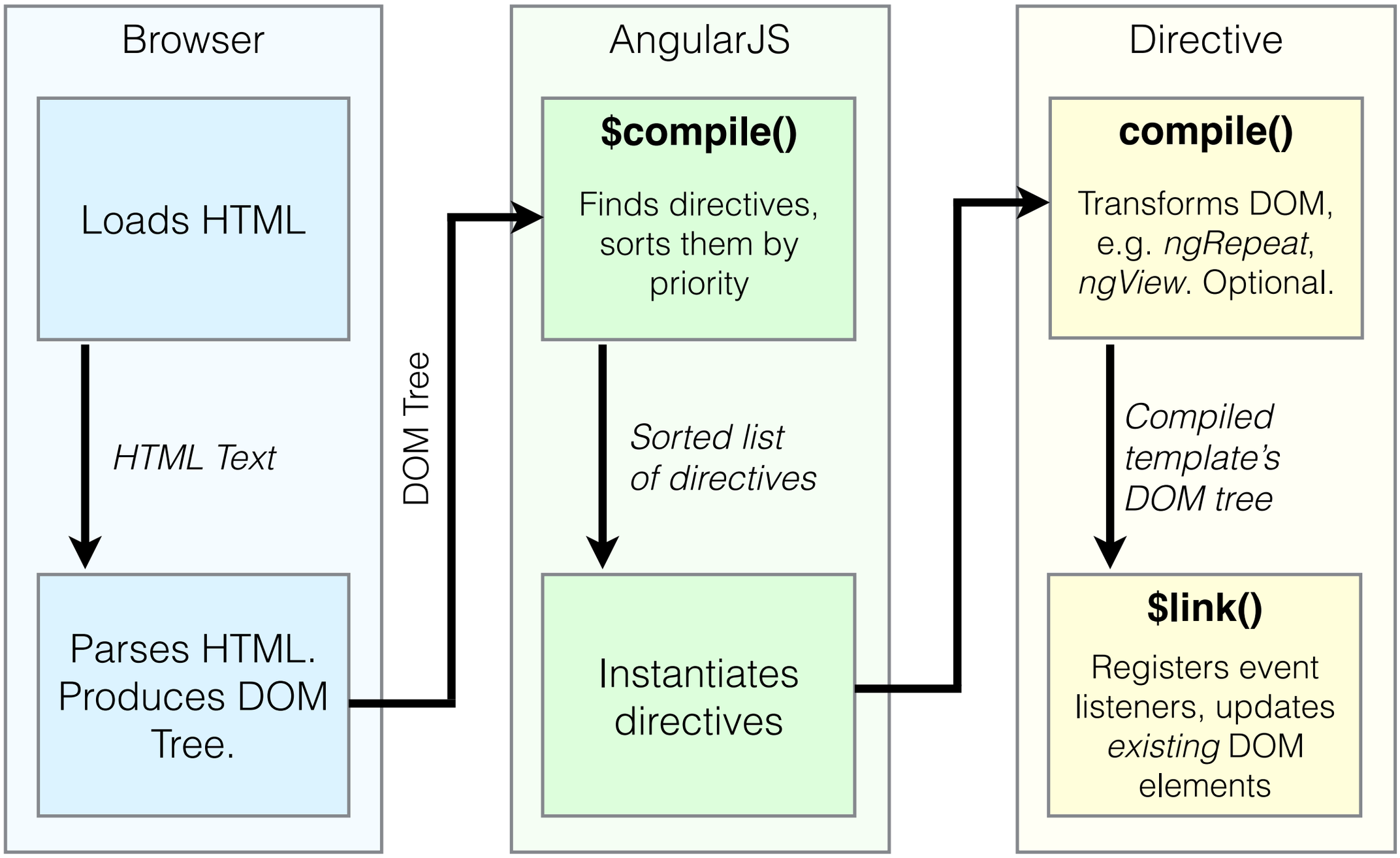
List of members

```
myModule.directive('directiveName', function factory (injectables) {
    return {
        priority: 0,
        template: '<div></div>', // or // function(tElement, tAttrs) { ... },
        // or
        // templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
        replace: false,
        transclude: false,
        restrict: 'A',
        scope: false,
        controller: function ($scope, $element, $attrs, $transclude, otherInjectables) { ... },
        controllerAs: 'stringAlias',
        require: 'siblingDirectiveName', // or // ['^parentDirName', '?optionalDirName', '?^optionalParent'],
        compile: function compile (tElement, tAttrs, transclude) {
            return {
                pre: function preLink (scope, iElement, iAttrs, controller) { ... },
                post: function postLink (scope, iElement, iAttrs, controller) { ... }
            };
            // or
            // return function postLink( ... ) { ... }
        }
        // or
        // link: {
        //  pre: function preLink(scope, iElement, iAttrs, controller) { ... },
        //  post: function postLink(scope, iElement, iAttrs, controller) { ... }
        // }
        // or
        // link: function postLink( ... ) { ... }
    };
});
```

compile(), link(), controller

```
myModule.directive('directiveName', function factory (injectables) {
    return {
        priority: 0,
        template: '<div></div>', // or // function(tElement, tAttrs) { ... },
        // or
        // templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
        replace: false,
        transclude: false,
        restrict: 'A',
        scope: false,
        controller: function ($scope, $element, $attrs, $transclude, otherInjectables) { ... },
        controllerAs: 'stringAlias',
        require: 'siblingDirectiveName', // or // ['^parentDirName', '?optionalDirName', '?^optionalParent'],
        compile: function compile (tElement, tAttrs, transclude) {
            return {
                pre: function preLink (scope, iElement, iAttrs, controller) { ... },
                post: function postLink (scope, iElement, iAttrs, controller) { ... }
            };
            // or
            // return function postLink( ... ) { ... }
        }
        // or
        // link: {
        //  pre: function preLink(scope, iElement, iAttrs, controller) { ... },
        //  post: function postLink(scope, iElement, iAttrs, controller) { ... }
        // }
        // or
        // link: function postLink( ... ) { ... }
    };
});
```

Directives Life Cycle



compile()

- Transform DOM in compile() *before* it's rendered
- Compiled template's DOM tree is cached
- Each instance of the directive receives a clone of the compiled DOM tree
- A place for the methods shared by all instances of the directive

link()

- `link()` is invoked as soon as the directive is linked to the DOM

- Use `link()` to bind to DOM events, update the DOM and watch scope's expressions.

- AngularJS separates compile() and link() for performance reasons

controller()

- Instantiated before preLink()
- The controller is injectable
- No DOM manipulations inside controller
- Use controller to provide public API (e.g. `ng-model`) for other directives
- Use case example: fetch server-side data based on the provided attribute value to display inside the directive

Scope

```
myModule.directive('directiveName', function factory (injectables) {
    return {
        priority: 0,
        template: '<div></div>', // or // function(tElement, tAttrs) { ... },
        // or
        // templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
        replace: false,
        transclude: false,
        restrict: 'A',
        scope: false,
        controller: function ($scope, $element, $attrs, $transclude, otherInjectables) { ... },
        controllerAs: 'stringAlias',
        require: 'siblingDirectiveName', // or // ['^parentDirName', '?optionalDirName', '?^optionalParent'],
        compile: function compile (tElement, tAttrs, transclude) {
            return {
                pre: function preLink (scope, iElement, iAttrs, controller) { ... },
                post: function postLink (scope, iElement, iAttrs, controller) { ... }
            };
            // or
            // return function postLink( ... ) { ... }
        }
        // or
        // link: {
        //  pre: function preLink(scope, iElement, iAttrs, controller) { ... },
        //  post: function postLink(scope, iElement, iAttrs, controller) { ... }
        // }
        // or
        // link: function postLink( ... ) { ... }
    };
});
```

Isolate Scope

- The **Isolate** scope is created by assigning an object to scope property

- Possible types of attributes:

```
scope: {
  user       : '=',  // Bind the user to the object given
  onLogin    : '&',  // Pass a reference to the method
  welcomeMsg: '@'    // Store the string associated by welcomeMsg
}
```

```
<login-widget user="model.currentUser"
              onLogin="model.hide()"
              welcomeMsg="Welcome to the Auction App!">
</login-widget>
```

- An attribute can be optional: **user: '?='**

Restrict

```
myModule.directive('directiveName', function factory (injectables) {
    return {
        priority: 0,
        template: '<div></div>', // or // function(tElement, tAttrs) { ... },
        // or
        // templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
        replace: false,
        transclude: false,
        restrict: 'A',
        scope: false,
        controller: function ($scope, $element, $attrs, $transclude, otherInjectables) { ... },
        controllerAs: 'stringAlias',
        require: 'siblingDirectiveName', // or // ['^parentDirName', '?optionalDirName', '?^optionalParent'],
        compile: function compile (tElement, tAttrs, transclude) {
            return {
                pre: function preLink (scope, iElement, iAttrs, controller) { ... },
                post: function postLink (scope, iElement, iAttrs, controller) { ... }
            };
            // or
            // return function postLink( ... ) { ... }
        }
        // or
        // link: {
        //  pre: function preLink(scope, iElement, iAttrs, controller) { ... },
        //  post: function postLink(scope, iElement, iAttrs, controller) { ... }
        // }
        // or
        // link: function postLink( ... ) { ... }
    };
});
```

The Restrict Property

- Determines how we can use directive in HTML
- Can be one of the following:

```
'A' - <span auction-navbar></span>
'E' - <auction-navbar></auction-navbar>
'C' - <span class="auction-navbar"></span>
'M' - <!-- directive: auction-navbar -->
```

Transclude

```
myModule.directive('directiveName', function factory (injectables) {
    return {
        priority: 0,
        template: '<div></div>', // or // function(tElement, tAttrs) { ... },
        // or
        // templateUrl: 'directive.html', // or // function(tElement, tAttrs) { ... },
        replace: false,
        transclude: false,
        restrict: 'A',
        scope: false,
        controller: function ($scope, $element, $attrs, $transclude, otherInjectables) { ... },
        controllerAs: 'stringAlias',
        require: 'siblingDirectiveName', // or // ['^parentDirName', '?optionalDirName', '?^optionalParent'],
        compile: function compile (tElement, tAttrs, transclude) {
            return {
                pre: function preLink (scope, iElement, iAttrs, controller) { ... },
                post: function postLink (scope, iElement, iAttrs, controller) { ... }
            };
            // or
            // return function postLink( ... ) { ... }
        }
        // or
        // link: {
        //  pre: function preLink(scope, iElement, iAttrs, controller) { ... },
        //  post: function postLink(scope, iElement, iAttrs, controller) { ... }
        // }
        // or
        // link: function postLink( ... ) { ... }
    };
});
```

Transclude (cont.)

- Instructs AngularJS to compile the content of the host element
- The content becomes available inside the directive's template
- The content's scope is a sibling of the directive's scope
- Example modal window:

```
<!-- Usage -->
<auction-modal>
    <h1>Title</h1>
    <p>Modal window's message</p>
</auction-modal>

<!-- Directive's template -->
<div class="modal">
    <div ng-tansclude></div>
</div>
```

Example

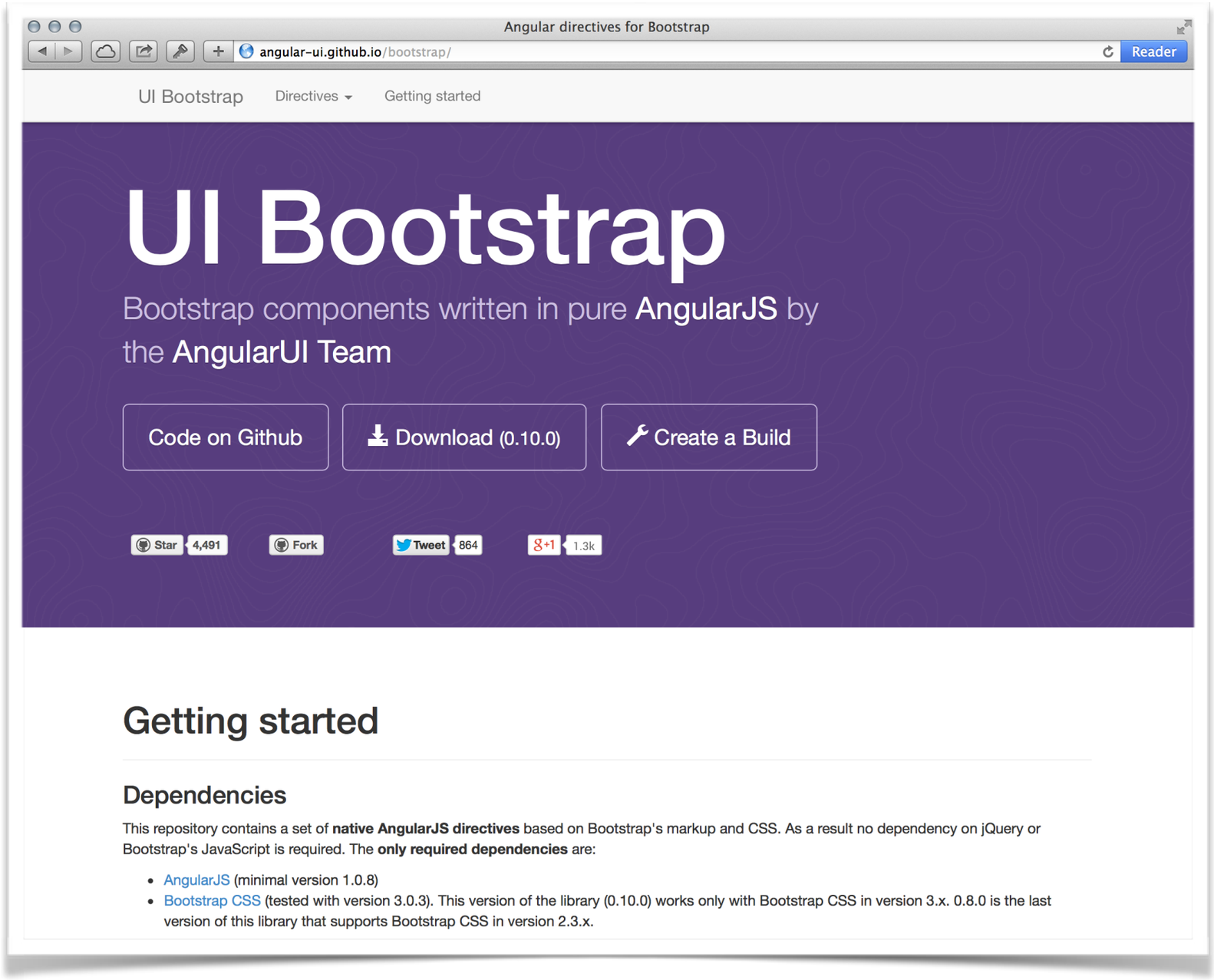
```
angular.module('auction').directive('auctionPriceRange',
  ['$timeout', ($timeout) => <ng.IDirective>{
    scope: {
      minPrice  : '@',
      maxPrice  : '@',
      lowPrice  : '=',
      highPrice : '='
    },
    restrict: 'E',
    templateUrl: 'views/partial/price-range.html',
    link: function (scope: any, el, attrs) {
      var slider: any = angular.element(el).find('input[type=text]'),
          min = scope.minPrice || 0,
          max = scope.maxPrice || 500;

      // Initialize slider
      slider.slider({min: min, max: max, value: [scope.lowPrice  || min, scope.highPrice || max]});

      // Slider -> numeric fields
      slider.on('slideStop', (e) => {
        scope.$apply(() => {
          if (scope.lowPrice  != e.value[0]) scope.lowPrice  = e.value[0];
          if (scope.highPrice != e.value[1]) scope.highPrice = e.value[1];
        });
      });

      // Numeric fields -> slider
      var curVal = () => slider.slider('getValue');
      var setSlider = (low, high) => slider.slider('setValue', [low, high]);
      scope.$watch('lowPrice',  (newVal) => setSlider(newVal, curVal()[1]));
      scope.$watch('highPrice', (newVal) => setSlider(curVal()[0], newVal));
    }
  }]);
```

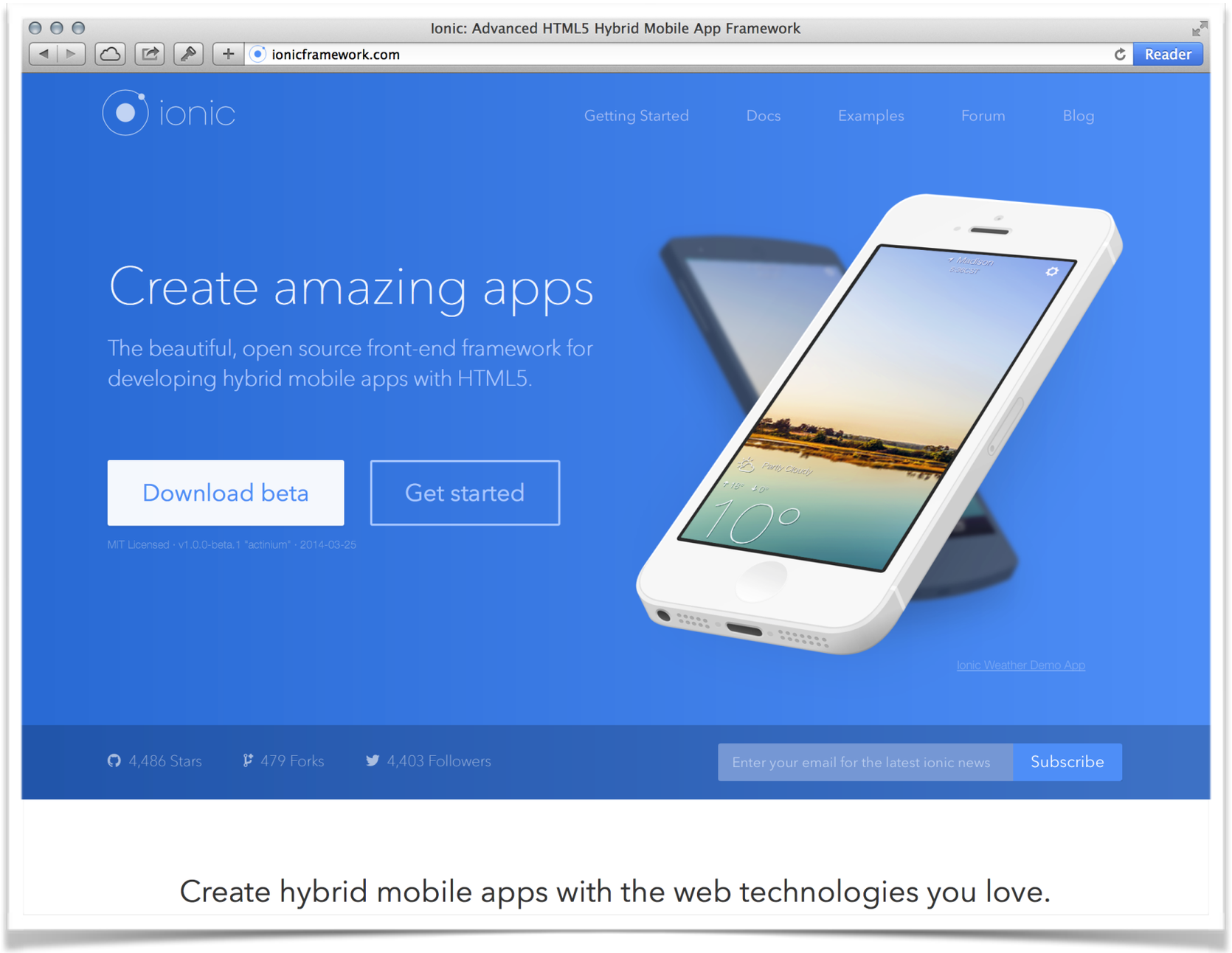
Angular UI



http://angular-ui.github.io/bootstrap/

FARATA

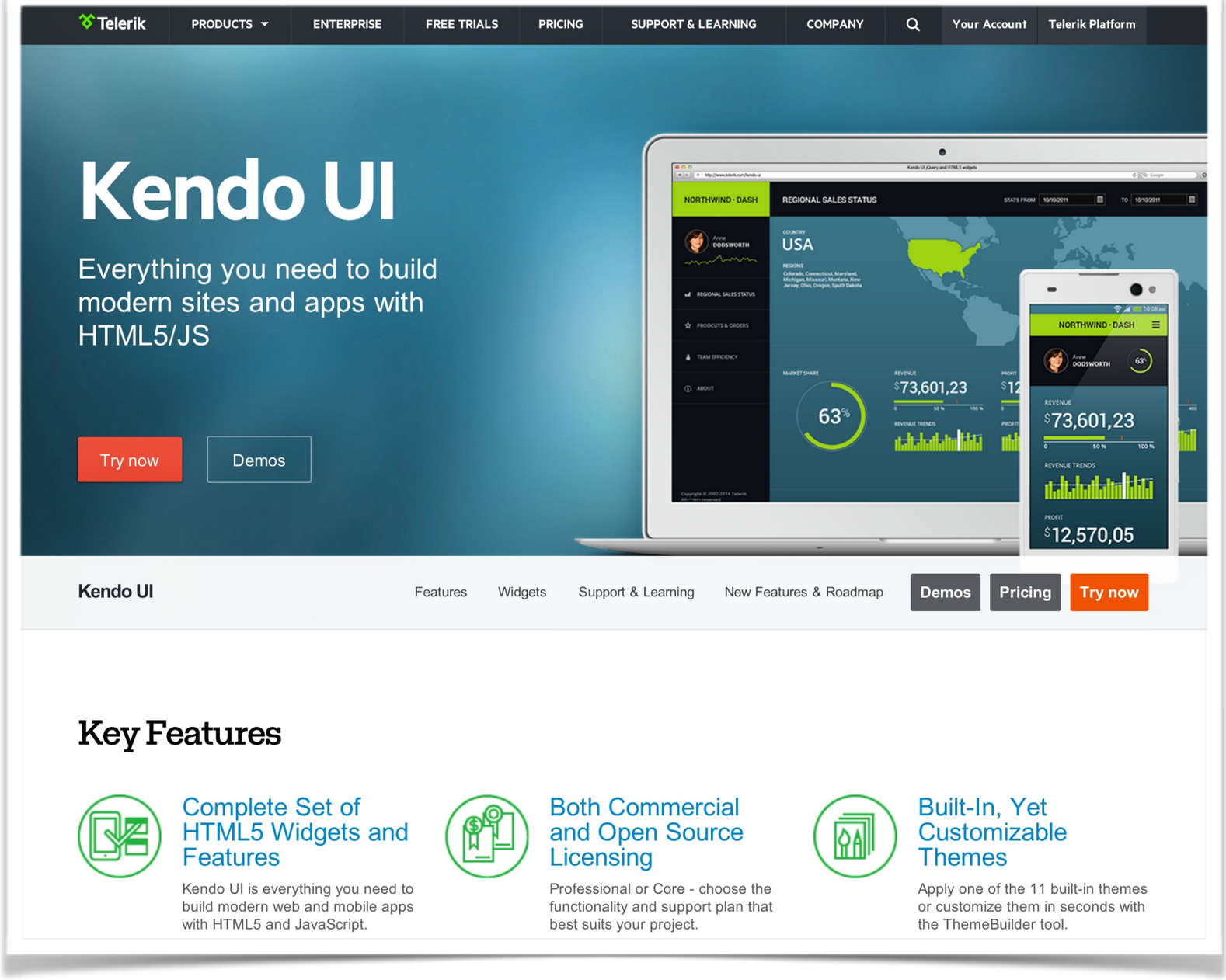
Ionic



http://ionicframework.com

FARATA

Kendo UI



telerik.com/kendo-ui

FARATA

Walkthrough 1

Search Form. Datepicker and Price Range Directives.

Use detailed instruction provided in `walkthrough_1_guide.html` file

AngularJS Forms

Form Validation

- Angular provides *client-side* validation features
- Use the client-side validation for instant feedback to the user. It doesn't make your app secure.
- Form validation available within `<form>` HTML element
- Form must have a name
- Use the `novalidate` attribute to disable native browser's validation
- AngularJS attaches custom behaviour to standard HTML5 validation rules
- Example: `<form name="search" novalidate>…</form>`

Validation rule examples

```
<input type="text" required />
```

```
<input type="text" ng-minlength="3" />
```

```
<input type="text" ng-maxlength="30" />
```

```
<input type="text" ng-pattern="[a-zA-Z]" />
```

```
<input type="email" name="email" ng-model="user.email" />
```

```
<input type="number" name="age" ng-model="user.age" />
```

```
<input type="url" name="homepage" ng-model="user.profileUrl" />
```

Form and Scope

- A form creates its fields on the scope
- Fields expose the validation properties
- The validation properties can be used for data binding
- To access a property use following syntax:

```
// Usage
formName.inputFieldName.property

// Example
searchForm.name.$invalid
```

Field Properties

```
// If the field isn't touched by user,
// returns true, otherwise - false
searchForm.productTitle.$pristine

// True - if user modified the field's value,
// otherwise - false
searchForm.productTitle.$dirty

// True if the field is currently valid, otherwise - false
searchForm.productTitle.$valid

// True if the field is currently invalid, otherwise - false
searchForm.productTitle.$invalid
```

Styling Validation State

Each validation property has corresponding CSS classes that automatically added to the fields as soon as the state is changed:

```
.ng-pristine {}
.ng-dirty {}
.ng-valid {}
.ng-invalid {}
```

$error property

- Lists all the validators attached to the field and their current state:

```
searchForm.productTitle.$error = {
  required: false, // Field's value passes "required" validation rule
  minlength: true  // Field's value fails "ng-minlength" validation rule
};
```

Form Properties

- Form itself exposes the same properties as individual fields:

```
// If the field isn't touched by user,
// returns true, otherwise – false
searchForm.$pristine

// True – if user modified the field's value,
// otherwise – false
searchForm.$dirty

// True if the field is currently valid, otherwise – false
searchForm.$valid

// True if the field is currently invalid, otherwise – false
searchForm.$invalid

searchForm.productTitle.$error = {
   required: [‘productTitle’, ‘category’] // List of invalid fields
};
```

Custom Validation

```
angular.module('auction')
  .directive('ensureUnique', ['$http', ($http) => {
    return {
      require: 'ngModel',
      link: function(scope, el, attrs, ctrl) {
        scope.$watch(attrs.ngModel, () => $http
          .post('/api/check/' + attrs.ensureUnique, {'field': ctrl.$modelValue})
          .success((data) => ctrl.$setValidity('unique', data.isUnique))
          .error((data) => ctrl.$setValidity('unique', false));
        );
      }
    }
  }]);
```

Interacting with RESTful Services

REST Principles (Roy Fielding)

- Every resource on the Web has an ID (URI)
- Use uniform interface: HTTP **Get, Post, Put, Delete.** Separation of concerns.
- A resource can have multiple representations (text, JSON, XML, PDF, etc.)
- Requests are stateless – no client-specific info is stored between requests
- You can link one resource to another(s)
- Resources should be cacheable
- A REST app can be layered

HTTP Methods

- GET Safe, Idempotent, cacheable
- PUT Idempotent
- DELETE Idempotent
- HEAD Safe, Idempotent
- POST None of the above

GET is for retrieval, POST for inserts, PUT – updates, DELETE - removal.

Idempotent: regardless of how many times a given method is invoked, the end result is the same.

$http Service

$http Service (cont.)

- XMLHttpRequest wrapper
- Basic usage:

```
this.$http({
  method: 'GET',
  url: 'data/featured.json'
})
.success((data, status, headers, config) => data.items)
.error((data, status, headers, config) => console.log(ERROR_MSG));
```

- $http() returns **HttpPromise**

HttpPromise

- HttpPromise extends Promise
- Handy methods: **success()**, **error()**
- The methods destructure HTTP Response object into **data, status, headers, config** parameters
- A *then(successCallback, errorCallback)* API is available.

Shortcut Methods

- get(), post(), put(), delete(), head(), jsonp()
- Example:

```
this.$http.get('data/featured.json')
.success((data, status, headers, config) => data.items)
.error((data, status, headers, config) => console.log(ERROR_MSG));
```

Interceptors

- Allow to intercept all HTTP requests before they are sent or after they arevreceived from the server
- Four types of interceptors:
  - **request** - modify HTTP request before it is sent
  - **response** - modify HTTP response before it is passed to the call site
  - **requestError** - handle errors of the previous request *interceptor*
  - **responseError** - handle errors of the previous response *interceptor*

```
angular.module('auction').factory('myInterceptor', ($q) => {
  return {
    'request': (config) => { /*...*/ },
    'response': (response) => { /*...*/ },
    'requestError': (rejection) => { /*...*/ },
    'responseError': (rejection) => { /*...*/ }
  }
});
angular.module('auction').config(($httpProvider) => {
  $httpProvider.interceptors.push('myInterceptor');
});
```

ngResource

$resource Service

- Simplifies communication with RESTful back-ends
- Comes with ngResource module
- Developed by AngularJS team

- To add it to your project install it with bower:

```
bower install angular-resource —save
```

- Add as dependency to main app's module:

```
angular.module('auction', ['ngResource']);
```

Usage Example

```
// Create a resource
var Product = $resource('/rest/products/:productId.json', { productId: '@id' });

// Get a single product
var p = Product.get({ id: '123' },
  (resp) => { /* success callback */ },
  (err)  => { /* error callback*/ });

// Get a product collection
var p = Product.query({ page: 2 }, (resp) => { /* success callback */ });

// Saves using POST
Product.save({}, { id: 123 }, (resp) => {}, (err) => {});

// Removes product using DELETE
// There is a synonym method Product.remove()
Product.delete({}, { id: 123 }, (resp) => {}, (err) => {});


// To create a custom request with PUT method:
auction.factory('Product', ($resource) =>
  $resource('/notes/:id', null, {
    'update': { method:'PUT' }
  })
);
```

Returned Object

- $resource-generated methods return a wrapped version of the requested object
- A wrapped object provides additional methods: *$save(), $remove(), $delete()*
- Example:

```
// Get the product and immediately update it:
Product.get({ id: 123 }, (product) => {
  product.price = 3.99;
  product.$save();
});

// Equivalent to:
Product.save({ id: 123 }, { price: 3.99 });

// $resource methods work asynchronously:
var p = Product.get({ id: 123 });
p.$promise.then((product) => { /*...*/ });
```

Restangular

Main Benefits

- Uses HTTP methods in a RESTful way
- Hides URLs
- Nested resources
- Leverages promises
- Clean API
- Complete list of benefits is here

Usage Examples

```
angular.module('auction').factory('ProductService', (Restangular) => {
  // Create a resource object
  var Product = Restangular.all('products');

  // URL is automatically constructed GET /products
  var allProducts = Product.getList();

  // Restangular knows how to construct URL using unique ID: GET /products/123
  var oneProduct = Restangular.one('products', 123);

  // Access nested resource: GET /products/123/reviews
  var reviews = oneProduct.getList('reviews');

  // Custom query string params and headers: GET /products/123/reviews?minRating=4
  var reviews = oneProduct.getList('reviews', { minRating: 4 }, { /* custom headers */ })

  // Create a new object: POST /products/123/reviews
  reviews.post({ rating: 5, comment: '' });

});

// Configure base URL
angular.module('auction').config((RestangularProvider) => {
  RestangularProvider.setBaseUrl('/rest/v1');
});
```

Returned Object

- Returned object is a promise:

- Has additional $object property:

```
reviews.post({ rating: 5 }).then(
  (review) => $scope.reviews = reviews.getList().$object,
  (error) => { /* Handle error message */ }
);
```


Walkthrough 2

Refactoring ProductService to use Restangular

- Use detailed instruction provided in `walkthrough_2_guide.html` file.

Additional Resources

- 5 Guidelines For Avoiding Scope Soup in Angular
- AngularJS Forms
- Form validation with AngularJS
- Build custom directives with AngularJS
- Restangular on Angular
- Official $resource documentation

The Next Project Review

1. Before starting working on the homework complete walkthroughs 1 and 2.
2. Implement the following Search Form validation rules:
    a. Product title should be at least 3 characters long
    b. Max Auction Close Day shouldn't be older than the current date.
    c. Yhe Number of Bids should be a positive value
3. Improve Search Form integration into the app:
    1. Add data binding for search form fields.
    2. When you navigate from the Search page to the Product Details page all the Search Form fields persist their values on the form. (Hint: since services are singletons you can use them to share the data among controllers).
    3. When you click the "Find More" button on the Search Page, a search criteria (form field values) should be reflected in the browser's address bar as query string. So you can copy the link and use it to open Search Page at the same state in a different browser's tab.
    4. When either Search Page of Product Page are opened, Search form fields should be pre-populated with the values from the query string.
    5. When you click the "Find More" button on the Product Details page, user should be navigated to the Search Page and all the search criteria should be reflected in the address bar as query string.
    6. The search criteria should be passed as JavaScript object to ProductService.find() method.
    7. When users click Search button in the Navigation Bar, they should be navigated to the Search Page, and Title input field in the Search Form should be pre-populated with the value from Navabar's input.
4. Read AngularJS topics from Additional Resources section.
5. *Extra challenge*: add the grunt plugin html2js (https://www.npmjs.org/package/grunt-html2js) to combine all the views in a single JS file and register them inside AngularJS $templateCache, so all of the will be loaded along with index.html page.
6. Review a proposed solution at http://farata.github.io/modernwebdev-showcase/homework4/dist