Modern Web Development for Java Programmers

Unit 3. AngularJS in Depth. Build automation with Grunt. Package Management with Bower.

Unit 3 Timeline

- Homework 2 Review 30 min
- AngularJS Recap 10 min
- AngularJS Scopes 20 min
- AngularJS Routing 15 min
- *Walkthrough 1*: Enabling routing for the auction app 20 min
- Break 10 min
- AngularJS Directives 30 min
- AngularJS Filters 15 min
- *Walkthrough 2*: Decomposing the auction app UI using directives 20 min
- Tools: Yeoman, Bower, Grunt 30 min

AngularJS Recap

Major Players

- Modules
- Controllers
- Directives
- Filters
- Services

Modules

- Each app is at least one module
- Help to organize the code
- Can have other modules as dependencies
- DI container per module
- Declaratively bootstrap application
- To create a module - two parameters:

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

- To get existing module - one parameter:

```
angular.module('auction');
```

Controllers

- Handle user interactions
- Orchestrate models and services
- Do not interact with HTML directly
- Use **$scope** to display data on view
- Registered in DI container using *controller()*

Directives

- Attach behaviour to HTML elements
- Can have visual as well as non-visual effect (*ngRepeat* vs *ngApp*)
- Represented as HTML element's attributes (e.g. *ng-app*)
- Can interact with view directly
- Enable UI decomposition (*ng-include*)
- Enable reusable UI components (will learn more later this class)
- Registered in DI container using *directive()*

Filters

- Formats expression's value
- Registered in DI container using *filter()*
- Example:

```
<span>{{ product.price | currency }}</span>
```

Expressions

- Code snippets placed in curly braces within HTML template - `<span>{{ name }}</span>`
- Similar to JavaScript, but:
  - Executed within **scope** context
  - Forgiving to **undefined** and **null**
  - Not control flow statements (consider directive)
  - You can pipe filters to expression value like this - `<span>{{ name | uppercase }}</span>`

AngularJS Scopes

What is Scope?

Scope is a JavaScript object that keeps application models available as the data binding source on views.

```
app.controller('MainCtrl', function ($scope) {
    $scope.name = 'John';
});
```

Scope

```
{
  name: 'John'
}
```

```
<div ng-controller="MainCtrl">
  <p>{{ name }}</p>
</div>
```

Scope Hierarchies

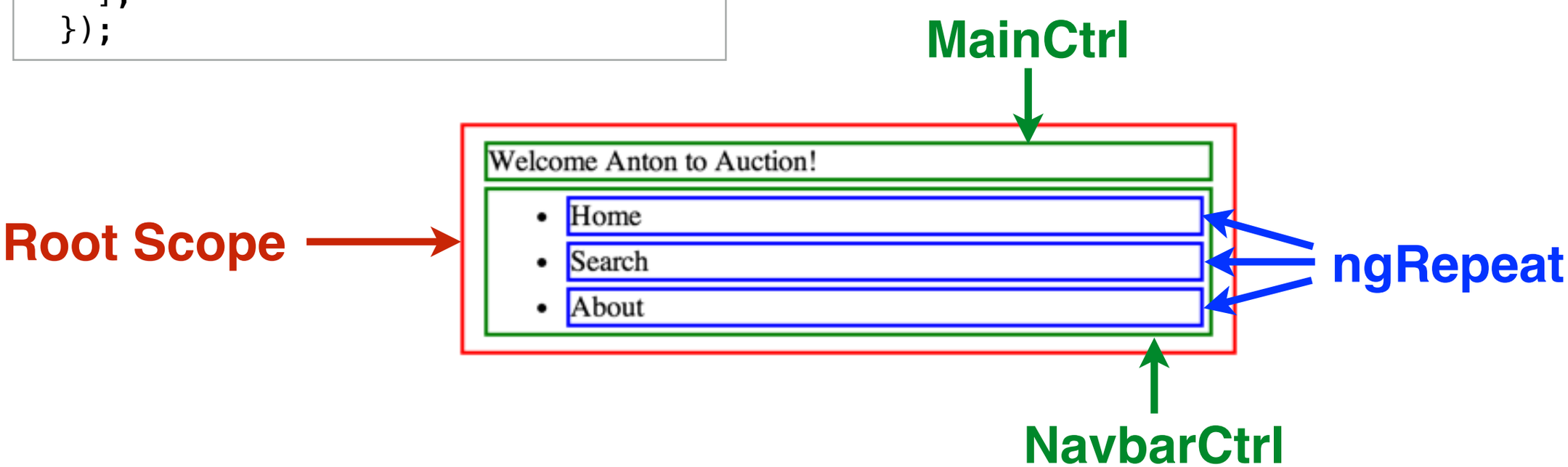
- Each application has only one **rootScope**
- Directives can create **child scopes** (e.g. ng-controller, ng-repeat)
- Child scopes **prototypically** inherit from their parents
- Directives can create **isolated scopes** (more on this later in this unit)

Scope Hierarchy Example

```
app.controller('MainCtrl',
  function ($scope, $rootScope) {
    $scope.appName = 'Auction';
    $rootScope.currentUser = 'Anton';
  });

app.controller('NavbarCtrl',
  function ($scope) {
    $scope.menuItems = [
      'Home',
      'Search',
      'About'
    ];
  });
```

```
<div ng-controller="MainCtrl">
  Welcome {{ currentUser }} to {{ appName }}!
</div>
<ul ng-controller="NavbarCtrl">
  <li ng-repeat="item in menuItems">
    {{ item }}
  </li>
</ul>
```



Scope Hierarchy Example

- Scopes Hierarchy mimics the DOM structure
- To get the scope for any element (for **debugging only**):

```
angular.element(domEl).scope();
```

```
<!DOCTYPE html>
▼<html ng-app="auction" class="ng-scope">
  ▶<head>…</head>
  ▼<body class="auction-app">
    <p ng-controller="MainCtrl" class="ng-scope ng-binding">Welcome Anton to Auction!</p>
    ▼<ul ng-controller="NavbarCtrl" class="ng-scope">
      <!-- ngRepeat: item in menuItems -->
      <li ng-repeat="item in menuItems" class="ng-scope ng-binding">Home</li>
      <!-- end ngRepeat: item in menuItems -->
      <li ng-repeat="item in menuItems" class="ng-scope ng-binding">Search</li>
      <!-- end ngRepeat: item in menuItems -->
      <li ng-repeat="item in menuItems" class="ng-scope ng-binding">About</li>
      <!-- end ngRepeat: item in menuItems -->
    </ul>
  </body>
```

Events Propagation

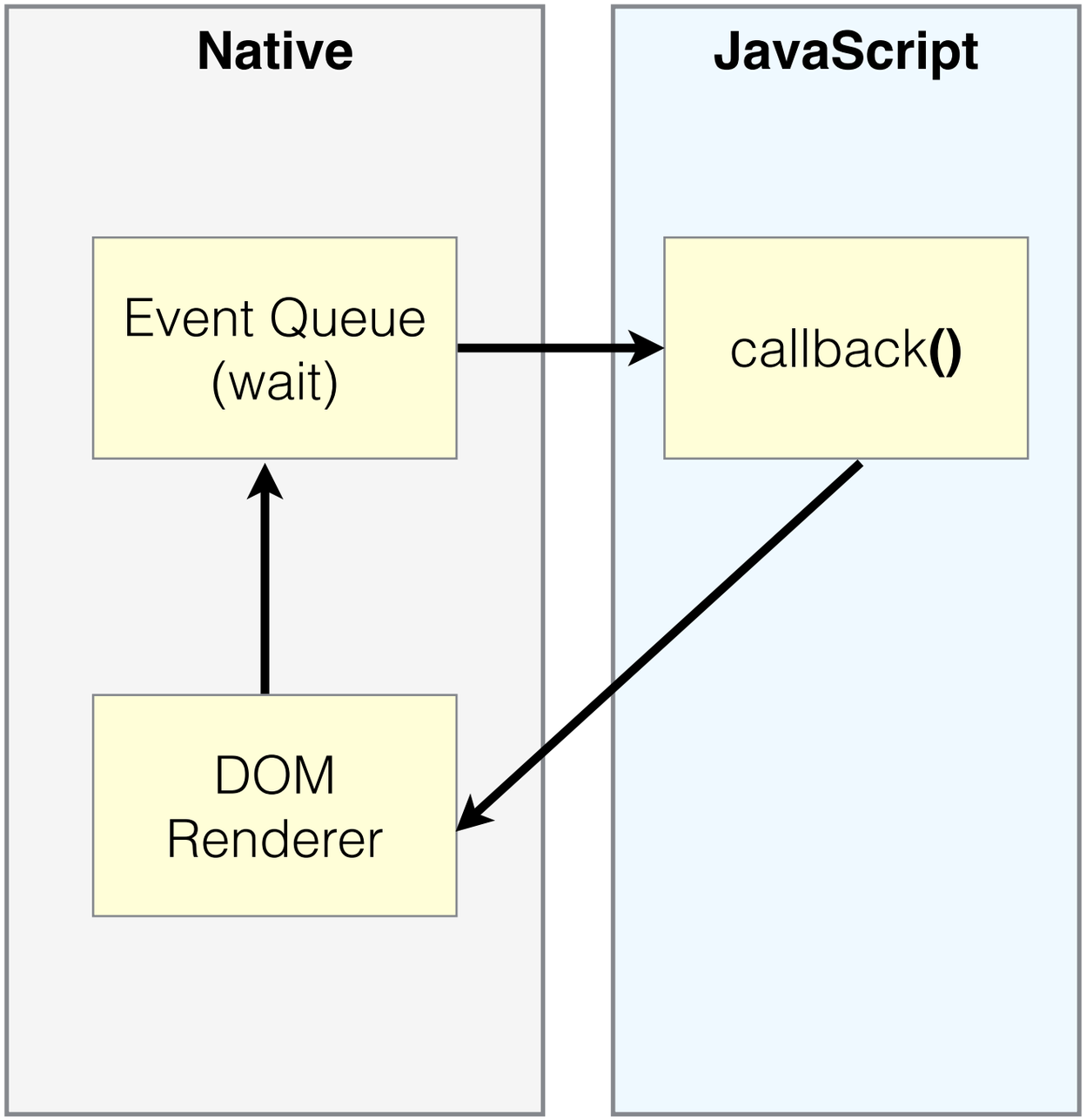
- Scope can fire 2 types of events:
  - **$emit(name, args)** - bubbling event, goes through the hierarchy of parent scopes up to the rootScope
  - **$broadcast(name, args)** - propagates event down through the entire hierarchy of child scopes
- Scope can listen to events - **$on(name, listener)**

How Scopes Work

- To make data-binding work AngularJS needs to:
  - observe model changes
  - modify DOM
  - observe DOM changes
  - modify models

How Scopes Work

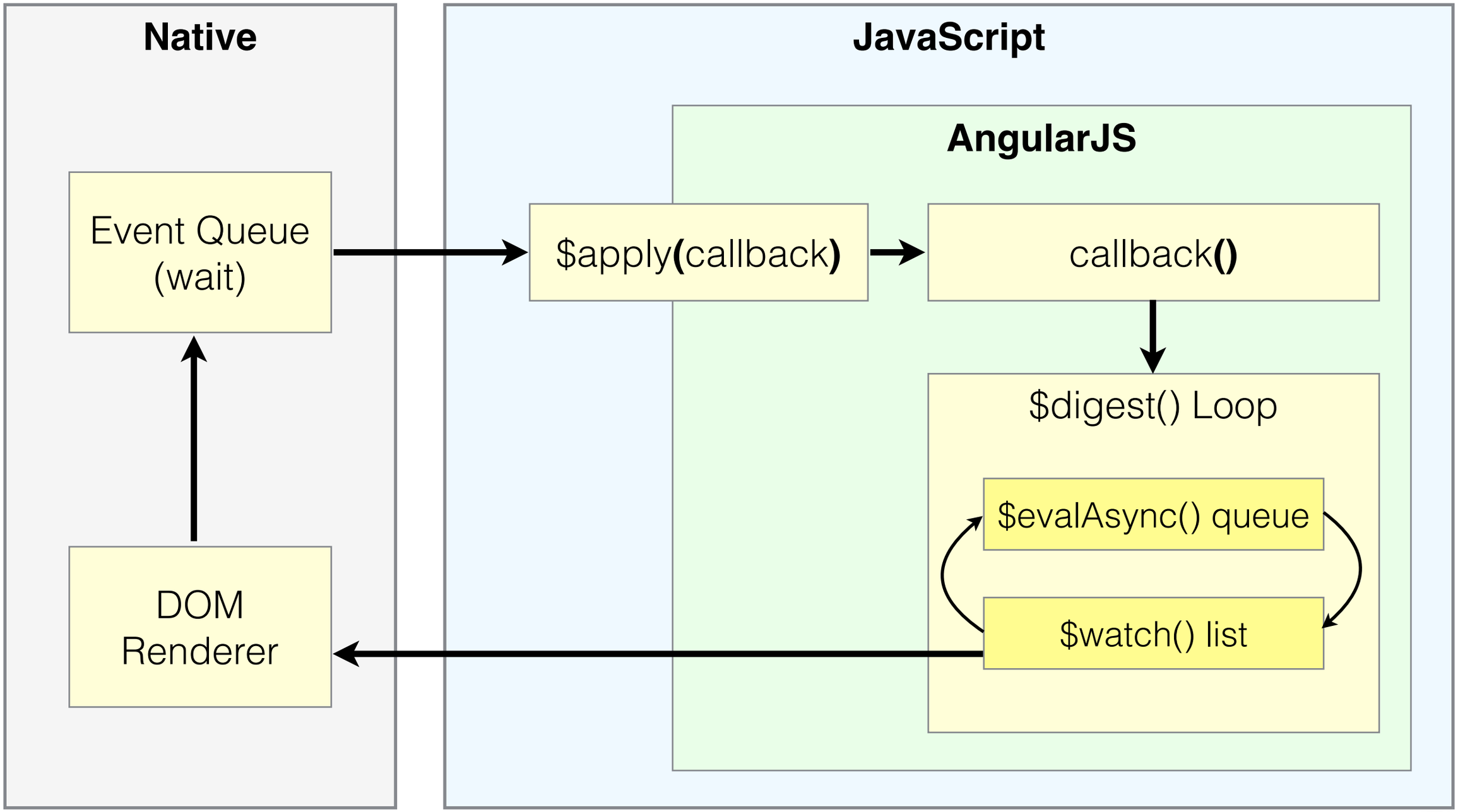
Native browser's event loop



WARNING: Next two slides can cause headaches

How Scopes Work

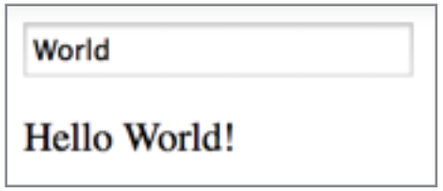
AngularJS modified event loop



How Scopes Work: Example

```
<body ng-controller="MainCtrl">
    <input ng-model="name" type="text">
    <p>Hello {{ name }}!</p>
</body>
```

```
angular.module('auction', [])
  .controller('MainCtrl', function($scope) {
    $scope.name = '';
  });
```



1. App launches, passes *configuration* phase and enters *run* phase.
2. **ng-model** and **input** directive set up *keydown* listener on the **<input>** element
3. AngularJS compiles HTML and sets up **$watch** on **{{ name }}** changes
4. App enters browser's event loop
5. Pressing 'W' causes browser fire *keydown* event on the **<input>**
6. **input** directive captures the change and calls **scope.$apply()**. Execution enters AngularJS modified event loop.
7. Inside **scope.$apply()** input directive calls **ngModelController.$setViewValue()** which updates internal state of view value and applies new value to **name** property. No **$digest** at this time.
8. **scope.$apply()** finishes execution and **$digest** loop begins.
9. **$watch** list detects a change on name property and notifies interpolation responsible for **{{ name }}** expression, which in turn updates DOM.
10. Execution exists AngularJS event loop, exits the keydown event and the JavaScript execution context.
11. The browser re-renders the view with update text.

AngularJS Routing

What is Routing?

- Routing enables deep-linking for single-page applications.
- Supports HTML5 History API as well as “hashbang” (/#!)
- In AngularJS represented as **$route** service:
- Maps URLs to views (HTML templates) and controllers

Leaves on a separate module

```
angular.module('ngRoute', ['ng']).provider('$route', $RouteProvider);
```

Registered with provider(), hence $routeProvider available at configuration phase

$routeProvider

- Allows configuring mapping at configuration phase

Added as dependency to the app

Config phase

Provider suffix

Path to partial view (HTML template)

Single mapping

Controller's name as it's registered in DI container

Default route

Notice, never contains "hashbang"

```
angular.module('auction', ['ngRoute'])
  .config(['$routeProvider', function ($routeProvider) {
    $routeProvider
      .when('/', {
        templateUrl: 'views/main.html',
        controller: 'MainCtrl'
      })
      .when('/search', {
        templateUrl: 'views/search.html',
        controller: 'SearchCtrl'
      })
      .otherwise({
         redirectTo: '/'
      });
  }]);
```

Where Display HTML View?

- Inside HTML element marked with ng-view directive
- Single ng-view for the entire app
- ng-view always in sync with URL according to $routeProvider's mapping

```
<div class="container" ng-view></div>
```

How Navigate to a Route?

- Clicking on a link:

```
<a href="#/search">Submit</a>
```

- Using **$location** service:

```
$location.url('/search');
```

- No named routes.

Route parameters

1. Define named placeholder

```
$routeProvider.when('/product/:id', {
  templateUrl: 'views/search.html',
  controller: 'SearchCtrl'
});
```

Name should match

2. Substitute placeholder in a template

```
<a href="#/product/{{ productId }}">Show Product</a>
<a ng-href="#/product/{{ productId }}">Show Product</a>
```

3. Access parameter in a controller

```
angular.module('auction')
  .controller('ProductCtrl', function ($routeParams) {
    var productId = $routeParams.id;
  });
```

Promises

- Promise is an object that wraps a value that will be available later on as the result of an asynchronous operation.

- Represented in AngularJS as **$q** service.

```
getFeatured() {
  var deferredProducts = this.$q.defer();

  this.$http.get('data/featured-products.json')
    .success((data) => deferredProducts.resolve(data.items))
    .error(() => deferredProducts.reject());

  return deferredProducts.promise;
}
```

Route's Dependencies

- Route can define dependencies it must to obtains before navigating to view.
- Dependencies defined using route's **resolve** property.
- Dependencies will be injected into target controller.
- If a dependency is promise, $route service will wait until the promise is either resolved or rejected.

resolve Example

```
$routeProvider.when('/product/:id', {
    templateUrl: 'views/product.html',
    controller: 'ProductCtrl',
    resolve: {
        product: ['$route', '$http', '$q', function ($http, $q) {
            var deferredProduct = $q.defer(),
                productId = route.current.params.id;

            $http.get('/product/' + id)
                .success((data) = deferredProduct.resolve(data))
                .error(() => deferredProduct.reject());

            return deferredProduct.promise;
        }]
    }
});
```

Name should match

```
angular.module('auction')
  .controller('ProductCtrl', function (product) {
  });
```

TypeScript Trick

```
class ProductController {
  //...
  public static resolve = {
    product: ['$route', '$http', '$q',
      function ($http, $q) {
        var deferredProduct = $q.defer(),
            productId = route.current.params.id;

        $http.get('/product/' + id)
          .success((data) = deferredProduct.resolve(data))
          .error(() => deferredProduct.reject());

        return deferredProduct.promise;
      }]
  };
}
```

```
$routeProvider.when('/product/:id', {
    templateUrl: 'views/product.html',
    controller: 'ProductCtrl',
    resolve: controller.MainController.resolve
});
```


Walkthrough 1

Enabling routing for the auction app

Walkthrough 1: Steps

- Import IntelliJ IDEA modules `unit2` and `unit3` provided in the handouts.
- Use application in directory `unit2/homework` as starting point for the walkthrough.
- Refactor the app in order to use routing and mg-view instead of `ng-if` and `ng-include`.
- Solution for the walkthrough located in the directory `unit3/walkthroughs/w1`.

AngularJS Directives

Directives

- Attach behaviour to the DOM elements
- Can have visual and non-visual effect (*ng-controller* vs *ng-repeat*)
- Solve two problems:
  - UI decomposition
  - Reusable components

How They Look Like

- Can be represented in several forms:
  - HTML element's attribute: `ng-app`, `data-ng-app`
  - HTML element's: `<auction-navbar>`
  - CSS classes `<div class="auction-navbar">`

UI Decomposition

```
angular.module('auction')
  .directive('auctionNavbar', function () {
    return {
      scope: true,
      restrict: 'E',
      templateUrl: 'views/partial/navbar.html'
    };
  });
```

Available as auction-navbar in HTML

New child scope is created for directive. Default - `false`.

Must be used as HTML element: `<auction-navbar>`

Path to the HTML template (partial view)

Restrict Property

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

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

Scope Property

- Use **isolated** scope for reusable UI components
- To create **isolated** scope:

```
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>
```

Reusable Components

- Must use isolated scope
- Use `link` function to bind to DOM events and update the DOM.
- `link` function is invoked as soon as the directive is linked to the DOM
- Use `controller` property if need to provide public API (e.g. `ng-model`)

Reusable Component Example

```
angular.module('auction')
  .directive('auctionNavbar', function () {
    return {
      scope: {
        reviews: '='
      },
      restrict: 'E',
      templateUrl: 'views/partials/AuctionRatingBreakdown.html',
      link: (scope) => {
        var groupedByRating = _.groupBy(scope.reviews, r => r.avgRating),
            bars = [];

        for (var i = 5; i >= 1; i--) {
          var nStarReviews = groupedByRating[i] || [],
              nStarCount = nStarReviews.length,
              totalReviews = scope.reviews.length;

          bars.push({
            rating: i,
            count: nStarCount,
            share: (nStarCount / totalReviews * 100) + '%'
          });
        }

        scope.bars = bars;
      }
    };
  });
```

AngularJS Filters

Filter Features

- Transforms format of an expression value
- Can be used in HTML and directly invoked from code.
- Take at least one parameter - the value to transform.
- Can take arbitrary number of parameters.

Filter Example

```
var names = ['John', 'Mike', 'Kate'];
```

```
<span>{{ names | join : ', ' }}</span>
```

```
angular.module('auction')
  .filter('join', function (array, separator) {
    return array.join(separator);
  });
```

```
'John, Mike, Kate'
```

Walkthrough 2

Decomposing the auction app UI using directives

Walkthrough 2: Steps

- Use application in directory `unit3/walkthoughs/w1` as starting point for the walkthrough.
- Refactor the app in order to use custom directives for UI decomposition. Move navigation bar and footer from `index.html` to separate directives. Directives should be represented as custom HTML elements.
- Solution for the walkthrough located in the directory `unit3/walkthroughs/w2`.

Node.js

- Platform for executing JavaScript code outside the web browser
- Built on top of Chrome's JavaScript engine - V8
- Event-driven non-blocking I/O model
- Allows writing server-side JavaScript applications
- We will use as the engine for JavaScript development tools
- Pre-built installers for all popular platforms here
- Instructions for installing from package managers - here

npm

- npm is Node.js package manager
- We use npm for development-tools dependencies
- Huge <u>collection</u> of packages
- Use Node.js installation instructions to install npm
- Available on the PATH as **`npm`**

npm: package.json

- package.json describes npm package. The most important for us - dependencies.
- Two types of dependencies:
  - **dependencies** - we won't use since we don't publish production-ready npm package
  - **devDependencies** - available only during development (e.g. TypeScript compiler, Grunt, development web server, etc.)
- To install all dependencies listed in package.json: `npm install`
- To install new dependency: `npm install grunt-ts`
- Use `--save` and `--save-dev` to automatically update package.json along with installing a new package.

npm: version range styles

- Full list here - The semantic versioner for npm

```
{
  "name": "auction",
  "version": "0.1.0",
  "dependencies": {},
  "devDependencies": {
    "grunt": "^0.4.2",>=0.4.2-0 <1.0.0-0
    "grunt-concurrent": "~0.4.1", “Reasonably close”: >=0.4.1-0 <0.5.0-0
    "load-grunt-tasks": "0.2.1"  Exactly 0.2.1 version
  }
}
```

YO

GRUNT

BOWER

Yeoman

- Scaffolding tool
- Installed with `npm install -g yo`
- Available on the PATH as `yo`
- Rich <u>collection</u> of *generators*
- Generators assemble best-practices in provided templates
- To install generator: `npm install -g generator-angular`
- To use generator: `yo angular`

Bower

- Bower is a package manager
- Package can have any layout and any type of assets
- We use bower for runtime dependencies (the ones we actually use in code)
- Huge <u>collection</u> of packages
- To install bower: `npm install -g bower`
- Available as `bower` on the PATH

Bower: bower.json

- bower.json describes app's package. The most important - *dependencies*.
- Two types of dependencies:
  - **dependencies** - packaged with production version of the app
  - **devDependencies** - available only during development (e.g. unit testing libraries)
- To install all dependencies listed in bower.json: `bower install`
- To install new dependency: `bower install angular-resource`
- Use `--save` and `--save-dev` to automatically update bower.json along with installing a new package.

Bower: version range styles

- Full list here - The semantic versioner for npm

```
{
  "name": "auction",
  "version": "0.1.0",
  "dependencies": {
    "angular": "1.2.14",  Exactly 1.2.14 version
    "angular-route": "~1.2.14",  >=1.2.14-0 <1.3.0-0
    "bootstrap": "^3.1.1"  >=3.1.1-0 <4.0.0-0
  },
  "devDependencies": {
    "DefinitelyTyped": "*"  any version
  }
}
```

GRUNT

The JavaScript Task Runner

Grunt

- Grunt is a universal automation tool for web apps
- We use it to automate all development processes
- Grunt comes as dependency in package.json
- We need to install command-line interface to grunt:

```
npm install -g grunt-cli
```

- Available on the PATH as `grunt`
- Grunt plugins installed with npm:

```
npm install grunt-ts --save-dev
```

Gruntfile

- Loads Grunt plugins containing tasks
- Defines all configuration for tasks Grunt can execute
- Creates new tasks
- Each task can be invoked with `grunt` *`task`*
- Optionally target can be specified: `grunt` *`task:target`*

Gruntfile.js Example

```
module.exports = function(grunt) {

    grunt.initConfig({
        pkg: grunt.file.readJSON('package.json'),
        concat: {
            options: {
                separator: ';'
            },
            dist: {
                src: ['src/**/*.js'],
                dest: 'dist/<%= pkg.name %>.js'
            }
        },
        qunit: {
            files: ['test/**/*.html']
        },
        watch: {
            files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
            tasks: ['qunit']
        }
    });

    grunt.loadNpmTasks('grunt-contrib-concat');
    grunt.loadNpmTasks('grunt-contrib-qunit');
    grunt.loadNpmTasks('grunt-contrib-watch');

    grunt.registerTask('test', ['qunit']);

    grunt.registerTask('default', ['qunit', 'concat']);

};
```

Node.js way to encapsulate modules

Initializes grunt configuration

Read package.json to be able to refer to its properties

Task's configuration. Has the same name as task.

Common name for all tasks

Unique set of available options for each task

Target can have an arbitrary name

<%= %> - refers to a property in the config

<% %> - executes arbitrary JavaScript code

Loads the plugin installed with npm

Creates custom task available in the command-line

default task can be invoked with `grunt`

Files

- Most Grunt tasks operate on files
- Grunt supports several source/destination formats:

Compact format

```
dist: {
    src: ['src/bb.js', 'src/bbb.js'],
    dest: 'dest/b.js'
}
```

Example from auction app

```
less: {
  dist: {
    files: [{
      expand: true,
      cwd: '<%= yeoman.app %>/styles',
      src: '{,*/}*.less',
      dest: '.tmp/styles',
      ext: '.css'
    }]
  }
}
```

Compact format

```
dist: {
    files: {
        'dest/a.js': ['src/aa.js', 'src/aaa.js'],
        'dest/a1.js': ['src/aa1.js', 'src/aaa1.js']
    }
}
```

Files Object Format

```
dist: {
    src: ['src/bb.js', 'src/bbb.js'],
    dest: 'dest/b.js'
}
```

Additional Resources

- Grunt And Gulp Tasks For Performance Optimization
- AngularJS Scopes
- AngularJS Directives
- AngularJS Forms
- Restangular on Angular

Homework 3

- Using AngularJS routing refactor the auction app in a way to use routing.
- Using AngularJS directives decompose the auction app's UI into separate directives.
- Develop Product Details page that displays a single product. Mockup provided in the handouts - *Single Item.png*. The route to the Product Details page, should contain a product ID.
- Leverage TypeScript features while developing the app (classes, optional type annotations, generics, etc.)
- Structure your code using Models, Services, Controllers and Directives.
- Read AngularJS topics from Additional Resources section.
- Read documentation for the Grunt directives used in the auction app's Gruntfile.js