Practical Development of Web Applications with JavaScript and AngularJS

Unit 2. AngularJS Basics. Becoming Productive with JavaScript.

Overview of JavaScript Frameworks

Why use HTML5 frameworks

- They deal with cross-browser compatibility
- Make your application more structured
- May include reusable components
- Make programmers more productive
- Lower the amount of manually written code

Frameworks and Libraries

Frameworks expect you to programs using well defined rules.

Libraries just offer reusable components

There are two major types of frameworks:

- Feature complete
- Lightweight (MVC + Binding + Routing)

Feature Complete Frameworks

- Suitable for back-office applications
- Include a library of UI components
- Have good tooling
- Steeper learning curve
- May be difficult to customize
- The size of the app 1MB+

Lightweight Frameworks

- Mainly deal with application structure, navigation, AJAX
- Work best for consumer-oriented websites
- Can be combined with 3-party libraries
- Easier to learn

AngularJS

HTML enhanced for web apps!

Why AngularJS?

- Good choice for single-page applications
- MVC with minimal coding
- Easy routing
- Lightweight, yet pluggable with well-defined modular architecture
- Follows modern Web standards

Single-Page Application (SPA)

- No full Web page reloads
- Only certain pages get refreshed as results of AJAX calls
- Parts of the page may toggle from visible to hidden using CSS
- The Web page states doesn't get reset
- The user gets a feel of a desktop app
- The SEO may suffer, need special tricks

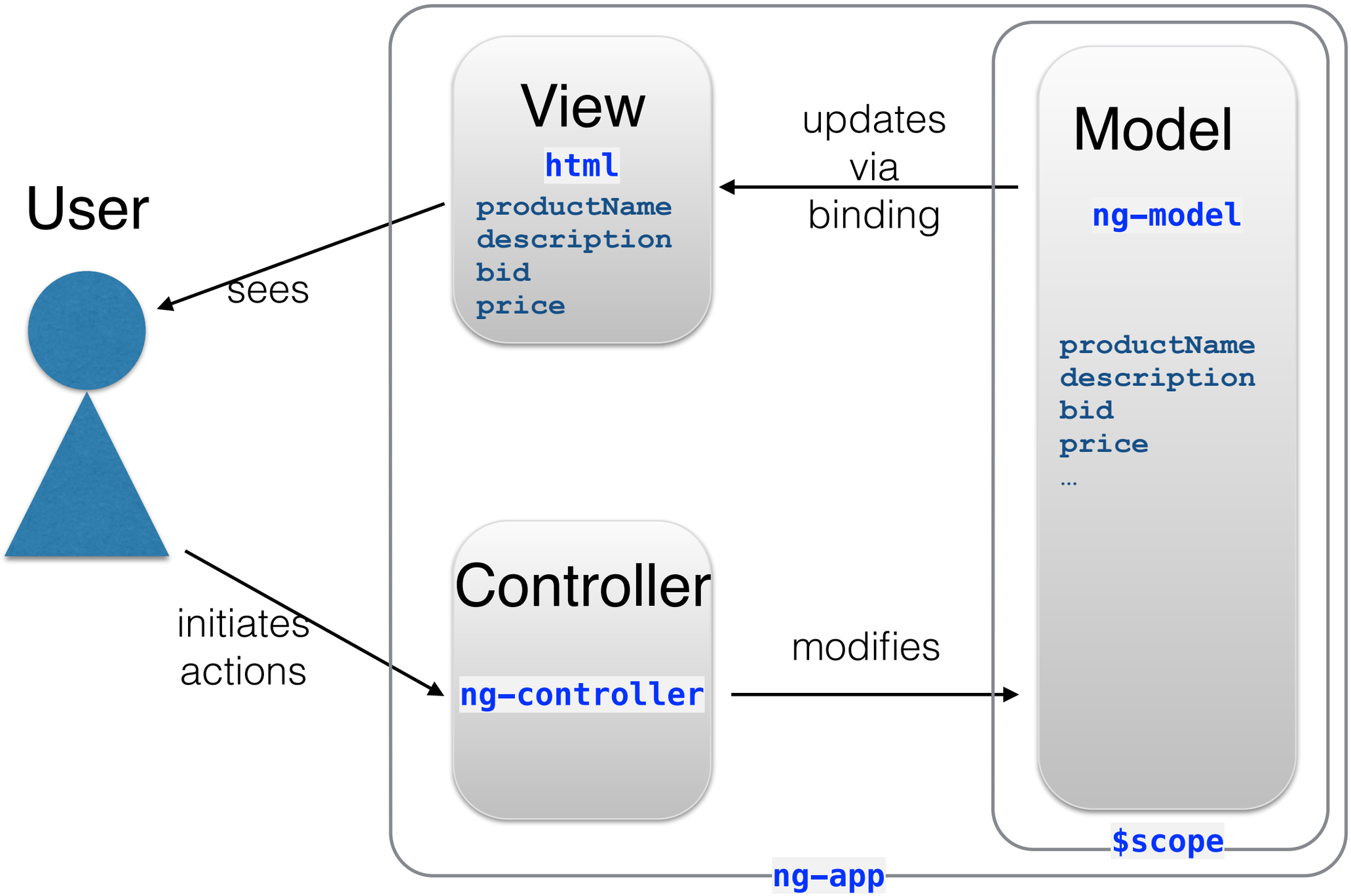
Professional SPA features

- Modularization
- Controllers handles DOM manipulations and AJAX
- HTML templating
- Routing with deep linking
- Real time communication via Web sockets
- Use of HTML5 Local storage

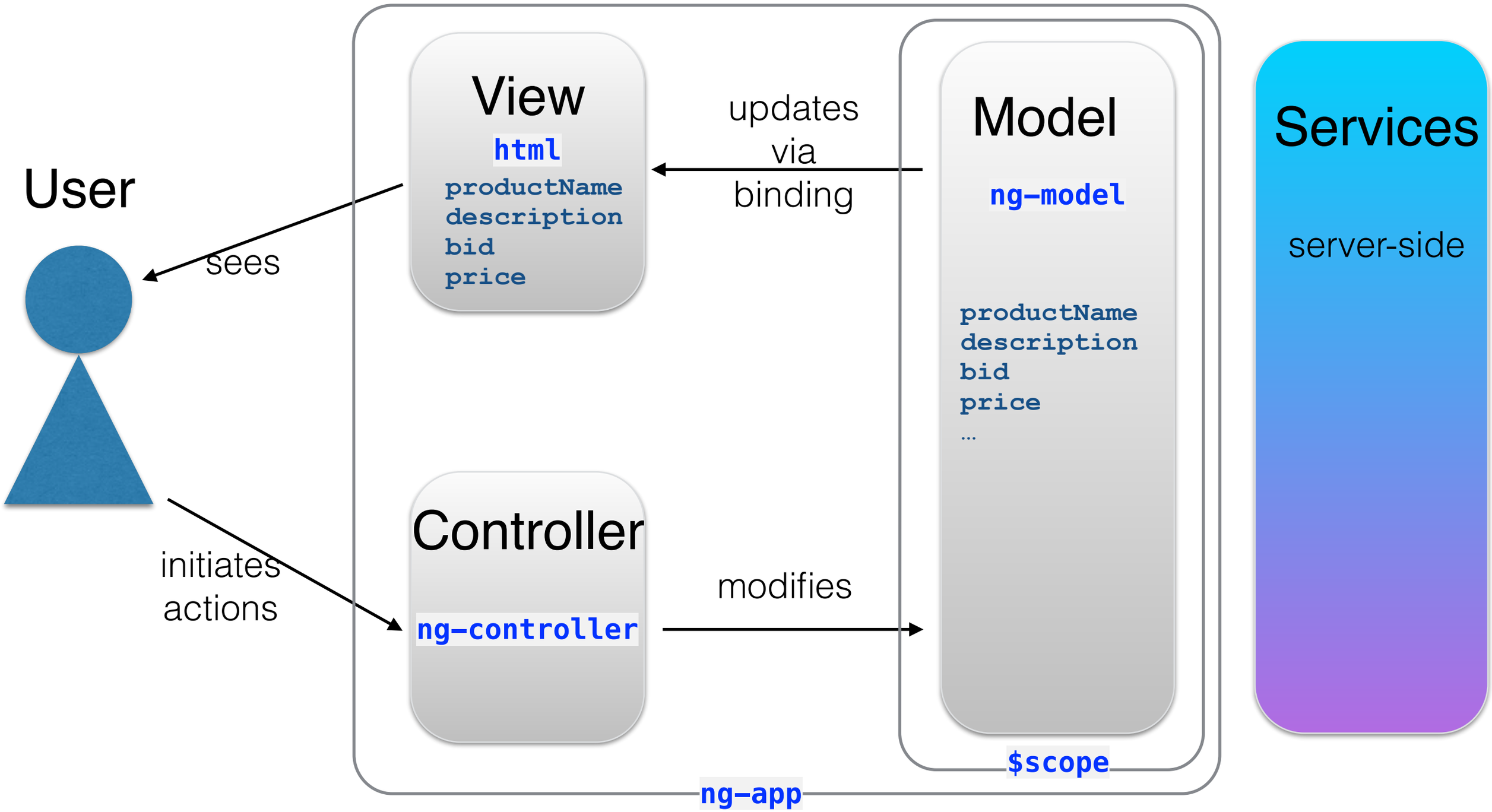
Read more about SPA at

https://en.wikipedia.org/wiki/Single-page_application

MVC in Angular App



MVCS



Minimal AngularJS App

- Single HTML file:

```
<!DOCTYPE html>

<!-- STEP 2: Bootstrap AngularJS -->
<html ng-app>
<head lang="en">
    <meta charset="UTF-8">
    <title>Unit 2. Walkthrough 2.</title>
</head>
<body>
    <!-- STEP 3: Define a new variable in the root scope. -->
    <p>Type here: <input type="text" ng-model="message"/></p>

    <!-- STEP 4: Add data binding to the "message" variable. -->
    <p>Should appear: <strong>{{ message }}</strong></p>

    <!-- STEP 1: Add AngularJS dependency. -->
    <script src="angular.js"></script>
</body>
</html>
```

Defines the scope of Angular app

Creates a variable in the model

Binds view to an existing variable from the model

Walkthrough 1

Create an empty IDEA project and import the module from the handouts according to instructions from import_code_manual.

Use directory **`walkthroughs/w1`** from the handouts as the starting point. Open **`w1.html`** file in the IDEA's editor:

1. Include AngularJS library using **`<script>`** tag:
   <script src="angular.js"></script>

2. Bootstrap AngularJS app by addin **`ng-app`** directive to <html> tag:
   <html ng-app>.

3. Define a new variable in the root scope and enable two-way data-binding for the input element as using **`ng-model`** directive: <input type="text" ng-model="message"/>

4. Inside the element add data-binding to the **`message`** variable using expression:
   {{ message }}

5. Open **`w1.html`** file in Chrome browser and see if the binding works.

6. Compare Angular and pure JavaScript versions in the solutions directory

Walkthrough 1: Comparing Angular and JavaScript versions

w1.html (/Users/yfain11/Downloads/Unit2_Handouts/Unit2_CodeSamples/walkthroughs/w1-solution)

```
<!DOCTYPE html>

<!-- STEP 2: Bootstrap AngularJS -->
<html ng-app>
<head lang="en">
    <meta charset="UTF-8">
    <title>Unit 2. Walkthrough 1.</title>
</head>
<body>
    <!-- STEP 3: Define a new variable in the root scope. -->
    <p>Type here: <input type="text" ng-model="message"/></p>

    <!-- STEP 4: Add data binding to the "message" variable. -
    <p>Should appear: <strong>{{ message }}</strong></p>

    <!-- STEP 1: Include AngularJS -->
    <script src="angular.js"></script>
</body>
</html>
```

w1_pureJS.html (/Users/yfain11/Downloads/Unit2_Handouts/Unit2_CodeSamples/walkthroughs/w1...

```
<!DOCTYPE html>
<html >
<head lang="en">
    <meta charset="UTF-8">
    <title>Unit 2. Walkthrough 1. Old fashioned way</title>
</head>
<body>

    <p>Type here: <input id="inputField" type="text"/></p>

    <p>Should appear: <span id="spanArea"></span></p>

\<script>
    var inputField = document.getElementById('inputField');
    var spanArea = document.getElementById('spanArea');

    inputField.addEventListener('keyup', function(){

        spanArea.innerHTML = inputField.value;
    });
</script>

</body>
</html>
```

w1-solution/w1.html

w1-solution/w1_JS.html

JavaScript Modules

```
// Wrap everything into a self-executable anonymous function
(function () {
  // Declare ‘use strict’; directive.
  'use strict';

  // Create constructor function for the object.
  var ProductService = function ($http) {
    // Instance attributes go here:
    this.$http = $http;
  };

  // Instance methods go here:
  ProductService.prototype = {

    getProducts: function () {},

    find: function () {}
  };

  // “Static” members
  ProductService.$inject = ['$http'];
}());
```

Angular Players

- **Controllers** - handle user interactions, orchestrate models and services
- **Directives** - assign custom behavior to HTML elements
- **Filters** - format the expression's value
- **Binding Expressions** - code snippets placed in curly braces within HTML template
- **Modules** - each app consists of one or more modules

Sample Angular App

index.html

```
<!DOCTYPE html>
<html ng-app="auction">
<head lang="en">
    <meta charset="UTF-8">
    <title>Unit 2. Walkthrough 3.</title>
</head>
<body ng-controller="IndexController">
    <p>Type here: <input type="text" ng-model="model.message"/></p>
    <p>Should appear: <strong>{{ model.message | uppercase }}</strong></p>

    <script src="angular.js"></script>
    <script src="w3.js"></script>
</body>
</html>
```

Directives

Filter

The Binding Expression

The only var on global space

w3.js

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

auction.controller('IndexController', function ($scope) {
    $scope.model = {
        message: 'Initial message'
    };
});
```

Module

Injecting a scope into the controller

Controllers

- Handle user interactions
- Creates the model variables on the $scope object
- Provide data-binding source (model) for views
- Receive control during routing
- Never manipulate the view directly
- A view can have multiple controllers

How to use a controller

```
'use strict';

(function () {
  var MainController = function ($scope) {
    $scope.awesomeThings = [
      'HTML5 Boilerplate',
      'AngularJS',
      'Karma' ];
  };

  angular.module('store').controller('MainController', MainController);
}());
```

A scope shares variables between view and controller

```
<div ng-controller="MainController">
   <ul>
      <li ng-repeat="thing in awesomeThings">{{ thing }}</li>
   </ul>
</div>
```

A new variable `thing` is created for each `ng-repeat` iteration

Scopes

- A scope shares variables between view and controller
- A scope is a place where the model data are located
- A scope is the **only** source for data-binding
- The root scope is implicitly created for the entire app
- Child scopes can be created for controllers and directives
- If a data-binding variable isn't found in the child scope, Angular looks in the prototypal inheritance chain.

Directives

- Attach behaviour to the DOM elements
- Can have visual and non-visual effects (`ng-controller` vs `ng-repeat`)
- Directives offer:
  - UI decomposition
  - Reusable components

Directive Usages

- 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">`

Routing

- Enables deep-linking for single-page applications
- Uses the fragment identifier **#,** doesn't reload the page
- Supports HTML5 History API as well as "hashbang" (/#!)
- Is represented as `$route` service:
- Maps URLs to views (HTML templates) and controllers

Lives in 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

```
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: '/'
      });
  }]);
```

One-to-one mapping

Path to a partial view (HTML template)

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

Default route

Never contains "hashbang"

Filters

- Transform 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'];
```

Filter name

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

Filter implementation

```
angular.module('auction').filter('joinData',
  (array, separator) => array.join(separator));
```

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

Modules

- Help avoiding monolithic applications
- Split one JS file into several
- Allow to specify dependencies
- Can be loaded in any order or in parallel

Module Usages

```
// Create a module
var auction = angular.module('auction', []);

// Get an existing module
var auction = angular.module('auction');

// Create a module with dependencies
var auction = angular.module('auction', ['ngRoute']);
```

```
<script src="angular-route.js"></script>
```

Identifying the View

- Add attribute marked with ng-view directive inside the HTML element
- Declare a single ng-view for the entire app
- ng-view is 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.

Walkthrough 2

Developing Online Auction in Angular

Walkthrough 2

- In this walkthrough we will use Angular to develop the Home and Search pages. We'll use the $http object to read the data from the json files.

- In this version we'll implement navigation between the Auction Home and Search pages *without* using routing. We will achieve the same result as in the project for Unit 1, but using various Angular directives:

  - **ng-init** - to declare and initialize a variable on the current scope
  - **ng-include** - to download HTML template, apply data to the template and insert generated markup inside the HTML element it's attached to

- Use directory `walkthroughs/w2` from the handouts as the starting point.

- Open `walkthrough_2_guide.html` file from the handouts and follow instructions.

Tools

The Tools We Use

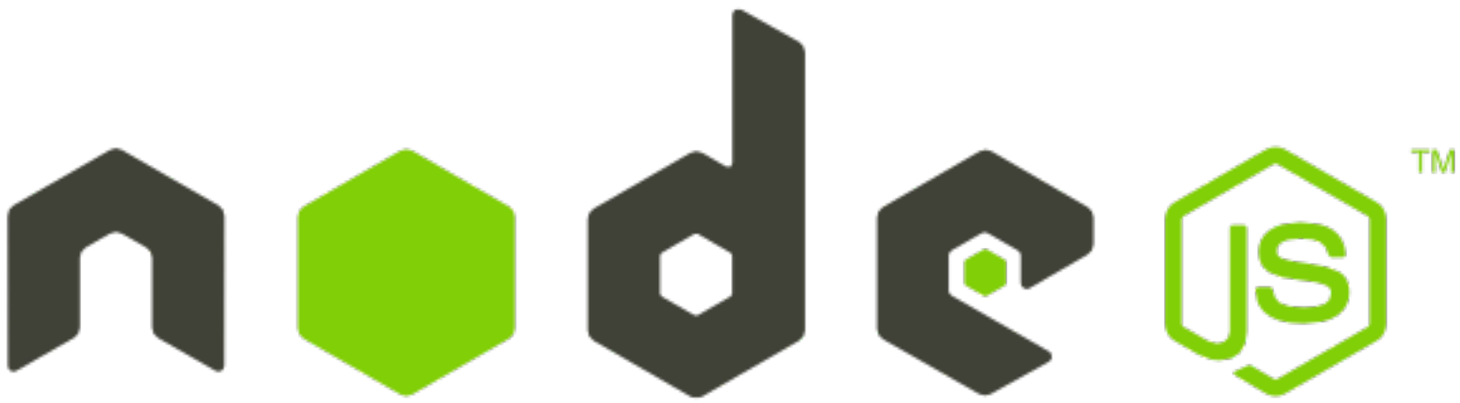
- **node.js** - JS framework plus a **runtime** for all development tools listed below

- **npm** - node package manager used for installing and managing development tools

- **yeoman** - a scaffolding tool used to generate the initial structure of an application

- **bower** - package manager for **application** dependencies

- **grunt** - a build tool; We'll use it to automate the build processes



Node.js

- A platform for executing JavaScript code outside the web browser
- Built on top of Chrome's JavaScript engine - V8
- Uses event-driven non-blocking I/O model
- Allows writing server-side JavaScript applications that access files, support TCP, UDP, HTTP, SSL, and compression.

- We will use Node.js as a runtime for JavaScript development tools
- Has pre-built installers for all popular platforms at nodejs.org/download/
- Instructions for installing from package managers are here: https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager

npm

- npm is a Node.js package manager for development-tools and their dependencies

- Has a repository of 50K+ packages at https://www.npmjs.org

- Use Node.js installation instructions to install npm

- To run, type **`npm`** in the command window

package.json

- package.json is a file that describes npm package. We'll use json properties to configure dependencies.
- Two types of dependencies:
  - **dependencies** - packages that are needed if you want to publish your app in the npm registry
  - **devDependencies** - packages that are used only during development only (e.g. Grunt, development web server, etc.)

- To install all dependencies listed in package.json: `npm install`
- To install a new dependency (e.g. grunt-ts): `npm install grunt-ts`
- Use `--save` and `--save-dev` to automatically update package.json along with installing a new package.

package.json example

- Lists all development dependencies

```
{
  "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"    The 0.2.1 version
  }
}
```


YO

GRUNT

BOWER

Yeoman

- A scaffolding tool for generating the initial project structure.
- It's an npm package and is installed with `npm install -g yo`
- Has a rich collection of generators at *yeoman.io/community-generators.html*
- To install a generator: `npm install -g generator-angular`
- To run, type `yo` in the command window, e.g. `yo angular`

Bower

- Bower is a package manager for the **application** dependencies
- Package can have any layout and any type of assets
- We use bower for runtime dependencies (the ones we actually use in code)
- Huge collection of packages

- To install bower: `npm install -g bower`
- To run, type `bower` in the command window

Bower: bower.json

- bower.json describes the application's package. We'll use json propertied to configure application *dependencies*.
- Two types of dependencies:
  - **dependencies** - packages needed for production version of the app
  - **devDependencies** - packages needed 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 ranges

- 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

- Grunt is a build tool for web apps
- We use it to automate all development processes
- Grunt is configured as a dependency in npm's package.json
- We need to install command-line interface to grunt:

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

- To run, type `grunt` in the command window
- Grunt plugins installed with npm:

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

Gruntfile.js

- A JavaScript that loads Grunt plugins containing tasks
- Defines all configurations for tasks that Grunt can execute
- Can create new tasks
- Each task can be invoked separately with `grunt` *`task`*
- Optionally a target can be specified: `grunt` *`task:target`*

Sample Gruntfile.js

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`

```
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']);

};
```

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'
}
```

Typical Workflow

- npm install
- bower install
- grunt build

Walkthrough 3

Mastering Tools. Reimplementing Home Page.

Open `walkthrough_3_guide.html` file from the handouts and follow instructions.

Frequently used AngularJS services

- $scope - access to the current scope
- $rootScope - access to the root scope
- $http - HTTP requests
- $location - to work with window.location
- $window - to access window object

AngularJS directives you may need for the homework

- **ng-app** - determines the scope of AngularJS app and automatically bootstraps the app

- **ng-view** - tells AngularJS which HTML element should be used to include rendered view while navigating using routing service

- **ng-repeat** - iterates through a collection of objects, instantiates a template it's attached to once per item

Additional Resources

- AngularJS Developer Guide - http://docs.angularjs.org/guide
- Collection of articles - http://www.ng-newsletter.com/posts/
- Collection of AngularJS learning resources - https://github.com/jmcunningham/AngularJS-Learning
- AngularJS Google style guide (disregard Google Closure part)
- AngularJS community style guide
- AngularJS docs on directives: https://docs.angularjs.org/api
- Grunt plugins can be found here: npmjs.org

The Next Project Review

This project is about refactoring the app to use routing instead of `ngInit` and `ngInclude` directives to navigate among the pages. Use directory `homework2` from the handouts as the starting point.

1. Add `angular-route` library as dependency to `bower.json`. Run `bower install` to download the library locally. Run `grunt wiredep` to add reference to the library into `index.html`.

2. Open `app/scripts/app.js` file and add `ngRoute` module as dependency for the main application module - `auction`. Configure `$routeProvider` the same way we did in class in *AngularJS Routing* section, but use your own `templateUrl` and controller values. Use `controllerAs` option to automatically expose controller's fields and methods on the scope.

3. Open `app/views/index.html` file. Replace `ngInit` and `ngInclude` directives with `ngView` directive. Replace `ngClick` with `ngHref` directives for Search button and brand name link that we use for navigation. `ngHref` should have correct value as we discussed in *How Navigate to a Route* section.

4. After you complete run grunt build command, it will generate `dist` directory in the root directory of your app. The content of `dist` can be deployed on the Web.

5. Watch the video explaining tools we used for this project (it was recorded for another group) http://meet46948477.adobeconnect.com/p670ho42mc2 . The passcode is tools.

5. Review a proposed solution at http://farata.github.io/modernwebdev-showcase/homework2/dist/#/